Motion Analysis and Performance Improved Method for 3D LiDAR Sensor Data Compression

Chenxi Tu, Eijiro Takeuchi, Alexander Carballo, Chiyomi Miyajima, and Kazuya Takeda,

***Abstract*—Continuous point cloud data is being used more and more widely in practical applications such as mapping, localization and object detection in autonomous driving systems, but due to the huge volume of data involved, sharing and storing this data is currently expensive and difficult. One possible solution is the development of more efficient methods of compressing the data. Other researchers have proposed converting 3D point cloud data into 2D images, or using tree structures to store the data. In a previous study targeting streaming point cloud data, we proposed an MPEG-like compression method which utilizes simultaneous localization and mapping (SLAM) results to simulate LiDAR's operating process. In this paper, instead of imitating MPEG, we propose new strategy for more efficient reference frame distribution and more natural frame prediction, and use a different algorithm to encode the residual, greatly improving the algorithm's performance and its stability in different scenarios. We also discuss how various parameters affect compression performance. Using our proposed method, streaming point cloud data collected by LiDAR sensors can be compressed to 1/50th of its original size, with only 2 cm of Root Mean Square Error for each detected point. We evaluate our proposed method by comparing its performance with several other existing point cloud compression methods in three different driving scenarios, demonstrating that our proposed method outperforms them.**

***Index Terms*—Point cloud, Data compression, 3D LiDAR**

I. Introduction

A Point cloud is a collection of points spread across a 3D space, which can be used to represent a local environment. Point cloud data collected by LiDAR sensors is currently being used for driving environment representation by many autonomous driving systems [1], including those used by Google and Uber [2], and the LiDAR systems used to collect this data are also becoming cheaper. For example, Velodyne's HDL-64, one of the most popular 3D LiDAR sensors, used to cost around $100,000, but a less expensive, less powerful version, the VLP-16, now costs only about $4,000, and Velodyne predicts the cost may fall to $50 in the future. These trends suggest that the use of point cloud data is becoming an industry standard and that it will continue to be used in autonomous driving applications in the foreseeable future. Shared and stored streaming point cloud data are also likely be important components of accident investigation and future V2V/V2X systems.

Streaming point cloud data is a type of "big data", however. One hour of point cloud data from the HDL-64 sensor mentioned above could represent over 100GB of data, which is too large to realistically share or store using currently available technology. Thus, developing methods of compressing this data has become an indispensable task.

Most previous studies on the compression of point cloud data from LiDAR systems have generally used one of two strategies. The first is to convert 3D point clouds into a 2D format (height map, panorama, etc.) while using an existing 2D compression approach to reduce redundancy [3], [4], [5]. Using a tree structure is another widely used strategy for compressing point clouds [6], [7].

Compressing streaming data is much more complex than compressing static data, in part because of the problem of temporal redundancy. Converting raw LiDAR packet data into a 2D format and using some of the frames to predict the others is an efficient strategy [8]. In contrast to video data, pixels in adjacent frames of 2D formatted LiDAR data not only translate but their values (usually representing distance) also change according to the driving scenario. In addition, as low-resolution sampling of motion through real-world environments, some pixels (points) may not appear in the following frame. Furthermore, adjacent pixels do not always have similar motion because laser modules in the LiDAR are usually not distributed in a line or directed at a fixed point with only linear variation in their orientation. All of these phenomena make it difficult to directly use the pixels of reference frames to predict the remaining frames, as is done during video compression. We need to understand the motion of each point and determine which point is being detected by which laser beam.

In our previous work [9], we proposed using SLAM information and LiDAR simulation to solve these problems. Influenced by video compression algorithms, we continued to search for neighboring pixels in reference frames to find appropriate reflection points, and used Huffman coding to deal with the residuals. But in driving scenarios, the value and position of a pixel may vary greatly between adjacent frames, so searching for neighboring pixels may not help us find the point we are actually looking for. Additionally, Huffman coding does not take into account spatial redundancy. Furthermore, our previous method uses more reference frames than are really needed, which needlessly increases the volume of the data.

To resolve these issues, this paper expands upon our previous study [9], making the following four, main contributions:

All of the authors are with the Department of Intelligent System, School of Informatics, Nagoya University, Furo-cho, Chikusa-ku, Nagoya 464-8603 Japan.

C.Tu tu.chenxi@g.sp.m.is.nagoya-u.ac.jp
E.Takeuchi takeuchi@coi.nagoya-u.ac.jp
A.Carballo alexander@g.sp.m.is.nagoya-u.ac.jp
C.Miyajima miyajima@nagoya-u.jp
K.Takeda kazuya.takeda@nagoya-u.jp
Manuscript received ;

- A new motion analysis strategy, which optimizes the number and locations of the reference frames, allowing for more efficient utilization of the information in these frames.
- A new searching strategy using a grid map, which allows for more natural and stable frame prediction.
- A new encoding method based on JPEG-LS, further reducing spatial redundancy in the coding module.
- More discussion about the influence of parameters, and more experiments.

This paper is organized as follows: In Section II, we discuss related work in the area of point cloud compression. In Section III, we provide an overview of our proposed algorithm, while the details of each module are explained in Sections IV, V and VI. In Section VII, we visualize decompressed point clouds and compare our proposed method with various existing compression methods quantitatively, using driving data collected in various driving contexts.

II. Related Work

When compressing point cloud data, the key is to reduce spatial redundancy. However, a point cloud itself is sparse and disordered, which makes it difficult to compress directly in its original format. Before compression, point clouds are usually converted into a different format and then quantized and encoded to reduce spatial redundancy. Several studies proposing various approaches to formatting and compressing point cloud data have been published.

3D modeling has been an important subject in the field of computer graphics research for some time. 3D mesh compression [10], which has been studied for more than 25 years, can be regarded as the forerunner of 3D point cloud compression, and it directly inspired several early point cloud compression methods. The earliest mesh compression methods targeted static 3D objects, and some very early methods were only single-rate mesh coders [11]. In order to transmit 3D objects via communication channels of limited bandwidth, progressive mesh coders were developed [12]. Inspired by 3D mesh compression, various 3D point cloud compression methods were then developed, many of which were height map-based methods which used strategies similar to the one used in 3D mesh compression. Pauly and Gross [13] were the first to propose using height maps to process point cloud data. Other approaches, such as those of Hubo et al. [14] and Ochotta and Saupe [15], [16] were then developed to utilize height mapping for point cloud compression using different coding methods. Schnabel et al. [17], [18] suggested using geometric primitives such as planes, spheres, cylinders, cones and tori in conjunction with height maps to compress point cloud data. A method of real-time, continuous point cloud data compression based on a height map was then proposed by Golla and Klein [3]. Their method used pre-processing to split the data into chunks, and then utilized JPEG to compress a 2D matrix, outperforming previous height map-based methods. Other methods did not use height maps per se, but also divided point clouds into patches, such as the method proposed by Kalaiah and Varshney [19], which divided point clouds into statistical representations which were constructed using clustering-based hierarchical principal component analysis (PCA) of the point geometry. The method proposed by Morell et al. [20] used triangles to represent each plane.

Some researchers have tried converting 3D point cloud data into a 2D format, rather than decomposing one frame of point cloud data into multiple images, as with height map-based methods. Houshiar and Nüchter [4] proposed mapping point clouds onto panorama images using equirectangular projection. Kohira and Masuda [5] mapped point cloud data onto 2D pixels using GPS time and the parameters of the laser scanner. Directly converting 3D point cloud data into a 2D image will inevitably result in the loss of information, therefore some methods utilize the LiDAR scanner's operating principle and target the raw LiDAR data, known as packet data. Yin et al. [21] targeted this raw packet data and focused on the LiDAR system's data format. In a previous study [8], we converted packet data into a 2D matrix and then used an existing image compression method to compress the data. In addition to image compression-based methods, learning-based methods have also demonstrated their usefulness in 2D data compression applications, even outperforming existing methods [22]. Methods using auto-encoders [23] or RNNs [24] also have potential for compressing 2D formatted LiDAR data, especially packet data, which is usually in an irregular 2D format. Instead of converting point cloud data into a 2D format, Hayes also took into account the operating principle behind LiDAR and proposed converting point clouds into a form that is susceptible to wavelet transformation [25].

