Ph.D Thesis

High Accuracy Real-Time 6D SLAM based on 3D Laser Range Finder

Principal supervisor Prof. Yasutaka Fujimoto

Division of Electrical and Computer Engineering

Graduate School of Engineering

Yokohama National University, Japan

Student number 18QC591

Jiayi Wang

May 2021

Abstract

In this thesis we proposed methods to improve the performance of the laser based 6D SLAM in accuracy and computational cost.

One key problem for the laser based scan matching is finding corresponding points pairs. Therefore, we proposed solutions aiming at decreasing the possibility of the incorrect correspondences or mistakenly matching.

In the first, we present a method to extract features such as the edges and corners in real-time from 3D point clouds to decrease the probability of the incorrect correspondences. To extract such features , we propose the use of a trained neural network (NN). The NN used in our feature extraction scheme is a simple backpropagation (BP) NN with two hidden layers, which allows us to build a real-time system for the purpose of 6D SLAM.

Then we extend the neural network to recognize more objects by using a convolutional neural network. Comparing with only classifying feature and no-feature, we extend the second method and it can cluster more objects such as points of trees or truck, etc.. In our consideration, matching the same objects can avoid the incorrect corresponding in ideal situation. For scan registration, we also improve the Particle Swarm Optimization (PSO) to match the recognized points. Using PSO to match the recognized object's points in each neighboring scan can help decrease incorrect correspondences and enhance the robustness of scan matching. Compared with state-of-art methods, the first proposed method and the extended method achieved good performance in the KITTI Odometry Benchmark and our SLAM experiments.

Table of Contents

List of Figures

List of Table

1. Introduction

1.1 Background

Simultaneous localization and mapping (SLAM) is an important problem for many autonomous robotic applications, such as autonomous vehicles and rescue robots. The SLAM problem involves creating appropriate representation for both the observation and motion models [1]. Lots of researchers have been working to solve the SLAM problem, and the most widely used sensors are categorized into laser-based, sonar-based, and vision-based systems. In addition, some sensors are used to better estimate the robot state and the outside world, such as, compasses, infrared technology, and the Global Positioning System (GPS). However, all these sensors are susceptible to certain errors, and also have several range limitations, such as, in indoor environments or in tunnels. Most autonomous robots and self-driving cars have to be capable of localizing themselves in real-time, ideally using their own sensors without help from external information such as GPS.

In the past few years, for vision-based SLAM, there has been dramatic advances in RGB-D-based [2, 3] SLAM systems. Most of these methods use dense reconstructions of the environment which have the advantage of using all available information, and thus do not rely on feature extraction. However, there are also other state-of-the-art visual approaches, such as Stereo LSD-SLAM [4], a stereo vision-based complete SLAM system with pose graph optimization and currently best-performing approach SOFT SLAM [5], which is based on careful feature selection and tracking. Current 3D laser-based SLAM, which is also our approach to SLAM, mainly accomplishes the estimation by relying on feature-based methods [6, 7], voxel grid-based approaches [8], or point cloud down sampling [9], which all decrease the data used for scan matching and the computational cost. However, most of laser-based SLAM has been investigated using many approaches [10] [11] which

primarily make use of the ICP algorithm. The ICP [12] algorithm is widely used for point cloud registration and has also been extended to point cloud-based SLAM [13, 14, 15, 16].

In fact, neighboring 3D point clouds scanned by LiDAR can never contain completely correct point-to-point correspondences due to random errors, discreteness of LiDAR sensor data, or the dynamic environment. In addition, the irregular motion particularly irregular high-speed motion, of a robot with installed LiDAR significantly increases the possibility of incorrect correspondences. In our consideration, the key problem of the scan matching is the incorrect corresponding. Incorrect correspondences can cause the local best or a totally wrong solution for scan matching. Incorrect ICP correspondences, will lead it to find the local best or a definitely incorrect transformation when matching two point clouds. To solve the correspondence problem of ICP-based registration, many studies have investigated various types of feature extraction, such as extracting color information [31], intensity values [32], or normals [33]. In point cloud registration, incorrect ICP correspondences can be avoided by extracting edges or planes from 3D point cloud as feature points [6]. Using the extracted feature points from neighboring source 3D point clouds for registration can decrease the possibility of incorrect correspondences. Therefore, our solution for a high accuracy 3D scan matching is to decrease the probability or eliminate the incorrect correct correspondences. To decrease the probability of the incorrect correspondences, we proposed to use the feature points. In our consideration, the feature points are the points with the same physical meaning. In this paper we selected the very common features edges and corners of the real world. There are so many edges and corners in the real 3D world such as the intersection of the wall and ground.

For the 3D laser-based SLAM using the ICP, we propose the feature-based method for point cloud registration because a vital issue in the use of ICP is accuracy in identifying corresponding point clouds [17]. A pair of mistakenly identified corresponding point clouds, usually stemming from a dynamic environment or the low frequency of a laser range sensor, tends to create a major error. If we can filter the point clouds of each scan and its successor by a few features and match only the extracted feature points, we can increase the likelihood of accurately finding the corresponding point clouds. For example, if we extract the shapes of the edges or corners from two neighboring scans (3D point clouds) and discard all of the

point clouds of the ground or walls, we can avoid mistakenly matching the extracted feature points (the edges and corners) with the no-feature points (the ground and walls). The other methods for the feature extraction such as the method that creating the descriptor using the color information [31], intensity values [32], normals [33] or the conventional method extracting the edges using the normal vector, are causing too much computational consumption that SLAM system cannot run in real time. Therefore we make much effort to decrease the computational time while we can also ensure the accuracy of the 6D SLAM. The proposed method uses a simple trained BPNN to extract the the feature points to ensure the real-time issue and the accuracy of feature extraction. In addition, we also introduced the PCA [28] [29] to fast uniform the inputs for the proposed BPNN. Using the PCA is one of the key points why we can keep the accuracy and the real-time. We also decrease the computational time by decreasing the input size of the proposed BPNN.

Besides the ICP algorithm for laser-based SLAM, other studies have presented attempts to solve the SLAM problem using approaches based on particle swarm optimization (PSO) [18, 19], demonstrating that the PSO algorithm, with a two-dimensional (2D) occupancy voxel grid map, may be used to obtain an accuracy higher than that of the ICP algorithm. Therefore, we also consider to introduce the PSO algorithm to our SLAM system. However, the method using the PSO in [18, 19] is for the 2D point clouds matching and the 3D SLAM. We modify many parameters by many simulations and tests to extend the conventional PSO for 3D SLAM into the 6D SLAM. At first, the PSO in 6D SLAM performed not well because of the heavy computational cost. The computational consumption is much larger than in 3D SLAM since the robots' movement need to be calculated is twice more. Therefore, we put much effort in decrease the computational consumption by introducing the parallel calculation for each particles. PSO is a metaheuristic and it can search a large solution space efficiently without knowing much given information. In metaheuristic, the meta in the PSO shows that it has the better searching capability than the conventional heuristic. In addition, the heuristic shows that the PSO can search a solution which is closest to the real solution, in a reasonable computational time [20]. In our SLAM system, the accuracy and the computational time are the top 2 important issues. Therefore, the PSO is very suitable for our SLAM system. The 6 dimensional movement of the robot

is irregular and each dimension is independent. Therefore, different swarms of particles can search its own dimension without affected by the other dimension's searching process. However, each dimension can summarize and communicate with the other dimensions' solution at the end. Therefore, the PSO is very suitable for the SLAM problem who has irregular movement. In addition, the searching space is easy to decide since the searching space is related to the velocity of the robot. Therefore, in the 6D SLAM, the searching space for the SLAM can be easily decided by the velocity of the robot or the command value. The optimization of the proposed methods in this thesis is based on the 3D grid occupancy map. The theoretical maximal error depends on the grid size of the occupancy map. However, by optimizing the matched number of points or the proposed score, the error should be less than the theoretical maximal error.

In this paper, we present a feature-based 6D SLAM algorithm and a voxel grid-based approach using the PSO algorithm for the optimization. We employ the feature points (the edges and corners) where many surfaces meet a widely utilized approach [21], together with its variants [22, 23], to focus on the calculation of the normal vectors and the curvature of a point and its surroundings. Unfortunately, the computational cost of calculating normal vectors, stemming from the neighboring point search based on the Euclidean distance or the kd-tree, is too high to be applied to a real-time 6D SLAM system. Therefore, to build a local 3D occupancy grid map, we can use machine learning or neural networks (NN) to learn and recognize the structures of the edges or corners, skipping the standard search for neighboring points. Because the classification of feature points is not a complex problem, we select a simple but classical NN with a low computational cost, called the backpropagation NN (BPNN) [24]. To define the inputs for the BPNN, we take inspiration from the concept of sub-windows, used in various studies investigating image processing [25], and extend it to a 3D model named slide-window. Instead of calculating the closest point or using the kd-tree, each point is assigned a sub-cube that is used to determine its surroundings. To avoid repetitive training, we introduce the principle component analysis (PCA) as the [28, 29], which directly uses the PCA to match the point clouds. However, we introduce this to uniform the coordinate for the sub-cube to unite the input for the proposed BPNN. This is an important part in our system, because the quantity of sub-

cubes in on scan is very large, without using the PCA to uniform the coordinate, the computational consumption is too heavy. However, without uniforming the coordination, the BPNN cannot be trained so well because of the repetitive training. We also extend the methods [18, 19] which are using the PSO algorithm into the 3D laser-based SLAM [34]. As our survey, the only method using the PSO aim at matching the original 2D point clouds. However, the proposed method extend and make the PSO can match the 3D point clouds using the feature points. The PSO algorithm presents a large computational load that may bypass the best local solution if it is not correctly initialized, which has led researchers to explore its computational cost and investigate the associated initialization problems. Because the initial location of each particle is very important for the PSO algorithm, we propose to use the ICP algorithm first to roughly match the 3D point clouds and calculate the 6DoF movement of particles, after which the PSO algorithm is initiated using the solution obtained by the ICP algorithm. By employing an initialization randomly around the rough solutions from the ICP instead of a totally random value assignment, the PSO algorithm requires a smaller number of particles and can easily avoid getting trapped during iterations by using a smaller computational load. In the optimization using the PSO algorithm, we present a way to find the best solution which can best match the extracted feature points with the global map stitched by the previous feature data. To using the PSO, the parameter is hard to decide. Therefore, we reference many studies that improves the original PSO as [30] and we also decided the parameters of the PSO by many test in simulations.

However, using extracted edges and planes as the feature points to be input point clouds for ICP registration involves finding the minimized distance error metric between two neighboring feature point clouds rather than calculating the minimized distance between the whole neighboring point clouds. Typically, the best transformation between two neighboring point clouds is the local best transformation between the whole neighboring point clouds.

Beside extracting the feature points from the 3D point clouds, we extend and propose another method to match whole neighboring point clouds with a low probability of incorrect correspondences. To address the incorrect correspondence problem, rather than using

feature points for point clouds registration, we prefer to identify the meaning of each point inside the source 3D point clouds and match neighboring points that have the same meaning. To identify the meaning of each point, we introduce machine learning with a convolutional neural network (CNN) to recognize the objects in the source 3D point clouds. The input of the proposed CNN is created by using the sliding window. The sliding window and CNN have already been used in object recognition for 3D point clouds [36] [37]. The proposed object recognition method uses the sliding-window concept; however, the setting of our sliding window has more elements. In addition, in the proposed method, the usage of the sliding window to build the input to the CNN also differs from the previous studies. We decrease the computational time by zip the sliding windows into images. To zip the sliding windows into images, we introduce many opened source methods such OpenCV [38], softmax [40] and bilinear interpolation [39]. We preset the label of each object to be recognized and train the CNN with serial 3D point clouds which each point is marked a label. Then, we can input any source point cloud to the trained CNN and obtain an output 3D point cloud with each point is marked with a recognized label. Comparing with the conventional methods that uses the PSO to match 2D point clouds. We not only extend the PSO into the 3D world, but also modify the evaluation function of the PSO algorithm such that the algorithm can evaluate incorrect correspondences using the recognized object. We assign a high score to matched points that have the same label and low scores to matched points that have different labels. Therefore, we can match the whole neighboring point clouds by optimize their matching score. Thus, we can also find the best transformation for the whole point clouds by considering incorrect correspondences.

The key contributions of our work are as follows:

• We designed a 26 dimensional vector as the input of the proposed BPNN which can represent a group of 3D points in the proposed BPNN.

• We uniformed the rotation and the coordination of all the inputs of the proposed BPNN by analyzing their principal components.

• We present an efficient way to extract feature points (the edges and corners) from a 3D point cloud using the BPNN.

• We extend the BPNN to CNN and we can recognize the preset objects of a 3D point

cloud using the proposed CNN.

• We propose a way to optimize the 3D laser-based SLAM by using the PSO algorithm and we also present a way to let the PSO can matched the recognized objects.

• Compared with the state-of-art methods, the proposed SLAM system can run in real-time and achieve the best accuracy in 3D laser-based SLAM.

Our experimental evaluation will be primarily based on the KITTI Odometry Benchmark [26]. We will show the feature extraction result and the object recognition, but we cannot evaluate how accurate the extracted feature is. We also show the result of the SLAM and compare it with other state-of-the-art methods [6, 27, 35, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50] with a good result in the KITTI dataset.

2. Algorithms

2.1 6D SLAM with Feature Extraction

2.1.1 Feature Extraction using a Neural Network

In this section, we describe the design of the proposed BPNN for the feature extraction of 3D point clouds and present some details of the sub-cube concept, as well as how this concept can be used to generate inputs for the network. Firstly, we show the process of extracting feature points using the proposed BPNN, as presented in Fig.2.1. As the flowchart in Fig.2.1 shows, we explain each step of the proposed method for feature extraction from 3D point clouds.

2.1.2 Sub-Cube for a 3D Point Cloud

Instead of calculating the Euclidean distance or using a kd-tree to obtain the corresponding points for our method, we build local 3D occupancy grid maps of each scan and use a sub-cube for each occupied point, as detailed in Fig.2.4. The local occupancy grid map has a resolution of 10 cm (in accordance with the global occupancy map) and is built using only the current scan data, whereas the global map is stitched together using all of the obtained scan data. The sub-cubes can help decrease the computational cost because they can down sample the original 3D point cloud and avoid the search for the closest points. For each sub-cube used in our experiment, we set our point of focus (marked in green in Fig.2.4) as the center and define 7^3 smaller cubes in its vicinity (marked in blue in Fig.2.4). The size of the sub-cube depends on the resolution of the local occupancy grid map (10 cm), and both the size of the sub-cubes and resolution of the local occupancy grid map determine the range of points or information used to classify the feature points. For example, the

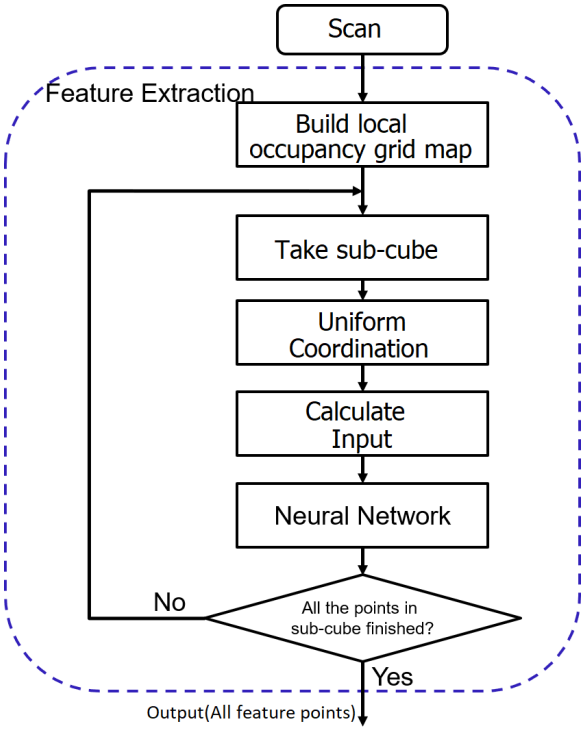

Fig. 2.1: Flow chart of a feature extraction.

information that we use to judge a feature point in the experiment reported here comprises all points inside a cube with a volume of $(70cm)^3$. We decided to use 7^3 sub-cubes because this amount of information produced adequate simulation results. When using 3^3 or 5^3 sub-cubes, we obtained large training errors, whereas our tests with 9^3 sub-cubes led to large training errors and increased the computational cost. The obtained training error is shown in Fig.2.2. We have found that the smaller cubes cause larger errors because their smaller size means that they contain fewer points, implying only an inadequate amount of information can be used to classify the edges. In our experiment, the size of the sub-cubes and the resolution of the local occupancy grid map are selected for the edge classification instead of the point cloud registration.

2.1.3 The Input of the proposed Neural Network

We describe the formation of inputs for the proposed BPNN.

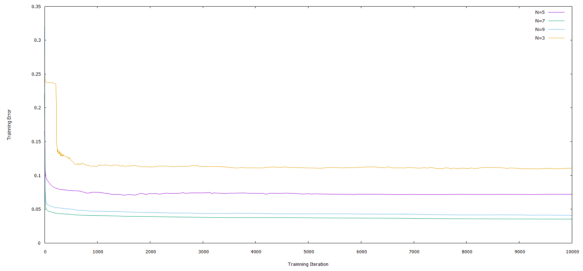

Fig. 2.2: Training error of the proposed neural network for various sub-cube sizes.

