Testing and Evaluation of LiDAR SLAM Algorithms Based on the Combination of Virtuality and Reality Method

1st Xing Chen
Department of Automation
Tsinghua University
Beijing, China
chenxing23@mails.tsinghua.edu.cn

2nd Junyi Bao
Department of Automation
Tsinghua University
Beijing, China
baojy20@mails.tsinghua.edu.cn

3rd Xinyao Han
Department of Automation
Tsinghua University
Beijing, China
hanxy24@mails.tsinghua.edu.cn

4nd Lihui Peng
Department of Automation
Tsinghua University
Beijing, China
lihuipeng@tsinghua.edu.cn

5nd Yi Zhang
Department of Automation
Tsinghua University
Beijing, China
zhyi@tsinghua.edu.cn

6nd Danya Yao
Department of Automation
Tsinghua University
Beijing, China
yaody@tsinghua.edu.cn

2024 IEEE International Conference on Imaging Systems and Techniques (IST) | 979-8-3503-7821-4/24/$31.00 ©2024 IEEE | DOI: 10.1109/IST63414.2024.10759188

***Abstract*—Simultaneous localization and mapping (SLAM), as an important means of sensing the surrounding environment, has a direct impact on the execution efficiency of autonomous grounding vehicles (UGV). At present, the most common means used to map the environment is to use hardware devices such as LiDAR, inertial measurement unit (IMU) or camera to collect data with environmental information, and then run the SLAM algorithm to construct the environment map. It is crucial by designing a system that can integrate testing and evaluation of the SLAM algorithm to promote the optimization iteration of the algorithm. In this paper, we not only collected the data from different sensors on a real UVG in the physical world test field, but also imported these data into the virtual simulation platform to construct a map of the real world test field. By maneuvering a vehicle in this virtual test field and gathering the data from the corresponding sensors installed, a typical LiDAR SLAM algorithm based on the combination of virtuality and reality was tested and evaluated as a case study.**

***Index Terms*—LiDAR SLAM, combination of virtuality and reality, dataset generation, test and evaluation**

I. Introduction

SLAM is an algorithm for constructing map models and localizing robots/intelligent vehicles simultaneously. The SLAM algorithm is proposed to solve the problem of whether there is a way for a robot that is placed into a certain location in an unknown environment to decide which direction it should move next while gradually drawing a map of this environment. The purpose of the algorithm to construct the map is to describe the region of interest of the intelligent vehicle in the surrounding environment, while the purpose of localization is to establish the positional relationship between the intelligent vehicle and the surrounding environment. SLAM algorithms are usually categorized into LiDAR SLAM algorithms, visual SLAM algorithms and so on. The LiDAR SLAM uses the point cloud data scanned by LiDAR and the angular velocity with acceleration data collected by IMU as the dataset, while the visual SLAM uses the image data collected by the on-board camera as the dataset. Generally speaking, LiDAR has higher accuracy in feature extraction and ranging, and IMU is able to correct the position of the vehicle, so the mapping results of LiDAR SLAM algorithm are better in most cases. The visual SLAM is more advantageous in terms of cost because it only uses one kind of hardware device, the camera, whose price is much lower than that of the LiDAR device. Recently, the results of Yuan et al. have shown that a multi-sensor fusion scheme combining LiDAR SLAM with vision SLAM is also feasible [1].

Given the importance of the SLAM algorithm for autonomous driving, the evaluation of their maping capabilities is an important task. For the LiDAR SLAM algorithm, it is generally very difficult to directly compare the maps constructed by the algorithm due to the large volume of point cloud data used in the mapping process. Instead, the mapping trajectory is often used as an indirect object to test and evaluate the performance of the SLAM algorithm. By comparing the self driving trajectory calculated by the SLAM algorithm with the actual driving trajectory of the vehicle, the error information of the SLAM algorithm in estimating the self-vehicle position can be obtained, which can to some extent reflect the accuracy of the algorithm in recovering the point cloud position during mapping.