Other compression methods store and compress 3D point cloud data more directly, without trying to tie clustered points back to a mesh. Krüger et al. [26] used spherical grids to pack a 3D space. Researchers have also used tree structures. Gumhold et al. [27] used prediction trees, and Hubo et al. [6] used kd-trees, while Schnabel and Elseberg [28], [29] used octrees, a method which is now widely used in the 3D graphics domain. Kammer et al. [7] also used an octree-based method to represent the spatial location of each point, using a double-tree structure to calculate the difference (exclusive or) between the octrees of adjacent frames, allowing time series redundancy to also be reduced for streaming. Kammer's method can be used to control information loss quantitatively and allows the real-time compression of streaming point cloud data. Since the open code for this method is available to other researchers via the Point Cloud Library, this method is currently a popular method for the compression of streaming point cloud data. Thanou et al. [30] proposed a method of streaming point cloud data compression which includes a special motion estimation module; here, octrees are used to divide 3D point clouds into many occupied voxels, while the points in each voxel (leaf) are represented by a weighted, undirected graph. Most previous methods have been designed to compress a single point cloud, and have only dealt with spatial redundancy, therefore they are not suitable for the compression of streaming data. The streaming point cloud compression methods which have been proposed, such as [8], [7], [30], aim at reducing temporal redundancy as well as spatial redundancy. However, methods

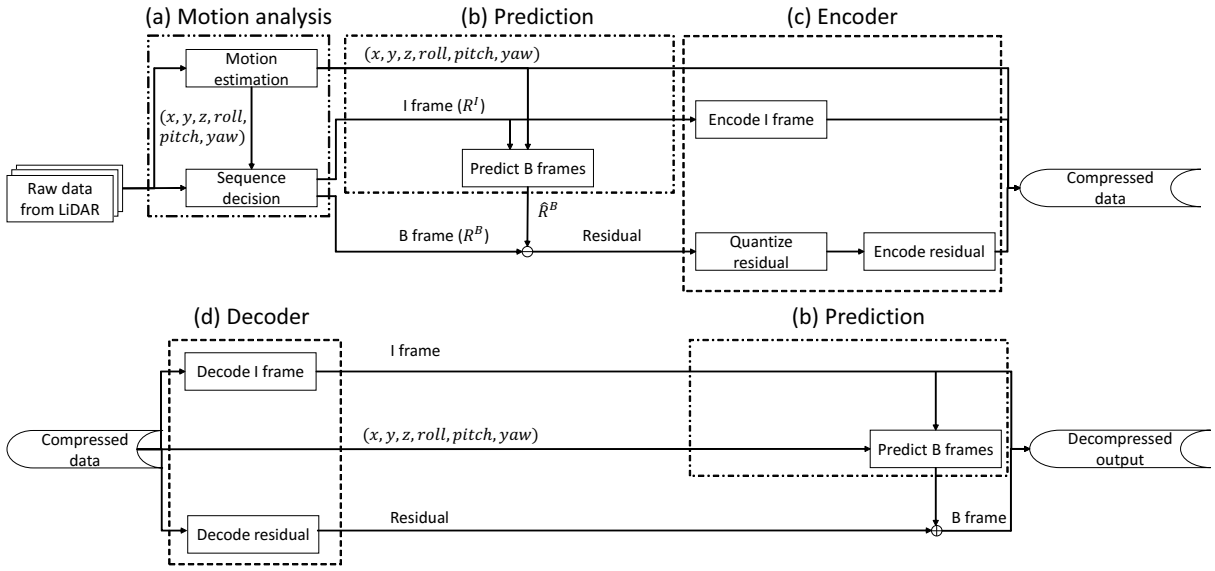

Fig. 1: Flowcharts illustrating the proposed method. The upper flowchart shows the compression process while the lower one shows decompression. For definitions of R^I, R^B and $\hat{R}^B$, please refer to Section IV.

which only compress adjacent frames [7], [30] have been unable to achieve superior performance. Using less reference frame data to predict more information is an important strategy for improving the efficiency of streaming data compression. It is also important to use reference frame information to accurately predict the content of the other frames.

In our previous work [9], we proposed predicting LiDAR frames by simulating the process used by LiDAR to collect data, which is more natural than using video compression methods on 2D formatted LiDAR data [8]. However, we still employed a searching strategy and coding method used for video compression, which led to unstable prediction and unsatisfactory compression performance. In this paper, instead of imitating video compression techniques, we propose a new method of searching for appropriate points for frame prediction more similar to what a LiDAR system actually does. We also propose a motion analysis strategy in order to use less reference frame data for more efficient prediction, which is important when compressing streaming data. Furthermore, we use a new encoder to reduce spatial redundancy. All of these improvements work together, enabling the method proposed in this study to greatly outperform the method proposed in our previous work [9].

III. Overview

Our proposed method can be divided into three components: (a) motion analysis, (b) prediction and (c) encoding, as shown in Fig. 1. The motion analysis module selects reference frames from the data, the prediction module reduces temporal redundancy, and the encoder reduces spatial redundancy among individual frames.

(a) Motion analysis is the first step in the compression process, and is composed of two processes: motion estimation and sequence decision.

Motion estimation here means obtaining the yaw, pitch and roll (x, y, z) of the LiDAR data in each frame. We use Simultaneous Localization and Mapping (SLAM) based on the Normal Distribution Transform (NDT) [31][32] to obtain this information. Giving a pair of point clouds, SLAM can calculate their relative positions and orientations by searching for the best shape matching. The same results can also be obtained by using a special device, such as an IMU, or by using various other matching algorithms. Using motion information, the sequence decision module can optimize the number and locations of the reference frames.

Our use of a motion analysis module is a unique feature of our proposed method, compared with other streaming data compression methods. Most other methods utilize a prediction + encoder structure, but the reference frames are usually chosen at a constant interval.

(b) Prediction is the second step of our method. In the following sections, we will call reference frames I-frames (intra-coded frames) and the remaining frames, which are predicted using the reference frames, will be called B-frames (bi-directionally predicted frames), a conventional designation used for video compression. The prediction module attempts to predict the B-frames, using data from the I-frames and the motion estimation results. The residuals between the predicted B-frames and the true B-frames are then calculated.

(c) Encoding is the last step of our compression method. During coding, the I-frames are compressed losslessly, while the residuals of the B-frames are quantized and coded. In this study, we used a different coding method than in our previous paper. Our improved method considers spatial redundancy among residuals, improving performance.

(d) The decoding process can be thought of as the inverse of the process conducted during encoding.

IV. Prediction

Prediction is the most important step for many streaming data compression approaches, since it can be used to efficiently

reduce temporal redundancy. In our method, an original data sequence is divided into I-frames and B-frames before prediction occurs, so that some number of B-frames are enclosed between pairs of I-frames in sequences such as "IBBBIBBBI". The purpose of the prediction module is to use each pair of I-frames to predict the enclosed B-frames, as shown in Fig 2(a).

To deal with the sparsity and disorder of point clouds, we store the collected LiDAR packet data in a 2D format. Then, to predict the packet data of the enclosed B-frames, we combine the point clouds of the surrounding I-frames to create a 3D representation of the scanned environment, and simulate laser beam emissions to predict the reflected LiDAR signals within the B-frames.

The whole process can be divided into four steps: coordinate system unification, beam simulation, reflection point searching and beam mapping. In the following subsections, we will discuss each step in detail. Figs. 2(b) and (c) illustrate the prediction process.

We use P^I and P^B to represent point clouds, which are 3D arrays, of I-frames, B-frames respectively. Points in P^I and P^B are denoted by $P^I_{i\in[1...N]}(x,y,z)$ and $P^B_{i\in[1...N]}(x,y,z)$, where N represent the number of points per frame. R^I and R^B are 2D matrices which denote the raw packet data of P^I and P^B, respectively. Elements in R^I and R^B are denoted by $R^I_{i\in[1...M],j\in[1...K]}(d)$ and $R^B_{i\in[1...M],j\in[1...K]}(d)$, respectively, where d represents the distance information of one reflection point. Subscript i, representing a row, has a mapping relationship with the LiDAR beam pitch angle, while j, representing a column, has a mapping relationship with its yaw angle. More details are provided in this paper's Appendix. M represents the number of points that the LiDAR system obtains per emission, which is usually equal to the number of laser emitters in the LiDAR system. K represents the number of laser emissions per frame, which depends on the frequency of rotation in the case of spinning LiDAR emitters, therefore $N = M \times K$. The calibration process used to convert raw packet data R into point cloud P is denoted as $P = f(R)$.

A. Coordinate system unification

As shown in Fig. 2, 3D LiDAR systems sample an environment by continuously emitting pulsed laser beams and collecting the reflected signals, capturing the distance to each surrounding object. To predict B-frames using a LiDAR simulation, we need to obtain information about the surrounding environment.

In our proposed method, each B-frame is surrounded by two I-frames, one representing the environment in the future and one representing the environment in the past. By combining the point clouds of the two I-frames, we can roughly estimate the 3D environments of the enclosed B-frames.

The point cloud data of each frame has its own coordinate system, based on the current orientation and location of the LiDAR unit. The coordinate systems of these frames need to be unified in order to link the I-frames and B-frames together. Using motion estimation information obtained from SLAM, we can determine the x, y and z coordinates for each frame/coordinate system, as well as yaw, pitch and roll, in relation to a global coordinate system. Using Equation (1), P^I can then be unified in relation to the global coordinate system:

$$\begin{bmatrix} x_g \\ y_g \\ z_g \end{bmatrix} = R_{\text{yaw}} \cdot R_{\text{pitch}} \cdot R_{\text{roll}} \cdot \begin{bmatrix} x_t \\ y_t \\ z_t \end{bmatrix} + \begin{bmatrix} c_x \\ c_y \\ c_z \end{bmatrix} \tag{1}$$

- x_t, y_t, z_t are the Euclidean 3D x, y, z coordinates of the point cloud at time t
- x_g, y_g, z_g are the corresponding x, y, z coordinates of the global coordinates system
- c_x, c_y, c_z are the translation obtained using SLAM
- $R_{\text{roll}}, R_{\text{pitch}}, R_{\text{yaw}}$ are the rotation matrices for roll, pitch and yaw angles obtained using SLAM

Then, using the inverse of Eq. (1) and motion estimation information from the B-frames, the point clouds of the two I-frames, P^{I_1} and P^{I_2}, can be unified to the enclosed B-frames: $\widetilde{P}^{I_1}$ and $\widetilde{P}^{I_2}$. By combining the points in $\widetilde{P}^{I_1}$ and $\widetilde{P}^{I_2}$ together, we can obtain a rough 3D environment, $\{\widetilde{P}^{I_1}, \widetilde{P}^{I_2}\}$, as shown in Fig. 3(a).

