Object Extraction Method for Mobile Robots using Fast Growing Neural Gas

Alfin Junaedy*, Hiroyuki Masuta*, Naoyuki Kubota**, Kei Sawai*, Tatsuo Motoyoshi*, Noboru Takagi*
*Dept. of Intelligent Robotics
Toyama Prefectural University
Toyama, Japan
u154003@st.pu-toyama.ac.jp, {masuta, k_381, motoyosh, takagi}@pu-toyama.ac.jp
**Graduate School of System Design
Tokyo Metropolitan University
Tokyo, Japan
kubota@tmu.ac.jp

2022 IEEE Symposium Series on Computational Intelligence (SSCI) | 978-1-6654-8768-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/SSCI51031.2022.10022121

***Abstract*—In this work, we present a fast object extraction method that can be applied to embedded computers in mobile robots. Currently, perceiving the real-world in 3D space is becoming increasingly important, and robotics is no exception. Many types of robots have utilized 3D point cloud data as part of their mission, such as rescue robots, humanoid robots, explorer robots, etc. All of them make use of 3D sensor data as the input. However, a problem occurs when large amounts of data are generated, especially for embedded computers that are usually installed in mobile robots. Therefore, we propose an object extraction method for 3D point cloud data that is lightweight and can be applied to the embedded computers. It is developed based on a simplification of fast growing neural gas algorithm, called as simplified FastGNG. As opposed to the standard GNG algorithm, FastGNG uses a multi-scale batch learning strategy to enhance the learning speed. Thus, in this work, we applied the algorithm to extract information of the point could data as a way to compress the large amounts of data. The experimental results have shown that the proposed method is able to reach 36.2 fps which is real-time capable, and hence suitable for embedded computers.**

Keywords—object extraction, 3D point cloud data, fast growing neural gas, depth image compression, mobile robot

I. Introduction

In recent years, representing the real-world in 3D visual scenes is becoming increasingly important [1], and robotics is no exception. Rescue robots in [2-4] use point cloud data from onboard laser scanner to build a 3D environment or map. The denser the point cloud, the more accurate the result. On the other hand, 3D plane segment information has been used for a humanoid robot to climb stairs [5]. It detects 3D planar surfaces form depth images by applying Hough Transformation methods. Explorer robots in [6, 7] employ simultaneous localization and mapping (SLAM) techniques for reconstructing the environment. All of them make use of 3D sensor data as the input. However, a problem occurs when large amounts of data are generated, especially for embedded computers that are usually installed in mobile robots [8, 9]. It is difficult for them to take advantage of all the data due to limited resources [10]. Moreover, they have to keep running and collecting the data. Therefore, real-time applications cannot be achieved efficiently.

In order to address this problem, we propose an object extraction method for 3D point cloud data that is lightweight and can be applied to the embedded computers. Object extraction is one of the ways to compress or simplify the large amounts of data [11], and thus the processing efficiency can be improved as the computational load of the main algorithm is reduced. The proposed method is developed based on a simplification of fast growing neural gas (FastGNG) algorithm, called as simplified FastGNG. FastGNG is a modification of the standard GNG algorithm which has a new growing method to enhance the learning speed [12]. Meanwhile, the standard GNG uses an iterative method as the learning strategy [13]. Hence, it is difficult to increase the learning speed since a sample node is added to a current network after errors are accumulated with many times of sampling data [12].

Previously, we have developed a mobile robot system for extracting plane information from a light detection and ranging (LiDAR) camera [14]. It uses a particle swarm optimization-seeded region growing (PSO-RG) as the main algorithm to detect planar surfaces under an embedded computer. However, the accuracy is still low as it only has eight nodes for each plane to represent the detected objects. Moreover, a maximum of 15.1 fps is the best performance and sometimes drops to 7.2 fps for the worst case.

There are many techniques out there for 3D data compression to reduce the computational cost. E.g., compression by depth downsampling [15], colorization [16], depth image warping [17], and the most popular compression is by segmentation [18] or feature extraction [11]. However, it is still difficult for them to be applied for embedded computers. The Hough transform [19] and Random sample consensus (RANSAC) [20] are usually used for feature extraction. Unfortunately, a multi-resolution plane segmentation using a combination of both is not real-time capable since it requires more than one second to run the algorithm [21].

Therefore, in comparison with our previous work [14], we have two main targets in this paper. First, perform real-time object extraction over 15.1 fps. Second, increase the accuracy of the extraction results. In this case, we use the same system configuration as before [22]. The mobile robot uses a Raspberry Pi as the main controller which is one of most popular embedded computers for robotics. It also has a 2D SLAM algorithm for exploration purposes [23]. However, SLAM is not the focus of this paper.

The mobile robot system applied a shared control method. It means that the algorithms are divided to a PC controller and the mobile robot. The PC controller is used by operators to control, interact, and monitor the mobile robot. Hence, all the observation data need to be sent from the mobile robot to the PC controller, including the object extraction results. The smaller the data size, the faster the communication time to support the real-time applications. In addition, the communication system uses a combination of high-bandwidth (i.e., Wi-Fi) and low-bandwidth (i.e., LoRa) networks that have bandwidths of up to 350 Mbps and 22 kbps, respectively. Therefore, another purpose of this object extraction method is to reduce the 3D point cloud data in terms of size to support both the high and low-bandwidth networks. The remainder of this paper is organized as follows. Section II explains the system overview. Section III describes the proposed algorithm. Section IV shows the experimental results. Finally, section V is the conclusion and future work.