For the test work of the SLAM algorithm, the collection of real data is time-consuming and laborious, and the performance of the hardware equipment is also very demanding. In this paper, we adopt the strategy of combining real and virtual data. The field map collected from the reality is imported into CARLA, an open source autonomous driving research simulator [2]. Then, a vehicle equipped with LiDAR and IMU

devices is controlled in the virtual simulation platform to travel along a set track. The data collected by the sensors while the vehicle is traveling is recorded. Further, this paper deploys the open-source LiDAR SLAM algorithm LIO-SAM developed by Shan et al. [3] in order to focus on the test and evaluation of the LiDAR SLAM algorithm. We input the point cloud data of LiDAR and inertial guidance data of IMU collected from the real vehicle and the corresponding data generated in the virtual simulation platform into the algorithm for mapping. Finally, for the output mapping results of the algorithm, we use the trajectory evaluation tool **evo** [4] to evaluate the accuracy of the predicted trajectories, and then measure the mapping capability of the SLAM algorithm.

II. Related Works

In terms of evaluating SLAM algorithms, different scholars have used different indicators and methods to evaluate different algorithms. Huletski et al. introduced some general indicators for estimating the performance and quality of SLAM algorithms such as localization accuracy, dataset processing time, peak memory consumption and some specific indicators only applicable to visual SLAM such as camera frame processing [5]. Krinkin et al. estimated the accuracy of the Laser SLAM algorithm by comparing the output trajectory with the ground truth trajectory based on The MIT Stata Center dataset that provides ground truth trajectories [6]. Filipenko et al. compared the operational results of 2D LiDAR-based SLAM, monocular camera-based SLAM and stereo camera-based SLAM based on a dataset collected indoors with a mobile robot, using absolute trajectory error as an evaluation metric [7]. Li et al. defined four indicators: the tracking accuracy, initialization quality, tracking robustness, and relocalization quality to evaluate the SLAM systems for mobile AR [8]. Zhang et al. evaluated 7 RGB-D SLAM systems using direct-based/indirect-based tracking, volumetric-based/point-based mapping and with/without loop detection strategy as indicators [9]. Wu et al. ran five SLAM algorithms on the Kitty dataset and then used evo to evaluate their absolute trajectory error and relative pose error [10]——two evaluation indicators proposed by Sturm et al. [11].

Furthermore, some scholars have designed evaluation frameworks for the SLAM algorithm. Filatov et al. presented a framework to quantitatively evaluate SLAM algorithms based on Jenkins, involving three indicators: the promotion of occupied and free cells, the amount of partners in a map and the amount of enclosed areas [12]. Schops et al. propose a novel, well-calibrated benchmark for SLAM evaluation that includes a training set, a test set without public ground truth, and an online evaluation service [13]. Bujanca et al. proposed a benchmarking suite named SLAMBench 3.0 that supports scene understanding and dynamic SLAM, which integrated relevant algorithms, datasets and indicators for evaluating SLAM systems [14]. Bettens et al. developed a rapid test simulation platform for SLAM algorithms connected to the Robotic Operating System (ROS) using the Unreal Engine [15]. Sobczak et al. developed a LiDAR simulator that can provides accurate 3D point clouds in real time, which is compatible with ROS [16].

III. Mapping with Data from A Vehicle in A Real World Test Field

A. Data Collection

The real vehicle data used in this paper is collected from a real world test field, which is a mega automobile test drive field located in downtown Beijing. A 16-line ring-scanning LiDAR RS-LiDAR-16 manufactured by RoboSense is equipped on an unmanned vehicle to scan the point cloud information of the surrounding environment. In addition, the vehicle is equipped with an IMU sensor, consisting of three single-axis accelerometers and three single-axis gyroscopes, which are used to measure the vehicle's three-axis acceleration and three-dimensional angular velocity data. The point cloud data and inertial guidance data are collected and stored temporarily in the unmanned vehicle at approximately 4 Hz and 150 Hz respectively. In actual operation, the unmanned vehicle takes about 8 minutes to collect data around the entire field.

