Hindawi
Journal of Sensors
Volume 2022, Article ID 7213044, 17 pages
https://doi.org/10.1155/2022/7213044

Research Article

A SLAM Algorithm Based on Edge-Cloud Collaborative Computing

Taizhi Lv ,[1,2] Juan Zhang ,[1] and Yong Chen [2]

[1]*School of Information Technology, Jiangsu Maritime Institute, Nanjing 211170, China*
[2]*Nanjing Longyuan Microelectronic Company Limited, Nanjing 211106, China*

Correspondence should be addressed to Juan Zhang; 404050569@qq.com

Received 6 April 2022; Revised 13 September 2022; Accepted 28 October 2022; Published 12 November 2022

Academic Editor: Yunchao Tang

Simultaneous localization and mapping (SLAM) is a typical computing-intensive task. Based on its own computing power, a mobile robot has difficult meeting the real-time performance and accuracy requirements for the SLAM process at the same time. Benefiting from the rapid growth of the network data transmission rate, cloud computing technology begins to be applied in the robotics. There is the reliability problem caused by solely relying on cloud computing. To compensate for the insufficient airborne capacity, ensure the real-time performance and reliability, and improve the accuracy, a SLAM algorithm based on edge-cloud collaborative computing is proposed. The edge estimates the mobile robot pose and the local map using a square root unscented Kalman filter (SR-UKF). The cloud estimates the mobile robot pose and the global map using a distributed square root unscented particle filter (DSR-UPF). By using sufficient particles in the cloud, DSR-UPF can improve the SLAM accuracy. The cloud returns the particle with the largest posteriori probability to the edge, and the edge performs edge-cloud data fusion based on probability. Both the simulation and the experimental results show that the proposed algorithm can improve the estimation accuracy and reduce the execution time at the same time. By transferring the heavy computation from robots to the cloud, it can enhance the environmental adaptability of mobile robots.

1. Introduction

Since the beginning of the 21st century, autonomous navigation technology has received increasing attention [1, 2]. It is among the disruptive technologies predicted by academic institutions and groups, such as the McKinsey Global Institute, Citi GPS MIT, and the ARK investment management company [3, 4]. Simultaneous localization and mapping (SLAM) is the key to realize autonomous navigation, and it has been widely used in unmanned vehicles, unmanned underwater vehicles, unmanned aerial vehicles, augmented reality, and in other fields.

The traditional SLAM algorithms rely on the mobile computing resource carried by a robot to estimate the environmental map and the robot pose. SLAM is a computing-intensive task, and the limited computing power makes mobile robots difficult to meet the real-time requirement. Benefiting from the rapid development of wireless communication technology, cloud technologies have been applied in robotics. The cloud robot was first proposed by Kuffner [5]. It combines cloud computing technology and robotics and has become a trending research direction [6]. A growing number of cloud platforms, applications, and services have been proposed for robotics. DaVinci, Rapyuta, RoboEarth, sensor-cloud, etc. transfer the heavy computation from robots to the safe computing environment in cloud [7, 8]. The humanoid robot developed by the Aldebaran Robotics relies on the cloud platform to realize speech recognition, face detection, and video acquisition [9]. A fast object capture method based on Dex-Net as a Service (DNaaS) is proposed and verified by the PR2 robot [10]. Pandey et al. designed a cloud underwater robot to collect and monitor marine data, and it completed tasks by sharing local and remote resources [11].

These above studies have proven the practicability and validity of cloud robotics. As a result, new solutions for SLAM have emerged. These solutions divide the SLAM process into two stages: data acquisition in the robot and data computing in the cloud. Arumugam et al. adopted a Hadoop cluster to boost the particle sampling of the FastSLAM algorithm [12]. This algorithm can improve the efficiency of map construction and shares data with other robots using the software as a service (SaaS) model. However, Hadoop is designed for batch processing and not essentially designed for real-time performance [13]. Its poor real-time performance limits its application in online SLAM environments. Kamburugamuve et al. used a distributed framework to implement the GMapping algorithm [14]. This framework is based on the IoTCloud platform which connects smart devices to cloud services for real-time data processing [15]. Lidar and inertial sensor data are sent to the cloud as stream events. The cloud executes SLAM computing, and the results of robot position and mapping are returned to the robot. IoTCloud is designed to make a connection between the Internet of Things (IoT) devices and the cloud, and its application to the real-time SLAM needs to be verified. The fault tolerance of this framework also should be improved. By the RoboEarth platform, C2TAM transfers the SLAM computing from the robot to the cloud. It uses the parallel tracking and mapping (PTAM) algorithm to achieve multirobot cooperative tracking and map creation tasks [16]. The limitation is that the high requirement for bandwidth is needed, and it is difficult to apply to other sensing environments, such as laser radar. Xu and Bian designed a cloud robotic application platform based on the container technology. The SLAM computation as microservices deployed in the container is provided [17]. The real-time and stability of this method is needed further verification. Liu et al. designed an RGB-D SLAM method based on the cloud computing [18]. It executes mapping, location, map fusion, and loop detection in the cloud. However, the feature detection, pose estimation, and 3D point cloud stitching are still executed in the robot, and the computational load of the robot is still heavy.

To overcome the disadvantage of insufficient capacity and the overburdening in the SLAM process of mobile robots, a SLAM algorithm based on edge-cloud collaborative computing is proposed. The cloud executes a distributed square root unscented particle filter (DSR-UPF) to build the global map by stream computing. For meeting the real-time requirement, the edge builds a local map through a square root unscented Kalman filter (SR-UKF) to implement the fast SLAM computing. The main contributions of this paper are stated as follows:

(1) Comparing the traditional cloud-based SLAM algorithms, the proposed algorithm not only ensure the real-time performance but also improves the accuracy and efficiency by the edge-cloud collaborative architecture

(2) Comparing the batch process of traditional cloud-based SLAM algorithms, stream computing with high real-time and low latency is used to construct the cloud computing framework. By distribute parallel computing, DSR-UPF can takes advantage of its sufficient power to improve the execution performance, and the accuracy is improved by the sufficient particles. To overcome the drawback of linear approximations to the motion function in traditional particle filter SLAM algorithms, each particle uses SR-UPF to samples particle in a highly accurate manner

(3) In EKF-SLAM, the computation increases exponentially with the increase of landmarks. To ensure the real-time performance of the edge, SR-UKF is used to build the local map. By only dealing with the recently observed landmarks, it reduces the computation

2. Background

2.1. SLAM Problem. SLAM is a process that enables a mobile robot to build a map and locate itself at the same time in an unknown environment. The goal of SLAM is to obtain the best estimation of the robot pose and the map based on the control data $u_{1:t}$ and observation data $z_{1:t}$ from the start to time t. It can be described by a posteriori probability distribution as follows:

$$p(x_t, \Theta | z_{1:t}, u_{1:t}), \tag{1}$$

where x_t is the robot pose at time t and Θ is the entire map.

According to the Bayes rule, the law of total probability, and the Markov hypothesis, the posteriori probability distribution of the robot pose and the environmental map can be rewritten as follows:

$$\begin{gathered}\underbrace{p(x_t, \Theta | z_{1:t}, u_{1:t})}_{\text{The posterior distribution at time } t} \propto \underbrace{\eta p(z_t | x_t)}_{\text{The observation model}}, \\ \int \underbrace{p(x_t | x_{t-1}, u_t)}_{\text{The motion model}} \underbrace{p(x_{t-1}, \Theta\theta | z_{1:t-1}, u_{1:t-1})}_{\text{The posterior distribution at time } t-1} dx_{t-1}.\end{gathered} \tag{2}$$

In general, the integral in Eq. (2) cannot be evaluated in a closed form. The extended Kalman filter (EKF) and the particle filter (PF) are simple approximations of the general Bayes filter.