B. Beam simulation

3D LiDAR systems collect point cloud data by emitting pulsed laser beams and capturing the reflected signals. To predict a B-frame's packet data R^B from $\{\widetilde{P}^{I_1}, \widetilde{P}^{I_2}\}$, we need to simulate the LiDAR system's laser beams and calculate the reflection signals, i.e., the measured distances, for each laser. For our purposes, we must know the rotation angle of the LiDAR scanner when the beams are being emitted. This information is stored directly in the LiDAR system's packet data and has a mapping relationship with the columns of the 2D raw packet data matrix. More details are provided in the Appendix of this paper.

LiDAR systems have many laser beam emission modules located at different positions in the scanner, so the point of origin of each particular laser beam is different, i.e., the simulated origin of the beams is not point (0,0,0). By using the manufacturer's calibration file, which has the necessary correction parameters, for each distance value $R^B_{i,j}$ we can simulate the corresponding laser emission with a line $l_{i,j}$. Note that the LiDAR system's calibration file is used to convert the packet data into point cloud data, and that each type of LiDAR system has a unique calibration file.

C. Reflection point searching

Using 3D environment $\{\widetilde{P}^{I_1}, \widetilde{P}^{I_2}\}$ and simulated laser beams $l_{i,j}$, in this step we determine which points could be detected and signal reflected by it. Using Euclidean distance as our criterion, we select the closest point in $\{\widetilde{P}^{I_1}, \widetilde{P}^{I_2}\}$ to each simulated beam $l_{i,j}$ to represent the obstacle which is reflecting the beam back to the sensor.

Since a point cloud frame can have over 100,000 points, directly traversing all of the possible points to find the closest one would be prohibitive, thus an effective searching strategy is important to whole prediction process. In our previous study [9], in which we imitated image compression methods,

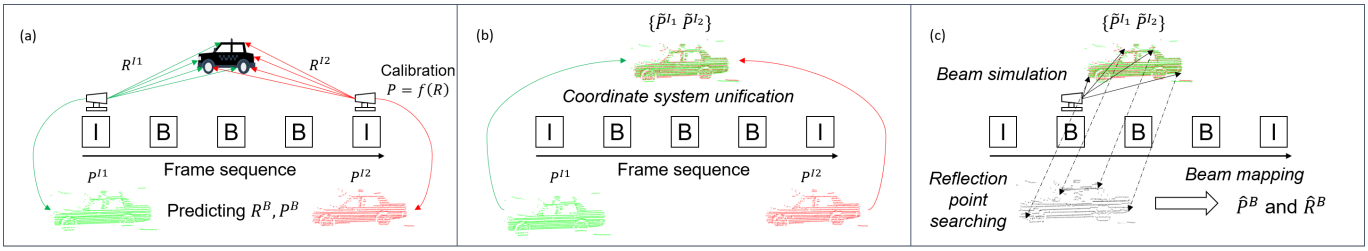

Fig. 2: Prediction process procedure.

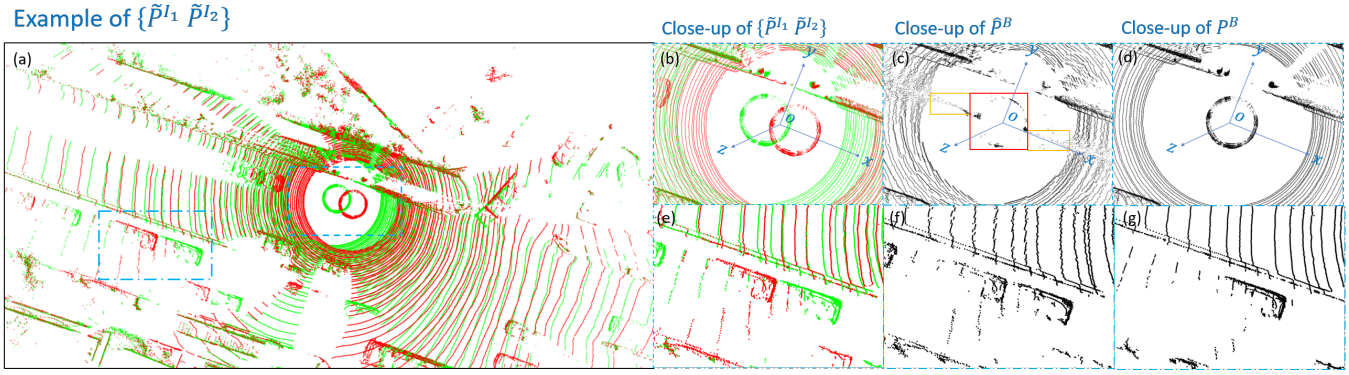

Fig. 3: Example of $\{\widetilde{P}^{I_1}, \widetilde{P}^{I_2}\}$ after coordinate system unification (Green points represent $\widetilde{P}^{I_1}$, red points represent $\widetilde{P}^{I_2}$), plus, continuing from left to right, close-ups of two patches in $\{\widetilde{P}^{I_1}, \widetilde{P}^{I_2}\}$, $\hat{P}^B$ and P^B.

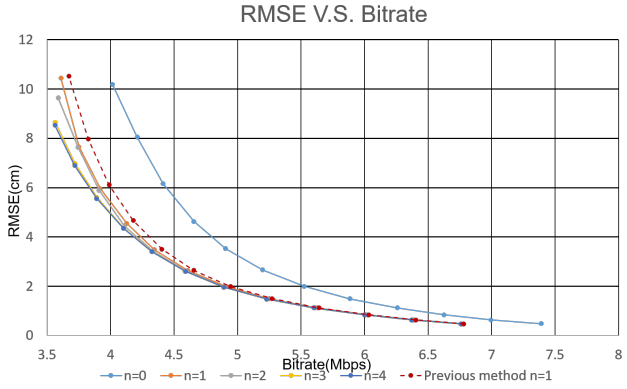

Fig. 4: RMSE of decompression vs. Bitrate of compression output for different values of n. Dotted line shows performance using the reflection point searching method proposed in our previous study [9].

for each beam $l_{i,j}$ we chose the points in $\widetilde{P}^{I_1}, \widetilde{P}^{I_2}$ which corresponded to the neighbor of $R^{I_1}_{i,j}$ and $R^{I_2}_{i,j}$ as our candidate points. However, if the LiDAR unit/vehicle is moving at a high rate of speed, these candidate points may not include the actual reflection points that we are seeking.

To resolve this issue, in this paper we propose a new searching strategy for more efficient and stable prediction. For $\widetilde{P}^{I_1}$ and $\widetilde{P}^{I_2}$, we build two polar grid maps, G^{I_1} and G^{I_2}, respectively. The columns of grid map G represent yaw angle while its rows represent pitch angle. By calculating the pitch and yaw angles for each point in $\widetilde{P}^{I_1}$ and $\widetilde{P}^{I_2}$, we can assign the points to grid maps. For each simulated beam $l_{i,j}$, by using its pitch and yaw angle we find the corresponding grid in G^{I_1} and G^{I_2}, and select points within that grid, and in its neighboring grids, as candidate points. Parameter n controls the range from which neighboring grids can be chosen as candidates for the reflection point. For a grid $G_{a,b}$, and its neighboring grids within range n, here means grids $\{G_{i,j}\}$, while $i \in [a-n, a+n], j \in [b-n, b+n]$.

As previously noted, the simulated beams do not originate exactly from $(0,0,0)$, which is the pole of the polar grid map, since the emitters are located at various places in the LiDAR scanner. This means that if we only rely on the yaw and pitch angles of the simulated beams, we may miss the true grid points we are looking for. For points which are far away from the emitter, this effect is very slight, but for points near the origin of the beam, the effect could be substantial, which is why we also need to include neighboring grids. Therefore, choosing the optimal value for parameter n is crucial.

To evaluate how n affects final compression performance, Root-Mean-Square Error (RMSE) between the decompressed data R' and original raw packet data R are used to measure information loss. The raw packet data only contains distance and angle information, and our compression method captures the angle information losslessly (for more details, please see this paper's Appendix). Storage volume needed after compression is measured in bitrate (Mbps). Fig. 4 shows RMSE vs. bitrate for different values of n

As shown in Fig. 4, when $n = 0$ our reflection point search is much less effective than when using other values, due to the bias in the starting point of each simulated beam, as explained previously. By including neighboring grids as candidates, $n = 1$ clearly outperforms $n = 0$. Although increasing n

leads to better compression/decompression performance, after n reaches a certain point further improvement becomes very modest. Thus, when n is sufficiently large, we can be almost 100% sure that the reflection point we are looking for is included in our search area, and that continuing to increase n further will not improve performance. Increasing n also greatly increases calculation cost. Since improvement after $n = 1$ is relatively slight, $n = 1$ is chosen as a cost-effective setting for our proposed method, thus for all of the following experiments $n = 1$.