The data collected by the real vehicle includes point cloud files stored in pacp format and inertial guidance data files stored in csv format. The pcap file can be replayed by the official Robosense LiDAR driver tool, *rslidar_sdk*. A third-party tool *rs_to_velodyne* is used to change the format of the replay data to the format corresponding to the *velodyne_points* topic. Then the data can be listened and recorded into a rosbag using the ROS record command *rosbag record*. The main information contained in the csv file includes timestamp, quaternion xyzw representing pose, angular velocity and linear acceleration in xyz axis, which can be converted into a rosbag with the help of python's rosbag library. The rosbag library has predefined the IMU class. When constructing the IMU object, it is only necessary to assign the corresponding properties according to the established template. We wrote code to merge the topics and data of the two rosbags together and complete the timestamp alignment. The final result is a rosbag with PointCloud2 and Imu sensor information.

B. Mapping Result

We deploy the LIO-SAM algorithm by modifying the algorithm parameters corresponding to the real world test field for this study. The point cloud map constructed during the operation of the algorithm is shown in Fig 1.

In Fig 1, the position occupied by the red, green and blue axes represents the location of the vehicle. The light blue line following it indicates the vehicle's motion trajectory recognized by the algorithm. The rings that moves along with the trajectory indicate the scanning range of the on-board LiDAR. When there are no objects around, the LiDAR only scans the ground and produces a series of panning circular point clouds. The left side of Fig 1 shows the fenced area of the site. It can be seen that the outline of the fence can be depicted by the algorithm. Since the point cloud was acquired in reality, some realistic details such as the railings of the fence are also represented in the map. The upper portion of

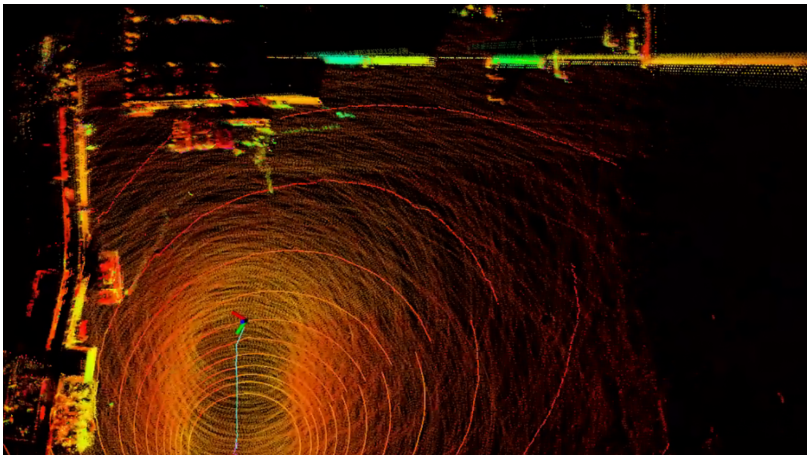


Fig. 1. Point cloud map from real collected data.

Fig 1 shows the garage area of the site, with the different garage doors being marked using different colors. The vehicles temporarily parked outside the garage and next to the fence can be clearly observed. In addition, it can be seen that although the vehicle is still traveling close to the fence on the left side, the outline of the garage on the right side is already shown in the map, reflecting the LiDAR's ability to implement long range scanning.

IV. Mapping with Data from the Virtual Test Field

A. Data Collection

The virtual data used in this paper is collected from the CARLA simulation platform. Before using the simulation platform, it is necessary to import customized maps into it in order to run the self-collected maps. Here we accomplish this function with the help of RoadRunner. RoadRunner is an interactive editor developed by MathWorks for designing 3D scenes for simulation and test of autonomous driving systems. This experiment uses this editor to implement functions such as road 3D scene modeling, import and export of road networks, and setting of road driving points.