II. System Overview for Data Measurement

A. Processing Hardware

First, we installed an Intel RealSense LiDAR camera to the mobile robot, as shown in Fig. 1. We assumed the experimental environment to indoor terrain with an area of 16.59 m^2. There are cluttered obstacles in it. The mobile robot has a Raspberry Pi as the main controller in which all onboard computations are performed. Table I summarizes the specifications of both. Both of them are powerful enough and relatively small in size, and thus can be applied to small mobile robots.

The mobile robot also has a 2D SLAM algorithm from our previous works [22, 23]. It can explore the environment with an average speed of 0.2 m/s. Therefore, all algorithms should have real-time capabilities in order to match the pre-defined parameters and accomplish the missions.

B. Depth Camera

Basically, the LiDAR camera has two outputs, i.e., color and depth. In this approach, however, we only used the depth information. Considering the problems in [24], the color images sometimes result in unstable detections, especially for cluttered objects which have the same or similar color as the background. The same as the previous work [14], 3D analysis can be performed by using the depth information, making it more precise for extracting objects from the LiDAR camera.

Then, to get the real-world coordinates which are the same as the camera frame coordinates (x_c, y_c, z_c), the right-angled triangle calculation can be used (1-3).

$$x_c = z_g \cdot \tan\left[\left(x_g - \frac{res_w}{2}\right) \cdot \frac{FoV_w}{res_w}\right] \qquad (1)$$

$$y_c = z_g \cdot \tan\left[\left(y_g - \frac{res_h}{2}\right) \cdot \frac{FoV_h}{res_h}\right] \qquad (2)$$

$$z_c = z_g \qquad (3)$$

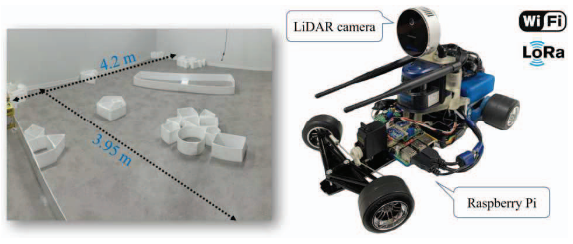


Fig. 1. Mobile robot with a LiDAR camera for depth measurement.

TABLE I. Hardware Specifications

Raspberry Pi 4 Model B	Intel RealSense LiDAR Camera L515
• Quad core Cortex-A72 (ARM v8) @ 1.5 GHz • 4 GB of LPDDR4-3200 SDRAM • 2.4 GHz and 5.0 GHz IEEE 802.11ac wireless • OpenGL ES 3.1; Vulkan 1.0	• Physical: dia. = 61 mm; height = 26 mm; weight = 100 g • Depth: res. 640×480 30 fps; FoV = 70°×55°; sensing range = 0.5 m ~ 9 m • Color (RGB): res. = 1920×1080 30 fps; sensor = 2 MP; FoV = 70°×43° • IMU • Interface: USB-C 3.1

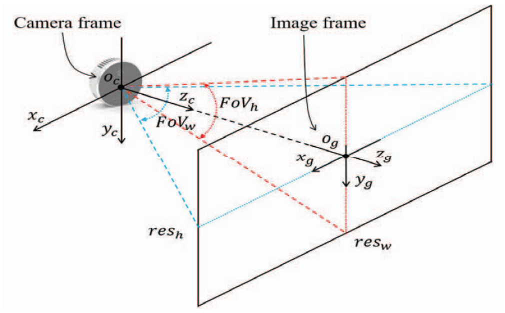


Fig. 2. Image frame projection form the camera position.

Where, x_g and y_g are the pixel coordinate, while z_g is the depth measurement at x_g, y_g. This information can be known from the LiDAR camera. The field of view (FoV_w, FoV_h) and the resolution (res_w, res_h) are the camera's properties of the depth output. Fig. 2 shows the projection of the image frame from the camera position. In this case, the sensing range of the depth output is limited to 4 m to minimize distortion effects when detecting far objects.

III. Object Extraction Method

In order to increase the performance and accuracy of the previous work, we propose an object extraction method based on a modification and simplification of FastGNG algorithm [12]. Fig. 3 shows the architecture of the proposed method. The input is 3D point cloud data from the LiDAR camera and the output is clusters that consist of nodes and connections of the detected objects. It also simplifies the 3D point cloud data which are much larger in size, and so for the communication cost for sending them. The next subsections explain the details of each method. Starting from the simplification of FastGNG algorithm. Then, several post-processing methods are performed sequentially to correct defects of the previous step.

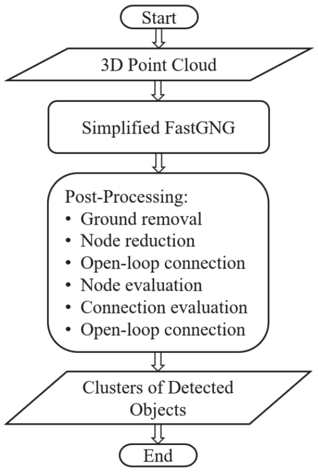


Fig. 3. Architecture of the proposed object extraction method.

A. *Simplified FastGNG*

FastGNG is one of unsupervised learning methods based on a modification of the standard GNG algorithm, with its main purpose to enhance the learning speed. General objectives such as clustering, feature extraction, and topological mapping can be performed using this algorithm.

In this paper, we made a simplification of the FastGNG algorithm, called as simplified FastGNG, to further increase the learning speed and match with the main purpose of this paper. Following are the main notations used for the algorithm.