By utilizing grid map with appropriate n, we can guarantee that all points around beam's emission angle are chosen as candidates during searching. In Fig. 4, the dotted line shows the performance of the reflection point searching method used in our previous study [9] when $n = 1$. When using about the same number of candidates, our new grid map-based approach clearly outperforms the previous method.

D. Beam mapping

Directly selecting reflection points from the environment built using I-frames is not enough, as these reflection points may not actually be located on the simulated beams. In order to ensure that all of the selected reflection points are located on beams, we map the reflection points to the corresponding simulated beams by fixing their y coordinates. Here, the y axis refers to the axis perpendicular to the direction in which the LiDAR unit is moving. After mapping, we acquire predicted B-frame point cloud $\hat{P}^B$. Fig. 3 shows an example of $\hat{P}^B$ in comparison with $\{\widetilde{P}^{I_1}, \widetilde{P}^{I_2}\}$ and P^B.

A point cloud of a driving scenario can be roughly divided into three parts: the static environment, dynamic objects and ground points. For objects in the static environment, such as buildings, the points in $\hat{P}^B$ are relatively accurate. However, for dynamic objects moving at high speeds, the proposed prediction method may contain obvious bias. For example, in Fig. 3(f), the close-up of $\hat{P}^B$, there appear to be two vehicles (using data from $\widetilde{P}^{I_1}$ and $\widetilde{P}^{I_2}$ shown by Fig. 3(e)), when in fact there is only one vehicle.

Ground points here refer to points reflected by the ground. As shown in Fig. 3(c) of $\hat{P}^B$, through beam mapping we could obtain some ground points which do not exist in $\widetilde{P}^{I_1}$ and $\widetilde{P}^{I_2}$, as shown in the red box. The remaining ground points could constitute many rough circles centered on o, like the points shown in Fig. 3(d) for P^B. This step may create some 'noise' however, as shown in the orange boxes.

By calculating the distance between each point in $\hat{P}^B$ and its corresponding emitter, we can obtain the predicted raw B-frame $\hat{R}^B$. Finally, we calculate the residual R^{res} between predicted B-frame $\hat{R}^B$ and actual B-frame R^B, and send it with R^I to the encoder.

V. Coding part

During coding, raw data matrix R^I and residual R^{res} between $\hat{R}^B$ and R^B are quantized and coded. R^I is losslessly compressed using a JPEG based compression method which is described in our previous work [8].

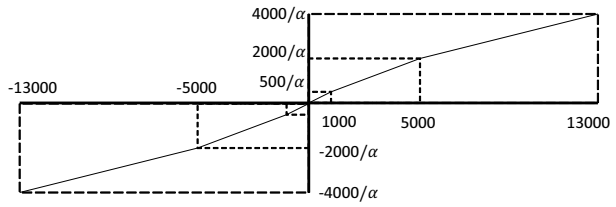

Fig. 5: Quantization of difference values. Up to three steps are used during quantization, depending on the size of the absolute difference value. After quantization, original data ranging from -13000 to 13000 cm is mapped to continuous integers range from $-4000/\alpha$ to $4000/\alpha$. Parameter α controls the range after quantization, and can also be used to tune compression ratio and loss.

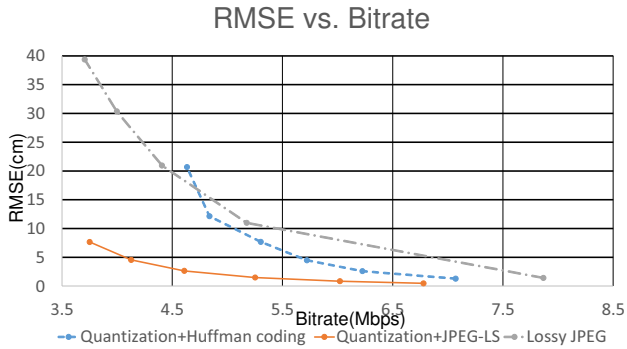

Fig. 6: Compression results using three different coding methods.

In [9], we used Huffman coding to code the residual. At that time, we thought of residual R^{res} simply as a sequence, but actually R^{res} between $\hat{R}^B$ and R^B also contains spatial redundancy. In this paper we propose utilizing a conventional lossless image compression method (JPEG-LS) as a coding tool when coding the residual, which further reduces spatial redundancy, resulting in lower data volume and more accurate decompression.

To encode the residual, we quantize each value $R^{res}_{i,j}$ as shown in Fig. 5 and then JPEG-LS [33] is used to encode the quantized values, allowing spatial redundancy to be further reduced. Thus, after simulating LiDAR's operating process to deal with temporal redundancy in the B-frames, and utilizing the 2D characteristics of the raw scanner data to deal with spatial redundancy in the I-frames and in the residual after prediction, compressed data is obtained.

A. Evaluation of different coding methods

In this section, we compare our proposed compression method, Quantization + JPEG-LS, with two other coding methods, Quantization + Huffman coding (the method used in our previous study) and lossy JPEG [34], in order to evaluate the effectiveness of our proposed method. Fig. 6 shows the results when using these three coding methods with the same difference values.

As shown in Fig. 6, the proposed method (solid line) achieved the best compression performance. The reason JPEG-LS outperforms Huffman coding is because JPEG-LS utilizes the 2D characteristics of the raw data to reduce spatial redundancy. Lossy JPEG, which is a popular 2D image compression method, achieved the worst compression results. This is because when using a lossy compression method, the amount of loss that can be tolerated is highly dependent on the application. Lossy JPEG was developed to "fool" human eyes when observing visual images, and quantization is performed after Discrete Cosine Transformation (DCT). Because R^{res} has a completely different distribution compared with normal images, Lossy JPEG's quantization factor is not suitable for R^{res} compression. Although it is difficult for us to detect loss after lossy JPEG compression with our naked eyes, quantization after DCT can lead to a large amount of bias, as Fig. 6 shows. Therefore, in our proposed method we chose to first quantize our values and then use JPEG-LS as a coding tool.

VI. Motion analysis

As explained in Section III, when using the proposed method the target data should be distributed into I-frames and B-frames at the beginning of the compression process. The I-frames, which are reference frames, are used to predict the B-frames located between them. For example, given the sequence "IBBBI", the second, third and fourth frames are predicted using information in the first and fifth frames.

As described in the previous section, the I-frames are compressed losslessly, which results in a relatively large volume of error-free data, while the B-frames are compressed using a lossy method, which results in less data but with some errors. How we decide to arrange the sequence of frames is highly important for the outcome of further processing.

In this section we explain how our sequence decision module optimizes I-frame and B-frame sequencing by analyzing the motion of the LiDAR unit.

A. Sequence decision algorithms

The key sequencing decisions are where we should place the I-frames and how often. In our prediction process, the information in each B-frame is based on the information coming from the two nearest I-frames, thus we need enough I-frames to provide information which is as accurate as possible for the B-frames. On the other hand, the data in the pair of I-frames enclosing the B-frames should not overlap too much, because when there are few changes between frames additional I-frames are unnecessary, and too many I-frames leads to a higher volume of compressed data.

In our previous study [9], we introduced three different sequencing methods and discussed their performance in detail. These methods can be summarized as follows:

Fixed number of B-frames: A fixed number of B-frames are used between each pair of I-frames for all of the data. This is the conventional method used in compression methods such as MPEG.

Fixed shift: Rather than fixing the number of B-frames, the driving distance traveled during the interval between two I-frames is fixed. The location of the LiDAR unit in each frame can be determined using motion estimation, so the shift distance is s_i per frame. The number of B-frames (n^B) between two I-frames then depends on two parameters, the threshold of shift s and lower limit n^B_{min}, while observing the following restriction:

$$n^B \geq n^B_{min} \quad \text{and} \quad \textstyle\sum_{i=1}^{n^B} s_i \leqslant s \tag{2}$$

Uniform motion splitting: In this method, we not only fix the shift distance between two I-frames, but we also separate uniform motion frames from frames which include acceleration or deceleration activity [9].

In our previous study, we analyzed the performance of the sequencing methods described above and concluded that simply increasing the number of B-frames between I-frames does not improve compression performance, and that setting I-frames among the low velocity frames is cost-effective. In light of these results, we proposed a new sequencing method which we called the "local minimum method".

Local minimum: The average velocity of the LiDAR within each pair of frames assumed to be the velocity observed in the latter frame. We then select the frames which have the local minimum velocity as I-frames. Between these minimum velocity I-frames, additional I-frames are inserted using the fixed shift method.

B. Comparative evaluation of sequence decision methods

A comparison of these different sequence decision methods is shown in Fig. 7, which represents 7 seconds of data as the vehicle is approaching a crosswalk.