CARLA runs simulations based on the Unreal Engine (UE). Maps imported into CARLA can be visualized by this engine. UE is a 3D rendering visualization engine developed by Epic. Although it was originally created as a game development tool, its value in expanding fields such as architecture and automobile has been gradually developed. This article uses its 4th generation version. After installation, type *make launch* in the CARLA directory to wake up its interface.

In order to communicate between CARLA and ROS, the carla-ros-bridge tool is also used here for vehicle control. It controls vehicle forwarding and steering by publishing commands in the form of *CarlaEgoVehicleControl* topics. Auto-routing logic is added to enable the vehicle to follow a set route. Finally, the *rostopic list* command is used to query the point cloud and IMU topics published by CARLA, which are "carla/ego_vehicle/lidar" and "carla/ego_vehicle/imu" Separately. While controlling the vehicle to run in the CARLA, the point cloud data and inertial guidance data are recorded as a rosbag using the *rosbag record <topic>* command. Both the point cloud data and the inertial guidance data were collected at a frequency of 20 Hz. It is worth noting that although LIO-SAM has certain requirements on the frequency of the inertial guidance data, preferably not lower than 100Hz, a small multiplier difference is acceptable due to the neat alignment of the point cloud data and the inertial guidance data generated by the simulation.

Another point need to be pointed out is that since the vehicle runs in a virtual simulation system, the speed can be set very fast. The data packet collected in the virtual test field contains only 90 seconds of data, while actually our vehicle had traveled most of the way around the map's established track at the end of the packet. This is difficult to achieve in a field experiment, as there are a number of constraints that need to be taken into account when running the vehicle, such as safety, equipment performance, and so on. In fact, the running time for the vehicle to travel around the field in reality is 466 seconds, which is about five times of the time taken by the simulation. This is also the advantage of the simulation system data collection compared to the real vehicle data collection.

B. Mapping Result

We modify the topic parameters of the SLAM algorithm to the corresponding parameters in the virtual test field, then start the algorithm and play the data packet. After the algorithm has run, the output map file is visualized to get the map building results, as shown in Fig 2.

It can be seen that the results produced using the two datasets are basically the same globally in terms of portraying the contours of the site. Even the contours of the point cloud map constructed using the simulation dataset are clearer. However, since the maps used for simulation are obtained by scanning the real site. Due to the accuracy of the equipment, some details are inevitably lost during the scanning process. Therefore, the constructed point cloud map contains less local details. Nevertheless, the simulation data has a unique

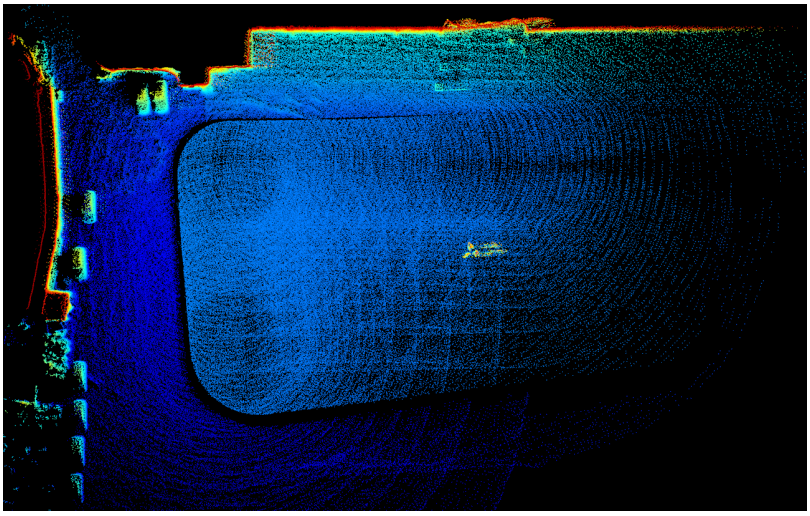


Fig. 2. Point cloud map from virtual simulation data.

advantage, which is the flexibility to edit the site map. For example, in Fig 2, we have arranged multiple parked cars at the edges of the site and pedestrians in the center of the site, without actually having them in reality. The algorithm is still able to scan the models and display them in the constructed map.