2.2. PF-SLAM. The EKF-SLAM (extended Kalman filter SLAM) and the PF-SLAM (particle filter SLAM) are two widely used solutions to the SLAM problem.

For PF-SLAM, the greater the particle number, the more accuracy SLAM can obtain. More particles mean more computation, and the traditional algorithm relying only on the robot's own resources cannot meet the real-time requirements. In reference [19], simulation experiments with 100, 300, and 500 particles are executed. The experimental results show that 500 particles obtained a more accurate estimation of the environment map and the robot pose than 100 particles and 300 particles. In order to control the particle number and improve the accuracy simultaneously, scholars have improved and optimized the process of the particle sampling and resampling, but the problem of massive computation of the SLAM algorithm based on particle filter has not been fundamentally solved [20–22].

2.3. Stream Computing. The traditional data processing architecture is mainly aimed at transactional data processing scenarios with low real-time requirements. It is difficult to apply in scenarios that have high real-time requirements. Stream computing uses distributed ideas and methods to process data in real-time. Stream computing can provide huge computing power for mobile robots to process data to ensure that low-cost robots can make more complex sensing data processes and decision-making [23].

3. The SLAM Architecture Based on Edge-Cloud Collaborative Computing

The system architecture of the proposed SLAM algorithm is shown in Figure 1. It is composed of the edge and the cloud. The edge is the mobile robot, and the cloud is a stream computing cluster.

The edge includes five modules: data acquisition, message client, local SLAM computing, edge-cloud fusion, and local storage. The data acquisition module is used to acquire the lidar data and the inertial navigation data and format them into a unified form. The message client realizes sealing, sending, receiving, and unsealing of messages. The local SLAM computing module performs a real-time SLAM computing to obtain the estimation of the mobile robot pose and the local map. The local storage module stores the map information in a certain time. Based on the probability distribution, the edge-cloud fusion module fuses the SLAM computing results from the edge and the cloud.

The cloud includes four modules: message server, coordination monitoring, distributed SLAM computing, and global storage. The message server realizes a high availability and load balancing message service by a Kafka cluster which completes a reliable communication between the cloud and mobile robots. The Hadoop distributed file system (HDFS) is used to store the global map, sensor data, path information, particle information, etc. The coordination and monitoring module realizes the coordination, monitoring, and management between the control node and the N computing nodes. As a distributed process coordination, zookeeper coordinates the SLAM computing actions and makes the distributed SLAM more reliable. The distributed SLAM computing module is based on the Flink stream computing platform.

4. The SLAM Process Based on Edge-Cloud Collaborative Computing

The process flow of the proposed algorithm is shown in Figure 2. In each control decision cycle, the acquired data is transferred to the edge and the cloud, which execute SLAM computing at the same time. In the edge, the robot pose and the local map are computed by the SR-UKF. The cloud performs loop detection, distributed SLAM computing, and priori knowledge correction in parallel. The distributed SLAM uses the DSR-UPF, and each particle is calculated separately. The particle with the largest posteriori probability is returned to the edge. The edge realizes the fusion between the edge SLAM result and the cloud SLAM result. The edge updates the state of the mobile robot and saves the local map. Finally, the cloud executes resampling according to the particle diversity and saves the global map information to the HDFS and the cache.

4.1. Process Flow of the Edge. The process flow of the edge is mainly divided into six steps: data acquisition, message transmission, edge SLAM, message reception, fusion of the edge and cloud SLAM results, and local map storage.

4.1.1. Data Acquisition. The mobile robot acquires the control data from the inertial navigation system (INS) and the observed data from the lidar.

4.1.2. Message Transmission. The mobile robot encapsulates the control data and the observed data in a control cycle into a message. Each message is marked with a unique sequence number, and this unique number is also encapsulated to the reply message from the cloud. It is used to identify an interactive process between the edge and the cloud. The Kafka client transmits this message to the cloud by the communication network.

4.1.3. The Edge SLAM Based on the SR-UKF. The SR-UKF (square root unscented Kalman filter) SLAM algorithm is the improvement of the EKF-SLAM which is the process of iterative updating for the system state and the covariance matrix. With the increase of landmarks, the system state vector and covariance matrix become larger, and the computation increase dramatically. In order to ensure the real-time performance, the edge uses the SR-UKF algorithm to build the local map, and only the recently observed landmarks are used for the map updating. It avoids the covariance matrix being too large and reduces the computation. Compared with EKF, SR-UKF avoids the linearization model error and improves the system accuracy.

The SR-UKF constructs the statistical characteristics of the system by $2L+1$ sigma points, which can characterize the system state vector. L is the dimension of the system state. First, the SR-UKF initializes the system mean, the square root of the covariance matrix, and the weights of sigma point as follows:

$$X_0 = x_{\text{init}} S_0 = \text{chol}(Q_{\text{init}}), \tag{3}$$

$$\omega_0^m = \frac{\lambda}{L+\lambda} \quad \omega_0^c = \frac{\lambda}{L+\lambda} + \left(1-\alpha^2+\beta\right), \tag{4}$$

$$\omega_i^m = \omega_i^c = \frac{1}{2(L+\lambda)} \qquad i = 1, 2 \cdots \cdots L+1 \cdots \cdots 2L, \tag{5}$$

where x_{init} and Q_{init} are the mean and the covariance of the initial state. chol is a Cholesky factorization that decomposes a symmetric, positive-definite matrix into the product of a lower-triangular matrix and its transpose [24]. $\lambda = \alpha^2(L+\kappa) - L$ is the scaling parameter. α determines the spread of the sigma points around the system mean, and it is usually set to $1e-4 \le \alpha \le 1$. β is used to incorporate the prior