- Step 1

  Define a sub-cube for one point inside the local occupancy grid map and form a point cloud P such that:

$$\boldsymbol{P} = (\boldsymbol{p}_1, \boldsymbol{p}_2, ..., \boldsymbol{p}_n) \tag{2.1}$$

$$\boldsymbol{p}_j = (x_j, y_j, z_j)^T \tag{2.2}$$

  where $\boldsymbol{P}$ is the point cloud of the sub-cube and $\boldsymbol{p}_1$ the point of focus or its center. The n in Eq.(2.1) should satisfy the condition $\xi_{low} < n < \xi^3_{high}$. ξ_{low} defines the threshold limit of the minimal number of points inside a sub-cube. If there are fewer points than the threshold inside a sub-cube, the information available to judge whether it is a feature is inadequate. The size of the sub-cube is denoted by ξ^3_{high}. In our case, ξ^3_{high} equals 7^3 and ξ_{low} is 10. The symbol p_j represents the jth point inside the sub-cube.

- Step 2

  When a robot rotates, the structure of the sub-cube usually deviates from that of the training samples, meaning that we cannot use each sub-cube directly. Therefore, all sub-cubes should be set in a uniform coordinate system. We represent the sub-cubes with eigenvectors of their covariance matrix and roughly rotate them into a uniform coordination. In fact, there are some studies that use the principal component analysis (PCA) such as [28] [29] for roughly aligning two similar point clouds. Therefor, we also absorb some part of the PCA that we calculate the covariance matrix of

each sub-cube and its eigenvector matrix. However, for some points, the covariance matrix might not be orthonormal matrix. Then, we judge the point as a non-feature point, as

$$\boldsymbol{p}_{cg} = \frac{1}{n}\sum_{j=1}^{j=n}\boldsymbol{p}_j \tag{2.3}$$

$$\boldsymbol{C}_{ov}(\boldsymbol{P}) = \frac{1}{n}\sum_{j=1}^{n}[(\boldsymbol{p}_j - \boldsymbol{p}_{cg})(\boldsymbol{p}_j - \boldsymbol{p}_{cg})^T] \tag{2.4}$$

where $\boldsymbol{p}_{cg}$ is the center of gravity of point cloud $\boldsymbol{P}$ and C_{ov} is the covariance of the sub-cube.

- Step 3

  We calculate the eigenvector matrix of $\boldsymbol{C}_{ov}$, $\boldsymbol{V}$, and the rotation matrix for the uniform coordination, $\boldsymbol{R}$, as follows:

$$\boldsymbol{V} = (\boldsymbol{v}_1\ \boldsymbol{v}_2\ \boldsymbol{v}_3) \tag{2.5}$$

$$\boldsymbol{I} = \boldsymbol{R}\boldsymbol{V} \tag{2.6}$$

$$\boldsymbol{R} = \boldsymbol{V}^{-1} \tag{2.7}$$

  where $\boldsymbol{v}_1$ is a unit eigenvector calculated using the minimum eigenvalue of the covariance matrix, $\boldsymbol{v}_3$ is calculated using the maximum eigenvalue and $\boldsymbol{v}_2$ is calculated using the remaining eigenvalues. To simplify the calculation, we define the uniform coordination through a unit matrix I in Eq.(2.6). The rotation matrix is the inverse of the eigenvector matrix.

- Step 4

  If the point cloud $\boldsymbol{P}$ is transformed into $\boldsymbol{P}_{uni}$ using the calculated rotation matrix, each point on the inside can be expressed as follows:

$$\boldsymbol{P}_{uni} = \boldsymbol{R}\boldsymbol{P} \tag{2.8}$$

$$\boldsymbol{P}_{uni} = (\boldsymbol{p}_1^{'}, \boldsymbol{p}_2^{'}, ..., \boldsymbol{p}_n^{'}) \tag{2.9}$$

$$\boldsymbol{p}_j^{'} = (x_j^{'}, y_j^{'}, z_j^{'})^T \tag{2.10}$$

The examples of the applied uniform coordination are presented in Fig.2.1.3, where the axes of the uniform coordination can be seen in red, blue, and green. The pink, yellow and light blue axes correspond to the eigenvectors v_1, v_2, and v_3, respectively. Two examples of sub-cubes that focus on the same corner, but with different viewpoints,are shown in Fig.2.1.3 and Fig.2.1.3. The point clouds inside the sub-cubes are shown in Fig.2.1.3 and Fig.2.1.3. The sub-cubes after the rotational transformation are shown in Fig.2.1.3 and Fig.2.1.3, where the point clouds, the sub-cubes, and the eigenvector V are colored pink, yellow and light blue, respectively. As the figure demonstrates, the differences between the subcube in the uniform coordination become smaller. These smaller differences between sub-cubes of a similar edge or feature lead to smaller differences between the inputs for the proposed NN. However, if the coordination is not made uniform, the NN will be trained chaotically, receiving different inputs for the same feature.

- Step 5

  The input sent to our BPNN is a 26-dimension vector in which each dimension denotes a part of the sub-cube. The separated parts are shown as Fig.2.5. Each dimension is defined as follows:

$$\boldsymbol{I}_j = (i_{1j}, i_{2j}, ..., i_{26j}) \tag{2.11}$$

$$i_{kj} = \sqrt{{x_j'}^2 + {y_j'}^2 + {z_j'}^2} \tag{2.12}$$

$$
k \;=\; \begin{cases}
1 & (x_j^{'} > 0, y_j^{'} > 0, z_j^{'} > 0) \\
2 & (x_j^{'} > 0, y_j^{'} > 0, z_j^{'} < 0) \\
3 & (x_j^{'} > 0, y_j^{'} < 0, z_j^{'} > 0) \\
4 & (x_j^{'} < 0, y_j^{'} > 0, z_j^{'} > 0) \\
5 & (x_j^{'} > 0, y_j^{'} < 0, z_j^{'} < 0) \\
6 & (x_j^{'} < 0, y_j^{'} < 0, z_j^{'} > 0) \\
7 & (x_j^{'} < 0, y_j^{'} > 0, z_j^{'} < 0) \\
8 & (x_j^{'} < 0, y_j^{'} < 0, z_j^{'} < 0) \\
9 & (x_j^{'} > 0, y_j^{'} > 0, z_j^{'} = 0) \\
10 & (x_j^{'} > 0, y_j^{'} < 0, z_j^{'} = 0) \\
11 & (x_j^{'} < 0, y_j^{'} > 0, z_j^{'} = 0) \\
12 & (x_j^{'} < 0, y_j^{'} < 0, z_j^{'} = 0) \\
13 & (x_j^{'} > 0, y_j^{'} = 0, z_j^{'} > 0) \\
14 & (x_j^{'} > 0, y_j^{'} = 0, z_j^{'} < 0) \\
15 & (x_j^{'} < 0, y_j^{'} = 0, z_j^{'} > 0) \\
16 & (x_j^{'} < 0, y_j^{'} = 0, z_j^{'} < 0) \\
17 & (x_j^{'} = 0, y_j^{'} > 0, z_j^{'} > 0) \\
18 & (x_j^{'} = 0, y_j^{'} > 0, z_j^{'} < 0) \\
19 & (x_j^{'} = 0, y_j^{'} < 0, z_j^{'} > 0) \\
20 & (x_j^{'} = 0, y_j^{'} < 0, z_j^{'} < 0) \\
21 & (x_j^{'} > 0, y_j^{'} = 0, z_j^{'} = 0) \\
22 & (x_j^{'} < 0, y_j^{'} = 0, z_j^{'} = 0) \\
23 & (x_j^{'} = 0, y_j^{'} > 0, z_j^{'} = 0) \\
24 & (x_j^{'} = 0, y_j^{'} < 0, z_j^{'} = 0) \\
25 & (x_j^{'} = 0, y_j^{'} = 0, z_j^{'} > 0) \\
26 & (x_j^{'} = 0, y_j^{'} = 0, z_j^{'} < 0)
\end{cases} \tag{2.13}
$$

All values in $\boldsymbol{I}_j$ are 0 in the beginning and can then be summarized using the following

equations:

$$\boldsymbol{S}_{um} = \sum_{j=1}^{m} \boldsymbol{I}_j \tag{2.14}$$

$$\boldsymbol{I}_{nput_n} = \frac{\boldsymbol{S}_{um}}{|\boldsymbol{S}_{um}|} \tag{2.15}$$

where the m is the number of points inside a sub-cube and the $\boldsymbol{I}_{nput_n}$ is the input to the proposed BPNN on the n-th point inside the scanned data.

2.1.4 Training of the proposed Neural Network

We designed our NN as depicted in Fig.2.6. To enable real-time 6D SLAM, we decided to use BPNN, a simple method devoid of complex calculations, which would lower the computational cost. The parameter of the proposed BPNN comprises two hidden layers with 512 neurons each. It was selected by conducting multiple tests that considered the computational cost and the error in the feature point classification. If the proposed BPNN is too complex, the computational cost is so heavy that the system cannot run in 10Hz. However, if the BPNN is too simple, the recognition accuracy is not enough (less than 90%). In our experiment, the recognition accuracy is higher than 95% and we can obtain a good performance in SLAM with the proposed NN. For the parameters of the BPNN, we tested many different numbers of the hidden layers and the neurons. Different parameters cause different training result. We tested the parameters with the same train rate, train samples and the validation samples. We selected the the numbers who has the best accuracy in the test as the parameters of the proposed BPNN. The test result and the tested is shown in Tab.2.1.

This network was provided only one output for each point, and it was trained by setting the sample outputs as 1 or 0, where 1 denotes the presence of feature points whereas 0 denotes their absence. If our network produces an output larger than 0.5, we consider that it has determined a feature point. The proposed feature extraction is based on the data of each scan data. The training data is taken from a simulated environment (Fig.2.7) with manually labeled feature points.

Table. 2.1: Tested parameters of BPNN.

Neurons of Layer 1	Layer 1	Layer 3	Accuracy
60	60	-	83%
64	64	-	85%
128	128	-	85%
256	256	-	90%
512	512	-	97%
1024	1024	-	96%
50	50	50	90%
100	100	100	93%
512	512	512	95%

In the experiment, we obtained more than $50,000$ positive samples (edges and corners) and $150,000$ negative samples. However, only 10% of the samples were used for testing and, the classification accuracy after $10,000$ epochs became higher than 95% while the learning rate was set to 0.01.

2.1.5 Particle Swarm Optimization and 6D SLAM using Feature Extraction

In this section, we describe how we use the extracted feature points for 6D SLAM and provide details on the application of PSO works in the system. As the process flowchart in Fig.2.8 shows, we use the extracted feature points and a combination of PSO and ICP to simultaneously estimate the 6DoF pose and build a map. In our process, ICP and PSO support each other. ICP can obtain a rough solution with small computational cost and help initiate PSO, which can use the solution, closer to the global best than a series of random solutions, to obtain the global best with fewer particles and iterations. In addition, ICP matches the data of the current scan with the data of the neighboring scans, whereas PSO matches the data of the current scan with the global map, formed by stitching

together all the scan data. Therefore, the solution obtained using ICP can be refined by PSO, ensuring that the solution is closer to the global best. The ICP algorithm used in our simulation is equal to the first ever presented [12]; it use the feature point clouds extracted by the proposed BPNN. To emphasize the connection between the proposed BPNN and the PSO algorithm, the result of the proposed NN is directly connected to the ICP method. After extracting feature points from the 3D point clouds by utilizing the proposed BPNN, we use only the feature points (filtered point cloud) and ICP for the point cloud registration. Then, the initiation of the particles for the PSO algorithm is performed by relying on the ICP solution. Therefore, the PSO algorithm acts as a refining step for the solution of the ICP algorithm. We present our method in steps.

- Step 1

  To obtain an estimation of the 6DoF movement, after obtaining the feature points from the current scan, we used the original ICP to match the feature points taken from the neighboring scan. Our calculations are as follows:

$$\Delta \boldsymbol{L}_{icp} = (\Delta \boldsymbol{t}_{icp}, \Delta \boldsymbol{q}_{icp}) \tag{2.16}$$

$$\Delta \boldsymbol{t}_{icp} = (\Delta x_{icp}, \Delta y_{icp}, \Delta z_{icp}) \tag{2.17}$$

$$\Delta \boldsymbol{q}_{icp} = (\Delta \theta_{icp}, \Delta \varphi_{icp}, \Delta \psi_{icp}) \tag{2.18}$$

  where $\Delta \boldsymbol{L}_{icp}$ is the estimation of the point cloud registration when using the ICP algorithm, while $\Delta \boldsymbol{t}_{icp}$ is the estimated translation vector, and $\Delta \boldsymbol{q}_{icp}$ the vector that contains the rotation angle.

- Step 2

  To initiate all of the particles of the PSO algorithm, we used $\Delta \boldsymbol{L}_{icp}$ to obtain:

$$\boldsymbol{u}_i(0) = (\Delta \boldsymbol{t}_i(0), \Delta \boldsymbol{q}_i(0)) = \Delta \boldsymbol{L}_{icp} + \boldsymbol{r}_i \tag{2.19}$$

$$\Delta \boldsymbol{t}_i(0) = (\Delta x_i(0), \Delta y_i(0), \Delta z_i(0)) \tag{2.20}$$

$$\Delta \boldsymbol{q}_i(0) = (\Delta \theta_i(0), \Delta \varphi_i(0), \Delta \psi_i(0)) \tag{2.21}$$

  where $\boldsymbol{u}_i(0)$ is the initial position of the particles, and $\boldsymbol{r}_i$ is a random vector with six elements. In our simulation, we obtain $\boldsymbol{r}_i \in [-0.02, 0.02]$. The index, i should fulfill

the condition of $1 \leq i \leq N_{op}$, where N_{op} is the maximum number of particles, which was set to 200 in our experiments.

- Step 3

  In the third step, we started the PSO algorithm with a few iterations. Firstly, we formed the rotation matrix as Eq.(2.31) using quaternions and the increment of angles:

$$w_i(t) = \cos\frac{|\Delta \boldsymbol{q}_i(t)|}{2} \tag{2.22}$$

$$x_i(t) = \frac{\Delta\theta_i(t)}{|\Delta \boldsymbol{q}_i(t)|}\sin\frac{|\Delta \boldsymbol{q}_i(t)|}{2} \tag{2.23}$$

$$y_i(t) = \frac{\Delta\varphi_i(t)}{|\Delta \boldsymbol{q}_i(t)|}\sin\frac{|\Delta \boldsymbol{q}_i(t)|}{2} \tag{2.24}$$

$$z_i(t) = \frac{\Delta\psi_i(t)}{|\Delta \boldsymbol{q}_i(t)|}\sin\frac{|\Delta \boldsymbol{q}_i(t)|}{2} \tag{2.25}$$

  where t denotes a solution that belongs to the t-th iteration of the PSO process.

- Step 4

  A set of the extracted features of current scan, $\boldsymbol{F}$, is transformed using the solutions for all of the particles and the robot's current pose:

$$\boldsymbol{F} = (\boldsymbol{f}_1, \boldsymbol{f}_2, ..., \boldsymbol{f}_q) \tag{2.26}$$

$$\boldsymbol{f}_j = (x_j, y_j, z_j)^T \tag{2.27}$$

$$\boldsymbol{f}'_{ij} = \boldsymbol{R}_i(t)(\boldsymbol{R}_{cur}\boldsymbol{f}_j + \boldsymbol{t}_{cur}) + \Delta\boldsymbol{t}_i(t) \tag{2.28}$$

  where q in Eq.(2.26) is the number of extracted feature points. $\boldsymbol{f}_j$ denotes one extracted feature point in $\boldsymbol{F}$ and $1 \leq j \leq q$. The rotation matrix and translation vector generated using the solution for the i-th particle in the t-th iteration are represented by $\boldsymbol{R}_i(t)$ as in Eq.(2.31) and $\Delta\boldsymbol{t}_i(t)$, respectively, whereas the rotation matrix and translation vector calculated using the currently estimated pose of the robot are marked as $\boldsymbol{R}_{cur}$ and $\boldsymbol{t}_{cur}$, respectively.

- Step 5

To evaluate the solutions generated by each particle, we calculated the number of points that can fit the current global map. The global map is a 3D occupancy grid map formed by stitching together all of the previous scans. In this paper, we set the resolution of the global map to 10 cm. The number of points inside each transformed $\boldsymbol{f}_{ij}^{'}$ can match that of the current global map, which is always upgraded from the data of the current scan. Our occupancy map is a binary map and is summarized as follows:

$$M_{ap}(\boldsymbol{p}) = \begin{cases} 1 & \text{if position } \boldsymbol{p} \text{ is occupied} \\ 0 & \text{otherwise} \end{cases} \tag{2.29}$$

$$N(\boldsymbol{u}_i(t)) = \sum_{j=1}^{q} M_{ap}(\boldsymbol{f}_{ij}^{'}) \tag{2.30}$$

where $N(\boldsymbol{u}_i(t))$ denotes the number of points that can match the global map using the i-th particle.