D	: a set of 3D point cloud data (input)
A	: a set of sample nodes
w_i	: a 3D vector of i-th node
$c_{i,j}$	: a connection between i-th and j-th node
L	: an internal loop parameter of the algorithm (mini-batch)
$l_{\min}$, $l_{\max}$	: a parameter for setting the minimum and maximum length of the connection
$P_{edge}(v)$	: the probability of a sample v is the edge. v is a 3D vector
$P_{node}(v)$	: the probability of adding a sample v to the network

The learning process is then arranged from the following steps:

Step 0: Generate a set of sample nodes (A) using $P_{edge}(v)$. It generates edges of the depth input to simplify the learning process. The constant α should be adjusted to match the depth input, and left-right-top-bottom node of v should be known.

$$P_{edge}(v) = \max(ed_1, ed_2) \tag{4}$$

$$ed_1 = \max(|dL - dR| - dL \cdot \alpha, |dL - dR| - dR \cdot \alpha) \tag{5}$$

$$ed_2 = \max(|dT - dB| - dT \cdot \alpha, |dT - dB| - dB \cdot \alpha) \tag{6}$$

$$dL = \left\|v - w_{\mathrm{left}(v)}\right\|,\ dR = \left\|v - w_{\mathrm{right}(v)}\right\| \tag{7}$$

$$dT = \left\|v - w_{\mathrm{top}(v)}\right\|,\ dB = \left\|v - w_{\mathrm{bottom}(v)}\right\| \tag{8}$$

Step 1: Generate two nodes using $P_{node}(v)$ and connect them. The constant β can be set to more than 1. At the beginning of the algorithm, set the $l_{\min}$ and $l_{\max}$ to any proper value. Then, initialize the phase counter iP to 1.

$$P_{node}(v) = \begin{cases} 1.0 & ds_1 \geq l_{\min} \wedge ds_1 \leq l_{\max} \\ 0.0 & ds_1 < l_{\min} \\ \tanh\left(\beta \frac{l_{\max}}{ds_1}\right) & otherwise \end{cases} \tag{9}$$

$$ds_1 = \left\|v - w_{s_1}\right\|,\ s_1 = \arg\min_{i \in A}\|v - w_i\| \tag{10}$$

$$c_{\mathrm{node1,node2}} = 1, iP = 1 \tag{11}$$

Step 2: Initialize the internal loop counter iL to 0. Define the $l_{\min}$, $l_{\max}$, and L. See parameter settings in section IV for more details.

Step 3: Generate a new node v with $P_{node}(v)$ using (9) and increase the loop counter iL. Then, calculate the nearest node s_1, and the second nearest node s_2.

$$iL = iL + 1 \tag{12}$$

$$s_1 = \arg\min_{i \in A}\|v - w_i\|,\ s_2 = \arg\min_{i \in A \setminus \{s_1\}}\|v - w_i\| \tag{13}$$

Step 4: Add the new node v to the network (r is a new index in the network), and update the connection c. Skip the connection if it already exists. Go back to step 3 if iL is less than L.

$$w_r = v \tag{14}$$

$$c\begin{cases} c_{s_1,r} = 1, c_{r,s_2} = 1, c_{s_1,s_2} = 0 & ds_{12} > ds_1 \wedge ds_{12} > ds_2 \\ c_{s_1,r} = 1, c_{s_1,s_2} = 1 & otherwise \end{cases} \tag{15}$$

$$ds_{12} = \left\|w_{s_1} - w_{s_2}\right\| \tag{16}$$

$$ds_1 = \left\|v - w_{s_1}\right\|,\ ds_2 = \left\|v - w_{s_2}\right\| \tag{17}$$

Step 5: If there are nodes that have more than two connections, remove the longest connection. Repeat the removal process until all nodes have a maximum two connections.

Step 6: As a result of connection deletion, remove the nodes that have no connection at all and increase the phase counter iP. Stop the process when the stopping criterion (i.e., network size or some performance measures) is satisfied or iP is more than or equal to the maximum phase. If not, return to step 2.

$$iP = iP + 1 \tag{18}$$

The learning process of simplified FastGNG is also summarized in Algorithm 1. It grows based on a multi-scale batch learning (MS-BL) process which divides the maximum number of nodes into several phases. Fig. 4 shows the phase update strategy in MS-BL. Compared with the standard GNG algorithm which uses an iterative method, FastGNG uses MS-BL strategy to update the networks for faster learning speed especially at the beginning of the phase level.

Algorithm 1 Simplified FastGNG

```
Input   : maximum phase $max\_phase$
Output  : nodes and connections
1:   Step 0                          #sample nodes $A$
2:   Step 1                          #initial nodes
3:   for $iP \leftarrow 1$ to $max\_phase$ do
4:       Step 2                      #parameter initialization
5:       for $iL \leftarrow 1$ to $L$ do
6:           Step 3                  #new nodes with $P_{node}(v)$
7:           Step 4                  #network update
8:       end for
9:       Step 5                      #network correction
10:      Step 6                      #network check
11:  end for
```

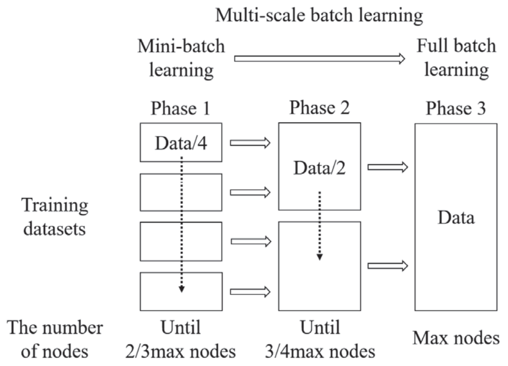


Fig. 4. Multi-scale batch learning strategy for faster learning speed [12].