V. Evaluation of SLAM Algorithm

A. Position and Pose Data

The information provided by the LiDAR and IMU does not include the position and pose data, which is a key piece of information that describes the mapping process of the SLAM algorithm. Therefore, the algorithm needs to fuse the input data from various sensors to estimate the self-vehicle position and pose data.The output interface for trajectories is already provided by LIO-SAM in the project setup, while the pose data need to be obtained by modifying the code to extract intermediate variables. Evo is a tool specifically developed to process, evaluate and compare the trajectory output of SLAM algorithms. It has certain requirements on the format of the input data, which needs to contain eight items of data, namely timestamp (in seconds), three-dimensional coordinates representing position and quaternion representing direction.

In addition to the estimated trajectory calculated by the algorithm, the evaluation also needs the reference trajectory as the ground truth. The acquisition of positional coordinate information for the reference trajectory requires the use of the GPS positioning function. Since the amplitude of the vehicle's motion varies very little at the scale of the GPS global view, the latitude and longitude variations of the GPS localization can be used to describe the vehicle's motion on the field plane, provided that there is a very accurate localization accuracy of the GPS. The direction quaternion information of the reference trajectory can be obtained by parsing the IMU data. If the IMU used does not provide the corresponding information, the three Euler angles roll, pitch, and yall of the vehicle can also be measured to calculate the direction quaternion indirectly using Eq. 1.

$$q = \begin{bmatrix} cos\frac{\gamma}{2} \\ 0 \\ 0 \\ sin\frac{\gamma}{2} \end{bmatrix} \begin{bmatrix} cos\frac{\beta}{2} \\ 0 \\ sin\frac{\beta}{2} \\ 0 \end{bmatrix} \begin{bmatrix} cos\frac{\alpha}{2} \\ sin\frac{\alpha}{2} \\ 0 \\ 0 \end{bmatrix} \tag{1}$$

where α, β, γ stands for roll, pitch, and yall in Euler angles, respectively. Multiplication of quaternion is defined as Eq. 2.

$$\begin{aligned} pq &= at - \vec{u}\cdot\vec{v} + a\vec{v} + t\vec{u} + \vec{u}\times\vec{v} \\ &= (at - bx - cy - dz) + (ax + bt + cz - dy)i \\ &+ (ay - bz + ct + dx)j + (az + by - cx + dt)k \end{aligned} \tag{2}$$

where $p = t + \vec{v} = t + xi + yj + zk$ and $q = a + \vec{u} = a + bi + cj + dk$. Substituting their into Eq. 1 gives the final calculation as Eq. 3.

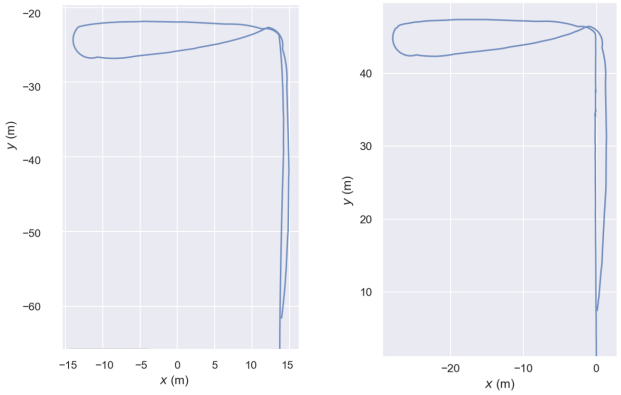
$$q = \begin{bmatrix} cos\frac{\alpha}{2}cos\frac{\beta}{2}cos\frac{\gamma}{2} + sin\frac{\alpha}{2}sin\frac{\beta}{2}sin\frac{\gamma}{2} \\ sin\frac{\alpha}{2}cos\frac{\beta}{2}cos\frac{\gamma}{2} - cos\frac{\alpha}{2}sin\frac{\beta}{2}sin\frac{\gamma}{2} \\ cos\frac{\alpha}{2}sin\frac{\beta}{2}cos\frac{\gamma}{2} + sin\frac{\alpha}{2}cos\frac{\beta}{2}sin\frac{\gamma}{2} \\ cos\frac{\alpha}{2}cos\frac{\beta}{2}sin\frac{\gamma}{2} - sin\frac{\alpha}{2}sin\frac{\beta}{2}cos\frac{\gamma}{2} \end{bmatrix} \tag{3}$$