- Step 6

  The best i-th particle that can match the largest number of points, from the first iteration to the t-th, is denoted by $\boldsymbol{p}_{best_i}$, while the best solution for all particles and iterations is denoted by $\boldsymbol{g}_{best}$.The best solution for all of the particles is that which can be used to match the current scan data with the current global map most efficiently. In the following iteration of the PSO processing, all particles will attempt to follow the solutions of $\boldsymbol{p}_{best_i}$ and $\boldsymbol{g}_{best}$, defined as:

$$\boldsymbol{p}_{best_i} = \underset{0 \le t' \le t}{\operatorname{argmax}}(N(\boldsymbol{u}_i(t'))) \tag{2.32}$$

$$\boldsymbol{g}_{best} = \underset{1 \le i \le N_{op}}{\operatorname{argmax}}(N(\boldsymbol{p}_{best_i})) \tag{2.33}$$

  The location of the i-th particle is denoted by u_i.

- Step 7

The PSO algorithm used in this study considers the update equations, which mark use of the current values of $\boldsymbol{p}_{best_i}$ and $\boldsymbol{g}_{best}$ such that

$$\begin{aligned} \boldsymbol{v}_i(t+1) = s(\boldsymbol{v}_i(t) + c_1 r_1(\boldsymbol{p}_{best_i} - \boldsymbol{u}_i(t)) \\ + c_2 r_2(\boldsymbol{g}_{best} - \boldsymbol{u}_i(t))) \end{aligned} \tag{2.34}$$

where $\boldsymbol{v}_i(t+1)$ is the velocity of each particle in the following iteration. It follows that:

$$\boldsymbol{u}_i(t+1) = \boldsymbol{u}_i(t) + \boldsymbol{v}_i(t+1) \tag{2.35}$$

$$s = \frac{2}{c_1 + c_2 - \sqrt{(c_1 + c_2)^2 - 4(c_1 + c_2)}} \tag{2.36}$$

where c_1 and c_2 represent the learning rate, such that $c_1 + c_2 > 4$ [30]. The symbols r_1 and r_2 represent random numbers in the interval [0, 1]. s is defined as in Eq.(2.71) and t is the ordinal number of iterations, which should be smaller than the maximal number. For the parameters of the Eq.(2.34), we have many tests, and we selected the parameters who has the best accuracy. The test is evaluated by the many simulations of scan matching whose ground truth is manually decided. The tested parameters and the results are shown in Tab.2.2.

Table. 2.2: Tested parameters of PSO.

Max Iterations	s	c_1	c_2	Error in test
200	0.56	2.8	1.3	0.02m
200	1.0	0.75	0.75	0.5m
200	0.56	2.1	2.0	0.2m
200	1.0	1.0	1.0	NG
200	0.72	2.5	2.5	1.5m
200	1.0	2.8	1.3	NG
200	0.56	1.3	2.8	NG

- Step 8

If $N(\boldsymbol{g}_{best})$ is large enough or if the number of iterations reaches the maximum set number, the evaluation process is stopped and $\boldsymbol{g}_{best}$ is considered the final solution. In case the PSO process does not satisfy either of those conditions, the algorithm reverts to Step 3.

- Step 9

  After the PSO process was finalized, we updated the current location using $\boldsymbol{g}_{best}$ and built the global map as follows:

$$\boldsymbol{L}_{cur} = (\boldsymbol{q}_{cur}, \boldsymbol{t}_{cur}) := \boldsymbol{L}_{cur} + \boldsymbol{g}_{best} \tag{2.37}$$

$$M_{ap}(\boldsymbol{R}_{cur}\boldsymbol{f}_j + \boldsymbol{t}_{cur}) = 1, \quad \text{for } 1 \leq j \leq q \tag{2.38}$$

  where $\boldsymbol{R}_{cur}$ is calculated using $\boldsymbol{q}_{cur}$, which is the updated current orientation of the robot and quaternion. M_{ap} in Eq.(2.38) is the global occupancy grid map, initiated by 0 at the start of the algorithm and for all the features. We build the map by setting the grid into 1.

2.2 6D SLAM with Object Recognition

2.2.1 Process of the algorithm

A flowchart outlining the proposed SLAM system is shown Fig.2.12, and we explain it in detail in this section. For the correction of scan distortion caused by motion, we employ a two-step method that is very similar to existing methods [6][8]. Here, we match the current scan data with the previous data to calculate the velocity using the ICP and compensate scan distortion using the calculated velocity.

In the proposed SLAM system (Fig.2.12), there are three scan registration processes, and each process updates the estimated current location for the next process. Two of these processes use the PSO for scan registration. However, these two PSO processes differ. The main differences are related to their evaluation functions and the grid resolution of their

global occupancy map. In our experiment, the first PSO process employed a larger grid resolution for the global map to match the scan with the global map roughly but quickly. The result of this rough match is used by the next process, i,e., the ICP, as the initial position. With this initial position, the ICP can optimize its result quickly and accurately. However, considering incorrect correspondences, the second PSO process is used to refine the ICP result. Here, we apply a smaller grid resolution to the second PSO, and the grid resolution of the local map for object recognition should be equal to the global map of the second PSO process. Compared to the first PSO process, the evaluation function in the second PSO process optimizes object matching rather than only maximizing the number of matched points.

After the three scan registration processes are complete, we update the global maps for the two PSO processing. As mentioned previously, both global maps are 3D occupancy grid maps, where one global map is constructed with larger grid resolution for a roughly initial matching, and the other is constructed with smaller grid resolution to facilitate precise matching.

In addition, to ensure the robustness of the proposed SLAM system, the evaluation function of the second PSO process also considers the scores when the object dose not match or there is no preset object in the scan. Here, the second PSO process still works the same as the first PSO process but with a smaller grid resolution. Therefore, even if there is no object or incorrect object recognition, the proposed SLAM system can function normally.

2.2.2 Sliding Window for 3D Point Cloud

The input of a CNN should have a uniformed format. For object recognition in computer vision, the sliding window approach is a frequently deployed paradigm. The sliding window and CNN have already been used in object recognition for 3D point clouds [36] [37]. The proposed object recognition method uses the sliding-window concept; however, the setting of our sliding window has more elements. In addition, in the proposed method, the usage of the sliding window to build the input to the CNN also differs from the previous studies.

First, with the proposed object recognition method 3D point clouds, the only required input is a 3D point cloud as Eq.(2.39) and the element inside is a 3D point as Eq.(2.40).

$$\boldsymbol{P}_{input} = (\boldsymbol{p}_1, \boldsymbol{p}_2, ..., \boldsymbol{p}_n) \tag{2.39}$$

$$\boldsymbol{p}_j = (x_j, y_j, z_j)^T \tag{2.40}$$

where $\boldsymbol{P}_{input}$ is the input 3D point, cloud and n is the number of the input points. x_j,y_j and z_j are the coordinates of point $\boldsymbol{p}_j$ inside the input point cloud.

To extract a sliding window, we must build local 3D occupancy grid maps for the input point cloud (Algorithm 1).

Algorithm 1 Algorithm for building local grid map

Input: 3D point cloud $\boldsymbol{P}_{input}$

Output: Local occupancy grid map L_{map} and the set of occupied grid $\boldsymbol{G}$

Initialisation: All values of L_{map} are 0 and $\boldsymbol{G}$ is an empty set.

1: **for** $j = 1$ to n **do**
2: **if** $(L_{map}(R_o(\boldsymbol{p}_j/r_{gl})) = 0)$ **then**
3: $L_{map}(R_o(\boldsymbol{p}_j/r_{gl})) = 1$
4: $\boldsymbol{G} = \boldsymbol{G} \cup \{R_o(\boldsymbol{p}_j/r_{gl})\}$
5: **end if**
6: **end for**
7: **return** L_{map} and $\boldsymbol{G}$

In Algorithm 1, the input is a 3D point cloud Eq.(2.39), and the output is occupancy grid map L_{map} and a set of occupied grids $\boldsymbol{G}$. Initially, the value of all grids in L_{map} are set to 0, and set $\boldsymbol{G}$ is empty. In addition, a grid in L_{map} with a value of 1 means that grid is occupied, an a value of 0 means that the grid is unoccupied. R_o (lines 2–4) is a function to take round numbers for all dimensions of vectors, and r_{gl} is the grid resolution of the local grid map L_{map}. We transform each point into the coordinates of a grid by dividing the map resolution r_{gl} (line 2) and taking the round number. In lines 2 to 4 in Algorithm 1, if a point visits an unoccupied grid (line 2), the grid is set to occupied (line 3), and the

coordinate of the grid is added to set $\boldsymbol{G}$ (line 4). However, if a point visits an occupied gird, the point will be represented by this occupied grid. After checking the entire input point cloud, the algorithm is terminated, and set $\boldsymbol{G}$ can represent all occupied grids for all input points.

After building the local grid map for the input point cloud, we obtain a set of occupied grids Eq.(2.41). We took the rounded number as $R_o(\boldsymbol{p}_j/r_{gl})$ for all points; thus, some points, that are sufficient close to each other, are represented by the same grid and its coordinates. Each grid in the local grid map can represent all input points which are inside this grid. In addition, to calculate using the girds rather than using the input points directly can avoid repeated calculations for the same input value.

$$\boldsymbol{G} = \{\boldsymbol{g}_1, \boldsymbol{g}_2, ..., \boldsymbol{g}_m\} \tag{2.41}$$

$$\boldsymbol{g}_i = (x_i^g, y_i^g, z_i^g)^T \tag{2.42}$$

where $\boldsymbol{G}$ is the set of occupied grids. Each occupied grid $\boldsymbol{g}_i$ in $\boldsymbol{G}$ is a three-dimensional vector representing an occupied grid's coordinates. Here x_i^g, y_i^g and z_i^g are the coordinates of an occupied grid $\boldsymbol{g}_i$. Set $\boldsymbol{G}$ is formed by all occupied grids; thus, all elements in set $\boldsymbol{G}$ should satisfy $L_{map}(\boldsymbol{g}_i) = 1$. The number of occupied grids is m, and one grid can represent one or more points inside; thus, the m should be less than n.

In addition, for each occupied gird, we set it as the center of the sliding window (Fig.2.9) and extract the surrounding information as follows.

$$\begin{aligned} \boldsymbol{S}_i(l, w, h) = L_{map}(\boldsymbol{g}_i + (l, w, h)^T), \\ \text{for } 1 \leq i \leq m, -\tfrac{d_l}{2} \leq l \leq \tfrac{d_l}{2}, \\ -\tfrac{d_w}{2} \leq w \leq \tfrac{d_w}{2}, -\tfrac{d_h}{2} \leq h \leq \tfrac{d_h}{2} \end{aligned} \tag{2.43}$$

where $\boldsymbol{S}_i$ is the sliding-window taken from L_{map}, the sliding window is a three-dimensional matrix, and d_l, d_w, and d_h are the numbers of grids along the length, width and height of the sliding window, respectively, and all of them are round numbers. l, w, and h are round numbers for an iterator who visits and checks all grids of the local map in the range of the sliding window.

The local occupancy grid map is built using only the current scan data. The sliding window helps reduce computational cost because it can avoid searching the closest points. The size of the sliding window depends on the grid resolution of the local occupancy grid map and the maximum size object to recognize. The entire maximum size object should be overlapped by the sliding window. In this study, the target objects do not have significant size differences; thus, a single size sliding window can cover all types of objects.

2.2.3 Neural Network Inputs

Here, we describe how we build the input for the CNN from an extracted sliding window. Our approach to form the input is inspired by three-view-image. A three-view-image is sufficient to represent a simple 3D object. In addition, using a three-view-image can reduce computational costs, and the image type input is more suitable for the CNN. Therefore, we build the input image for the CNN by transforming the 3D points in the sliding window to a three-view-image. The input image is a grayscale image with three channels. Examples of the three-view-image are shown in Fig.2.10.

For the three-view-image, we must zip the sliding window into top-view, side-view and front-view images.

To zip the sliding window from the top-view point, we create an image with $d_h \times d_l$ pixels. Here, each pixel, at location (l, w) corresponds to the grids in the same location in the sliding windows. However, if there are several occupied girds at the same location (l, w), we only use the highest grid to calculate the pixel value.

$$\begin{aligned} &I_i^{'}(l,w) = \frac{\max(h)}{d_h} + \frac{1}{2}, \\ &\text{for } \boldsymbol{S}_i(l,w,h) = 1 \text{ and } -\tfrac{d_h}{2} \leq h \leq \tfrac{d_h}{2} \end{aligned} \tag{2.44}$$

where $I_i^{'}$ is the top-view image, and (l, w) is the coordinate of a pixel of this image. However, if all the heights h in coordinate (l, w) of the sliding windows are unoccupied, the pixel value $I_i^{'}(l, w)$ is set to 0.

In addition to the top-view image calculation, all pixels of the side-view and front-view images are calculated in a similar manner. Note that the calculation of each pixel of

the side-view image only uses the grid in the nearest side, and the front-view image only uses the most frontal occupied grid. The equations used to calculate the side-view and front-view images are given in Eq.(2.45) and Eq.(2.46), respectively.

$$I_i^{''}(w,h) = \frac{\max(l)}{d_l} + \frac{1}{2}, \quad \text{for } \boldsymbol{S}_i(l,w,h) = 1 \text{ and } -\frac{d_l}{2} \leq l \leq \frac{d_l}{2} \tag{2.45}$$

$$I_i^{'''}(l,h) = \frac{\max(w)}{d_w} + \frac{1}{2}, \quad \text{for } \boldsymbol{S}_i(l,w,h) = 1 \text{ and } -\frac{d_w}{2} \leq w \leq \frac{d_w}{2} \tag{2.46}$$

where $I_i^{''}$ and $I_i^{'''}$ denote the side-view and front-view images. However, similar to the top-view image, if an entire column or entire row of grids of $\boldsymbol{S}_i$ are unoccupied, the pixel value of $I_i^{''}$ or $I_i^{'''}$ is set as 0.

After we obtain three images $I_i^{''}$, $I_i^{''}$ and $I_i^{'''}$, we resize them to $R_s \times C_s$ using OpenCV [38] with bilinear interpolation [39]. We then combine the three resized grayscale images into one image with three channels. The three channels concept is more like the three RGB channels; therefore, the final input to the proposed CNN is an image with three channels with $R_s \times C_s$ pixels for each channel. In our experiment, $R_s \times C_s$ was set to 51×51 pixels for each input image.

2.2.4 Output of Convolutional Neural Network and Object Recognition

Here, we describe how the output from the trained neural network is used to mark the labels of the input point cloud. As shown in Fig.2.11, the output of the proposed CNN is a d-dimensional vector, where d is the number of objects. It is easy to recognize d objects with a CNN, with softmax [40] to generate the probability of each object.

We can extract sliding window $\boldsymbol{S}_i$ for each occupied grid g_i in the local occupancy grid map. In addition, for each extracted sliding window $\boldsymbol{S}_i$, we can generate an input image for the trained CNN. Therefore, we can obtain output $\boldsymbol{O}_i^{'}$ for each sliding window $\boldsymbol{S}_i$ and each occupied gird g_i as follows.

$$\boldsymbol{O}_i^{'} = (o_1, o_2, ..., o_d) \tag{2.47}$$

where $\boldsymbol{O}_i^{'}$ is the output of the proposed CNN. $\boldsymbol{O}_i^{'}$ is a d-dimensional vector, where the value of d depends on the number of target objects to recognize. There are d elements inside $\boldsymbol{O}_i^{'}$, where each element o indicates the probability to be one of the target objects.

After the output is obtained from the proposed CNN, we do not directly determine if occupied grid g_i belongs to one of the target objects with the highest probability in $\boldsymbol{O}_i^{'}$. Considering the robustness of object recognition, we use all outputs in a range to vote on object should be selected as g_i. Here, one occupied grid can be overlapped by one or several sliding windows, created by itself and the other occupied grids, and each sliding window has an output; thus, we summarize all overlapping outputs to vote on which target object should the grid g_i belong to. To summarize all overlapped outputs, we must first reassign the outputs to a new sliding windows using Eq.(2.48).

$$\begin{aligned} \boldsymbol{O}_i^S(l,w,h) = \begin{cases} \boldsymbol{O}_i^{'} & \text{if } \boldsymbol{S}_i(l,w,h) = 1 \\ 0 & \text{otherwise} \end{cases} \\ \text{for } 1 \leq i \leq m,\ -\tfrac{d_l}{2} \leq l \leq \tfrac{d_l}{2}, \\ -\tfrac{d_w}{2} \leq w \leq \tfrac{d_w}{2},\ -\tfrac{d_h}{2} \leq h \leq \tfrac{d_h}{2} \end{aligned} \tag{2.48}$$

Here, $\boldsymbol{O}_i^S$ is another sliding window, i.e., a three-dimensional matrix. However, each element in $\boldsymbol{O}_i^S$ is a d-dimensional vector (the same as the $\boldsymbol{O}_i^{'}$). We copy the value of the output from the CNN to each occupied grid in this sliding-window.