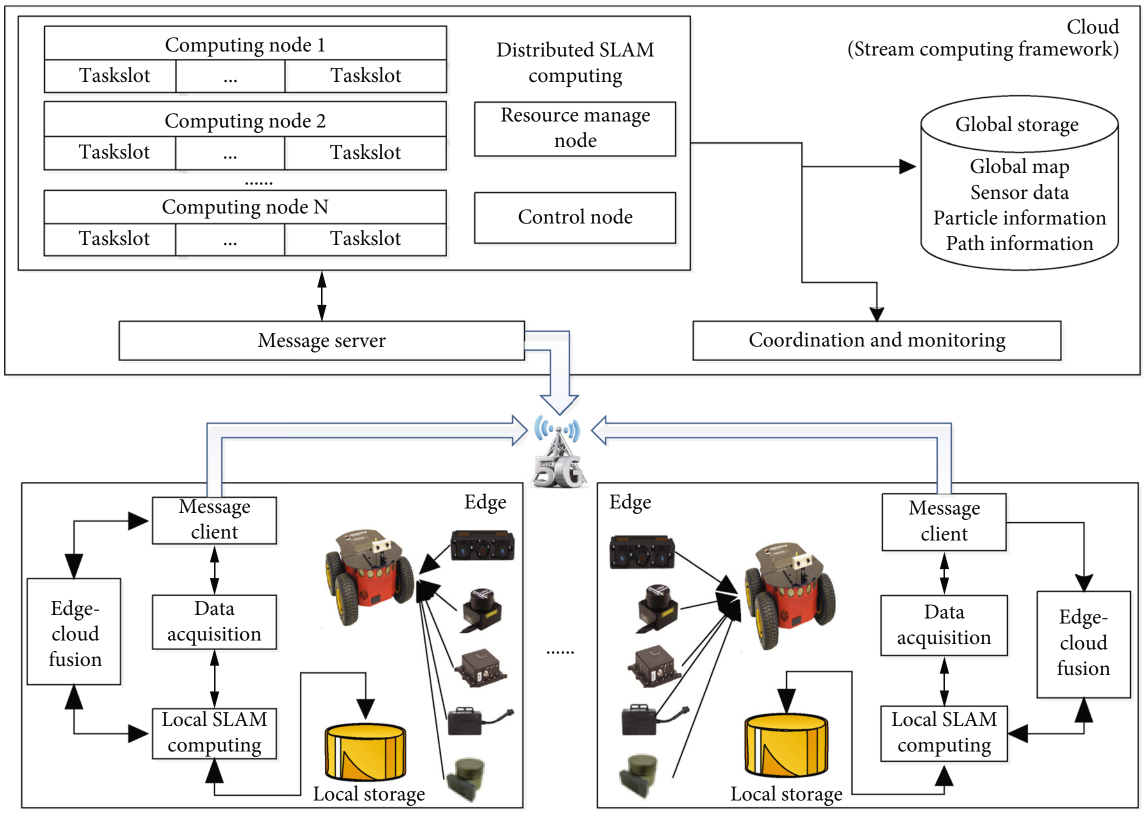


Figure 1: The SLAM system architecture based on edge-cloud collaborative computing.

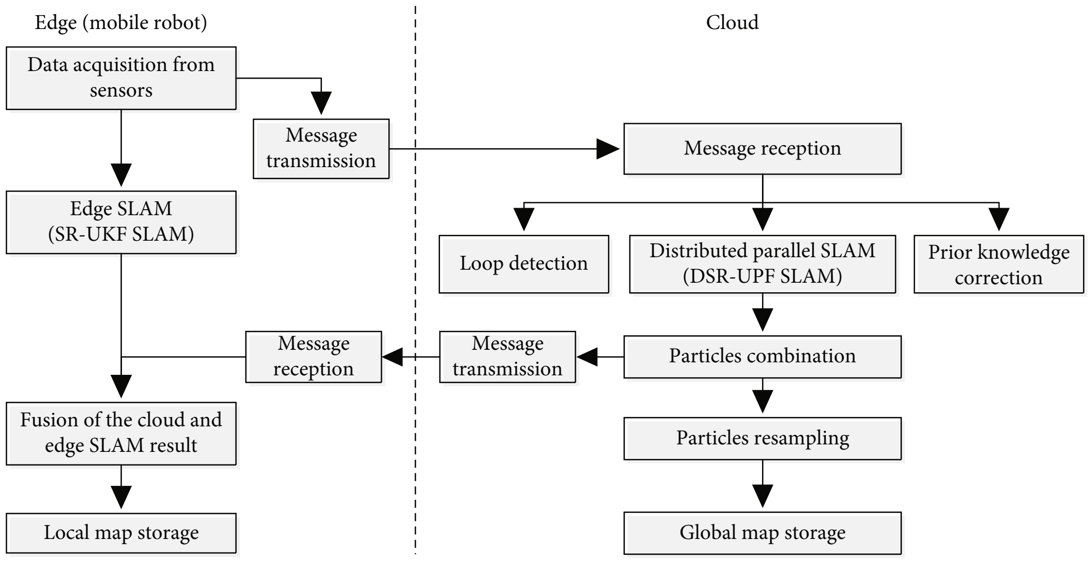


Figure 2: The flow of the SLAM process based on edge-cloud collaborative computing.

distribution of the state variable, and it is set to 2 for a Gaussian distribution. κ is a secondary scaling factor, and it is set to 0. ω_i^m is the mean weight of the i-th sigma point, and ω_i^c is the covariance weight.

At time t, the mean and the square root of the covariance matrix of the system state are x_{t-1} and S_{t-1}. The iterative update of x_{t-1} and S_{t-1} is by sigma point calculation, prediction, and observation update.

(1) Calculate the Sigma Point.

$$X_{t-1} = [x_{t-1} x_{t-1} + \eta S_{t-1} x_{t-1} - \eta S_{t-1}], \quad (6)$$

where the scale η is calculated as follows:

$$\eta = \sqrt{L + \lambda}. \quad (7)$$

(2) Prediction. The Sigma point at time t is calculated as follows:

$$X_t^* = g(X_{t-1}, u_t), \tag{8}$$

where g is the motion model function and u_t is the control data.

The predicted mean and the square root of covariance matrix are calculated by Eqs. (9)–(12). $\widehat{S}_t$ is calculated by a QR decomposition, which decomposes a matrix into a normal orthogonal matrix Q and an upper triangular matrix R. To ensure the semipositive of the square root of the covariance matrix, the Cholesky update (cholupdate) function [25] is used to update $\widehat{S}_t$.

$$\widehat{x}_t = \sum_{i=0}^{2L} \omega_i^m X_{t,i}^*, \tag{9}$$

$$\widehat{S}_t = \text{qr}\left\{ \left[\sqrt{\omega_1^c} \right] \left(X_{t,1:2L}^* - \widehat{x}_t \right) \sqrt{Q} \right\}, \tag{10}$$

$$\widehat{S}_t = \text{cholupdate}\left\{ \widehat{S}_t, X_{t,0}^* - \widehat{x}_t, \omega_o^c \right\}, \tag{11}$$

where qr is to denote a QR decomposition of a matrix. The sigma point is updated as follows:

$$\widehat{X}_t = \left[\widehat{x}_t \quad \widehat{x}_t + \eta \widehat{S}_t \quad \widehat{x}_t - \eta \widehat{S}_t \right]. \tag{12}$$

(3) Update by Observation. By using the observation model, the landmark observation at time t is predicted as follows:

$$\widehat{Z}_t = h(X_t), \tag{13}$$

$$\widehat{z}_t = \sum_{i=0}^{2L} \omega_i^m \widehat{Z}_{t,i}. \tag{14}$$

QR decomposition and cholupdate are used to update $\widehat{S}_{Z_t}$ as follows:

$$\widehat{S}_{Z_t} = qr\left\{ \left[\sqrt{\omega_1^c} \left(\widehat{Z}_{t,1:2L} - \widehat{z}_t \right) \sqrt{R} \right] \right\}, \tag{15}$$

$$\widehat{S}_{Z_t} = \text{cholupdate}\left\{ \widehat{S}_{Z_t}, \widehat{Z}_{t,0} - \widehat{z}_t, \omega_0^c \right\}. \tag{16}$$

The Kalman gain is calculated as follows:

$$P_{x_t z_t} = \sum_{i=0}^{2L} \omega_i^c (X_{t,i} - \widehat{x}_t)\left(\widehat{Z}_{t,i} - \widehat{z}_t\right)^T, \tag{17}$$

$$K_t = \frac{P_{x_t z_t} / \widehat{S}_{Z_t}^T}{\widehat{S}_{Z_t}}. \tag{18}$$

Finally, the mean and the square root of the covariance matrix are updated by the Kalman gain.

$$x_t = \widehat{x}_t + K_t (Z_t - \widehat{z}_t), \tag{19}$$

$$U = K_t \widehat{S}_{Z_t}, \tag{20}$$

$$S_t = \text{cholupdate}\left(\widehat{S}_t, U, -1\right). \tag{21}$$

4.1.4. Message Reception. The edge receives messages by the Kafka client. To ensure the system reliability, the exact one mode is adopted to ensure that message processing and submission feedback are in the same transaction or atomicity.

4.1.5. Local Map Storage. The local map and the robot position from the result of the edge-cloud fusion are stored in the edge.