FastGNG adds new nodes after one mini-batch learning using weight parameters. Although it improves the learning speed, this also results unnecessary nodes added to the network that require a node deletion process. Thus, the simplified FastGNG avoids that by improving the correctness of the probability functions ($P_{edge}(v)$, $P_{node}(v)$), and additional computational costs can be saved. Furthermore, due to higher correctness of the probability functions, a new node and its connection can be added directly to the network in each step of the mini-batch without the node deletion process. This is the most important difference between simplified FastGNG and the FastGNG algorithm. The simplified FastGNG can reject a node with a zero-probability value, and generate a new appropriate node instead. Moreover, it also filters the sample nodes at the beginning of the algorithm by applying $P_{edge}(v)$ for improving the correctness of the sample nodes.

The same as FastGNG algorithm, the simplified FastGNG tries to cover the entire sample nodes as fast as possible in the first phase. This can be achieved by setting higher $l_{\min}$ and $l_{\max}$ with smaller L value. In the second phase, $l_{\min}$ and $l_{\max}$ are reduced to reach the sample nodes in more detail. On the other hand, L is increased to support the detailed nodes by generating more new nodes based on the probability function, and so for the third and larger phases. The parameter settings for these variables can be seen in section IV. More conditions, of course, can be added to get different results. However, the computational cost may increase.

B. Ground Removal

One of purposes of developing this algorithm is to simplify the 3D point cloud data. This is very important especially when sending them using the low-bandwidth communication. Therefore, the ground information is removed from the detection results. In 3D point cloud data, the ground is basically a set of nodes lying on a flat plane, as shown in Fig. 5. They have no neighbors in the y axis. Thus, we can use this information to determine whether a particular node is lying on the ground or not (19, 20). The set of 3D point cloud data D is needed for this calculation as the reference nodes.

$$Gnd(v) = \begin{cases} 0 & dG_x < g_x \wedge dG_y > g_y \wedge dG_z < g_z \\ 1 & otherwise \end{cases} \quad (19)$$

$$dG_x = |v_x - w_i x|, dG_y = |v_y - w_i y|, dG_z = |v_z - w_i z|, i \in D \quad (20)$$

Where, g_x, g_y, and g_z are the thresholds for ground detection. They should be adjusted based on the input characteristic, i.e., minimum distance between nodes (see parameter settings in section IV). After applying the $Gnd(v)$ to all simplified FastGNG nodes, the ground nodes ($Gnd(v) = 1$) are then removed from their network.

C. Node Reduction

The same as before, this method reduces redundant nodes from the detection results to further compress the 3D point cloud data. The node reduction is based on the angle between two connections, as shown in Fig. 6. In principle, the straight connections can be simplified by removing nodes along these connections. Hence, in order to determine these straight connections, we can use θ information (21). If the angle is more than a specific value, the node between them can be removed.

$$\cos\theta = \frac{\vec{a}\cdot\vec{b}}{|\vec{a}||\vec{b}|} \quad (21)$$

Both vector $\vec{a}$ and $\vec{b}$ can be known from the corresponding nodes since they have x, y, and z value. In this case, we determined that 160° is the threshold for node reduction. After removing the specific nodes, the connections should also be updated.

D. Open-Loop Connection

A perfect object extraction should have closed loop networks around the detected objects. Until this step, there are open-loop and closed loop networks from the detection results. Therefore, a correction is needed in order to minimize the open-loop networks. We use the combination of angle and distance to generate a new connection for the open-loop networks. The probability of a simplified FastGNG node v to be connected to the open-loop nodes, $P_{open}(v)$, can be defined using (22, 23).

$$P_{open}(v) = \eta_1 \cdot \theta + \eta_2 \cdot dC \quad (22)$$

$$dC = \|v - w_{open_i}\|, i \in A \quad (23)$$

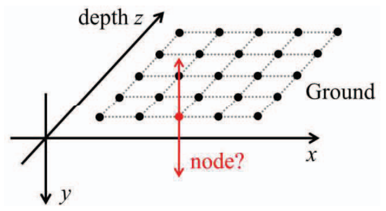


Fig. 5. Nodes lying on the ground.

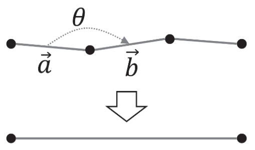


Fig. 6. Node reduction based on the angle between two connections.

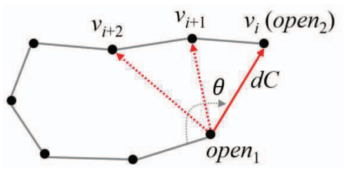


Fig. 7. An example of the open-loop connection. The $open_1$ node is connecteed to the v_i node as a result of the method.

Where, the θ value is the same as in (21). The $P_{open}(v)$ prioritizes nodes having larger angle and shorter distance from the open-loop nodes by using a positive value of η_1 and a negative value of η_2 (see parameter settings for more details). At last, nodes having the highest $P_{open}(v)$ are connected to the open-loop nodes for closing the network connection. Fig. 7 shows an example of the open-loop connection.