Then all the outputs of the proposed CNN are reassigned to their new sliding windows. For all of these sliding windows, whose elements are d-dimensional vectors, we summarize them into a new occupancy voxel map. The resolution of the new occupancy voxel map L_{sum} is r_{gl}, which is the same as the local occupancy grid map L_{map}. Each voxel in L_{sum} is a d-dimensional vector, and the values of all dimensions of all vectors are initially set to 0. All new sliding windows $\boldsymbol{O}_i^S$ are summarized in the local occupancy voxel map as follows.

$$\boldsymbol{L}_{sum}(\boldsymbol{g}_i + (l,w,h)^T) = \sum_{i=1}^{n} \sum_{l=-\frac{d_l}{2}}^{\frac{d_l}{2}} \sum_{w=-\frac{d_w}{2}}^{\frac{d_w}{2}} \sum_{h=-\frac{d_h}{2}}^{\frac{d_h}{2}} \boldsymbol{O}_i^S(l,w,h) \tag{2.49}$$

where $\boldsymbol{L}_{sum}$ is a three-dimensional matrix whose elements are d-dimensional vectors. Each value of the local map $\boldsymbol{L}_{sum}$ in coordinate $\boldsymbol{g}_i + (l,w,h)^T$ is calculated by summarizing all sliding windows $\boldsymbol{O}_i^S$ that overlap there. In addition, $\boldsymbol{g}_i + (l,w,h)^T$ is the coordinate of the other occupied girds inside the range of the sliding window of occupied grid $\boldsymbol{g}_i$. Therefore, it is possible for one occupied grid to be overlapped by one or several close sliding windows. Therefore, Eq.(2.49) means that one output from the proposed CNN in coordinate $\boldsymbol{g}_i$ will add its value to all occupied grids in the range $[-\frac{d_l}{2}, \frac{d_l}{2}]$, $[-\frac{d_w}{2}, \frac{d_w}{2}]$ and $[-\frac{d_h}{2}, \frac{d_h}{2}]$.

After this summarization, each occupied grid corresponds to a d-dimensional vector in the final local voxel map Eq.(2.50). The vector summarizes all surrounding outputs from the proposed CNN to increase object recognition robustness. In addition, each value, s, in vector $\boldsymbol{L}_{sum}(\boldsymbol{g}_i)$ indicates a probability to be a preset object. Therefore, if we find which dimension has the maximum value of the vector $\boldsymbol{L}_{sum}(\boldsymbol{g}_i)$, we can determine which object or label should set as the occupied grid as Eq.(2.52). After checking all input points for their maximum probability inside $\boldsymbol{L}_{sum}$, we mark all labels to the input point cloud according to Eq.(2.51).

$$\boldsymbol{L}_{sum}(\boldsymbol{g}_i) = (s_1, s_2, ..., s_d) \tag{2.50}$$

$$\boldsymbol{L}_l = (l_1, l_2, ..., l_n) \tag{2.51}$$

$$l_j = F_{max}(\boldsymbol{L}_{sum}(R_o(\boldsymbol{p}_j / r_{gl}))), \text{for } 1 \leq j \leq n \tag{2.52}$$

where $\boldsymbol{L}_l$ represents the labels of $\boldsymbol{P}_{input}$. Here, l_j is a round number in the range $[1, d]$ that indicates what object the point $\boldsymbol{p}_j$ is. The function F_{max} finds which dimension has the maximum value in vector $\boldsymbol{L}_{sum}(\boldsymbol{g}_i)$ or $\boldsymbol{L}_{sum}(R_o(\boldsymbol{p}_j / r_{gl}))$. In addition, $\boldsymbol{g}_i$ represents one or several input points because some input points can have the same result for the calculation of $R_o(\boldsymbol{p}_j / r_{gl})$. Therefore, $\boldsymbol{L}_{sum}(R_o(\boldsymbol{p}_j / r_{gl}))$ in Eq.(2.52) can be represented by $\boldsymbol{L}_{sum}(\boldsymbol{g}_i)$ in Eq.(2.50).

2.2.5 6D SLAM using the Recognized Objects

Here, we describe the scan registration step using PSO. As shown in Fig.2.12, two different PSO processes are employed in the proposed method, i,e., ordinary PSO to match the scan data without using recognized points and a modified PSO process that uses recognized points for scan registration. The difference between these two PSO processes in scan registration is the evaluation function. The evaluation of the ordinary process finds a transformation to maximize the number of matched points between the source scan and target point clouds. However, the PSO process that uses the recognized points simultaneously considers the number of matched points and the matches of recognized objects. For ICP scan registration, the algorithm is equal to the most common ICP [16]. In addition, ICP matches the data of the current scan with the data of the neighboring scans, whereas PSO matches the data of the current scan with the global map by stitching together all scan data. Here, the global map for the ordinary PSO process is an occupancy grid map stitched using the scan data and the global map is a 3D matrix whose elements are 0 or 1, where 0 and 1 mean unoccupied and occupied, respectively. However, the global map for the second PSO process, that uses the recognized points, is a 3D matrix whose elements are round numbers from 0 to d, where 0 and d denote unoccupied and preset object's label, respectively.

- Step 1

  For 6DoF movement, PSO must estimate and optimize six values for each DoF; therefore, each particle of the PSO is a six-dimensional vector, and all particles are initiated random.

$$\boldsymbol{A}(0) = \{\boldsymbol{a}_1(0), \boldsymbol{a}_2(0), ..., \boldsymbol{a}_b(0)\} \tag{2.53}$$

$$\begin{aligned} \boldsymbol{a}_k(0) = (&\Delta\theta_k(0), \Delta\varphi_k(0), \Delta\psi_k(0), \\ &\Delta t_k^x(0), \Delta t_k^y(0), \Delta t_k^z(0)) \end{aligned} \tag{2.54}$$

  where $\boldsymbol{A}(0)$ is the set of all particles in the 0th generation, and b is the number of particles. Considering our experimental environment and the real-time issue, where the SLAM system's processing frequency must be higher than sensor's frequency, we

set b to 200. $\Delta\theta_k$, $\Delta\varphi_k$ and $\Delta\psi_k$ denote the roll, pitch and yaw rotation values, respectively, and they are initiated randomly. $\Delta t_k^x(0)$, $\Delta t_k^y(0)$, and $\Delta t_k^z(0)$ are the translation values for the x, y and z axes and they are also initiated randomly. In our situation, the random numbers for rotation are selected randomly in the range $[-0.03, 0.03]$, and the translation value is selected randomly in the range $[-0.2, 0.2]$.

- Step 2

  For all particles, we form transformation matrix $\boldsymbol{T}_k$ using quaternions according to Eq.(2.60):

$$\Delta\boldsymbol{q}_k(t) = (\Delta\theta_k(t), \Delta\varphi_k(t), \Delta\psi_k)(t) \tag{2.55}$$

$$w_k(t) = \cos\frac{|\Delta\boldsymbol{q}_k(t)|}{2} \tag{2.56}$$

$$x_k(t) = \frac{\Delta\theta_k(t)}{|\Delta\boldsymbol{q}_k(t)|}\sin\frac{|\Delta\boldsymbol{q}_k(t)|}{2} \tag{2.57}$$

$$y_k(t) = \frac{\Delta\varphi_k(t)}{|\Delta\boldsymbol{q}_k(t)|}\sin\frac{|\Delta\boldsymbol{q}_k(t)|}{2} \tag{2.58}$$

$$z_k(t) = \frac{\Delta\psi_k(t)}{|\Delta\boldsymbol{q}_k(t)|}\sin\frac{|\Delta\boldsymbol{q}_k(t)|}{2} \tag{2.59}$$

  where t denotes a solution belonging to the t-th iteration of the PSO process.

- Step 3

  The input source point cloud in Eq.(2.39), i.e., $\boldsymbol{P}_{input}$, is transform using the solutions for all particles as follows.

$$\begin{bmatrix} \boldsymbol{P}_k^{'} \\ \mathbf{1} \end{bmatrix} = \boldsymbol{T}_k(t) \begin{bmatrix} \boldsymbol{P}_{input} \\ \mathbf{1} \end{bmatrix} \tag{2.61}$$

$$\boldsymbol{P}_k^{'} = (\boldsymbol{f}_1^k, \boldsymbol{f}_2^k, ..., \boldsymbol{f}_n^k) \tag{2.62}$$

  The transformation matrix generated using the solution for the k-th particle in the t-th iteration is represented by $\boldsymbol{T}_k(t)$. $\boldsymbol{P}_k^{'}$ is the point cloud after being transformed using the $\boldsymbol{T}_k(t)$.

- Step 4

  To evaluate the solutions generated by each particle of the original PSO process without using the object recognition, we calculate the number of points that can fit the current global map G_o using Eq.(2.63). However, to evaluate solutions for the PSO process that uses object recognition, we calculate the score of points whose label fit the current labeled global map G_l using Eq.(2.64). Here, global maps G_o and G_l are three-dimensional occupancy maps formed by stitching together all of previous scans using Eq.(2.75) and Eq.(2.76). However, each element of G_o is filled using 0 or 1, and G_l is filled using the recognized object's label. Here, r_{go} and r_{gl} are the grid resolution of the occupancy global maps G_o and G_l, respectively.

$$E_o(\boldsymbol{f}_j^k) = \begin{cases} 1 & \text{if } G_o(R_o(\boldsymbol{f}_j^k / r_{go})) = 1 \\ 0 & \text{otherwise} \end{cases} \tag{2.63}$$

$$E_l(\boldsymbol{f}_j^k) = \begin{cases} s_1 & \text{if } G_l(R_o(\boldsymbol{f}_j^k / r_{gl})) = l_j \neq 0 \\ s_2 & \text{if } G_l(R_o(\boldsymbol{f}_j^k / r_{gl})) \neq l_j \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.64}$$

  where E_o is the score of each transformed point. Here, if the point matches the occupied grid of G_o, the score is set to 1. However, the calculation of score in Eq.(2.63) does not consider incorrect correspondences. For the PSO process that uses the recognized points, E_l is also the score of each transformed point obtained using the k-th particle. However, if the point only matches the occupied grid of G_l without matching the label, the score is set to s_2. In addition, if the point simultaneously matches the label, the score is set to s_1. In our case, s_1 is 1.0, and s_2 is 0.1, which means that we consider the correspondence importance much more than the number of matched points. However, when the object is recognized incorrectly or there is no preset object in the scan, we provide a low score to ensure that the system can function normally in such cases.

- Step 5

After evaluating all transformed points using the k-th particle (i.e., the solution), we summarize their score using Eq.(2.65) or Eq.(2.66). Here, $S_{co}(\boldsymbol{a}_k(t))$ denotes the total score of the k-th particle. However, for the PSO process that does not consider recognized objects, we use Eq.(2.65) to calculate the score, and Eq.(2.66) is used calculate the score for PSO that does consider the object recognition.

$$S_{co}(\boldsymbol{a}_k(t)) = \sum_{j=1}^{n} E_o(\boldsymbol{f}_j^k) \tag{2.65}$$

$$S_{co}(\boldsymbol{a}_k(t)) = \sum_{j=1}^{n} E_l(\boldsymbol{f}_j^k) \tag{2.66}$$

- Step 6

  From the first iteration to the t-th, the best k-th particle that can obtain the highest score $S_{co}(\boldsymbol{a}_k(t))$, is denoted by $\boldsymbol{p}_{best_k}$, and the best solution for all particles and iterations is denoted by $\boldsymbol{g}_{best}$

$$\boldsymbol{p}_{best_k} = \underset{0 \le t' \le t}{\operatorname{argmax}}(S_{co}(\boldsymbol{a}_k(t'))) \tag{2.67}$$

$$\boldsymbol{g}_{best} = \underset{1 \le k \le b}{\operatorname{argmax}}(S_{co}(\boldsymbol{p}_{best_k})) \tag{2.68}$$

  where $\boldsymbol{p}_{best_k}$ and $\boldsymbol{g}_{best}$ are six-dimensional vectors, and each dimension's meaning is the same as in Eq.(2.54).

- Step 7

  In the PSO process, each particle can upgrade itself by attempting by following the local best and global best solutions $\boldsymbol{p}_{best_i}$ and $\boldsymbol{g}_{best}$ in a balanced manner:

$$\begin{aligned} \boldsymbol{v}_k(t+1) = s(\boldsymbol{v}_k(t) + c_1 r_1(\boldsymbol{p}_{best_k} - \boldsymbol{a}_k(t)) \\ + c_2 r_2(\boldsymbol{g}_{best} - \boldsymbol{a}_k(t))) \end{aligned} \tag{2.69}$$

  where $\boldsymbol{v}_i(t+1)$ is the upgraded velocity of each particle for the next iteration. It follows that:

$$\boldsymbol{a}_k(t+1) = \boldsymbol{a}_k(t) + \boldsymbol{v}_k(t+1) \tag{2.70}$$

$$s = \frac{2}{c_1 + c_2 - \sqrt{(c_1 + c_2)^2 - 4(c_1 + c_2)}} \tag{2.71}$$

where c_1 and c_2 are learning rates, such that $c_1 + c_2 > 4$ [30]. Here, r_1 and r_2 are random numbers in the interval [0, 1], and s is a constant defined in Eq.(2.71).

- Step 8

  If the score $S_{co}(\boldsymbol{g}_{best})$ is sufficiently large or the number of iterations reaches the maximum iteration number, we stop the PSO processing and consider the $\boldsymbol{g}_{best}$ as the final solution. In our case, the maximum iteration number is set to 200. If the PSO process does not satisfy either condition, the algorithm reverts to Step 2.

- Step 9

  When the PSO process is complete, we update the current location of the robot using the best solution of the PSO process $\boldsymbol{g}_{best}$ as follows:

$$\boldsymbol{T}_{cur} = \boldsymbol{T}_{gbest}\boldsymbol{T}_{cur} \tag{2.72}$$

  where $\boldsymbol{T}_{cur}$ is the current location and gesture of the robot, which is updated using $\boldsymbol{T}_{gbest}$. $\boldsymbol{T}_{gbest}$ is the transformation matrix calculated using $\boldsymbol{g}_{best}$ and quaternions as in Eq.(2.60).

- Step 10

  If the final location is updated by the final PSO process (Fig.2.12), both global maps G_o and G_l of both PSO processes are updated at the same time with the last updated $\boldsymbol{T}_{cur}$. However, prior to the final PSO process, the current location is updated using the scan registration result of the ICP (Fig.2.12).

$$\begin{bmatrix} \boldsymbol{P}_{gl} \\ \mathbf{1} \end{bmatrix} = \boldsymbol{T}_{cur} \begin{bmatrix} \boldsymbol{P}_{input} \\ \mathbf{1} \end{bmatrix} \tag{2.73}$$

$$\boldsymbol{P}_{gl} = (\boldsymbol{f}_1^{'}, \boldsymbol{f}_2^{'}, ..., \boldsymbol{f}_n^{'}) \tag{2.74}$$

$$G_o(R_o(\boldsymbol{f}_j^{'}/r_{go})) = 1, \quad \text{for } 1 \leq j \leq n \tag{2.75}$$

$$G_l(R_o(\boldsymbol{f}_j^{'}/r_{gl})) = l_j, \quad \text{for } 1 \leq j \leq n \tag{2.76}$$

In Eq.(2.75), G_o is the global occupancy grid map without using the recognized label, which is filled with 0 or 1. In Eq.(2.76), G_l is the global occupancy grid map that records the recognized label, which is filled by round numbers from 0 to d. Note that this is the same as $\boldsymbol{L}_l$. In Eq.(2.75), each point $\boldsymbol{f}_j^{'}$, which is transformed from the input point using the final updated $\boldsymbol{T}_{cur}$, fills and updates the occupancy map G_o with round number 1. However, in Eq.(2.76), each point updates the labeled global map G_l with the labels inside $\boldsymbol{L}_l$. In addition, the grid resolution of global map G_l is r_{gl} and it is the same as the grid resolution of L_{map}.