4.2. Process Flow of the Cloud. The process flow of the cloud is mainly divided into six steps: message reception, parallel computing, particle combination, computing result transmission, resampling, and global map storage.

4.2.1. Message Reception. Based on the Kafka cluster, the control and the observation data from the mobile robot are received. The data are distributed to the loop detection, the distributed SLAM, and the priori knowledge correction modules for concurrent execution.

4.2.2. Parallel Computing

(1) Loop Detection. Loop detection compares the observation information in all periods stored in the cloud. The scan-to-scan method is used to loop detection, which matches two frames of lidar data. The Iterative Closest Point (ICP) algorithm is used to match two frames by calculating the rigidity change between two pairs of point clouds [26]. If the lasted received data are highly similar to the data at a certain time in history, a closed loop is formed. With the closed loop, the cumulative error is reduced.

(2) DSR-UPF SLAM. The DSR-UPF (distributed square root unscented particle filter) SLAM is a distributed parallel PF-SLAM algorithm. In the PF-SLAM, the particle number determines the algorithm accuracy. More particles can bring not only the higher accuracy but also the more computation. In order to improve the accuracy, the cloud adopts enough particles based on the cloud computing power, and it executes the PF-SLAM algorithm in a distributed manner. In the DSR-UPF SLAM, each particle is independently associated with the estimation of N landmarks. At time t, the particle set is shown in Figure 3. $x_t^{[i]}$ is the estimated pose of the i -th particle, and $\omega_t^{[i]}$ is the weight of this particle. $(\mu_t^{[i,j]}, \sum_t^{[i,j]})$ represents the mean and the covariance matrix of the j-th landmark estimation of the i-th particle, respectively.

As shown in Figure 4, each particle is estimated independently.

Each computing node predicts and updates the new pose by the SR-UKF, and it updates the landmark position by the extended Kalman filter (EKF). Landmark updating depends on whether the landmark is observed at time t. If the landmark is newly observed, its mean and covariance are initialized as

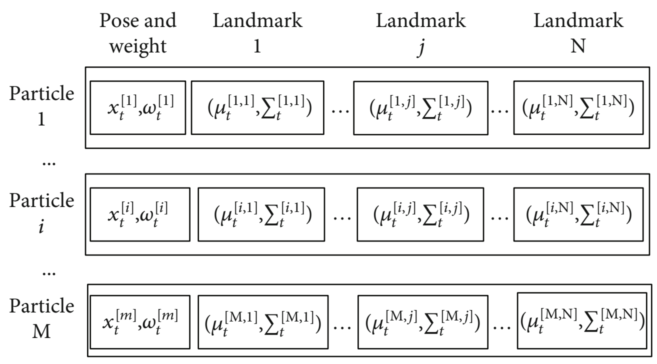

Pose and weight
Landmark 1
Landmark j
Landmark N
Particle 1
$x_t^{[1]}, \omega_t^{[1]}$
$(\mu_t^{[1,1]}, \Sigma_t^{[1,1]})$
$(\mu_t^{[1,j]}, \Sigma_t^{[1,j]})$
$(\mu_t^{[1,N]}, \Sigma_t^{[1,N]})$
Particle i
$x_t^{[i]}, \omega_t^{[i]}$
$(\mu_t^{[i,1]}, \Sigma_t^{[i,1]})$
$(\mu_t^{[i,j]}, \Sigma_t^{[i,j]})$
$(\mu_t^{[i,N]}, \Sigma_t^{[i,N]})$
Particle M
$x_t^{[m]}, \omega_t^{[m]}$
$(\mu_t^{[M,1]}, \Sigma_t^{[M,1]})$
$(\mu_t^{[M,j]}, \Sigma_t^{[M,j]})$
$(\mu_t^{[M,N]}, \Sigma_t^{[M,N]})$

FIGURE 3: The particle set.

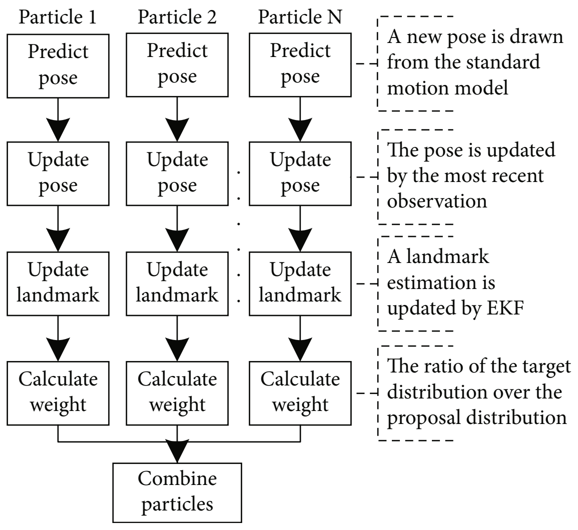

Particle 1
Particle 2
Particle N
Predict pose
Update pose
Update landmark
Calculate weight
Combine particles
A new pose is drawn from the standard motion model
The pose is updated by the most recent observation
A landmark estimation is updated by EKF
The ratio of the target distribution over the proposal distribution

FIGURE 4: The flow of DSR-UPF.

```
Input: M initial particles
Step:
1. particlesM1=SelectParticles(particles, $P_s$);
2. particlesM2=FastMutation(particlesM1, $P_m$);
3. particlesM3=Crossover(particlesM1, $P_c$);
4. newParticles=particlesM1+particlesM2+particlesM3;
Output: M new particles
```

ALGORITHM 1: Improved genetic resampling method.

follows:

$$\mu_t^{[i,N+1]} = h^{-1}\left(z_t, x_t^{[i]}\right), \tag{22}$$

$$\sum_t^{[i,N+1]} = H_{rz} R_t H_{rz}^T, \tag{23}$$

where h^{-1} is the inverse function of the observation model and H_{rz} is the Jacobian matrix of function h^{-1}. If the j-th landmark is not observed at time t, its mean and covariance remain unchanged. If the j-th landmark is observed at time t, the mean and the covariance are updated as follows:

$$\mu_t^{[i,j]} = \mu_{t-1}^{[i,j]} + \mathrm{K}_t\left(z_t - \hat{z}_t^{[i]}\right), \tag{24}$$

$$\sum_t^{[i,j]} = (I - K_t H_z) \sum_{t-1}^{[i,j]}, \tag{25}$$

where K_t is the Kalman gain and H_z is the Jacobian matrix of the observation model.

The particle weight is defined as the ratio of the target distribution over the proposal distribution. The i-th particle weight is defined as follows:

$$\omega_t^{[i]} = \frac{\text{Target distribution}}{\text{Proposal distribution}} = \frac{p\left(x_{1:t}^{[i]} | u_{1:t}, z_{1:t}\right)}{q\left(x_{1:t}^{[i]} | u_{1:t}, z_{1:t}\right)}. \tag{26}$$

The proposal distribution $q(x_{1:t}^{[i]} | u_{1:t}, z_{1:t})$ can be represented by a recursive form as follows:

$$q\left(x_{1:t}^{[i]} | u_{1:t}, z_{1:t}\right) = q\left(x_t^{[i]} \middle| x_{t-1}^{[i]}, u_t, z_t\right) q\left(x_{t-1}^{[i]} | u_{1:t-1}, z_{1:t-1}\right). \tag{27}$$