Focusing on the blue lines in Fig. 7(c) and Fig. 7(d), when the number of B-frames between two I-frames is fixed, as velocity decreases relatively less bitrate is needed and RMSE also decreases. Now note the red line, which represents the fixed shift sequencing method. Even at very low velocity, some frames still require a large bitrate and have large RMSE if they are located far from the I-frames, as illustrated within the red square in Fig. 7(a). Uniform motion splitting and local minimum can be seen as a trade-off between the other two methods. By inserting only a few I-frames when the LiDAR unit is traveling at a low velocity, bitrate and RMSE of the enclosed B-frames can be reduced. This is because when the vehicle is moving slowly, fewer I-frames are needed to accurately represent the relatively static environment in the many nearby B-frames. When compared with uniform motion splitting, the local minimum method proposed in this paper is more cost-effective because it only inserts one I-frame in the low velocity "valley" shown in Fig. 7(b).

Fig. 8 shows global compression performance using each of the four sequencing methods. At high bitrates, the B-frames are compressed almost as losslessly as the I-frames. In this case, no matter how we select the I-frames the results are almost the same. At high compression rates, i.e., at low bitrates, the proposed local minimum method, represented by the solid yellow line, clearly outperforms the other methods.

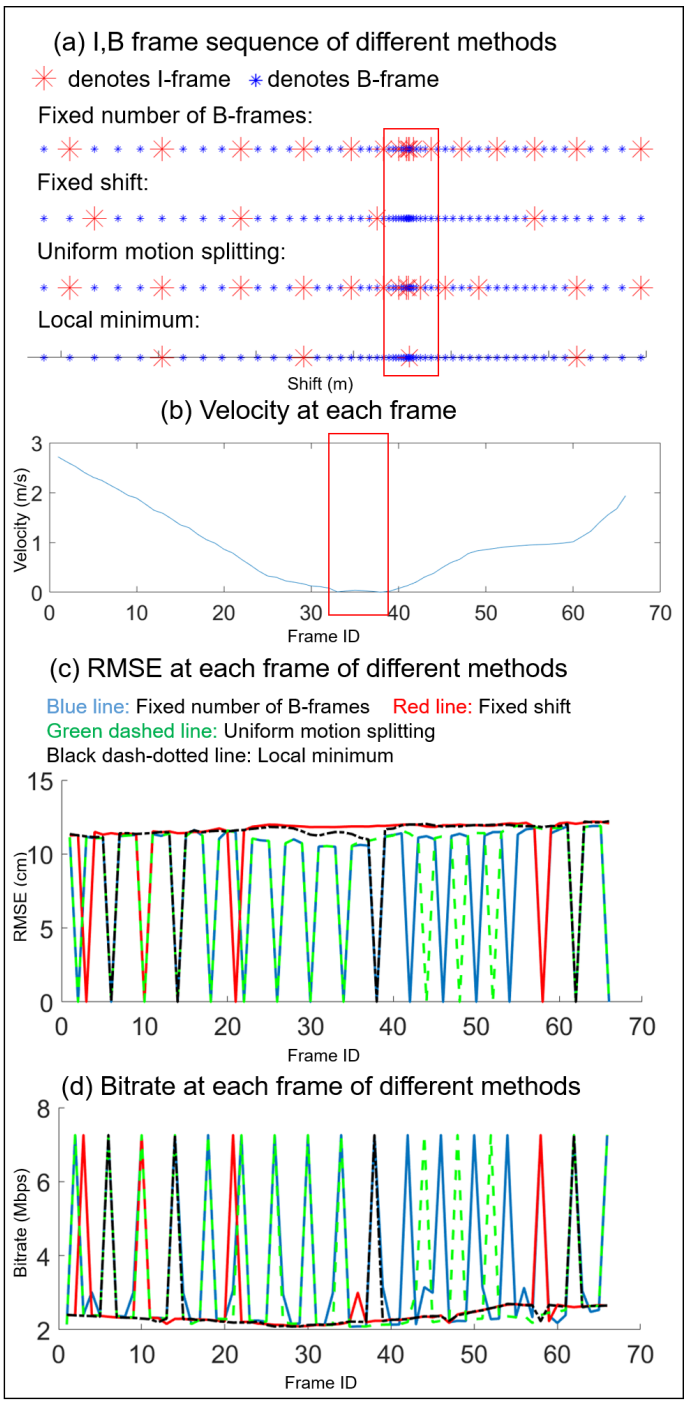

Fig. 7: Various frame sequencing strategies and their effects. In (a), the four frame sequencing methods are illustrated. Small blue asterisks represent the B-frames while the big red asterisks represent the I-frames. (b) shows the velocity of the LiDAR unit during each frame. (c) and (d) show RMSE and bitrate for each frame, respectively, when using each of the sequencing methods. Note that in the (c), points where the RMSE value equals 0 represent I-frames, because our proposed method compresses the I-frames losslessly. Similarly, in (d), points where the bitrate value is a little over 7 Mbps represent I-frames.

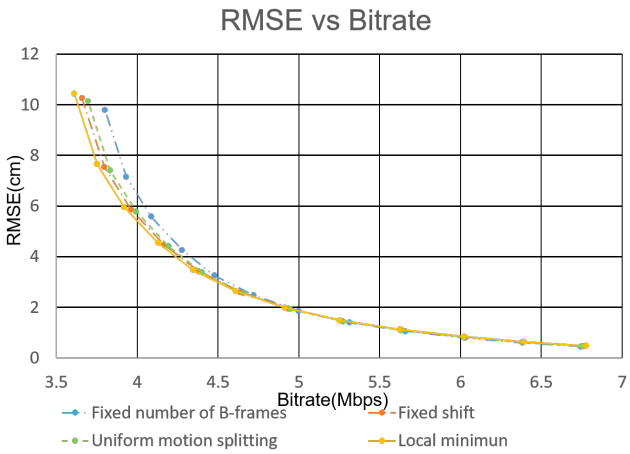

Fig. 8: Compression performance using four different sequencing methods.

VII. Evaluation

In this section, we evaluate the performance of the proposed method both subjectively and objectively. In Section VII-C, we visualize some examples of decompressed point cloud data and analyze them perceptually. In Section VII-D, we quantitatively evaluate the proposed method in terms of the volume of the compressed data and the error rate of the decompressed data, and compare its performance with that of existing point cloud compression methods. We do not compare our method with some other compression methods proposed in related work, either because the code for these methods is unavailable, or because the datasets used are not from autonomous driving situations and do not provide the raw LiDAR packet data, making a direct comparison difficult.

A. Test data

In order to evaluate the proposed method and compare it with other methods, the use of an open dataset is highly desirable. Some point cloud compression methods employ the fr1/room dataset [35], while in the autonomous driving domain the KITTI dataset [36] is often used. Unfortunately, the KITTI dataset does not provide the raw packet data which is required for our proposed method to function. Therefore, we use our own data from a Velodyne HDL-64E S2 sensor for our comparative evaluation (which can be downloaded here: https://drive.google.com/drive/u/0/folders/1qUG_kEqfoT3oCOIMHI0FOktY8Nk9hC3b). The data contains three, one-minute sets of driving data, representing three different driving scenarios: exiting a parking lot, driving in a residential area and driving on a major urban road. Fig. 9 shows the environments and routes used.

B. Implementation

In the motion analysis module, we set parameter $s = 3$ and parameter $n^B_{min} = 15$. During prediction, we build a 60×2000 grid map to search for appropriate reflection points, with each grid sector covering two points on average. In the coding module, we use parameter $\alpha = (\frac{3}{4})^i \quad i \in [0, 1, 2, ..., 11]$ to

Fig. 9: Locations and routes used for collection of driving data.

tune the trade-off between compression rate vs. information loss.

Our implementation is based on Matlab and uses only one core of an Intel Core i7-7820X CPU. Processing time depends on parameter n for reflection point searching. When n=0, compressing 1 minute of data takes about 30 minutes. When n=1, compressing 1 minute of data takes about 80 minutes.

There are methods which could be used to greatly accelerate the processing speed, such as using multiple cores in parallel, utilizing GPU acceleration (especially for accelerating reflection point searching, which is the biggest calculation cost) or customized FPGA, which is often used to accelerate MPEG into real-time. However, accelerating the proposed method to operate in real-time is outside the scope of this paper.

C. Visualizing the decompressed data

Fig. 10 shows examples of decompressed point cloud data at different Root-Mean-Square Error rates. When RMSE = 3.05 cm, it is almost impossible to detect differences in the decompressed point cloud samples with the human eye. However, as RMSE error rates increase the differences become more obvious. The error rate of decompressed data can be thought as the bias of point location. As RMSE increases, lines and planes become rougher, leading to the loss of detail. For example, focusing on the point cloud data in the boxes in Fig. 10, which represent a vehicle, in the original point cloud and even at low RMSE we can easily determine the orientation of the vehicle and even identify its front windshield. However, when RMSE=9.49 cm we can no longer easily ascertain this information. In addition, when the error rate is huge it is difficult to accurately assign boundary boxes to obstacles, which would be dangerous in an autonomous driving application.