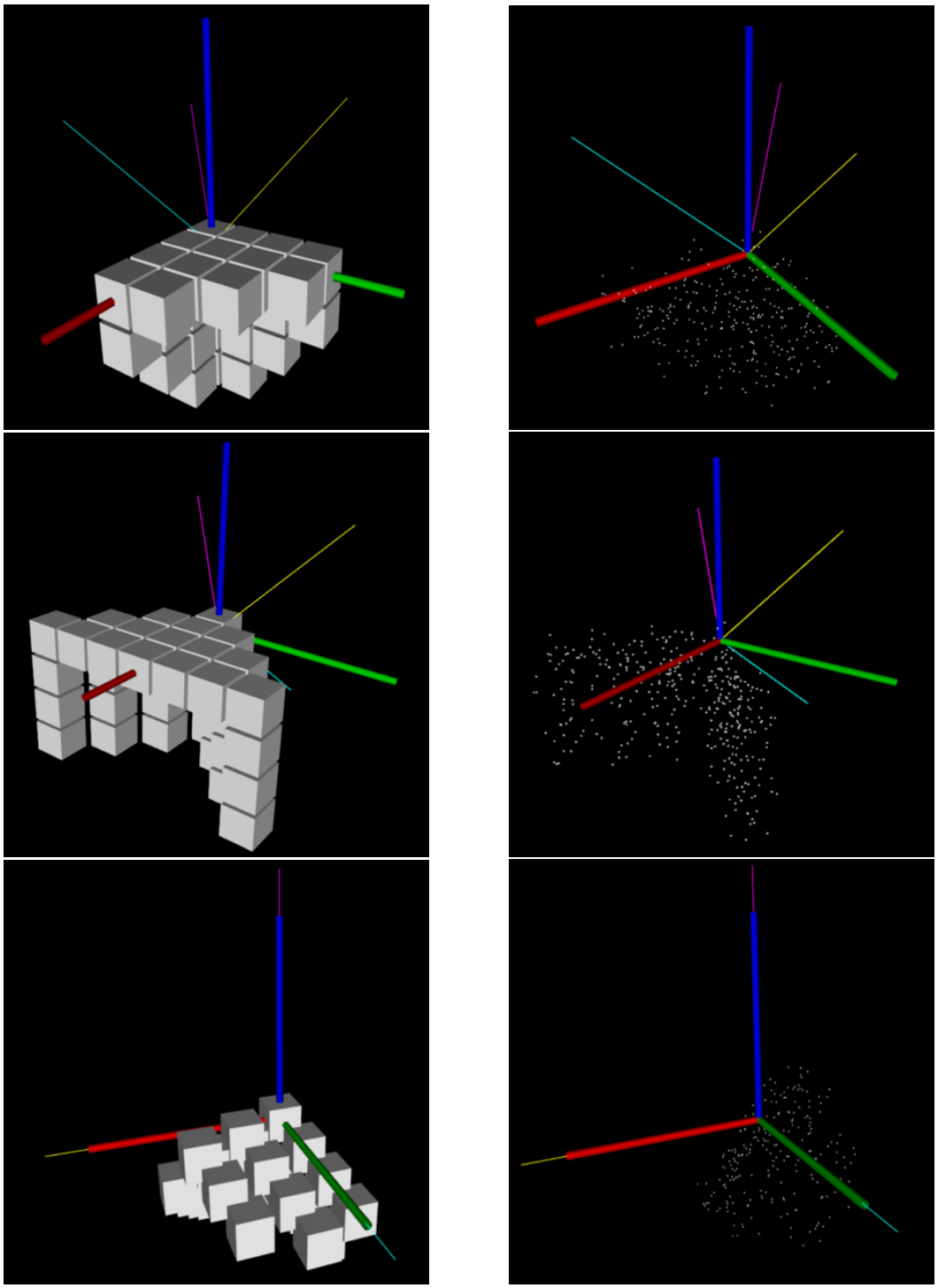

Fig. 2.3: Example of uniform coordination; (a) a corner sub-cube; (b) the point clouds inside the sub-cube in (a); (c) a sub-cube of the same corner as in (a), but from a different viewpoint; (d) the point clouds inside the sub-cube in (c); (e) the sub-cube in (a) after the transformation; (f) the sub-cube in (c) after the transformation.

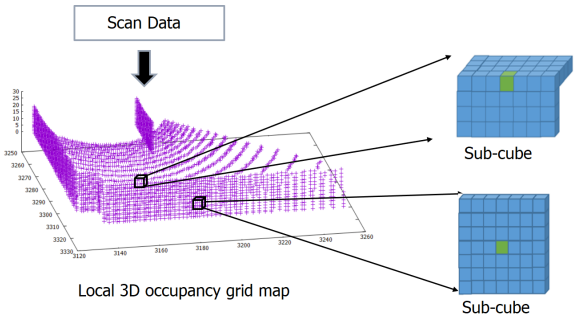

Fig. 2.4: Examples of sub-cubes: Sub-cubes from a local grid map of size 7^3.

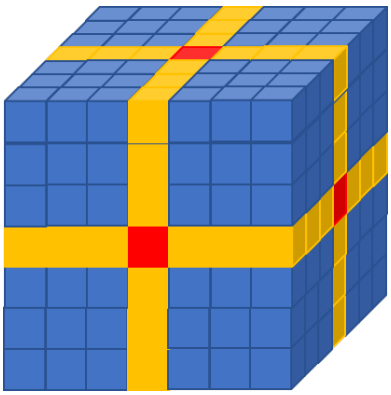

Fig. 2.5: Examples of sub-cubes: Sub-cube is separated into 8 blue parts, 12 yellow parts, 6 red parts for the 26-dimensional inputs (best viewed in color).

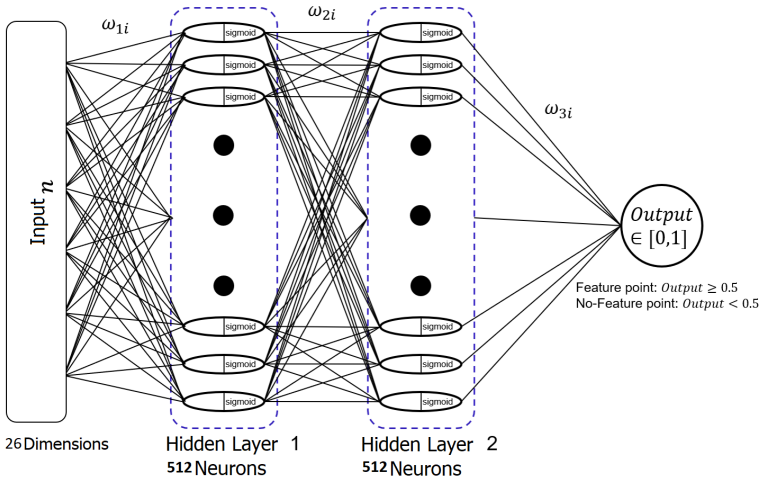

Fig. 2.6: Backpropagation neural network for feature extraction.

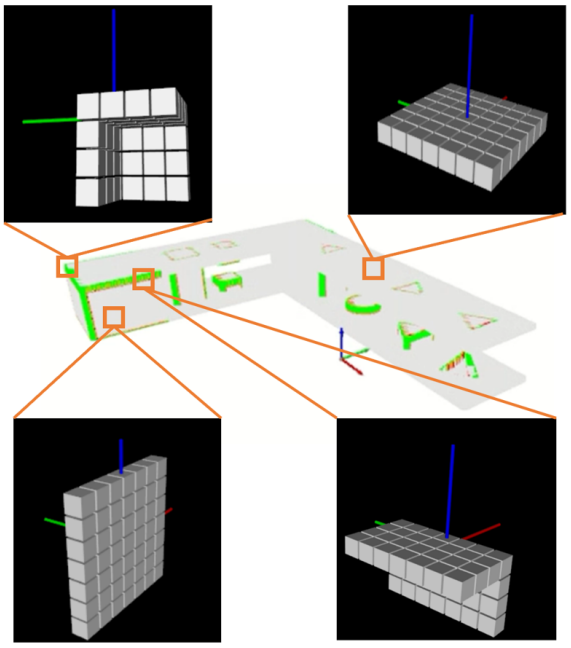

Fig. 2.7: Examples of training samples for the backpropagation neural network. Green points indicate examples of positive samples (edges and corners); gray points indicate examples of negative samples (wall and roof).

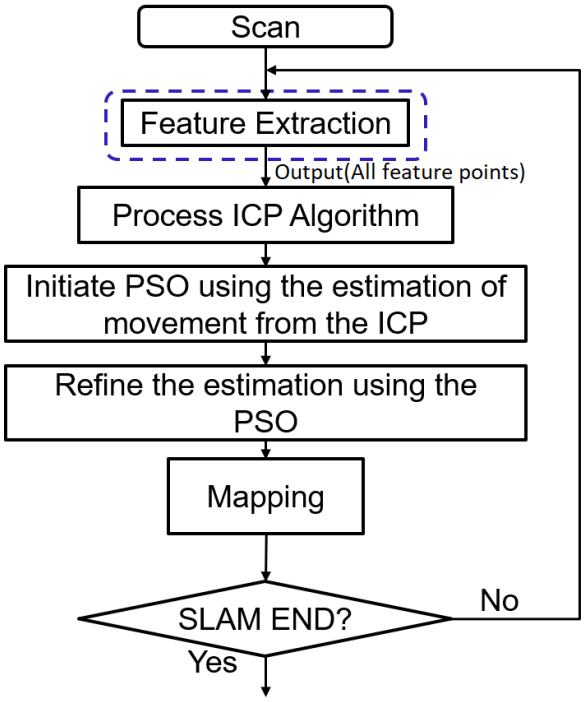

Fig. 2.8: Flowchart of the 6D SLAM.

$$R_i(t) = \begin{bmatrix} 2w_i^2(t) + 2x_i^2(t) - 1 & 2w_i(t)z_i(t) + 2x_i(t)y_i(t) & 2x_i(t)z_i(t) - 2w_i(t)y_i(t) \\ 2x_i(t)y_i(t) - 2w_i(t)z_i(t) & 2w_i^2(t) + 2y_i^2(t) - 1 & 2w_i(t)x_i(t) + 2y_i(t)z_i(t) \\ 2w_i(t)y_i(t) + 2x_i(t)z_i(t) & 2y_i(t)z_i(t) - 2w_i(t)x_i(t) & 2w_i^2(t) + 2z_i^2(t) - 1 \end{bmatrix} \quad (2.31)$$

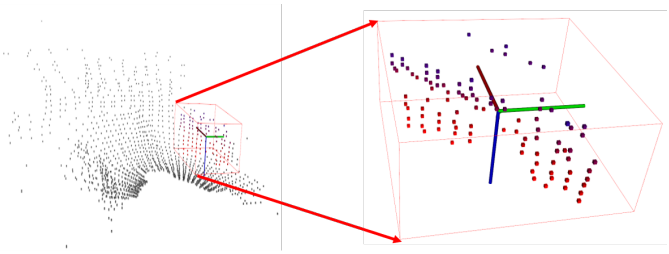

Fig. 2.9: Grid occupancy map and sliding window. Gray points are occupied grids. The rectangular cuboid with red edges is the sliding window. Colored points are occupied grids inside the sliding window. The three-dimensional axis is marked at the focused grid, i.e., the center of the sliding-window.

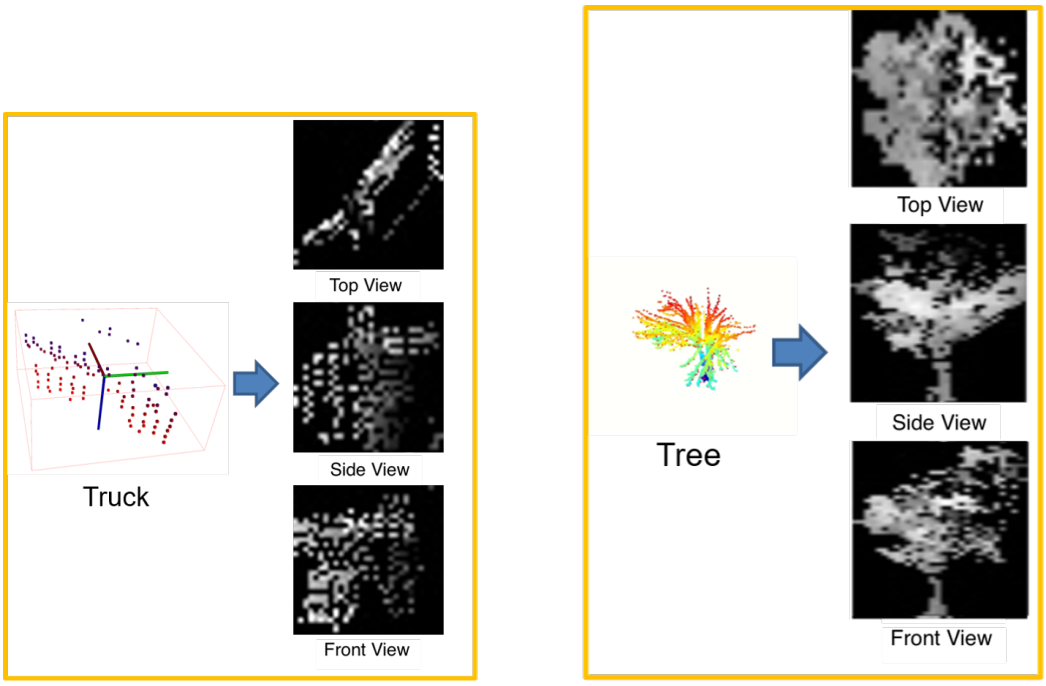

Fig. 2.10: Examples of transforming a sliding window into a three-view-image; (a) transforming a truck, (b) transforming a tree

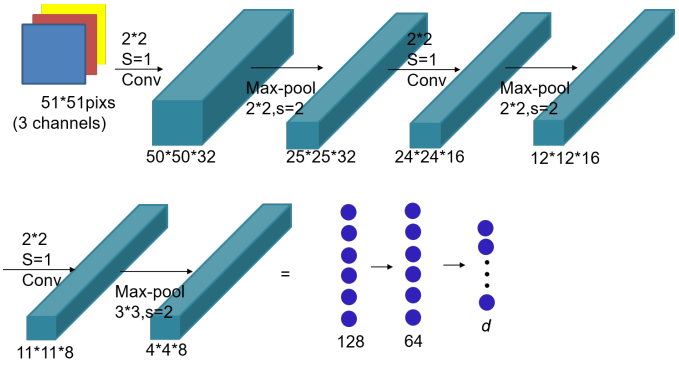

Fig. 2.11: CNN for object recognition.

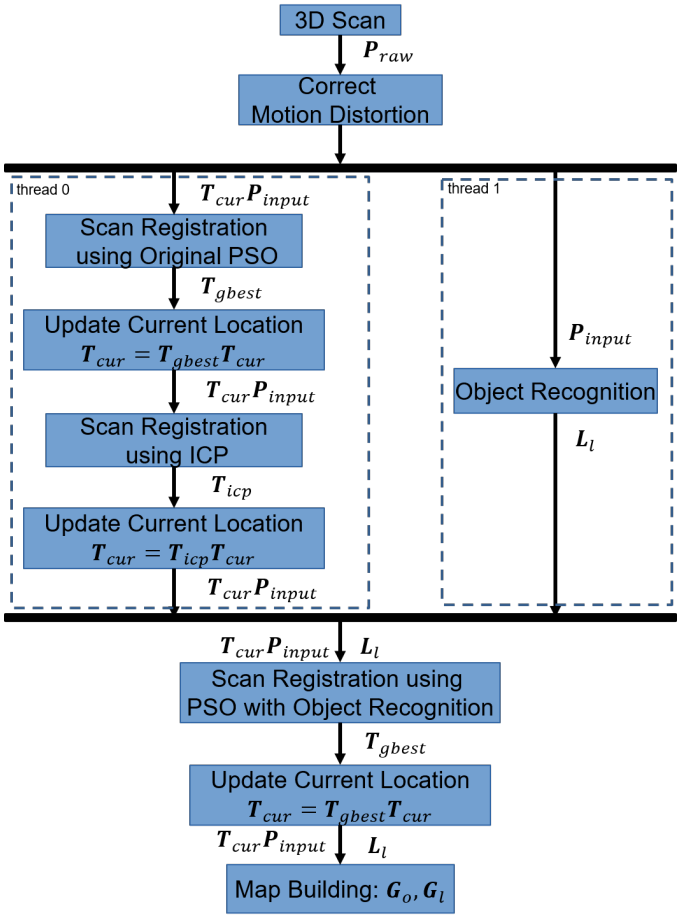

Fig. 2.12: Flowchart of 6D-SLAM.

$$\boldsymbol{T}_k(t) = \begin{bmatrix} 2w_i^2(t) + 2x_i^2(t) - 1 & 2w_i(t)z_i(t) + 2x_i(t)y_i(t) & 2x_i(t)z_i(t) - 2w_i(t)y_i(t) & \Delta t_k^x(t) \\ 2x_i(t)y_i(t) - 2w_i(t)z_i(t) & 2w_i^2(t) + 2y_i^2(t) - 1 & 2w_i(t)x_i(t) + 2y_i(t)z_i(t) & \Delta t_k^y(t) \\ 2w_i(t)y_i(t) + 2x_i(t)z_i(t) & 2y_i(t)z_i(t) - 2w_i(t)x_i(t) & 2w_i^2(t) + 2z_i^2(t) - 1 & \Delta t_k^z(t) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.60}$$

3. Experiments

3.1 Experiments of 6D SLAM using the Feature Extraction

3.1.1 Experiments of Feature Extraction

The first experiment focused on the extraction of feature points from a 3D map (point cloud). The left panel of Fig.3.1 shows the original 3D point cloud of an indoor room with many different surfaces. To simplify the visual representation, the roof of the room is not shown. The right panel of Fig.3.1 shows the roof and walls of the room, filtered using the trained BPNN, with edges and corners that are well extracted using our proposed method. We also test our method on the KITTI dataset and show some examples of the feature extraction in Fig.3.2. In the KITTI dataset, the edges, where the surfaces meet, are essentially extracted.