The above equation is the formula for calculating pose quaternion through Euler angles. Up to this point, all the position and pose data required for trajectory evaluation can be obtained.

B. Trajectory Evaluation

Since the data we obtained from simulation is rich in information, the SLAM algorithm utilizes the simulation data to obtain more detailed mapping results compared to the results obtained from the data collected by the real vehicle. Therefore, the trajectory evaluation of simulation data is used here as an example, although it is also applicable to the trajectory evaluation of real vehicle data. The actual trajectory data of the closed loop of the unmanned vehicle driving along the field

and the trajectory data estimated by the SLAM algorithm are input into evo for visualization, and the results shown in Fig 3 are obtained.



(a) Reference Trajectory (b) Estimated Trajectory

Fig. 3. Visualization results of mapping trajectory.

It can be seen that the trend of the two trajectories is basically the same. The algorithm has some deviation in describing the details of the trajectory. Evo provides two indicators for measuring errors between trajectories, namely absolute pose error and relative pose error. Absolute pose error directly compares the corresponding poses between the estimated trajectory and the reference trajectory, which is generally used to test the global consistency of the trajectories. It is defined by Eq. 4.

$$APE_n = (\frac{1}{n}\sum_{i=1}^{n}||trans(F_i)||^2)^{\frac{1}{2}} \tag{4}$$

Where $trans(F_i)$ refers to the translational components of the relative pose error F_i. When assuming that there is a sequence of poses from the estimated trajectory $\{P_n\}$ and from the ground truth trajectory $\{Q_n\}$, F_i is defined by Eq. 5.

$$F_i = Q_i^{-1}SP_i \tag{5}$$

The relative pose error measures the local accuracy of the trajectory by comparing the pose increment of the estimated trajectory relative to the reference trajectory. It is defined by Eq. 6.

$$RPE_m = (\frac{1}{m}\sum_{i=1}^{m}||trans(E_i)||^2)^{\frac{1}{2}} \tag{6}$$

Given $\{P_n\}$, $\{Q_n\}$ and a fixed time interval Δ, E_i is defined by Eq. 7.

$$E_i = (Q_i^{-1}Q_{i+\Delta})^{-1}(P_i^{-1}P_{i+\Delta}) \tag{7}$$

Regardless of which method is used, it is true that the smaller the error, the higher the prediction accuracy, indicating that the algorithm has stronger mapping and localization capabilities. TABLE I lists the results of the algorithm mapping trajectory evaluation obtained using these two methods respectively.