Compression error is mainly the result of quantization, and it is difficult to determine a threshold/minimum standard for error because the acceptable level of quality when using lossy compression methods is always dependent on the applications they are being used for. Streaming point cloud data collected for autonomous driving is being used for multiple applications: mapping, localization [37], object detection/tracking [38], and so on [39]. Furthermore, each of these applications focuses on different information from the point cloud. For example, object detection applications are more sensitive to information about relatively nearby points, while localization using normal distributions transforms (NDTs) is oriented on more distant points. For this reason, we want the decompressed data to retain as much useful information as possible, although some error is inevitable. As shown in Fig. 10, at 9.49 cm of RMSE there is so much loss of detail that it interferes with determining the bounding box of the obstacle. Thus, we consider roughly 10 cm of RMSE as the largest error rate our compression method can tolerate when being used for autonomous driving data.

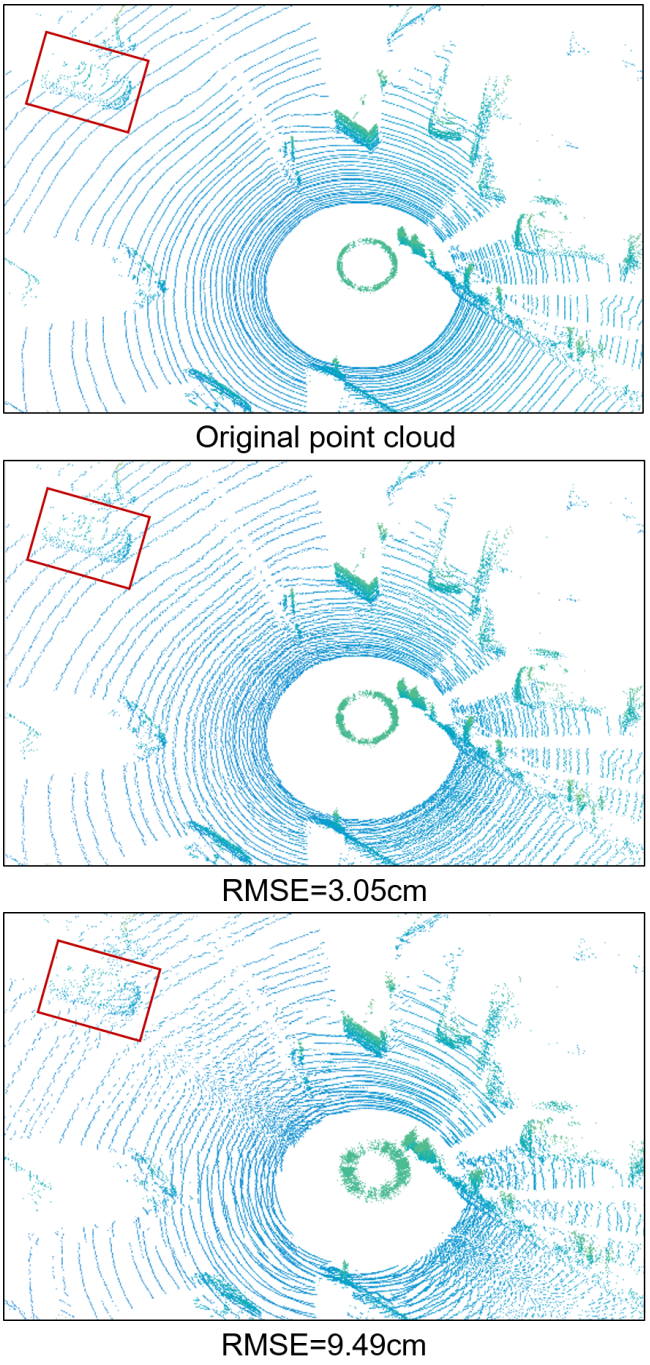

Fig. 10: Examples of decompressed LiDAR data at various rates of RMSE (colored by height).

D. Comparison of proposed method with other compression methods

In this section we compare the proposed method with other point cloud compression methods, as well as with the method proposed in our previous study [9]. As the proposed method targets raw data, in Section VII-D1 we compare the proposed method with other three methods which also target raw point cloud data. In Section VII-D2, we compare the proposed method with the octree compression method[7], which is probably the most popular generic point cloud compression method.

In the following subsection, we use two different evaluation criteria for two different type of algorithms respectively. For the compression methods which target raw packet data R and decompression data are raw data R' with point cloud $P' = f(R')$ like proposed method, using RMSE of $R'_{i,j}$ vs. Bitrate (Mbps) as the evaluation criterion is a natural choice.

However, when using compression methods which directly target point cloud P and whose decompression data is simply P', such as the octree compression method, we cannot directly calculate RMSE between the original point cloud input P and decompressed point cloud P' because the decompressed point cloud is disorderly and there is no direct correspondence between the data points in P and P'. To evaluate these types of methods we instead use Symmetric Nearest Neighbor Root Mean Squared Error (SNNRMSE) as our evaluation criterion.

In contrast to compression methods which target raw data and keep all the points, many other point cloud compression methods throw away some data points during compression. Therefore, using megabits per second (Mbps) directly as a performance criterion would result in an unfair comparison in this case, so instead we use bits per point (bpp) to evaluate the volume of the compressed data, a method which has been widely used as an assessment tool in other point cloud compression research.

1) ours vs. other raw data targeting methods: Some other compression methods besides our proposed method also target raw point cloud data, in order to utilize the potential 2D characteristics of the point cloud data collected by LiDAR scanners. In [9], we proposed several methods which were based on existing image compression methods. Yin [21] proposed a method called Adaptive Point Cloud (APC) and provided the source code, which we have modified in the following experiment in order to adapt it to our sensor. Here, we use the RMSE of the distance value in the raw data vs. Bitrate (Mbps) as the evaluation criterion, as previously mentioned. Fig. 11 shows the performance of the five selected raw data compression methods.

As shown in Fig. 11, our proposed motion analysis-based method (yellow line) outperforms the APC method, which uses fewer bits for each point without reducing spatial or temporal redundancy. The layered JPEG based method utilized JPEG to compress rearranged 2D matrices/images, each of which contains the data from one laser during a fixed period of time. Our motion analysis-based method achieved better compression results than the layered JPEG based method in relatively small RMSE situations, as shown in the red box, although the layered JPEG can compress data at a much smaller

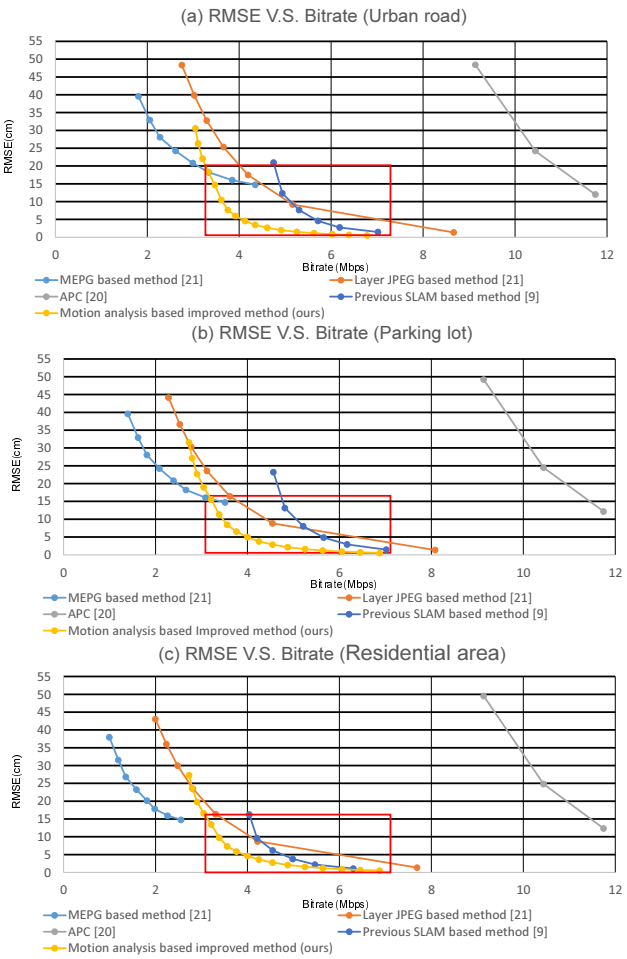

Fig. 11: RMSE vs. Bitrate using five different compression methods.

bitrate when RMSE is high. As a video compression method, MPEG provides a high compression ratio with a relatively large RMSE, but is not as effective in situations requiring low RMSE and does not perform as well as our proposed motion analysis-based method. By using these new motion analysis, prediction and encoder modules, the streaming point cloud compression method proposed in this paper greatly outperforms our previous work [9].