E. Node Evaluation

As mentioned earlier, the detection results are sometimes flawed. These can be caused by many factors, e.g., sensor accuracy, camera distortion, measurement condition, environment, and some others. Hence, a node evaluation is performed to mitigate these problems. Similar to the ground removal method, this method also uses the set of 3D point cloud data D as the reference nodes. Initially, the counter μ is defined as 0. It is then increased using $Ne(v)$ to determine whether the node v should be removed or not (24, 25).

$$Ne(v) = \begin{cases} \mu = \mu + 1 & dN_x \le n_x \wedge dN_y \le n_y \wedge \\ & dN_z \le n_z \\ \mu = \mu & otherwise \end{cases} \tag{24}$$

$$dN_x = |v_x - w_i x|, dN_y = |v_y - w_i y|, dN_z = |v_z - w_i z|, i \in D \tag{25}$$

The threshold for μ is adjusted based on the input characteristic. In this case, the node v should be removed when the μ is less than a specific value (i.e., 36). This means that there are no input nodes (references) around the node v, as shown in Fig. 8. Thus, it can be assumed that this is a false detection. The threshold values of n_x, n_y, and n_z are also set based on the input data (see parameter settings in section IV).

F. Connection Evaluation

The same as the node evaluation, this method also reduces defects of the previous methods. By using the set of reference nodes D, a connection (k, l) can be evaluated using (26-28).

$$Ce(k,l) = \begin{cases} 1 & dQ_x > q_x \wedge dQ_y > q_y \wedge dQ_z > q_z \\ 0 & otherwise \end{cases} \tag{26}$$

$$dQ_x = |t_x - w_i x|, dQ_y = |t_y - w_i y|, dQ_z = |t_z - w_i z|, i \in D \tag{27}$$

$$t_x = \frac{k_x + l_x}{2}, t_y = \frac{k_y + l_y}{2}, t_z = \frac{k_z + l_z}{2} \tag{28}$$

Where, q_x, q_y, and q_z are threshold values for the connection evaluation (see parameter settings for more details). This method calculates and compares the center point t of the connection to the reference nodes. Thus, the bad connections will have the $Ce(k, l) = 0$, and should be removed from the corresponding network. On the other hand, the center point on the correct connections will be surrounded by reference nodes, e.g., the connection k to k' which has $Ce(k, k') = 1$ as shown in Fig. 9.

IV. Experimental Results

A. Parameter Settings

All parameters for data measurements can be seen in Table II. α, β, $l_{\min}$, $l_{\max}$, and L are the parameters for the simplified FastGNG algorithm. In the first phase, $l_{\min}$ and $l_{\max}$ are set for higher value with lower L to cover the input data as fast as possible. Then, the values are gradually reduced for the higher phase level to cover the input data in more detail. On the other hand, the L value is increased with the increasing phase level to generate more nodes, and hence better detection results can be achieved. In addition to L, the internal loop can also be terminated when no nodes are added due to l_{min} and l_{max} values.

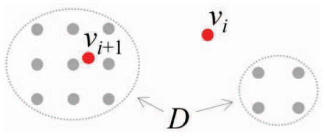


Fig. 8. Node evaluation for a false detection. Because thare are no reference nodes near the node v_i, this node is removed from the network.

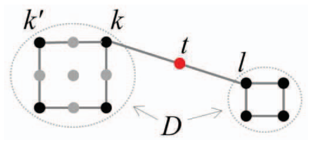


Fig. 9. Connection evaluation for a false detection. The k node is a direct connection of the l node, $c_{k,l} = 1$. This connection is removed from the network as the $Ce(k, l) = 0$.

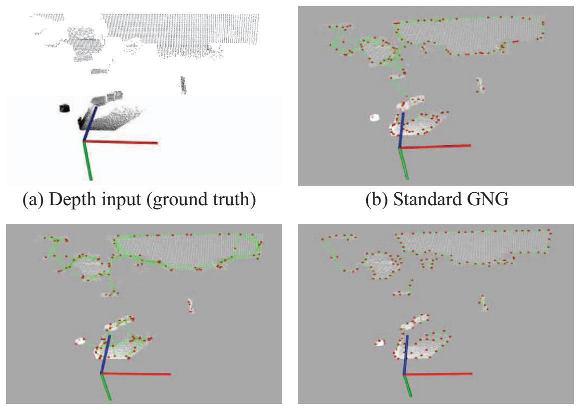
TABLE II. PARAMETER SETTINGS FOR DATA MEASUREMENT

Parameter	Value	Definition
α	3.9	The constant for generating a set of sample nodes *A*
β	1.55	The constant for $P_{node}(v)$
$l_{\min}$	$l_{\min} = \begin{cases} 0.225\text{ m} & iP = 1 \\ 0.150\text{ m} & iP = 2 \\ 0.100\text{ m} & iP \geq 3 \end{cases}$	Minimum length of a connection in the network for each phase level *iP*
$l_{\max}$	$l_{\max} = \begin{cases} 0.300\text{ m} & iP = 1 \\ 0.225\text{ m} & iP = 2 \\ 0.175\text{ m} & iP \geq 3 \end{cases}$	Maximum length of a connection in the network for each phase level *iP*
L	$L = \begin{cases} \text{ceil}\left\lceil \frac{\#A}{10} \right\rceil & iP = 1 \\ \text{ceil}\left\lceil \frac{\#A}{8} \right\rceil & iP = 2 \\ \text{ceil}\left\lceil \frac{\#A}{6} \right\rceil & iP \geq 3 \end{cases}$	The internal loop parameter for mini-batch learning. *#A* is the number of sample nodes. In this case, the *max_phase* is set to 3.
g_x, g_y, g_z	$g_x = 0.08$ m, $g_y = 0.035$ m, $g_z = 0.08$ m	The thresholds for ground removal
η_1, η_2	$\eta_1 = 0.01$, $\eta_2 = -0.95$	The parameters for open-loop connection
n_x, n_y, n_z	$n_x = 0.2$ m, $n_y = 0.2$ m, $n_z = 0.2$ m	The thresholds for node evaluation
q_x, q_y, q_z	$q_x = 0.06$ m, $q_y = 0.06$ m, $q_z = 0.06$ m	The thresholds for connection evaluation

The remaining parameters are used in the post-processing to correct defects. All of them are adjusted based on the depth input characteristic, i.e., minimum distance between nodes. Therefore, different camera settings will require different values for these parameters. In this case, we have adjusted them for 3D datasets of our laboratory and used in the experiments.