3.1.2 Experiment of 6D using Feature Extraction in Indoor and Outdoor Environment

In this experiment, we execute our proposed method on a two-floor map that contains both indoor and outdoor environments, as shown in Fig.3.3. For comparison, we also execute our proposed method without employing the feature points to show the importance of using the feature points. We performed 2,500 scans, each of which generated an average of 7,000 points. The sensor we used in this experiment was the YVT-X001 by HOKUYO. The pink line in the figure indicates the path followed by the robot. The red points in

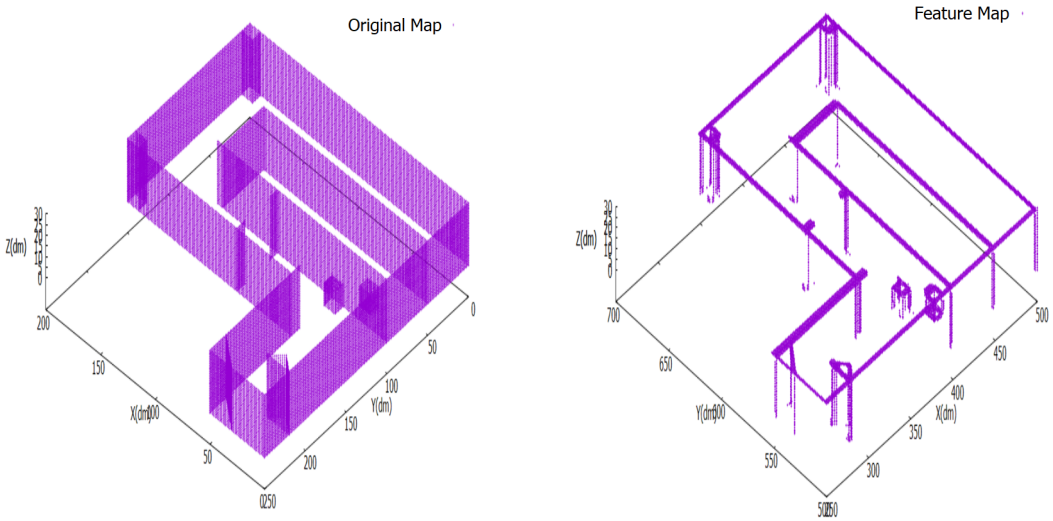

Fig. 3.1: Map for Experiment: (a) Original 3D point cloud; (b) feature-extracted point cloud.

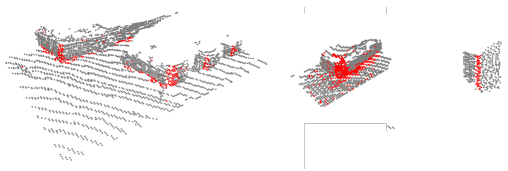

Fig. 3.2: Feature Extraction of the KITTI dataset. The red points are the extracted points that use the proposed BPNN.

Fig.3.1.2 and Fig.3.1.2 are the feature points extracted by our trained BPNN, while most of the features (the edges and corners) are shown to be accurately extracted.

For all of the 6D SLAM experiments, the computer used was a Core I9-9900K CPU 3.60 GHz with 32 GB RAM and an Nvidia GeForce RTX 2080TI with 11 GB RAM, while the computational cost of searching for the feature points from an average of 7,000 points per scan is less than 10 ms. The frequency of the sensor used in our experiment, YVT-X001, was 5 Hz (200 ms), while our method's computational cost was on average 30 ms. Our proposed BPNN makes it possible to address a real-time 6D SLAM system.

By using the feature points for the 6D SLAM, the performance of the pose estimation was also enhanced. The trajectory obtained using the feature points remained very close

Table. 3.1: Results on KITTI Odometry.

Method	Error Type	Sequences 00	01	02	03	04	05	06	07	08	09	10	Average	Average time(ms)	Sensor
Proposed 1	Rotation	0.154	**0.186**	**0.209**	**0.241**	0.248	0.254	0.144	0.264	**0.208**	0.193	**0.237**	**0.233**	60	Laser
	Translation	**0.478**	**1.012**	**0.632**	**0.477**	0.236	**0.256**	0.209	0.288	**0.633**	**0.457**	**0.600**	**0.504**		
Proposed 2	Rotation	0.260	0.275	0.346	0.246	**0.182**	0.255	**0.113**	**0.264**	0.234	0.194	0.237	0.239	98	Laser
	Translation	0.545	1.052	0.877	0.519	**0.226**	0.295	**0.186**	**0.272**	0.710	0.504	0.750	0.549		
LOAM[6]	Rotation	-	-	-	-	-	-	-	-	-	-	-	-	77	Laser
	Translation	0.8	1.4	0.9	0.7	0.6	0.7	0.6	1.1	0.8	0.8	0.8	0.8		
SuMa[27]	Rotation	0.23	0.5	0.4	0.5	0.3	**0.2**	0.3	0.6	0.4	0.2	0.3	0.3	15	Laser
	Translation	0.7	1.7	1.2	0.7	0.4	0.4	0.5	0.7	1.2	0.6	0.7	0.8		
SuMa++[41]	Rotation	0.22	0.46	0.37	0.46	0.26	**0.20**	0.21	0.19	0.35	0.23	0.28	0.29	10	Laser
	Translation	0.64	1.60	1.0	0.67	0.37	0.40	0.46	0.34	1.10	0.47	0.66	0.70		
IMLS[42]	Rotation	-	-	-	-	-	-	-	-	-	-	-	-	125	Laser
	Translation	0.50	0.82	0.53	0.68	0.33	0.32	0.33	0.33	0.80	0.55	0.53	0.55		
MC2[43]	Rotation	-	-	-	-	-	-	-	-	-	-	-	-	10	Laser
	Translation	0.51	0.79	0.54	0.65	0.44	0.27	0.31	0.34	0.84	0.46	0.52	0.52		
S4[44]	Rotation	-	-	-	-	-	-	-	-	-	-	-	-	20	Laser
	Translation	0.62	1.11	1.63	0.82	0.95	0.50	0.65	0.60	1.33	1.05	0.96	0.92		
PSF[45]	Rotation	-	-	-	-	-	-	-	-	-	-	-	-	20	Laser
	Translation	0.64	1.32	0.87	0.75	0.66	0.45	0.47	0.46	0.94	0.56	0.54	0.74		
LiTAMIN2[46]	Rotation	0.28	0.46	0.32	0.48	0.52	0.25	0.34	0.32	0.29	0.40	0.47	0.33	1	Laser
	Translation	0.70	2.10	0.98	0.96	1.05	0.45	0.59	0.44	0.95	0.69	0.80	0.85		
LO-Net[47]	Rotation	0.42	0.40	0.45	0.59	0.54	0.35	0.35	0.45	0.43	0.38	0.41	0.42	10	Laser
	Translation	0.69	1.42	1.01	0.73	0.56	0.62	0.55	0.56	1.08	0.77	0.92	0.83		
FALO[48]	Rotation	0.51	1.35	0.28	0.24	0.33	0.18	-	0.34	-	0.13	0.17	0.39	10	Laser
	Translation	1.28	2.36	1.15	0.93	0.98	0.45	-	0.44	-	0.64	0.83	1.0		
LoDoNet[49]	Rotation	0.69	0.28	0.57	0.98	0.45	0.59	0.34	1.64	0.97	0.35	0.45	0.66	-	Laser
	Translation	1.43	0.96	1.46	2.12	0.65	1.07	0.62	1.86	2.04	0.63	1.18	1.27		
S-LSD[4]	Rotation	0.26	0.36	0.23	0.28	0.31	0.18	0.18	0.29	0.31	0.26	0.33	0.26	30	Laser
	Translation	0.63	2.36	0.79	1.01	0.38	0.64	0.71	0.56	1.11	1.14	0.72	0.95		
SOFT[5]	Rotation	0.22	0.18	0.23	0.23	0.18	0.17	0.14	0.24	0.12	0.18	0.26	0.20	20	Stereo
	Translation	0.66	0.96	1.36	0.70	0.50	0.43	0.41	0.36	0.78	0.59	0.68	0.68		
LeGO-LOAM[50]	Rotation	1.05	1.02	1.01	1.18	1.01	0.74	0.63	0.81	0.94	0.98	0.92	1.0	-	Laser
	Translation	2.17	13.4	2.17	2.34	1.27	1.28	1.06	1.12	1.99	1.97	2.21	2.49		

Relative errors averaged over trajectories of 100 to 800 m in length: relative rotational error in degrees per 100 m and relative translational error in %. Bold numbers indicate the best performance for laser-based approaches.

to the ground truth in every dimension, as well as being stable and robust for the entire duration of the experiment. The details of the errors in translation and rotation are shown in Fig.3.4 and Fig.3.5. On the basis of presented figures, the estimation obtained in outdoor environments (around a distance of 150 m) is robust, whereas the conventional method, which does not use the feature points, yields large fluctuations. Fig.3.6 shows the trajectories of the estimated results obtained with and without the use of the feature points and ground truth. The error can be reduced because the use of the feature points reduces the likelihood of erroneously searching for the corresponding points. Because the points of the ground or walls are removed by the proposed BPNN, the remaining points with similar features are more likely to be the correctly corresponding points.

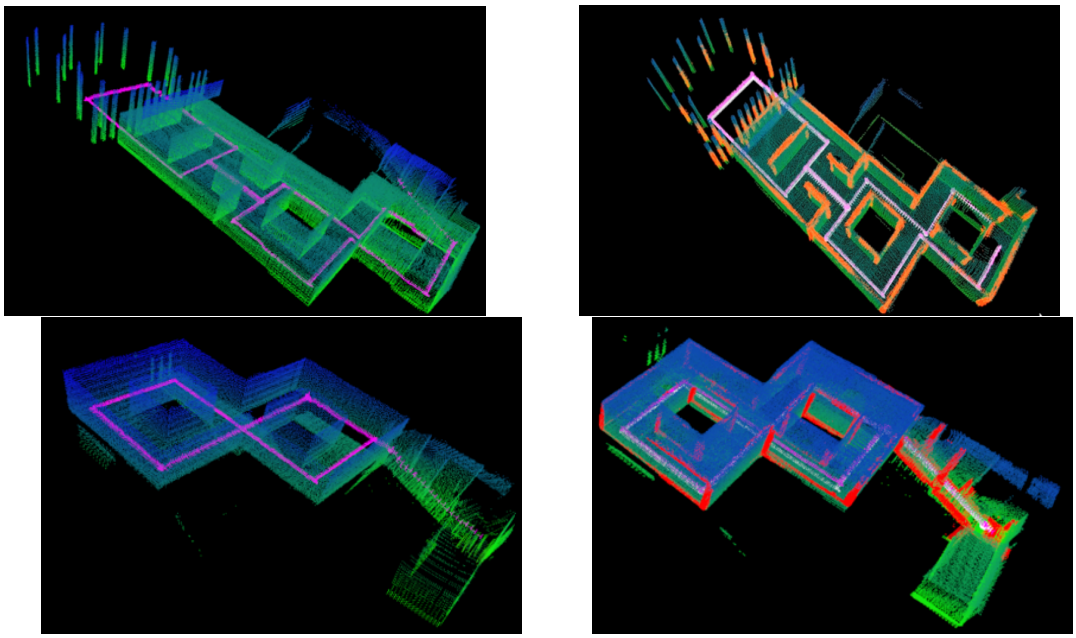

Fig. 3.3: Experimental environment and feature extraction: (a) Original environment (first floor); (b) extracted feature points (red points); (c) original environment (second floor); (d) extracted feature points (red points).

3.1.3 Experiment of 6D SLAM using Feature Extraction in KITTI benchmark

We evaluate our approach on the odometry datasets of the KITTI Vision Benchmark [26]. The provided point clouds were taken by the Velodyne HDL-64E S2 at a rate of 10 Hz. The dataset contains data from different street environments ranging from the inner city to highways. In particular, the highway datasets are a major challenge, as the sensor moves at a high speed, which can be faster than 80km/h, while other fast-moving dynamic objects are present and the environment are very commonly drawn from the real world. We use an Intel i9-9900K@3.6 GHz with 32 GB RAM, and an Nvidia GeForce RTX 2080TI with 11 GB RAM.

Tab.3.1 shows the results of our approach for all sequences in detail. The final estimated trajectory, compared with the ground truth, is shown in Fig.3.19. We compared the proposed method 1 and the proposed method 2 in the table and basically they obtain the

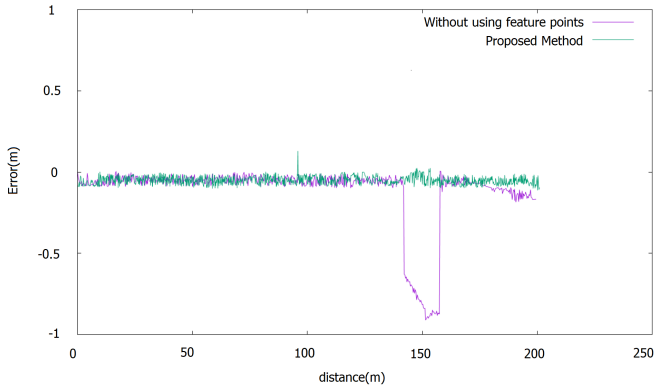

Fig. 3.4: Error of translation.

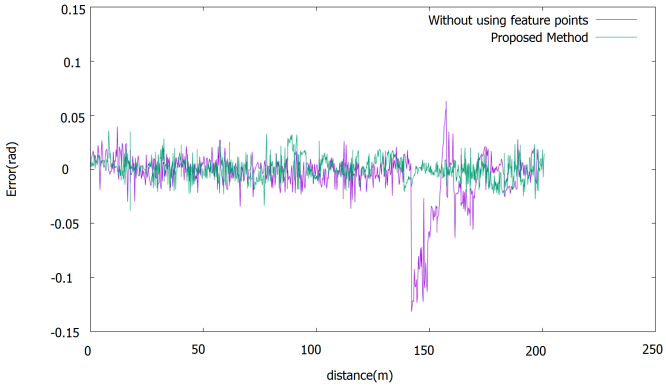

Fig. 3.5: Error of rotation.

best performance in more sequence. For the sequence of the data in highway, the proposed method 2 perform not such well since there are lack of the target objects. However, in the sequence where are full of urban or target objects, the proposed method 2 obtain a better results than the proposed method 1. Therefore, we believe that if we have more target objects, the proposed method 2 maybe a better method for accuracy. The differences seem not so large in the Tab.3.1. Therefor, we show the final estimated traces of the proposed method and the SUMA by using the sequence 02 of the KITTI benchmark as Fig.3.7. Though there are only 0.7% difference in the value, the final position of error can be more than 200m after a very long trace. In the Tab.3.1, the SOFT perform the best in the average translational error. However, the method is using the stereo camera. Comparing

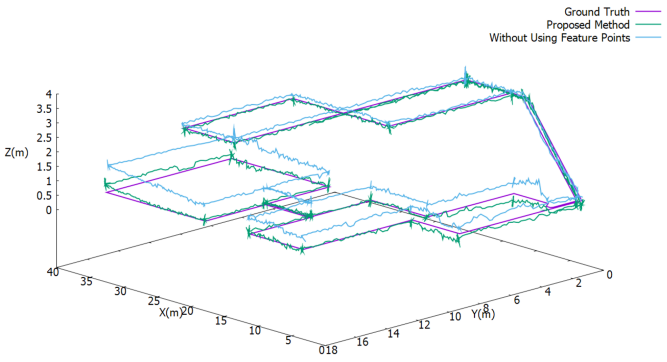

Fig. 3.6: Ground truth and estimated trajectory.

with the other state-of-the-art methods that only use the point clouds data (Laser) the proposed method have the best accuracy in estimation of translation and rotation. In the environments where have less target objects the proposed method 2 perform worse than the proposed method 1. However, even there are not target objects the proposed method 2 can still work robust because the method can still optimize the matched number of points with a low but highest score. Most methods match the scan by using the ICP methods such as SUMA and LOAM. The differences are how to fix the input scan data or how to fix the result of the ICP to obtain a higher accuracy in SLAM. The SUMA try to fix the input data by using a unique map to build the features for the ICP and with their feature points, the results are accuracy than the conventional ICP. The method LOAM also aim at fix the input data by correcting the moving distortion in each scan and extracting the boundary from the point clouds. The boundary is similar to our features, the edges. However, we do not use the boundary since we consider that most boundary of the 3D point clouds are caused by the limitation of the range of the 3D laser sensor. Therefore, we only the feature, where many planes are intersecting, because this kinds of features are more likely to be the correct correspondence and more likely to have the same physical meaning.