In general, the motion analysis-based compression method outperformed the image compression-based methods while requiring smaller RMSE, but the image compression-based methods achieved higher compression ratios at higher RMSE. Image compression-based methods can compress more data because they take units or blocks of several pixels at a time, for example a 4×4 group of pixels, while the motion analysis-based method processes individual points of data. Thus, image compression methods can lose more information, a feature which could possibly be used to improve the proposed method. In this study, our proposed method focuses on compressing data with high accuracy because we consider a low RMSE is a basic requirement for compression tasks in autonomous driving systems. It should be remembered that safety is always the key factor in autonomous driving applications, because

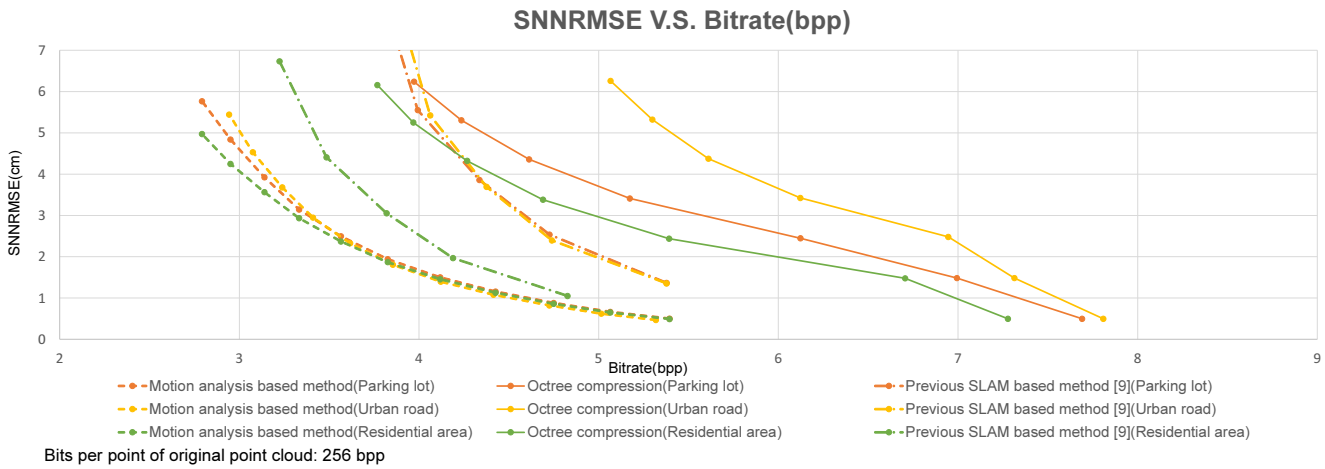

Fig. 12: SNNRMSE vs. Bitrate (in bits per point) for proposed motion analysis based method, octree compression method and our previous work [9].

mistakes caused by errors in decompressed data could lead to serious accidents. In order to avoid this, the decompressed data should have a very small RMSE.

So, in summary, we can say that our proposed method outperformed APC and the image compression-based methods, and that it is a more appropriate choice for autonomous driving applications.

2) ours vs. octree compression method and previous work [9]: Many data compression studies have directly targeted point cloud data, taking advantage of its 3D characteristics without utilizing the 2D characteristics of the raw LiDAR packet data. The octree-based compression method proposed by Kammer [7] is probably the most popular point cloud compression method, since it can compress streaming point cloud data in real-time and is supported by Point Cloud Library (PCL).

As we mentioned at the beginning of this subsection, the decompression output of octree compression is only the point cloud, and some data points may be lost during compression. Therefore, next we use SNNRMSE vs. bpp to compare decompression results P' from octree compression with decompression results when using the proposed motion analysis method and our previous work [9].

SNNRMSE is the conventional criterion used to quantify the difference between two point cloud frames. Given two point cloud frames P and Q, for each point p in P find the closest point q in Q (in Euclidean distance), where q is defined as $q = \mathrm{NN}(p, Q)$:

$$\mathrm{MSE}_{\mathrm{NN}}(P, Q) = \sum_{p \epsilon P} (p - q)^2 / |P| \tag{3}$$

Here $|P|$ represents the number of points in P. Then:

$$\mathrm{RMSE}_{\mathrm{NN}}(P, Q) = \sqrt{\mathrm{MSE}_{\mathrm{NN}}(P, Q)} \tag{4}$$

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

$$\begin{aligned} &\mathrm{RMSE}_{\mathrm{SNN}}(P, Q) \\ &= \sqrt{0.5\mathrm{MSE}_{\mathrm{NN}}(P, Q) + 0.5\mathrm{MSE}_{\mathrm{NN}}(Q, P)} \end{aligned} \tag{5}$$

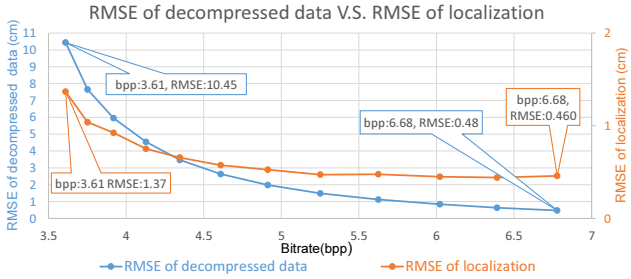

Fig. 13: Evaluating decompressed data by vehicle localization application.

Fig. 12 shows the results. The proposed motion analysis-based method (dashed lines) clearly outperforms the octree-based method and our previously proposed SLAM-based method [9]. Comparing the performance of the three methods in various driving scenarios, all of the methods performed best in the residential area and worst on the urban road. The reason is that the ego vehicle (the platform for the LiDAR unit) moves relatively slowly through the residential area, there are many similar buildings and there are fewer dynamic objects, such as other vehicles and cyclists. All of these factors make it easier for compression algorithms to reduce temporal redundancy. On the other hand, the urban road scenario is just the opposite. Note that the performance of the proposed method is more consistent across the various driving scenarios than that of the octree and our previous work [9]. This is because the prediction module in the proposed method, which uses a grid map, can predict the data points within a static environment relatively accurately even when the ego vehicle/LiDAR unit is moving at a high velocity, which allows our proposed method to function effectively under different driving conditions.

E. Evaluation by application

The evaluation of any lossy compression method should take into consideration possible application scenarios. In this

section, we use vehicle localization, which is one of the most important applications for point clouds in autonomous driving, to evaluate the applicability of our method. Given a point cloud map, we perform localization using decompressed point clouds which were compressed at various compression rates, and compare the accuracy of the localization results. Localization was based on the normal distribution transform (NDT) [31] and we used the implementations provided by the Autoware open source autonomous driving platform [32].

Using with original point cloud as the ground truth for the localization results, the RMSE of localization (for all frames) was then calculated to evaluate error. In Fig. 13, we compare the RMSE of localization with the RMSE of the decompressed data. We can see that at low compression rates, both the decompressed data and localization results are very accurate. However, when increasing the compression rate to reduce the bitrate from 6.68 bpp to 3.61 bpp, the RMSE of decompressed data increases almost 20 times (from 0.48 cm to 10.45 cm) while the RMSE of localization only increases 3 times (from 0.46 cm to 1.37 cm). In other words, the increase in the RMSE of the localization results was much slower than the increase in the RMSE of the decompressed data. As a result, even when the point cloud data is compressed at a low bitrate with relatively high RMSE, we can still perform localization accurately. As we can see in Fig. 13, although the decompressed point cloud had a RMSE of 10.4 cm and needed only 3.61 bpp (almost 1/70th of original point cloud), this led to only about 1.4 cm of RMSE in localization error, which is very small and well within the acceptable range for vehicle localization tasks. Thus, the results of this experiment demonstrate the potential of the proposed method for use in real applications.

VIII. Conclusion & Future Work

In this paper we have introduced a method of streaming point cloud compression which is an improved version of a method we proposed in a previous paper. We also analyzed how different parameter values affect compression performance. Experimental evaluation of the motion analysis-based compression method proposed in this paper demonstrated its ability to compress LiDAR point clouds into a small volume of data with very little information loss. This was achieved by using an efficient prediction module to reduce temporal redundancy, an encoder to reduce spatial redundancy and a motion analysis module to optimize reference frame distribution. In an experimental comparison of the proposed method with existing point cloud compression methods, our results showed that the proposed method achieved better overall performance when compressing and decompressing LiDAR data (i.e., high compression rates at low error rates), outperforming the other methods and demonstrating that it has great potential for use in autonomous driving systems.

As for future works, we will focus on optimizing our method for main applications, epically localization and object detection.

As mentioned before, the evaluation of any lossy compression method should take into consideration possible application scenarios. In autonomous driving, the most important applications for point cloud are localization and object detection. In Section VII-E, we showed that decompressed data from the proposed method could work well in localization tasks. How about object detection and how to future improve the performance of localization and object detection while keeping the same compression rate remains a future goal.

To ensure that our decompressed data can work well in these applications, we should selectively lose information in the encoder module. And the definition of important information varies between localization and object detection. In the future work, we will focus on designing a smarter encoder module by exploring and considering what is the most important information for popular localization and object detection algorithms. By discarding relatively inessential information, we believe to obtain better localization and object detection results at the same compression rate.

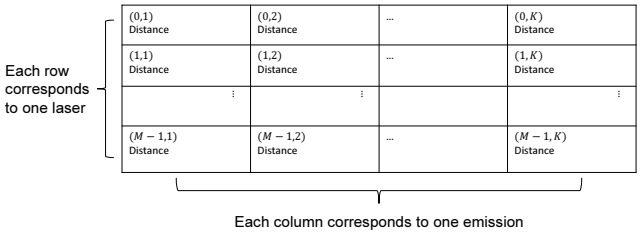

Fig. 14: Arranging raw packet data into a 2D matrix. Each row corresponds to one laser ID, and each column corresponds to one emission. The value of each pixel represents distance information.