TABLE I
MAPPING TRAJECTORY EVALUATION RESULTS OF SLAM

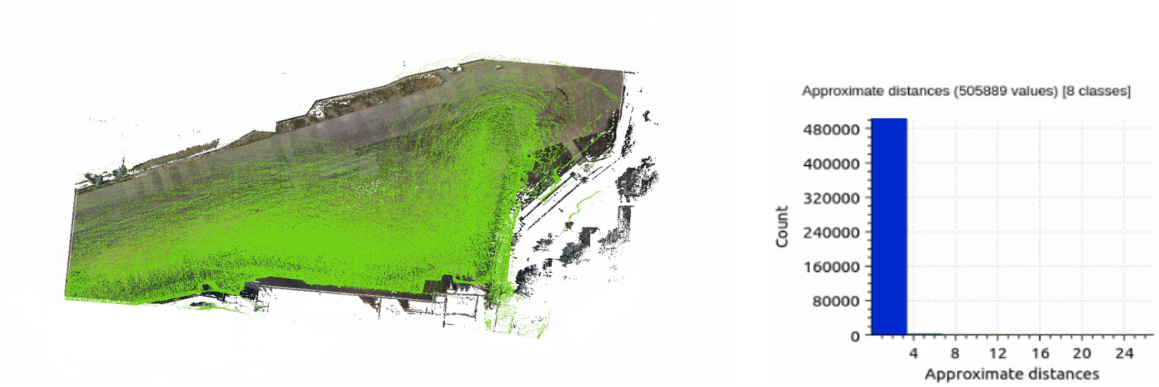
Indicator	max	mean	median	min	rmse	std
APE(m)	37.81	10.56	5.92	0.00	15.05	10.73
RPE(m)	1.53	0.33	0.22	0.00	0.44	0.30

It can be seen that in the global system, the average prediction error of the algorithm is about 10m. The difference between the maximum and minimum values is relatively large, but the minimum value of the error is 0. The local prediction error is much smaller, with a root-mean-square error of about 0.44m and an average deviation of about 0.33m, which indicates that the algorithm has a strong ability to track the trend of the actual trajectory. The local standard deviation is about 0.3m, indicating that the prediction fluctuations are small.

C. Point Cloud Evaluation

Although, it is very difficult to directly compare the maps created by the algorithm with the ground truth as mentioned earlier, some means can still be used to roughly match and compare the two, such as point cloud alignment method. The principle of the point cloud alignment algorithm is to obtain the estimated parameters of the matching model through the least squares solution, and then use the greedy algorithm to find the closest point as the corresponding point, so as to establish the correspondence between the points on the two point cloud maps [17]. Here we use **CloudCompare** software to realize this function [18]. The point cloud of the test field obtained by the high-precision scanner and the point cloud generated by SLAM algorithm are imported into the software. Firstly, the automatic alignment function of the software is used for the preliminary matching of the two point clouds. The matching target RMS difference, iteration number, and final overlap can be set. On the basis of the automatic alignment, manual fine-tuning is carried out according to the edge contours of the two point clouds, so as to realize a fine comparison. After the comparison is completed, the similarity of the shapes of the two point clouds can be observed, as shown in Fig4(a).

It can also be quantified by calculating the distance between the two point clouds. For each point in the compared point cloud, the closest point in the reference point cloud is found to match it. Then the distance between this pair of data points can be calculated. Next, a histogram of the distribution of distances between all pairs of data points is plotted globally, as shown in Fig4(b), which can reflect the distance between the two point clouds. The closer the overall distribution of the distance between the two point clouds is to 0, the higher the overlap between them. The overlap degree reflects the alignment effect of the two point clouds, which can test the similarity of the



(a) Schematic diagram of point cloud alignment

(b) Distribution map of distance between point clouds

Fig. 4. Point cloud alignment results.

two point clouds to some extent. That is to say, it is possible for two point clouds to have a high degree of overlap without a high degree of similarity, so this indicator only serves as a auxiliary validation.

VI. Summary and Future works

In this paper, we design a test and evaluation process for the LiDAR SLAM algorithm based on the combination of reality and virtuality. The process firstly proposes a dataset collection method of the LiDAR SLAM algorithm based on the virtual-reality combination. This is accomplished by running a vehicle on a selected site to collect sensor data, then scanning the entire site map and importing it into the virtual simulation platform. The corresponding sensors are deployed in the simulation platform and the vehicle is controlled to collect data. Two sets of data describing the same object but from different sources are thus constructed. Next, the datasets are fed into the SLAM algorithm and the algorithm is run to get the mapping results. The predicted trajectory from the mapping results are compared with the reference trajectory used as ground truth. The smaller the comparison difference, the more accurate the mapping and localization of the algorithm.