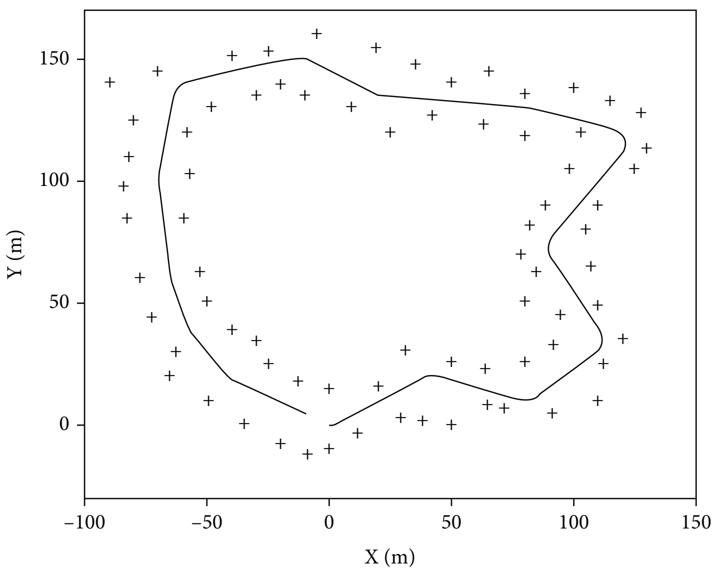
The Bayes rule is used to calculate the weight as follows:

$$\begin{aligned} \omega_t^{[i]} &\propto \frac{p\left(z_t \middle| x_t^{[i]}\right) p\left(x_t^{[i]} \middle| x_{t-1}^{[i]}, u_t\right) p\left(x_{t-1}^{[i]} | u_{1:t-1}, z_{1:t-1}\right)}{q\left(x_t^{[i]} \middle| x_{t-1}^{[i]}, u_t, z_t\right) q\left(x_{t-1}^{[i]} | u_{1:t-1}, z_{1:t-1}\right)} \\ &= \omega_{t-1}^{[i]} \frac{p\left(z_t \middle| x_t^{[i]}\right) p\left(x_t^{[i]} \middle| x_{t-1}^{[i]}, u_t\right)}{q\left(x_t^{[i]} \middle| x_{t-1}^{[i]}, u_t, z_t\right)}, \end{aligned} \tag{28}$$

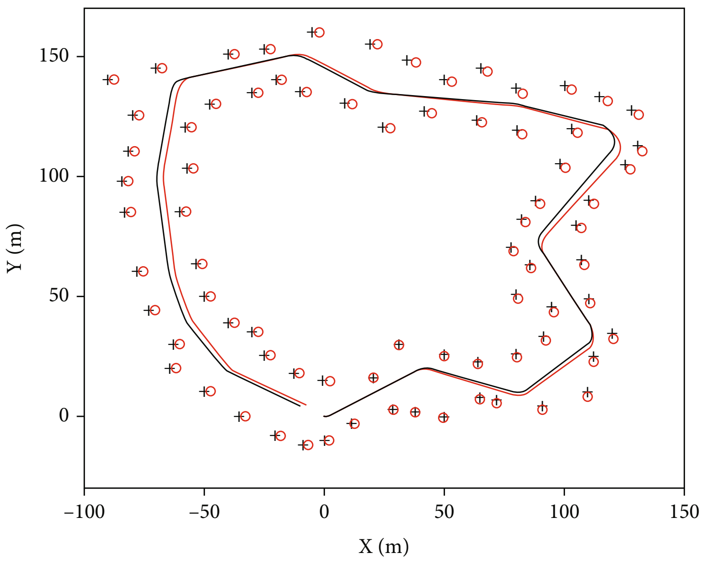
where the proposal distribution $q(x_t^{[i]} | x_{t-1}^{[i]}, u_t, z_t)$ is calculated as follows:

$$q\left(x_t^{[i]} \middle| x_{t-1}^{[i]}, u_t, z_t\right) = p\left(x_t^{[i]} \middle| x_{t-1}^{[i]}, u_t, z_t\right). \tag{29}$$

(3) Prior Knowledge Correction. Most SLAM algorithms assume that the control noise statistics Q and observation

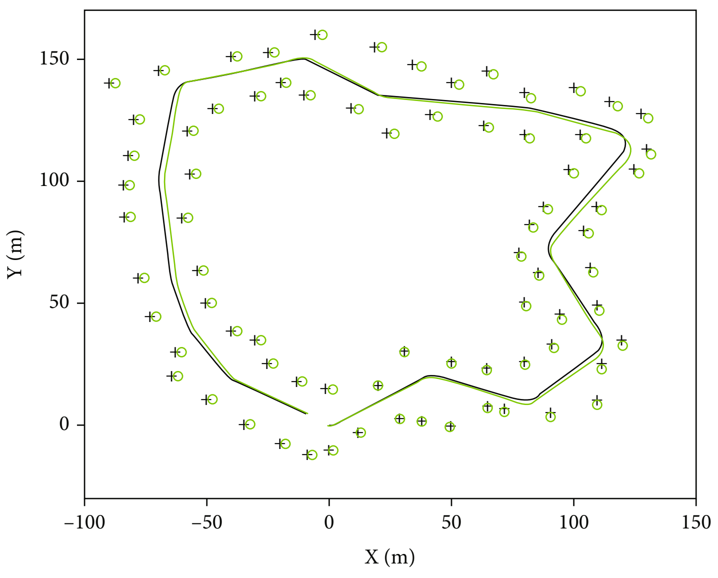


(a) Environmental map

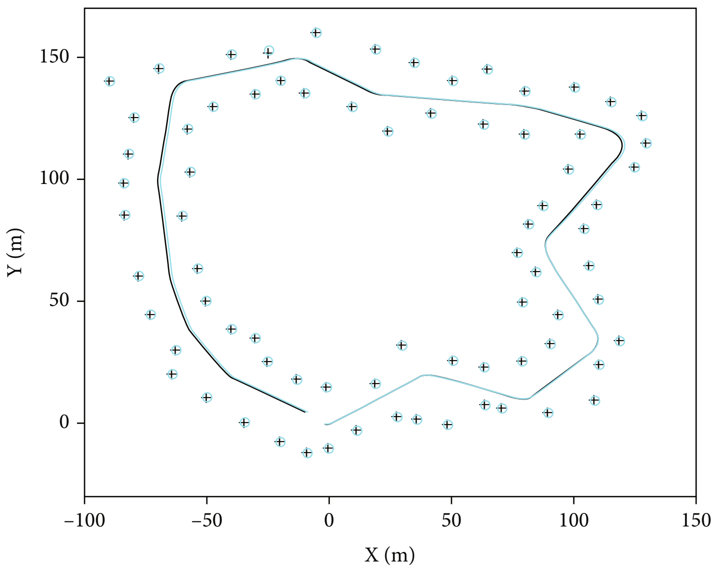


(b) FastSLAM 2.0

Figure 5: Continued.



(c) UFastSLAM



(d) The proposed algorithm

FIGURE 5: A comparison between the three algorithms in the simulation environment.

TABLE 1: The comparisons between the three algorithms.

Algorithms	RMSE_P (m)	RMSE_L (m)	Execution time (s)
FastSLAM 2.0	2.70	2.42	22.46
UFastSLAM	1.55	1.37	29.52
The proposed algorithm	1.37	1.21	19.28

noise statistics R are completely known and correct. Because of the complexity of the real world, this assumption is hard to be tenable. Incorrect a priori knowledge about the control and the observation noise matrices can seriously degrade the accuracy of these algorithms [27–30]. Based on the previous research [31], a dynamic fractional order and alpha stable distribution particle swarm optimization method is adopted, and the prior knowledge Q and R are adjusted dynamically by a fitness function. This fitness function is based on the inconsistency between the predicted observations and the observations. By introducing the prior knowledge correction step, it can reflect the real values of the prior knowledge more accurately.