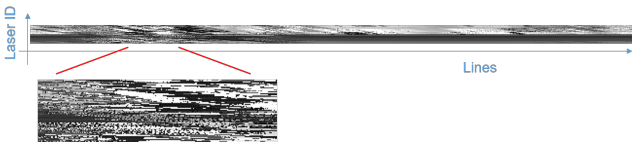

Fig. 15: Visualization of raw packet data in an image-like format. A pixel's grayscale value from black to white represents a distance from 0 to 13,000 cm. Noted that without calibration, we could not understand the raw data the way we can understand a depth map.

Appendix A
Potential 2D Characteristics of Point Cloud Data Collected by LiDAR Scanners

Each point in a set of point cloud data is represented by x, y, and z coordinates, identifying its location in a 3D space. Raw packet data, however, represents each point using a distance value, a rotation angle and a laser ID. Rotation angle here means the yaw angle of LiDAR, and all of a particular laser's emissions occur at the same yaw angle once per rotation, without taking calibration into consideration. Laser ID here represents pitch angle information. In LiDAR systems, every laser sensor is fixed at a specific location and angle, so that if the laser ID is known we can easily determine the pitch angle of the beam. In other words, raw packet data can be roughly

thought as a kind of polar-coordinate-like representation of 3D point cloud. After a calibration process $f(R) = P$, which uses a calibration file to correct angle, distance and starting locations, raw packet data R can be converted into a point cloud P in real-time.

By arranging raw packet data into a 2D matrix as shown in Fig. 14, the information can be stored losslessly. Fig. 15 shows an example of an image created using this type of raw packet data. Laser ID information is represented by row number i, while information about the rotation angle, which corresponds to column j, could be coded into a few bits of data because, generally, the difference between adjacent rotation angles is uniform. As a result, by using run length coding all of the rotation angle information can be efficiently captured. These potential 2D characteristics of point clouds are in fact based on LiDAR's own operating principle.

We need to store three elements, the x, y and z coordinates, for each point in a point cloud, but when utilizing the data in this image-like format, for the most part we only need one element, distance from the scanner to the detected point. Thus, the 2D format is itself already a kind of data compression.

REFERENCES

[1] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt *et al.*, "Towards fully autonomous driving: Systems and algorithms," in *IEEE Intelligent Vehicles Symposium (IV)*, 2011, pp. 163–168.

[2] C. Margulis and C. Goulding, "Waymo vs. uber may be the next edison vs. westinghouse," *J. Pat. & Trademark Off. Soc'y*, vol. 99, p. 500, 2017.

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

[4] 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.

[5] 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.

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

[7] J. Kammerl, N. Blodow, R. B. Rusu, S. Gedikli, M. Beetz, and E. Steinbach, "Real-time compression of point cloud streams," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2012, pp. 778–785.

[8] 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.

[9] 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.

[10] 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.

[11] H. Zhan, Z. Ding, and L. Zhou, "Connectivity compression for triangle meshes," in *Active Media Technology*. World Scientific, 2003, pp. 502–507.

[12] A. Khodakovsky, P. Schröder, and W. Sweldens, "Progressive geometry compression," in *ACM Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, 2000, pp. 271–278.

[13] 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.

[14] 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.

[15] 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.

[16] T. Ochotta and D. Saupe, "Image-based surface compression," in *Computer graphics forum*, vol. 27, no. 6. Wiley Online Library, 2008, pp. 1647–1663.

[17] R. Schnabel, S. Möser, and R. Klein, "A parallelly decodeable compression scheme for efficient point-cloud rendering." in *Symposium on Point Based Graphics(SPBG)*, 2007, pp. 119–128.

[18] R. Schnabel, S. Möser, and R. Klein, "Fast vector quantization for efficient rendering of compressed point-clouds," *Computers & Graphics*, vol. 32, no. 2, pp. 246–259, 2008.

[19] A. Kalaiah and A. Varshney, "Statistical geometry representation for efficient transmission and rendering," *ACM Transactions on Graphics (TOG)*, vol. 24, no. 2, pp. 348–373, 2005.

[20] V. Morell, S. Orts, M. Cazorla, and J. Garcia-Rodriguez, "Geometric 3D point cloud compression," *Pattern Recognition Letters*, vol. 50, pp. 55–62, 2014.

[21] 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.

[22] 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.

[23] 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.

[24] C. Tu, E. Takeuchi, A. Carballo, and K. Takeda, "Point cloud compression for 3d lidar sensor using recurrent neural network with residual blocks," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[25] J. Hayes, "LIDAR point cloud compression," Sep. 2017, US Patent 9,753,124.

[26] J. Kruger, J. Schneider, and R. Westermann, "Duodecim-a structure for point scan compression and rendering," in *Point-Based Graphics, Eurographics/IEEE VGTC Symposium Proceedings*, 2005, pp. 99–146.

[27] S. Gumhold, Z. Kami, M. Isenburg, and H.-P. Seidel, "Predictive point-cloud compression," in *ACM SIGGRAPH 2005 Sketches*, 2005, p. 137.

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

[29] 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.

[30] D. Thanou, P. A. Chou, and P. Frossard, "Graph-based compression of dynamic 3D point cloud sequences," *IEEE Transactions on Image Processing*, vol. 25, no. 4, pp. 1765–1778, 2016.

[31] 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.

[32] 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.

[33] M. J. Weinberger, G. Seroussi, and G. Sapiro, "The LOCO-I lossless image compression algorithm: Principles and standardization into JPEG-LS," *IEEE Transactions on Image processing*, vol. 9, no. 8, pp. 1309–1324, 2000.

[34] G. K. Wallace, "The JPEG still picture compression standard," *IEEE transactions on consumer electronics*, vol. 38, no. 1, pp. xviii–xxxiv, 1992.

[35] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of RGB-D SLAM systems," in *IEEE Intelligent Robots and Systems (IROS)*, 2012, pp. 573–580.

[36] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The KITTI dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[37] A. Y. Hata and D. F. Wolf, "Feature detection for vehicle localization in urban environments using a multilayer lidar," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 2, pp. 420–429, 2016.

[38] E. Arnold, O. Y. Al-Jarrah, M. Dianati, S. Fallah, D. Oxtoby, and A. Mouzakitis, "A survey on 3d object detection methods for autonomous driving applications," *IEEE Transactions on Intelligent Transportation Systems*, 2019.

[39] H. Guan, J. Li, Y. Yu, Z. Ji, and C. Wang, "Using mobile lidar data for rapidly updating road markings," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 5, pp. 2457–2466, 2015.

Chenxi Tu is currently a Ph.D. candidate in Informamatics at Nagoya University, Japan. He received his B.S. and M.S. degrees in school of Information and Communication Engineering from Beijing University of Posts and Telecommunications, Beijing, China, 2013 and in Information Science from Nagoya University, Nagoya, Japan in 2016 respectively.

His research focuses on point cloud data processing.

Eijiro Takeuchi received his Bachelor, Masters and Ph.D. degrees from the Intelligent Robot Laboratory, University of Tsukuba, Japan. From 2008 to 2014, he had been working at Tohoku University, Japan, as Assistant Professor. Since 2014, he started to work at Nagoya University, Japan. His main focus is localization, mapping in Robotics and autonomous driving. He is currently a Associate Professor at Graduate School of Informatics, Nagoya University.

Alexander Carballo received his Dr.Eng. degree from the Intelligent Robot Laboratory, University of Tsukuba, Japan. From 1996 to 2006, he worked as lecturer at School of Computer Engineering, Costa Rica Institute of Technology. From 2011 to 2017, worked in Research and Development at Hokuyo Automatic Co., Ltd. Since 2017, he is a Designated Assistant Professor at Institutes of Innovation for Future Society, Nagoya University, Japan. His main research interests are LiDAR sensors, robotic perception and autonomous driving.

Chiyomi Miyajima Chiyomi Miyajima received the B.E.,M.E., and D.E. degrees in computer science from Nagoya Institute of Technology, Japan. From 2001 to 2003, she was a Research Associate at the Department of Computer Science, Nagoya Institute of Technology. Since 2003, she has been an Assistant Professor at the Graduate School of Information Science, Nagoya University, Japan. She is currently a Designated Associate Professor at the Green Mobility Research Institute, Institutes of Innovation for Future Society, Nagoya University. Her research interests include analysis and modeling of driver behavior.

Kazuya Takeda (Senior Member, IEEE) received his B.E.E., M.E.E., and Ph.D. from Nagoya University, Japan. Since 1985 he had been working at Advanced Telecommunication Research Laboratories and at KDD R&D Laboratories, Japan. In 1995, he started a research group for signal processing applications at Nagoya University.

His main focus is investigating driving behavior using data centric approaches, utilizing signal corpora of real driving behavior.

He is currently a Professor at the Institutes of Innovation for Future Society, Nagoya University. He is also serving as a member of the Board of Governors of the IEEE ITS society.