In fact, this processes is not limited to the application of LiDAR SLAM algorithms, but also applicable to visual SLAM algorithms. As future work, on the one hand, we plan to collect more real vehicle data and simulation data in order to further enrich our algorithm evaluation system based on these data. On the other hand, we plan to apply the whole process to more slam algorithms for testing, so as to verify the universality of this evaluation system.

References

[1] Z. Yuan, J. Deng, R. Ming, F. Lang, and X. Yang, "Sr-livo: Lidar-inertial-visual odometry and mapping with sweep reconstruction," *IEEE Robotics and Automation Letters*, 2024.

[2] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[3] T. Shan, B. Englot, D. Meyers, W. Wang, C. Ratti, and D. Rus, "Lio-sam: Tightly-coupled lidar inertial odometry via smoothing and mapping," in *2020 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2020, pp. 5135–5142.

[4] M. Grupp, "evo: Python package for the evaluation of odometry and slam." https://github.com/MichaelGrupp/evo, 2017.

[5] A. Huletski, D. Kartashov, and K. Krinkin, "Evaluation of the modern visual slam methods," in *2015 Artificial Intelligence and Natural Language and Information Extraction, Social Media and Web Search FRUCT Conference (AINL-ISMW FRUCT)*. IEEE, 2015, pp. 19–25.

[6] K. Krinkin, A. Filatov, A. yom Filatov, A. Huletski, and D. Kartashov, "Evaluation of modern laser based indoor slam algorithms," in *2018 22nd Conference of Open Innovations Association (FRUCT)*. IEEE, 2018, pp. 101–106.

[7] M. Filipenko and I. Afanasyev, "Comparison of various slam systems for mobile robot in an indoor environment," in *2018 International Conference on Intelligent Systems (IS)*. IEEE, 2018, pp. 400–407.

[8] L. Jinyu, Y. Bangbang, C. Danpeng, W. Nan, Z. Guofeng, and B. Hujun, "Survey and evaluation of monocular visual-inertial slam algorithms for augmented reality," *Virtual Reality & Intelligent Hardware*, vol. 1, no. 4, pp. 386–410, 2019.

[9] S. Zhang, L. Zheng, and W. Tao, "Survey and evaluation of rgb-d slam," *IEEE Access*, vol. 9, pp. 21 367–21 387, 2021.

[10] J. Wu, S. Huang, Y. Yang, and B. Zhang, "Evaluation of 3d lidar slam algorithms based on the kitti dataset," *The Journal of Supercomputing*, vol. 79, no. 14, pp. 15 760–15 772, 2023.

[11] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of rgb-d slam systems," in *2012 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 2012, pp. 573–580.

[12] A. Filatov, A. Filatov, K. Krinkin, B. Chen, and D. Molodan, "2d slam quality evaluation methods," in *2017 21st Conference of Open Innovations Association (FRUCT)*. IEEE, 2017, pp. 120–126.

[13] T. Schops, T. Sattler, and M. Pollefeys, "Bad slam: Bundle adjusted direct rgb-d slam," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 134–144.

[14] M. Bujanca, P. Gafton, S. Saeedi, A. Nisbet, B. Bodin, M. F. O'Boyle, A. J. Davison, P. H. Kelly, G. Riley, B. Lennox *et al.*, "Slambench 3.0: Systematic automated reproducible evaluation of slam systems for robot vision challenges and scene understanding," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 6351–6358.

[15] A. M. Bettens, B. Morrell, M. Coen, N. McHenry, X. Wu, P. Gibbens, and G. Chamitoff, "Unrealnavigation: Simulation software for testing slam in virtual reality," in *AIAA Scitech 2020 Forum*, 2020, p. 1343.

[16] Ł. Sobczak, K. Filus, A. Domański, and J. Domańska, "Lidar point cloud generation for slam algorithm evaluation," *Sensors*, vol. 21, no. 10, p. 3313, 2021.

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

[18] D. Girardeau-Montaut, "Cloudcompare," *France: EDF R&D Telecom ParisTech*, vol. 11, no. 5, 2016.