4.2.3. Particle Combination. After all particles are updated, the effective particle number N_{eff} is calculated as follows:

$$N_{\text{eff}} = \frac{1}{\sum_{i=1}^{M} \left(\omega_t^{[i]}\right)^2}, \tag{30}$$

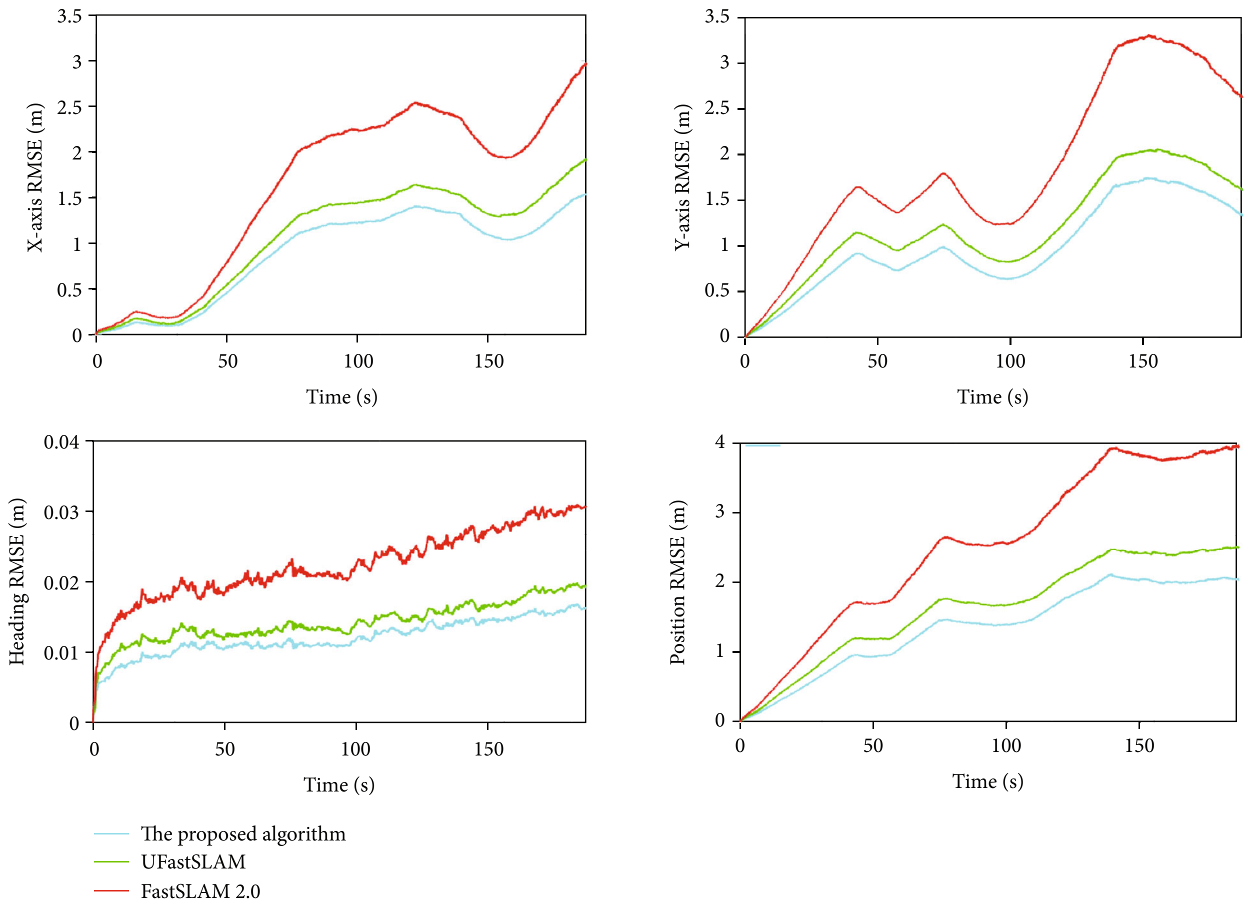


Figure 6: The RMSE comparisons on the x-axis, the y-axis, the heading, and the position.

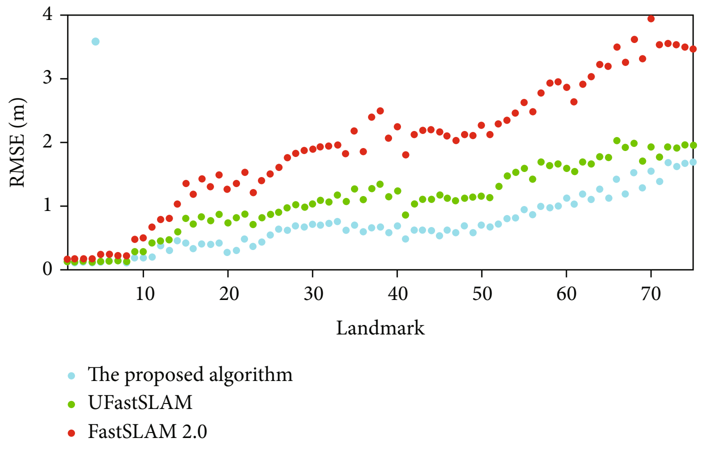


Figure 7: The RMSE comparison on landmark position.

where M is the number of particles, and $\omega_t^{[i]}$ is the normalized weight of the i-th particle. If N_{eff} is less than the specified threshold, the particle set is resampled. After resampling, all particle weights are reset as follows:

$$\omega_t^{[i]} = \frac{1}{M} \tag{31}$$

4.2.4. Computing Result Transmission. The particle with the largest weight is returned to the edge.

4.2.5. Particle Resampling. The SLAM algorithms based on particle filter will degenerate over time no matter how many landmarks in the environment and particles are used in the estimation [32]. Resampling aims to amend the particle degeneracy, but it always leads to the loss of the particle

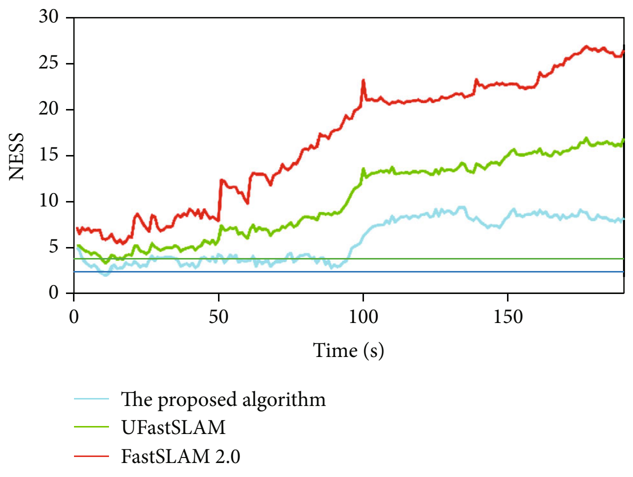


Figure 8: The average NEES comparison.

diversity, which is called particle depletion [33]. To improve the diversity of particles as much as possible, some biological evolutionary algorithms have been applied for resampling [34, 35]. To prevent particle degeneracy and improve particle diversity, an improved genetic resampling method is proposed. It uses a double roulette wheel as the selection operation and a fast metropolis Hastings (FMH) mutation as the mutation operation. The FMH mutation can improve the divergence problem of the traditional mutation methods and produce particles that can better reflect the target distribution [36]. The main procedures of particle resampling are listed as follows:

The SelectParticles method uses the double roulette selection to select $M1$ ($M1 = M \times Ps$) parent particles from M initial particles. The FastMutation method uses the FMH mutation to generate $M2$ particles from the $M1$ parent particles. The Crossover method uses the crossover operation to generate $M3$ particles from $M1$ parent particles.