For comparison with the state-of-the-art methods, we included here the reported results of LOAM [6] and SUMA [27], a laser-based odometry approach. Overall, we achieved an average rotational error of 0.0024 deg/m and an average translational error of 0.504% compared to a 0.8% translational error of LOAM and a 0.0032 deg/m and 0.8% translational

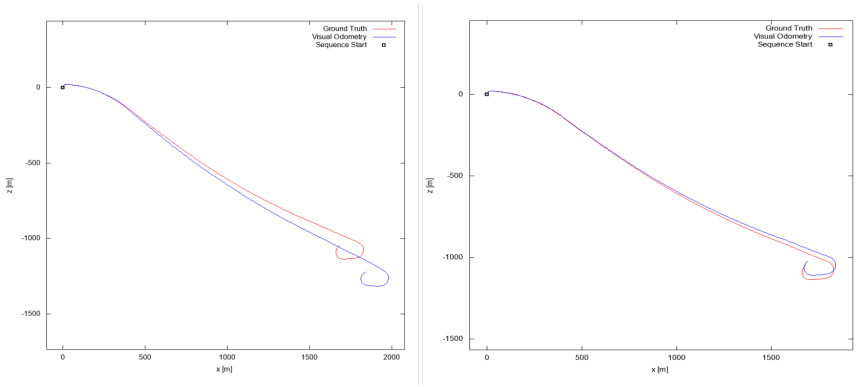

Fig. 3.7: Estimated trace in sequence 01. Left: SUMA with translation error 1.7%; Right: Proposed method with translation error 1.01%

error of SUMA in KITTI. The proposed method not only obtained a 25% improvement in rotational accuracy compared with the SUMA, but also received a 37% improvement in translational accuracy compared with the SUMA and the LOAM. Our proposed method performs more precisely than the state-of-the-art methods in the KITTI Vision Benchmark.

To prove the proposed method can estimate 6 dimensional solution for the 6D SLAM, we show the translation error of x, y and z axis, and the rotational error of roll, yaw and pitch angle in Fig.3.8 and Fig.3.9. There is not accumulating error in any sequences and any dimensions. Therefore, we can say that the proposed method can perform well without large error. For comparison in each dimension of the 6D SLAM, we also compare with the state-of-the-art method SUMA in Fig.3.10. Because both methods are using the ICP algorithm for part of the scan registration, the shape of the trend of the error is very similar. However, by using the feature points and the optimization of the PSO, error of the proposed method in each dimension is basically closer to zero than SUMA.

Fig. 3.8: Translational Error of the KITTI data set. The purple corresponds to the translational error in x axis, the green to the translational error in y axis, and the blue to the translational error in z axis (best viewed in color). (a) Sequence 00; (b) sequence 01; (c) sequence 02; (d) sequence 03; (e) Sequence 04; (f) sequence 05; (g) sequence 06; (h) sequence 07; (i) Sequence 08; (j) sequence 08; (k) sequence 10.

Fig. 3.9: Rotational Error of the KITTI data set. The purple corresponds to the rotational error in roll angle, the green to the rotational error in yaw angle, and the blue to the rotational error in pitch angle (best viewed in color). (a) Sequence 00; (b) sequence 01; (c) sequence 02; (d) sequence 03; (e) Sequence 04; (f) sequence 05; (g) sequence 06; (h) sequence 07; (i) Sequence 08; (j) sequence 08; (k) sequence 10.

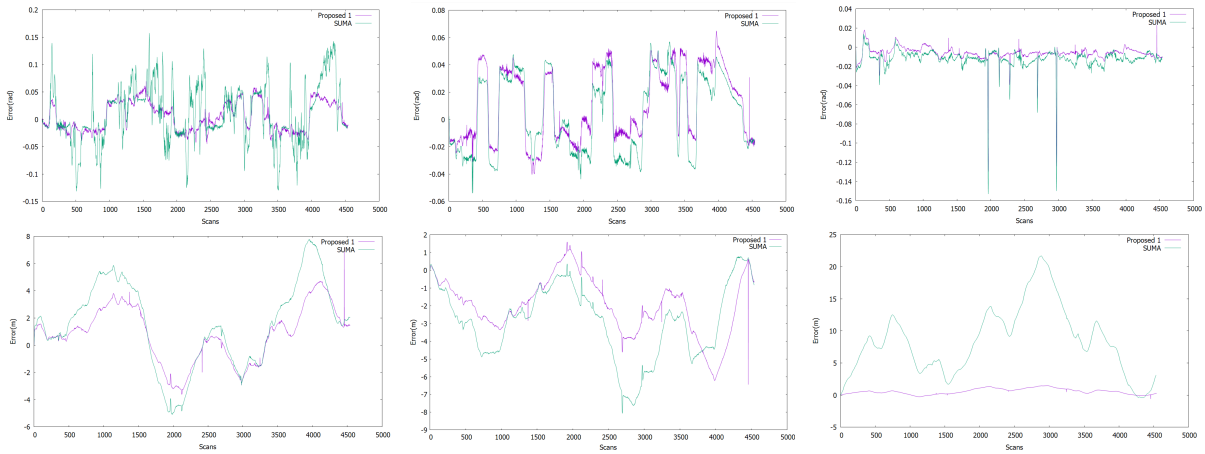

Fig. 3.10: Comparison of Estimated Error of the sequence 00 in the KITTI data set. The purple corresponds to the error of the proposed method 1, the green to the error of the SUMA (best viewed in color). (a) Error in Roll angle; (b) Error in Yaw angle; (c) Error in Pitch angle; (d) Error in X angle; (e) Error in Y angle; (f) Error in Z angle.

From visual inspection, we determined that our method cannot correctly estimate motion in an environment with very few meeting surfaces, such as highways with many trees and a lack of edges and corners.

For the runtime, Fig.3.11 shows the processing time needed to process sequence 00 from the KITTI Vision Benchmark. The processing time for each input for the proposed BPNN is the same. However, the number of points of each scan is not the same. The figure is showing the runtime for dealing with all the input points. Therefore, the fluctuation of the runtime is not because of the proposed method but the number of input points. The number of input points is shown in Fig.3.12. Overall, our approach runs at 60 ms in average, and is therefore able to process a scan at 10 Hz on average. For comparison, we run the LOAM and the SUMA in our equipment. The LOAM can run at 77 ms on average which is a little bit slower than the proposed method. The SUMA can run at 15 ms on average, which is the fastest in three SLAM systems, and sometimes the SUMA needs at most 30 ms for loop closure.

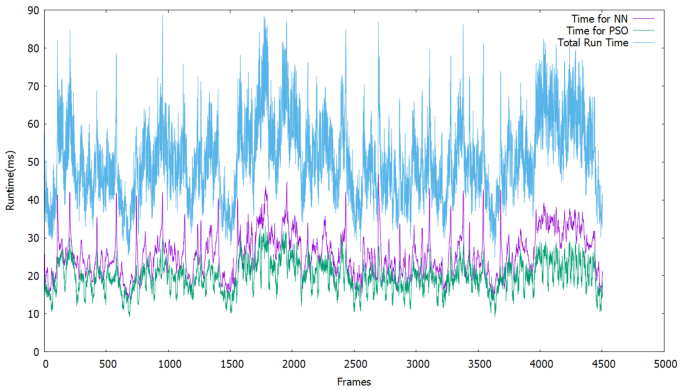

Fig. 3.11: Processing time needed to map sequence 00 from the KITTI Vision Benchmark.

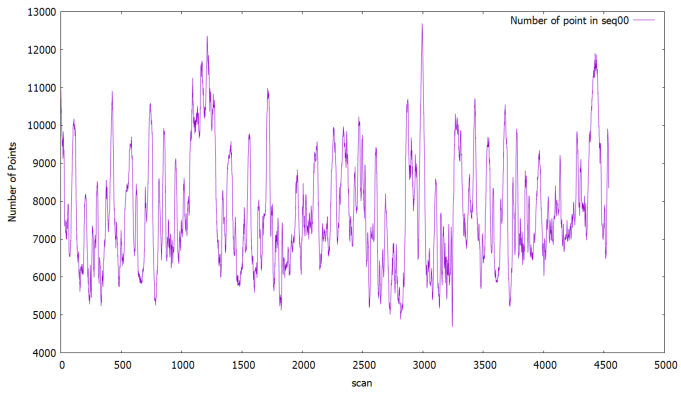

Fig. 3.12: The number of points in sequence 00 from the KITTI Vision Benchmark.

3.2 Experiments of 6D SLAM using the Object Recognition

3.2.1 6D SLAM using Object Recognition in Outdoor Experiment

We first discuss our primary experiment. In this experiment, a sequence of 3D point clouds was considered. The 3D point clouds were taken using one YVT-35x 3D laser scanner (HOKUYO). The sensor was set on the top of a wheel loader, which is about 3 meters high, and the wheel loader average moves in 20 km/h to 40 km/h in an excavation site, as

shown in Fig.3.13. Simultaneously, the ground truth was recorded using a high-accuracy Hiper II (Topcon) Global Positioning System (GPS). We evaluated the experiment by comparing the estimated location to the ground truth. In addition to the proposed SLAM system, we evaluated and compared the LOAM-SLAM [6], SUMA-SLAM [27], ISCLOAM [35], and MULLS methods, as well as a hybrid ICP and PSO method [34]. Most of these methods demonstrate good performance and a high rank in the KITTI benchmark. To use these methods in this experiment, we downloaded the corresponding programs from Github and executed them on our dataset. However, SUMA-SLAM could not obtain a reasonable result in this experiment. It appears that SUMA was built for 3D scanner sensor that can scan dense point clouds. However, in this experiment, each scan obtain an average of 1000 points in 50 ms and it is possible that this is overly sparse for SUMA.

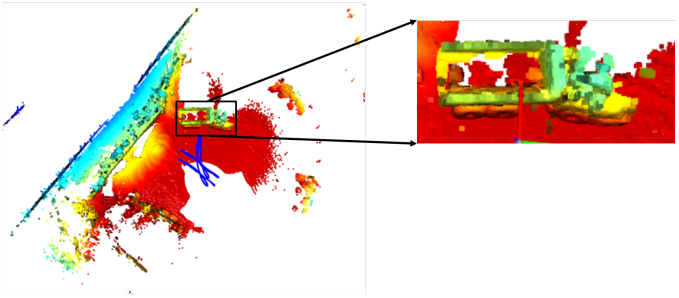

Fig. 3.13: Experimental environment. The map of the experiment environment was build using the proposed method. The right image shows the stationary dump truck in the experimental environment.

Three types of preset labels were considered in this experiment, i.e., labels the ground, truck, and other point. There was a single dump truck in the experimental environment, as shown in Fig.3.13. The dump truck was stationary in this experiment; thus, we considered it a landmark. The training data, which were used to train the proposed CNN to recognize the dump truck, were generated using a scanning model of the dump truck (Fig.3.14). The ground training data were generated by scanning a horizontal plane. The training data for other points were taken from a mountain slope part of the experiment data. After training, the average recognition accuracy was 85%, and the recognized result in shown in Fig.3.15. In this experiment, we used an Intel i9-9900K@3.6 GHz with 32 GB RAM, and an Nvidia GeForce RTX 2080TI with 11 GB RAM.

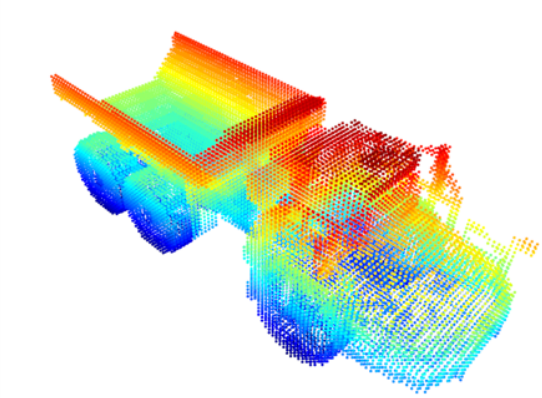

Fig. 3.14: Training data for dump truck: scanning model

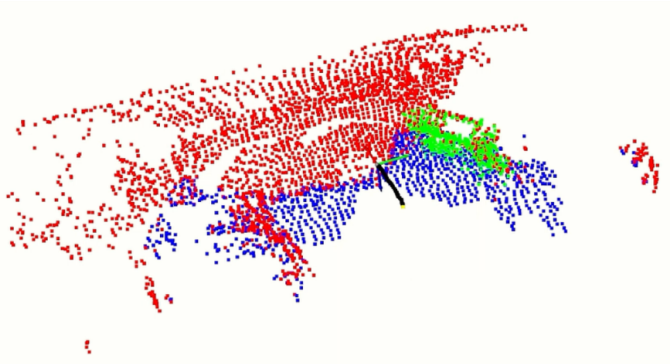

Fig. 3.15: Object recognition result obtained using the proposed method (red points: recognized as other points (mountain slope); blue points: recognized as the ground; green points: recognized as the dump truck)

Tab.3.2 compares the experimental results obtained for all compared methods. The final estimated trajectory compared to the ground truth is shown in Fig.3.16.

3.2.2 6D SLAM using Object Recognition in KITTI Experiment

In this experiment, we evaluated the proposed method on KITTI Vision Benchmark odometry dataset [26]. This 3D point cloud dataset was taken by a Velodyne HDL-64E S2 at a rate of 10 Hz. The sensor was installed on top of a passenger vehicle, and the vehicle moved in different street environments ranging from inner city to highway environment. Although there are some sequences of scans are lack of preset objects for the proposed

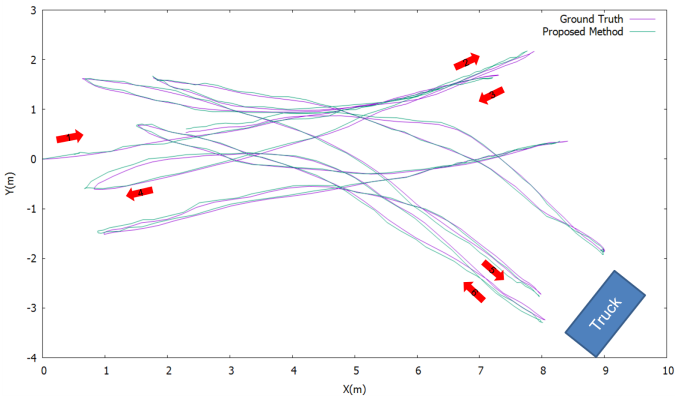

Fig. 3.16: Estimated trace of experiment. The moving direction and order are shown as the red arrows and the numbers inside. The location of the target object, truck, is shown as a blue box at the bottom right (purple line: ground truth; green line: estimated trace obtained using the proposed method)

Table. 3.2: Localization Results

Method	Maximal Error(m)	Average Error(m)	Average Time(ms)
Proposed 1	0.350	0.154	22
Proposed 2	0.290	0.101	55
ICP+PSO[34]	0.315	0.127	21
LOAM[6]	0.925	0.286	51
ISCLOAM[35]	0.812	0.243	44
MULLS	0.777	0.282	42

SLAM system, the proposed method worked normally.

For the preset objects we want to use for the proposed SLAM system, we selected trees, cars, walls, ground, and other points for the proposed CNN (Fig.3.17). In addition, the cars were dynamic in this experimental environment; thus, each point recognized as cars was deleted from the input point cloud. The training data were taken from part of the experimental data, and the training data were selected and labeled manually. The object recognition results obtained by the proposed CNN are shown in Fig.3.18. As can be seen, all recognized objects were essentially recognized correctly. However, there are still some objects has not been recognized. For example, in the bottom left of Fig.3.18, there are two trees (red points) were recognized as other points (gray points). With the KITTI dataset, we could not evaluate the accuracy of the proposed CNN because we did not have the ground truth data, i.e., labeled test samples. However, we primarily focus on

the performance of 6DoF-SLAM; thus, we mainly evaluated the accuracy of the proposed SLAM system on this dataset.

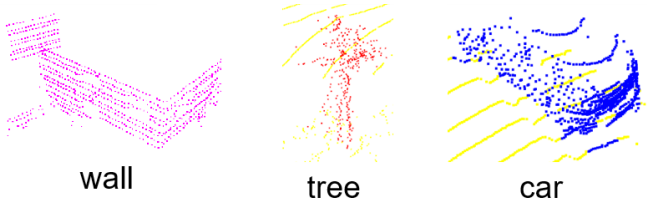

Fig. 3.17: Example of objects in KITTI dataset (pink: wall; red: tree; blue: car)

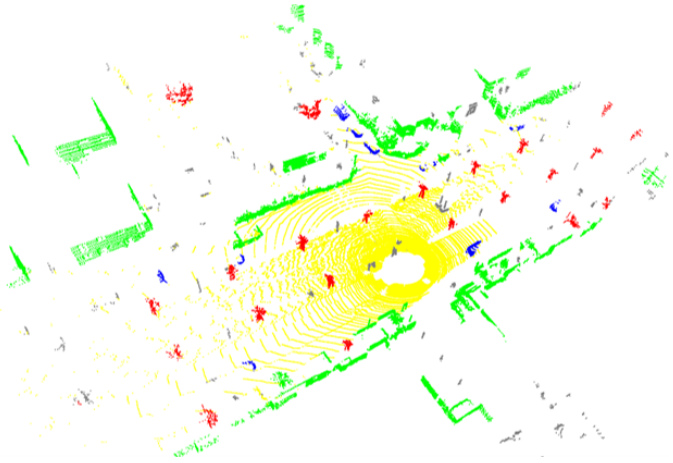

Fig. 3.18: Recognition result of one scan in KITTI (green: wall; red: tree; blue: car; yellow: ground; gray: other points)

Eleven sequences of 3D point cloud data with ground truth were captured using a high accuracy Global Positioning System/Inertial Navigation System (GPS/INS). To compare the proposed method to the state-of-the-art, we input the result of the 11 sequences (sequence 00 to 10) into the evaluated program provided by the KITTI Vision Benchmark, and the translation and rotation errors were calculated automatically. Overall, the proposed method obtained an average rotational error of 0.0025 deg/m, average translational error of 0.55% compared to 0.003 deg/m and 0.8% translational error of SUMA [27] and 0.8% translational error of LOAM [6] on KITTI (Tab.3.3). Therefor, the proposed method increase the rotational accuracy for 16.7% and translational accuracy for 31.3%.