B. *Comparison with Related Works*

Before applying the proposed method to the system, we performed several comparisons to verify its performance and stability. The standard GNG, FastGNG, and simplified FastGNG were executed for the same input dataset. Fig. 10 shows the detection results. In this case, they ran the object detection without any post-processing methods.


(a) Depth input (ground truth) (b) Standard GNG

(c) FastGNG (d) Simplified FastGNG

Fig. 10. Comparison of the proposed method with related works. White dots are the point cloud data of the objects (input).

TABLE III. PERFORMANCE COMPARISON WITH RELATED WORKS

Standard GNG					
No.	**Number of Nodes**	**Number of Connections**	**Iteration / Phase**	**Time [ms]**	**Mean Time [ms]**
1	150	159	3700	58.3	67.9
2	150	165	3780	61.8	
3	150	171	4120	85.7	
4	150	164	4070	68.3	
5	150	166	3840	65.4	
FastGNG					
6	148	278	3	22.5	22.2
7	149	269	3	23.2	
8	152	297	3	21.2	
9	151	278	3	21.5	
10	150	285	3	22.8	
Simplified FastGNG (proposal)					
11	155	146	3	13.6	16.2
12	144	132	3	18.2	
13	158	143	3	17.3	
14	152	137	3	16.0	
15	145	130	3	15.8	

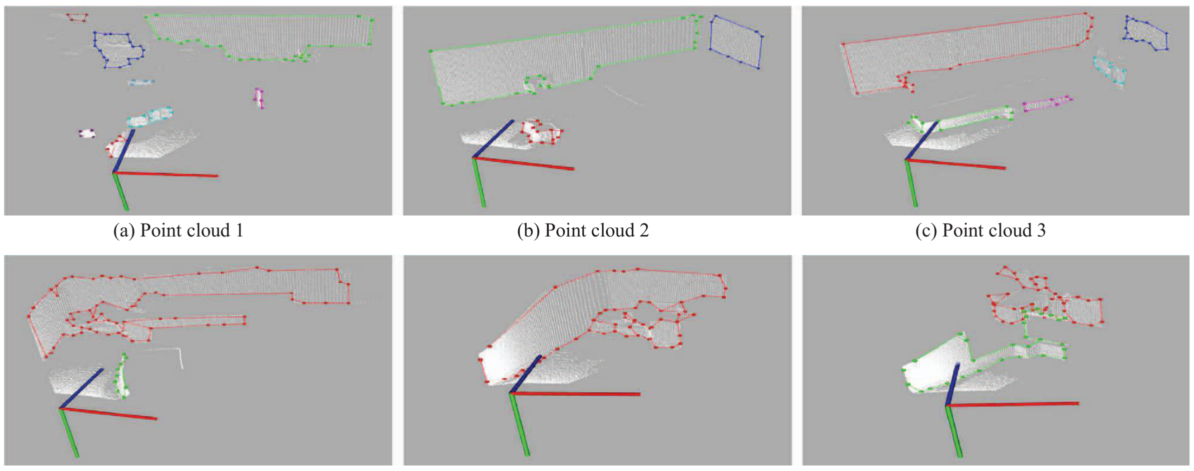
The results are also summarized in Table III. It is difficult to run for the same number of nodes in each trial, especially for FastGNG and simplified FastGNG. Based on the results, the standard GNG algorithm has the highest computational cost. It really depends on the iteration. Meanwhile, the FastGNG algorithm performs faster. It is roughly three times faster than the standard GNG algorithm. The simplified FastGNG has a similar performance as the FastGNG. It uses the same core of the FastGNG algorithm, however, in a simpler way in order to achieve an improved performance for the system.

C. *Object Extraction Results*

Finally, the simplified FastGNG algorithm is combined with the post-processing methods, as shown in Fig. 3. The input is 3D point cloud data and the final output is clusters of the detected objects. Fig. 11 shows the detection results. In the experiments, the 3D point cloud datasets of our laboratory were used. The datasets were previously collected using the LiDAR camera as described in section II. They all have the same resolution of 640×480 pixels.

The performance of the proposed method can be seen in Table IV. Based on the results, a maximum of 36.2 fps can be achieved. The mean value of 30.7 fps is fast enough as the LiDAR camera also has the same specification which is 30 fps. Table V shows the distance metric of the experiments. The values of mean distance of nodes and center of connections are close to zero. It means that the detection results are similar to the shape of the input point cloud. From more than 10000 points of point cloud to only less than 100 outputs (nodes and connections), higher compression ratios can also be achieved.

Compared with our previous work which was able to perform up to 15.1 fps [14], this is a 142% improvement. The accuracy is also improved since we have more nodes for each object compared to only eight in the previous, as shown in Fig. 12. The more the number of nodes the more detail the extraction results to represent the detected objects.