P_s, P_c, and P_m represent the probability of the selection, crossover, and mutation, respectively. P_s is set to 0.5, and the values of P_c and P_m are determined by the particle diversity V_d as follows:

$$V_d = \frac{H_t}{H_{\text{best}}}, \tag{32}$$

$$H_t = \sum_{i=1}^{M} f_2\left(x_t^{[i]}\right), \tag{33}$$

$$f_2\left(x_t^{[i]}\right) = \sum_{j=1}^{M}\left(\sum_{j=1}^{M}\left|x_t^{[i,k]} - x_t^{[j,k]}\right|\right), \tag{34}$$

where the function f_2 is to calculate the sum of the distances between the i-th particle and the other particles. H_{best} represents the maximum value of V_d from the beginning. P_c and P_m are calculated as follows:

$$\begin{cases} P_c = 0.8, P_m = 0.2 & \frac{3}{4} < V_d \le 1, \\ P_c = 0.6, P_m = 0.4 & \frac{1}{2} < V_d \le \frac{3}{4}, \\ P_c = 0.4, P_m = 0.6 & \frac{1}{4} < V_d \le \frac{1}{2}, \\ P_c = 0.3, P_m = 0.7 & 0 < V_d \le \frac{1}{4}. \end{cases} \tag{35}$$

4.2.6. Global Map Storage. The global map, sensor information, and all particles information are stored the distributed cache and the HDFS.

4.3. Process Flow of the Edge-Cloud Fusion. The edge-cloud fusion is the combination of the edge results and the cloud results. To improve the reliability and overcome the problems caused by transmission delay, the edge-cloud fusion depends on the data transmission time. If the cloud result is not returned to the edge in real-time, the edge-cloud fusion is no longer executed in the edge, and the edge result is directly used as the SLAM result.

Based on the fusion principle of the posterior probability function ratio, the fusion of SLAM results from the cloud and the edge is realized. The posteriori probability of the edge SLAM is shown as follows.

$$\begin{gathered} y_{lt} = p(z_t|x_{lt}, \Theta_{lt})p(x_{lt}|u_t, \bar{x}_{t-1}) = , \\ \prod_{j=1}^{N} p(z_t|x_{lt}, m_{lj})p(x_{lt}|u_t, \bar{x}_{t-1}) = , \\ \prod_{j=1}^{N} \frac{1}{\sqrt{2\pi R_t(m_j)}} e^{-(z_t(m_j)-\hat{z}_t(m_{lj}))^2/2R_t(m_j)} \frac{1}{\sqrt{2\pi Q_t}} e^{-(x_{lt}-\hat{x}_t)^2/2Q_t}, \end{gathered} \tag{36}$$

where x_{lt} is the edge estimation of the mobile robot pose at time t and Q_t is the covariance matrix of the estimation at time t. Θ_{lt} is the edge estimation of the environment map which includes the N landmark estimations. m_{lj} is the estimated means of the j -th landmark, and $R_t(m_j)$ is the relevant covariance matrix. x_{t-1} is the estimated pose at time $t-1$. $z_t(m_j)$ is the real observation of the j-th landmark, and $\hat{z}_t(m_{lj})$ is the predicted observation of the j-th landmark. $\hat{x}_t$ is the predicted robot pose based on the motion model.

The posteriori probability of the cloud SLAM is shown as follows.

$$\begin{gathered} y_{ct} = p(z_t|x_{ct}, \Theta_{ct})p(x_{ct}|u_t, x_{t-1}) = , \\ \prod_{j=1}^{N} p(z_t|x_{ct}, m_{cj})p(x_{ct}|u_t, x_{t-1}) = , \\ \prod_{j=1}^{N} \frac{1}{\sqrt{2\pi R_t(m_j)}} e^{-(z_t(m_j)-\hat{z}_t(m_{cj}))^2/2R_t(m_j)} \frac{1}{\sqrt{2\pi Q_t}} e^{-(x_{ct}-\hat{x}_t)^2/2Q_t}, \end{gathered} \tag{37}$$

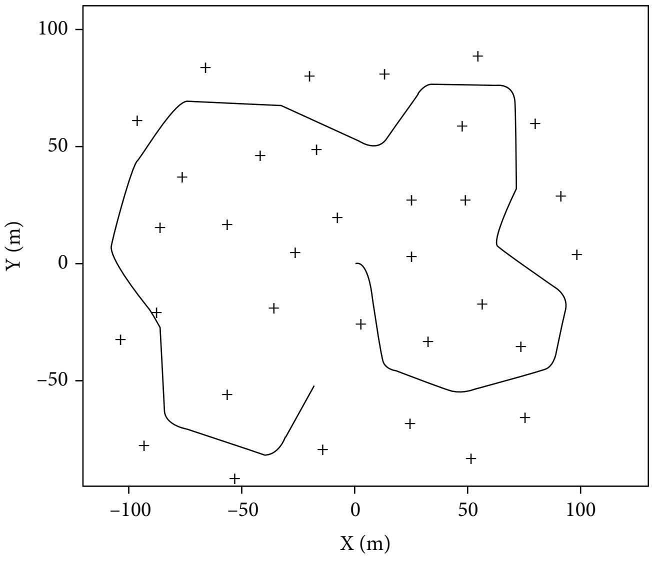


Figure 9: The map with the sparse landmarks.

Table 2: The comparisons between the three algorithms on the map with the sparse landmarks.

Algorithms	RMSE_P (m)	RMSE_L (m)	Execution time (s)
FastSLAM 2.0	7.45	9.39	20.54
UFastSLAM	4.27	5.08	26.37
The proposed algorithm	3.51	4.20	18.69

where x_{ct} is the cloud estimation of the mobile robot pose and Θ_{ct} is the cloud estimation of the environment map. m_{cj} is the j -th landmark estimation, and $\hat{z}_t(m_{cj})$ is the predicted observation of the j-th landmark.

The pose and map estimation fusion for the mobile robot are written as follows:

$$x_t = \frac{y_{ct}}{(y_{ct}+y_{lt})}x_{ct} + \frac{y_{lt}}{(y_{ct}+y_{lt})}x_{lt}, \tag{38}$$

$$m_j = \frac{y_{ct}}{(y_{ct}+y_{lt})}m_{cj} + \frac{y_{lt}}{(y_{ct}+y_{lt})}m_{lj}. \tag{39}$$

5. Experimental Results and Discussion

To verify the performance of the proposed algorithm, comparisons between the proposed algorithm, the FastSLAM 2.0 [37] and the UFastSLAM [38] are made. FastSLAM 2.0 and its variants become the most widely used laser SLAM solutions. They have been applied in the fields of unmanned vehicles, unmanned aerial vehicles, unmanned ships, mobile robots, and so on. As an important improvement of FastSLAM 2.0, UFastSLAM applies UKF to the iterative updating of the mobile robot pose. UFastSLAM is widely used not only in laser SLAM but also in visual SLAM [39].

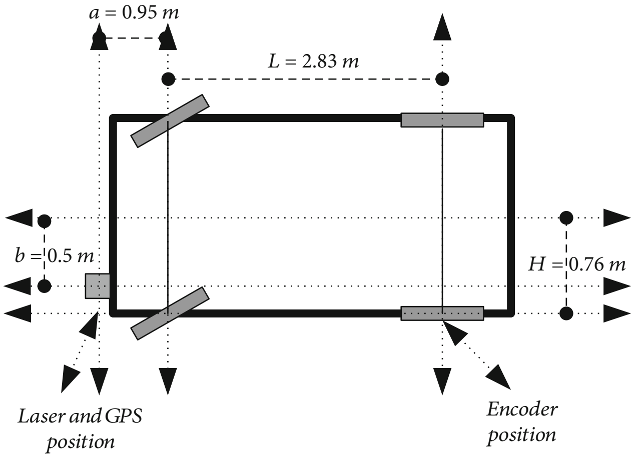


Figure 10: The motion model of the truck.