Table. 3.3: Results Obtained on KITTI Odometry Dataset (Sequences 00 to 10)

Method	Average Translation Error	Average Rotation Error
Proposed 1	0.233	0.504
Proposed 2	0.252	0.554
LOAM[6]	-	0.845
SUMA[27]	0.354	0.763

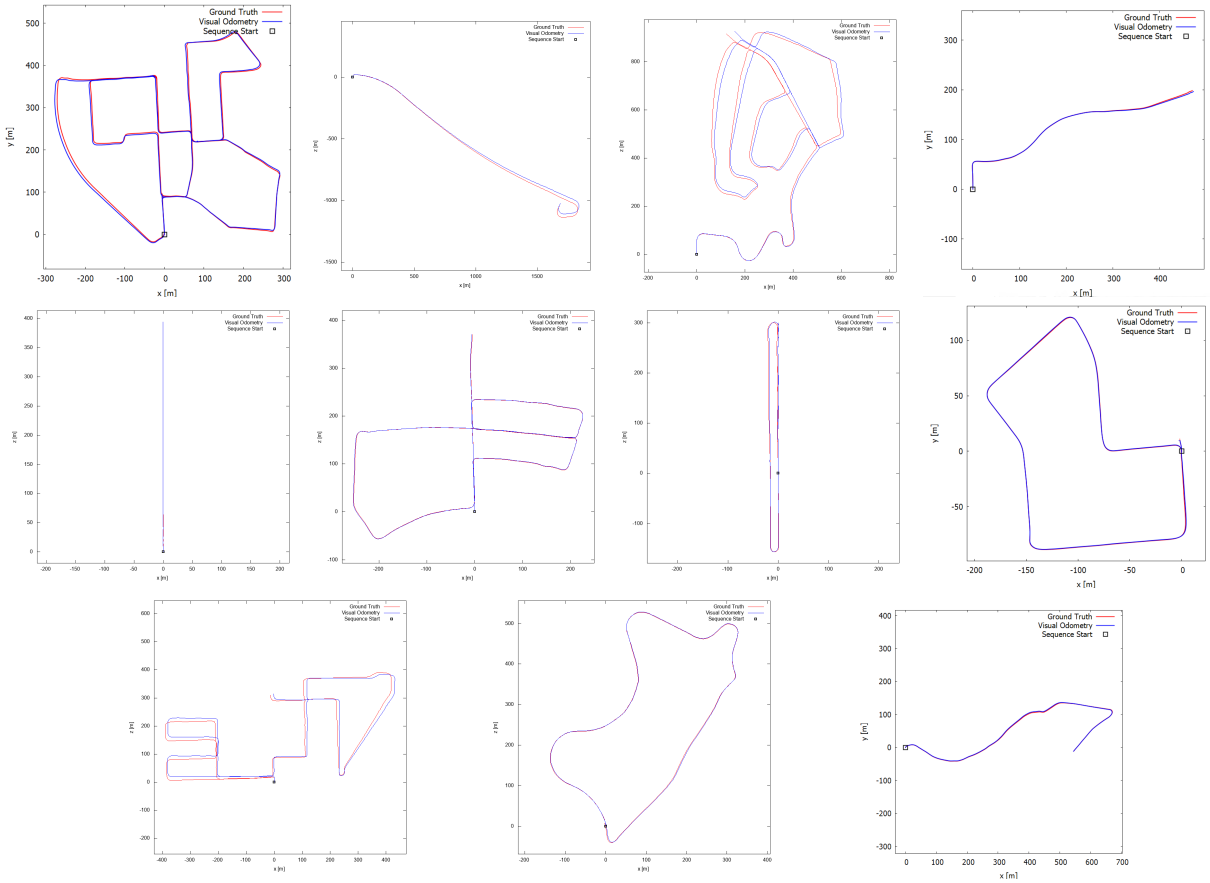

Fig. 3.19: Trajectories of the KITTI data set. The red corresponds to the GPS-based ground truth, the blue to our approach (best viewed in color). (a) Sequence 00; (b) sequence 01; (c) sequence 02; (d) sequence 03; (e) Sequence 04; (f) sequence 05; (g) sequence 06; (h) sequence 07; (i) Sequence 08; (j) sequence 08; (k) sequence 10.

4. Conclusions and Future work

4.1 Conclusions

In this thesis, we present 2 methods to improve the performance of the 6D SLAM in accuracy and computational cost.

In the first proposed method, we present a new way that using the BP Neural Network for feature extraction. Comparing to the conventional methods for feature extraction, the proposed method dramatically decrease the computational consumption by using the simple BPNN and it is possible to satisfy the real time 3D SLAM. The proposed input shape that using 26 elements to represent the whole slide-window and the fast method for coordinate uniform ensure the real-time of the feature extraction. For the real-time issue, we introduce the PCA to avoid the repetitive training and ensure the computational time since the PCA method are really fast. It is important to uniform the coordination of the slide windows by using the PCA. The conventional method for edges is 2s for 7000 points, while we can run in 0.03s for 18000 points.

In the simulation we successfully extract corners and edges from a 3D point cloud. With the help of the feature points and the optimization processing, the accuracy is very high. In the experiment, we can also see that, the first proposed method can obtain the best accuracy comparing with the state-of-the-art algorithm in 6D SLAM.

In the second proposed method, we still focus on how to avoid the incorrect corresponding point pairs for the ICP algorithm. We believe the incorrect corresponding points are the main problem for the accuracy of the 6D SLAM. Therefore, to avoid the incorrect corresponding, we propose a way to match the points with the same label and the same label means they are the same objects. Therefore, we also proposed a method to recognized the objects from the 3D point cloud by using a CNN. In the two experiments, we can see

that our method can susscefully recognized the target objects in a high accuracy. Then we also propose a way to match the points with the same labels. We give a high score to a match if a point match a point with the same label. It means that we look more important the qullity of the matching than the quantity of the matching. In the experiment, we can also see that the proposed 2 obtain a better result that some high ranking method in the KITTI benchmark.

Compared with the state-of-art methods, both the proposed methods can run in real-time and achieve a better performance in 3D laser-based SLAM. In the environment where has enough preset objects, the proposed method 2 can obtain a better result than the proposed method 1. However, in the environment such as highway (sequence 01 and 10) where is lack of preset objects, the proposed method 1 is better. The edges as the feature is very common but it can only decrease the incorrect correspondence. If the preset object is enough, the proposed method should be better.

4.2 Future work

In the Future work, we focus on overcoming the original error of the 3D laser sensor itself. We believe that the sensor has two main error. They are the random error and the discreteness. The random error is some kind like the noise of the sensor. The discreteness is the limitation of the sensor, because it is impossible to take the whole world by a laser sensor now. There are infinite points in the real world, but a sensor can only take less that 100,000 in one scan. Therefore, we can never take a real same point between 2 scans. However, if the point is close enough, they will be match by the ICP algorithm. Because they are not the same point, when we try to match them, there will be more or less error. Therefore, we proposed method to project the points into the plane then the random error will be average and decrease. If we have two intersecting planes, we can also interpolate a line, and the line is more like to obtain a same point. Because the line is also calculated and interpolated by the calculated planes and the error can be average and decrease.

Reference

[1] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part I," IEEE Robot. Automat. Mag., vol. 13, no. 2, pp. 99–110, 2006.

[2] R. A. Newcombe, et al., "KinectFusion: Real-time dense surface mapping and tracking," IEEE Int. Symp. Mixed and Augmented Reality, 2011.

[3] A. Dai, et al., "Bundlefusion: Real-time globally consistent 3d reconstruction using on-the-fly surface reintegration," ACM Trans. Graphics, vol. 36, no. 4, 2017.

[4] J. Engel, et al., "Large-scale direct SLAM with stereo cameras," IEEE/RSJ Int. Conf. Intell. Robotics Syst., 2015.

[5] I. Cvisic, et al, "SOFT-SLAM: Computationally efficient stereo visual simultaneous localization and mapping for autonomous unmanned aerial vehicles," J. Field Robotics, vol. 35, no. 4, pp. 578–595, 2017.

[6] J. Zhang and S. Singh, "LOAM: Lidar Odometry and Mapping in Real-time," Robotics: Science and Syst., vol. 2, no, 9. 2014.

[7] J. Zhang and S. Singh, "Low-drift and real-time lidar odometry and mapping," Autonomous Robots, vol. 41, no. 2, pp. 401–416, 2017.

[8] F. Moosmann and C. Stiller, "Velodyne slam," IEEE Intell. Vehicles Symp. (IV), pp. 393–398, 2011.

[9] M. Velas, M. Spanel and A. Herout, "Collar line segments for fast odometry estimation from velodyne point clouds," IEEE Int. Conf. Robotics and Automation, 2016.

[10] M. G. Dissanayake, et al., "A solution to the simultaneous localization and map building (SLAM) problem," IEEE Trans. robotics and automation, vol. 17, no. 3, pp. 229–241, 2001.

[11] A. Nchter, et al., "6D SLAM-3D mapping outdoor environments," J. Field Robotics vol. 24, no. 8–9, pp. 699–722, 2007.

[12] P. J. Besl and N. D. McKay, "Method for registration of 3-D shapes," Sensor fusion IV: control paradigms and data structures, vol. 1611, pp. 586–606, 1992.

[13] D. Haehnel, T. Sebastian and W. Burgard, "An extension of the ICP algorithm for modeling nonrigid objects with mobile robots," Int. Joint Conf. Artificial Intelligence, Vol. 3, pp. 915–920, 2003.

[14] R. Kuramachi, et al., "G-ICP SLAM: An odometry-free 3D mapping system with robust 6DoF pose estimation," IEEE Int. Conf. Robotics and Biomimetics, 2015.

[15] M. Tomono, "Robust 3D SLAM with a stereo camera based on an edge-point ICP algorithm," IEEE Int. Conf. Robotics and Automation, pp.4306–4311, 2009.

[16] P. J. Besl and N. D. McKay, "Method for registration of 3-D shapes," Sensor fusion IV: control paradigms and data structures, vol. 1611, pp. 586–606, 1992.

[17] T. Zinber, J. Schmidt and H. Niemann, "A refined ICP algorithm for robust 3-D correspondence estimation," IEEE Proc. Int. Conf. Image Processing, vol. 2, 2003.

[18] H. Yudai and F. Yasutaka, "Experimental Verification of Path Planning with SLAM," IEEJ J. Industry Applications, vol. 5, no. 3, pp.253–260, 2016.

[19] N. Yuta and F. Yasutaka, "Validation of SLAM without odometry in outdoor environment," IEEE Int. Workshop on Advanced Motion Control, pp. 278–283, 2014.

[20] Beheshti, Zahra, and Siti Mariyam Hj Shamsuddin. "A review of population-based meta-heuristic algorithms," Int. J. Adv. Soft Comput. Appl 5.1 (2013): 1-35.

[21] S. Gumhold, X. Wang and R. S. MacLeod, "Feature Extraction From Point Clouds," Int. Meshing Roundtable, 2001.

[22] R. B. Rusu, et al., "Towards 3D point cloud based object maps for household environments," Robotics and Autonomous Syst, vol. 56, no. 11, pp. 927–941, 2008.

[23] R. B. Rusu, N. Blodow and M. Beetz, "Fast point feature histograms (FPFH) for 3D registration," IEEE Int. Conf. Robotics and Automation, 2009.

[24] R. Hecht-Nielsen, "Theory of the backpropagation neural network," Neural networks for perception, 1992.

[25] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," IEEE Conf. Computer Vision and Pattern Recognition, 2001.

[26] A. Geiger, P. Lenz and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," IEEE Conf. Computer Vision and Pattern Recognition, 2012.

[27] J. Behley and C. Stachniss, "Efficient Surfel-Based SLAM using 3D Laser Range Data in Urban Environments," Robotics: Science and Syst., 2018.

[28] W. Shengbin, W. Shaoqing and Z. Changhe, "An Iterative Closest Point Algorithm Based on Biunuque Correspondence of Point Clouds for 3D Reconstruction," Acta Optica Sinica, vol. 35, no. 5, pp. 252–258, 2015

[29] C. Yuan, X. Yu and Z. Luo, "3D point cloud matching based on principal component analysis and iterative closest point algorithm," IEEE Int. Conf. Audio, Language and Image Processing, 2016.

[30] M. Clerc, "The swarm and the queen: towards a deterministic and adaptive particle swarm optimization," IEEE Congress Evolutionary Computation, 1999.

[31] A. E. Johnson and S. B. Kang, "Registration and integration of textured 3-D data," Image and vision computing, vol. 17, no. 2, pp. 135–147, 1999.

[32] D. Akca, "Matching of 3D surfaces and their intensities," Journal of Photogrammetry and Remote Sensing, vol. 62, no. 2, pp. 112–121, 2007.

[33] K.-H. Bae and D. D. Lichti, "Automated registration of unorganized point clouds from terrestrial laser scanners," in Int. Archives of Photogrammetry and Remote Sensing (IAPRS), pp. 222–227, 2004.

[34] Jiayi Wang and Yasutaka Fujimoto, "Combination of the ICP and the PSO for 3D-SLAM," proc. IEEE Industrial Electronics Society Annual Conference (IECON), Beijing, 2017.

[35] H. Wang, et al, "Intensity Scan Context: Coding Intensity and Geometry Relations for Loop Closure Detection," arXiv:2003.05656, 2020.

[36] D. Z. Wang and I. Posner, "Voting for voting in online point cloud object detection," Robotics: Science and Syst., vol. 1, no. 3, 2015.

[37] M. Engelcke, et al., "Vote3deep: Fast object detection in 3d point clouds using efficient convolutional neural networks," IEEE Int. Conf. on Robotics and Automation (ICRA), pp. 1355–1361, 2017.

[38] G. Bradski, "The OpenCV Library," Dr. Dobb's Journal of Software Tools, 2000.

[39] K. T. Gribbon, and D. G. Bailey, "A novel approach to real-time bilinear interpolation," IEEE Int. Workshop on Electronic Design, Test and Applications, pp. 126–131, 2004.

[40] C. M. Bishop, "Pattern recognition and machine learning," springer, 2006.

[41] Chen, X., Milioto, A., Palazzolo, E., Giguere, P., Behley, J., and Stachniss, C. (2019, November). Suma++: Efficient lidar-based semantic slam. In 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (pp. 4530-4537). IEEE.

[42] Deschaud, J. E. (2018, May). IMLS-SLAM: scan-to-model matching based on 3D data. In 2018 IEEE International Conference on Robotics and Automation (ICRA) (pp. 2480-2485). IEEE.

[43] F. Neuhaus, T. KoB, R. Kohnen, and D. Paulus, “Mc2slam: Real time inertial lidar odometry using two-scan motion compensation,” in German Conference on Pattern Recognition, 2018.

[44] B. Zhou, Y. He, K. Qian, X. Ma, and X. Li, “S4-SLAM: A real time 3D LIDAR SLAM system for ground water surface multi-scene outdoor applications,” Autonomous Robots, pp. 122, 2020.

[45] Hen, B. Wang, X. Wang, H. Deng, B. Wang, and S. Zhang, “PSF-LO: Parameterized Semantic Features Based Lidar Odometry,” arXiv preprint arXiv:2010.13355, 2020.

[46] M. Yokozuka, K. Koide, S. Oishi, and A. Banno, “LiTAMIN2: Ultra Light LiDAR-based SLAM using Geometric Approximation applied with KL-Divergence,” IEEE International Conference on Robotics and Automation (ICRA), 2021.

[47] Q. Li, S. Chen, C. Wang, X. Li, C. Wen, M. Cheng, and J. Li, “LO-Net: Deep Real-time Lidar Odometry,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 84738482.

[48] I. Garc, M. Rentero, C. Salinas Maldonado, R. Izquierdo Gonzalo, and N. Hernandez Parra, “Fail-aware lidar-based odometry for autonomous vehicles,” Sensors, vol. 20, no. 15, p. 4097, 2020.

[49] C. Zheng, Y. Lyu, M. Li, and Z. Zhang, “Lodonet: A deep neural network with 2d keypoint matching for 3d lidar odometry estimation,” in Proceedings of the 28th ACM International Conference on Multimedia, 2020, pp. 23912399.

[50] Shan, Tixiao, and Brendan Englot. ”Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain.” 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2018.

Published documents

[1] Jiayi Wang and Yasutaka Fujimoto, "Comparison of ICP and PSO in 3D Map Matching, Proc. IEEJ Int. Workshop on Sensing, Actuation, and Motion Control, TT7-6, 2017.

[2] Jiayi Wang and Yasutaka Fujimoto, "Combination of the ICP and the PSO for 3D-SLAM," proc. IEEE Industrial Electronics Society Annual Conference (IECON), Beijing, 2017.

[3] Jiayi Wang and Yasutaka Fujimoto, "A High Accuracy Real-Time 6D SLAM with Feature Extraction using a Neural Network," IEEJ Journal of Industry Application (Accepted).