(a) Point cloud 1 (b) Point cloud 2 (c) Point cloud 3

(d) Point cloud 4 (e) Point cloud 5 (f) Point cloud 6

Fig. 11. Object extraction results of the proposed method. White dots are the point cloud data of the objects. Color-dots and color-lines are the nodes and connections, respectively, as the output of the algorithm. Different colors indicate different clusters.

TABLE IV. SUMMARY OF THE EXPERIMENTAL RESULTS

No.	**Input**	**Number of Nodes**	**Number of Connections**	**Number of Clusters**	**Time [ms]**
1	Fig. 11 (a)	77	80	8	32.7
2		74	77	7	36.1
3		69	75	8	35.4
4	Fig. 11 (b)	48	49	3	27.6
5		48	52	3	30.0
6		53	58	3	30.1
7	Fig. 11 (c)	61	67	5	34.6
8		54	59	5	33.5
9		57	62	5	35.4
10	Fig. 11 (d)	80	86	2	35.6
11		75	80	3	34.1
12		87	94	3	37.1
13	Fig. 11 (e)	54	60	1	32.1
14		43	46	1	29.7
15		46	50	1	30.2
16	Fig. 11 (f)	70	76	2	30.0
17		81	91	2	30.3
18		69	76	2	32.1
Frame Rate		Mean $= \frac{1\text{ s}}{32.6\text{ ms}} = 30.7$ fps Min $= \frac{1\text{ s}}{37.1\text{ ms}} = 26.9$ fps, Max $= \frac{1\text{ s}}{27.6\text{ ms}} = 36.2$ fps			

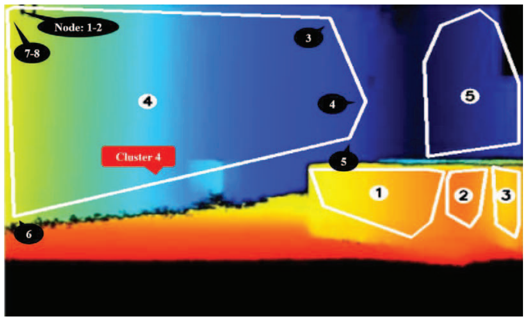
V. CONCLUSION

Perceiving the real-world in 3D space is becoming increasingly important, and robotics is no exception. The main problem of processing 3D data is the large amounts of data, especially for embedded computers in mobile robots. Thus, the proposed object extraction method is performed to compress or simplify the data. This will not only reduce the computational load of the main algorithm but is also necessary for communication purposes. As mentioned earlier, we need to send all the observation data from the mobile robot to a PC controller for control and monitoring purposes. The smaller the data size, the faster the communication time to support real-time applications. This is especially important for the low-bandwidth network which has limited bandwidth to send large data sizes.

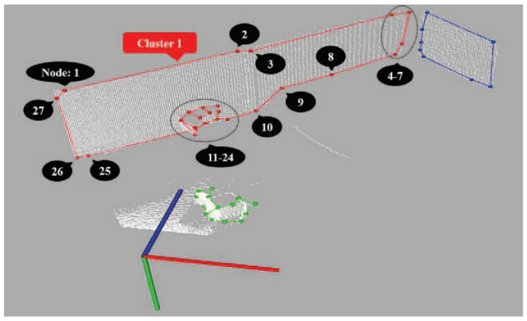
The simplified FastGNG algorithm also plays an important role since it is not only faster than the standard GNG algorithm but also simpler in computation. The experimental results have proven that the proposed method can achieve up to 36.2 fps which is real-time capable. Moreover, the algorithm was run using a Raspberry Pi on the mobile robot which typically has less power than general-desktop CPUs. Compared with the previous work that only has eight nodes for each plane, the proposed method can generate more nodes to represent the detected objects more precisely. Therefore, the extraction accuracy is also increased.

TABLE V. DISTANCE METRIC OF THE EXPERIMENTAL RESULTS

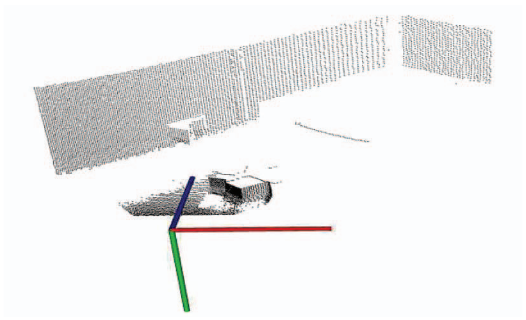
No.	**Input**	**Number of Points in Point Cloud**	**Number of**		**Mean Dist. of Nodes from Point Cloud**	**Mean Dist. of Center of Connections from Point Cloud**	**Min Distance between Connected Nodes**	**Max Distance between Connected Nodes**
			Nodes	**Connections**				
1	Fig. 11 (a)	8586	77	80	0.000 m	0.018 m	0.100 m	2.430 m
2	Fig. 11 (b)	11647	48	49	0.000 m	0.017 m	0.100 m	3.210 m
3	Fig. 11 (c)	10455	61	67	0.000 m	0.021 m	0.100 m	3.180 m
4	Fig. 11 (d)	10802	80	86	0.000 m	0.020 m	0.102 m	1.575 m
5	Fig. 11 (e)	13948	54	60	0.000 m	0.019 m	0.102 m	1.217 m
6	Fig. 11 (f)	14094	70	76	0.000 m	0.021 m	0.101 m	0.560 m



(a) Plane extraction result from the previous work [14]



(b) Simplified FastGNG final result (proposal)



(c) Ground truth of the input

Fig. 12. Simplified FastGNG final result compared with the previous work. Both (a) and (b) use the same 3D point cloud input (c). However, the result is plotted in 2D viewpoint for (a) and in 3D viewpoint for (b).

For the future work, we will utilize the object extraction results to build a 3D map of the environment in collaboration with the SLAM algorithm. Real-time system is still the main target for the implementation using the mobile robot system.

References

[1] K. Helin, J. Karjalainen, T. Kuula, and N. Philippon, "Virtual/mixed/augmented reality laboratory research for the study of augmented human and human-machine systems," 12th International Conference on Intelligent Environments, pp. 163-166, 2016.

[2] H. Balta et al., "Integrated data management for a fleet of search-and-rescue robots," J. Field Robotics, vol. 33, pp. 539-582, 2017.

[3] Y. Tian et al., "Search and rescue under the forest canopy using multiple UAVs," Int. J. Robotics Research, vol. 39, pp. 1201-1221, 2020.

[4] M. Schwarz et al., "NimbRo rescue: solving disaster-response tasks with the mobile manipulation robot Momaro," J. Field Robotics, vol. 34, pp. 400-425, 2017.

[5] K. Okada, S. Kagami, M. Inaba, and H. Inou, "Plane segment finder: algorithm, implementation, and applications," IEEE International Conference on Robotics and Automation, vol. 2, pp. 2120-2125, 2001.

[6] D. Geromichalos et al., "SLAM for autonomous planetray rovers with global localization," J. Field Robotics, vol. 37, pp. 830-847, 2020.

[7] Q. Sun, J. Yuan, and F. Sun, "RGB-D SLAM in indoor environments with STING-based plane feature extraction," IEEE Transactions on Mechatronics, vol. 23, no. 3, pp. 1071-1082, 2018.

[8] M. A. Al Mamun, M. T. Nasir, and A. Khayyat, "Embedded system for motion control of an omnidirectional mobile robot," IEEE Access, vol. 6, pp. 6722-6738, 2018.

[9] E. Rohmer et al., "Quince: a collaborative mobile robotic platform for rescue robots research and development," The 5th International Conference on the Advance Mechatronics, vol. 5, pp. 225-230, 2010.

[10] M. Inaba, S. Kagami, F. Kanehiro, Y. Hoshino, and H. Inoue, "A platform for robotics research based on the remote-brained robot approach," Int. J. Robotics Research, vol. 19, no. 10, pp. 933-954, 2000.

[11] H. Mols, K. Li, and U. D. Hanebeck, "Highly parallelizable plane extraction for organized point clouds using spherical convex hulls," IEEE International Conference on Robotics and Automation, pp. 7920-7926, 2020.

[12] N. Kubota, "Multiscopic topological twin in tobotics," The 28th International Conference on Neural Information Processing, Online, December 2021.

[13] B. Fritzke, "A growing neural gas network learns topologies," Advances in Neural Information Processing Systems, vol. 7, pp. 625-632, 1995.

[14] A. Junaedy, H. Masuta, K. Sawai, T. Motoyoshi, and N. Takagi, "A plane extraction method for embedded computers in mobile robots," IEEE Symposium Series on Computational intelligence, 2021.

[15] M. Georgiev, E. Belyaev, and A. Gotchev, "Depth map compression using color-driven isotropic segmentation and regularised reconstruction," Data Compression Conference, pp. 153-162, 2015.

[16] T. Sonoda and A. Grunnet-Jepsen, "Depth image compression by colorization for Intel RealSense depth cameras," Intel, Rev 1.0, 2021 [Online]. Available: https://dev.intelrealsense.com/docs/whitepapers.

[17] X. Wang et al., "Fast depth video compression for mobile RGB-D sensors," IEEE transactions on Circuits and Systems for Video Technology, vol. 26, no. 4, pp. 673-686, 2016.

[18] S. Yan, Y. Peng, G. Wang, S. Lai, and M. Zhang, "Weakly supported plane surface reconstruction via plane segmantation guided point cloud enhancement," IEEE Access, vol. 8, pp. 60491-60504, 2020.

[19] J. Overby, L. Bodum, E. Kjems, and P. M. Ilsoe, "Automatic 3D building reconstruction from airbone laser scanning and cadastral data using Hough transform," International Archives of Photogrammetry, Remote Sensing and Spatial information Sciences, vol. xxxv-B3, pp. 1-6, 2004.

[20] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," Communication of the ACM, vol. 24, no. 6, pp. 381-395, 1981.

[21] B. Oehler, J. Stueckler, J. Welle, D. Schulz, and S. Behnke, "Efficient multi-resolution plane segmentation of 3D point clouds," Proceedings of the 4th International Conference on Intelligent Robotics and Applications, pp. 145-156, 2011.

[22] A. Junaedy, H. Masuta, K. Sawai, T. Motoyoshi, and N. Takagi, "LPWAN-based real-time 2D SLAM and object localization for teleoperation robot control," J. Robot. Mechatron., vol. 33, no. 6, pp. 1326-1337, 2021.

[23] A. Junaedy, H. Masuta, K. Sawai, T. Motoyoshi, and N. Takagi, "Real-time simultaneous localization and mapping for low-power wide-area communication," IEEE Symposium Series on Computational Intelligence, pp. 1905-1912, 2020.

[24] M. Mary Synthuja Jain Preetha, L. Padma Suresh, and M. John Bosco, "Image segmentation using seeded region growing," International Conference on Computing, Electronics and Electrical Technologies, pp. 576-583. 2012.