5.1. Simulation Experiments. In the simulation environment, the system state is described by the Cartesian coordinate system. The motion and observation models are as follows:

$$x_t = \left[x_{x_t}, y_{x_t}, \theta_{x_t}\right]^T = g(x_{t-1}, u_t) = \begin{bmatrix} x_{x_{t-1}} + \Delta T V_t \cos\left(\theta_{x_{t-1}} + \alpha_t\right) \\ y_{x_{t-1}} + \Delta T V_t \sin\left(\theta_{x_{t-1}} + \alpha_t\right) \\ \theta_{x_{t-1}} + \dfrac{\Delta T V_t \sin(\alpha_t)}{L} \end{bmatrix} + \varepsilon, \tag{40}$$

$$z_t(m_j) = \left[l_t(m_j), \beta_t(m_j)\right]^T = h(x_t, m_j) = \begin{bmatrix} \sqrt{\left(x_{mj} - x_{x_t}\right)^2 + \left(y_{mj} - y_{x_t}\right)^2} \\ \arctan\left(\dfrac{y_{mj} - y_{x_t}}{x_{mj} - x_{xt}}\right) - \theta_{x_t} \end{bmatrix} + \delta, \tag{41}$$

where (x_{x_t}, y_{x_t}) represents the mobile robot position. The heading θ_{x_t} is the motion direction at time t. Its value range

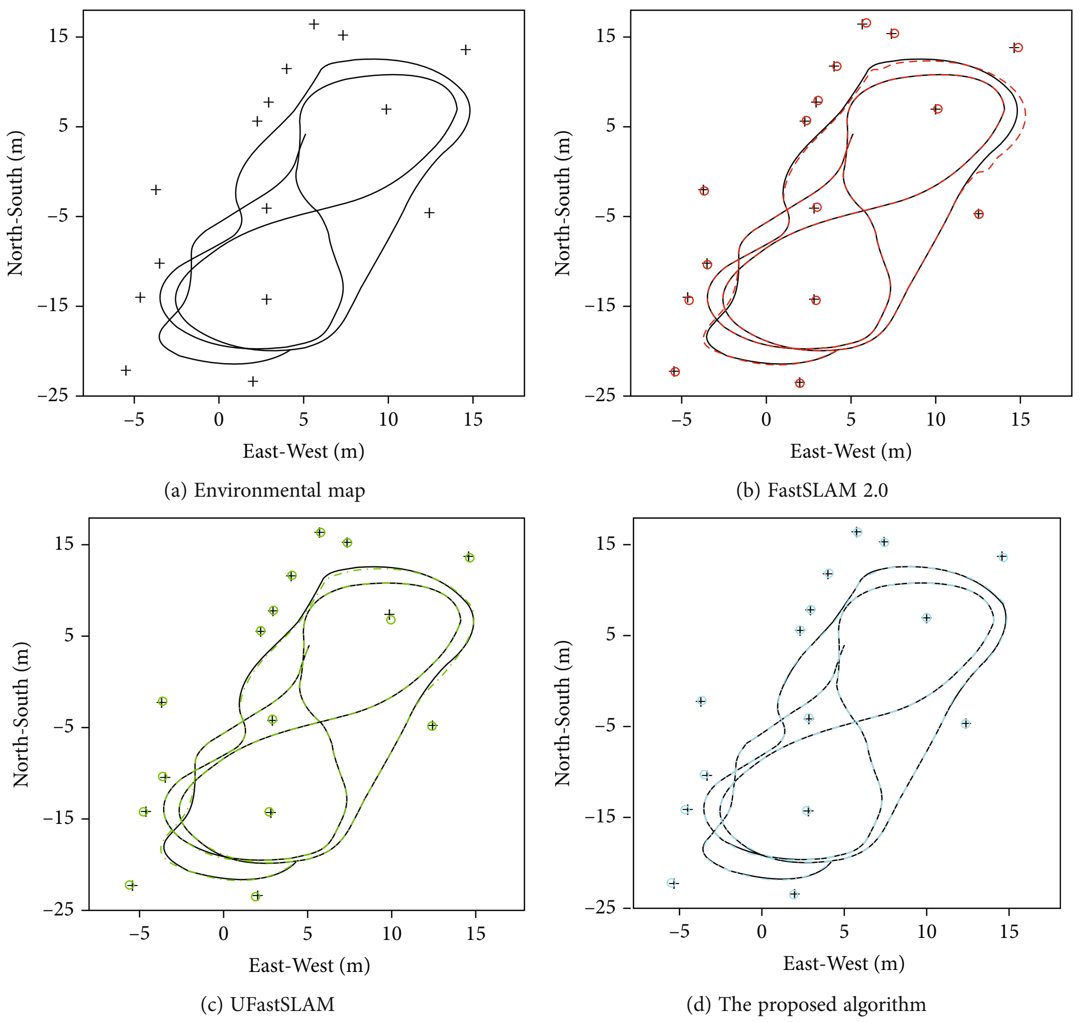


Figure 11: The comparison between the three algorithms based on the University Car Park dataset.

Table 3: The distance error comparison between the three algorithms based on the University Car Park dataset.

Algorithms	$d_{\text{r-ave}}$ (m)	$d_{\text{l-ave}}$ (m)	Execution time (s)
FastSLAM 2.0	0.153	0.096	9.36
UFastSLAM	0.129	0.081	11.09
The proposed algorithm	0.107	0.069	8.42

is [- π, π] rad. (x_{mj}, y_{mj}) is the j-th landmark position. u_t and z_t denote the control and observation data, respectively. u_t includes the velocity V_t and the steering angle α_t. $z_t(m_j)$ represents the range-bearing observation from the robot to the j -th landmark, $l_t(m_j)$ is the distance, and $\beta_t(m_j)$ is the angle from 0 to 2π clockwise. ΔT is the sampling interval, and L is the wheel base.

In the simulation environment, a low-cost embedded control board is simulated as the whole mobile robot system. As a control center, it is widely used for portable and low-cost mobile robot systems. The control board is equipped with 1.5 GHz CPU with 4 core processors and 2G memory and installed a robot operating system (ROS). The control board simulates the robot moving process according to the predefined path. To simulate the real environment, the speed and angle are added a random noise, and the data with noise are used as the control data. The distance and angle between the landmarks and the robot are calculated. The distance and angle with a random noise are taken as the observation data. This process simulates the acquisition process of control data and observation data. The maximum driving speed V_t is 3 m/s, and the maximum steering angle α_t is $\pi/6$ rad. The speed noise ε_v is 0.3 m/s, and the angle noise ε_β is $\pi/60$ rad. It is equipped a ranger bearing sensor with a π rad frontal view and a maximum range of 30 m. The range noise of the observation δ_l is 0.2 m, and the angle noise δ_β is $\pi/180$ rad. The cloud uses three servers to build a stream computing cluster by the Flink platform and the related components. The cloud server uses 16 G memory and 2.2 GHz CPU with 8 core processors.

As shown in Figure 5(a), the mobile robot moves in a 250 m × 200 m area with 75 landmarks. The robot starts from (0, 0) and moves according to a series of planned target points. The FastSLAM 2.0 and the UFastSLAM use 20

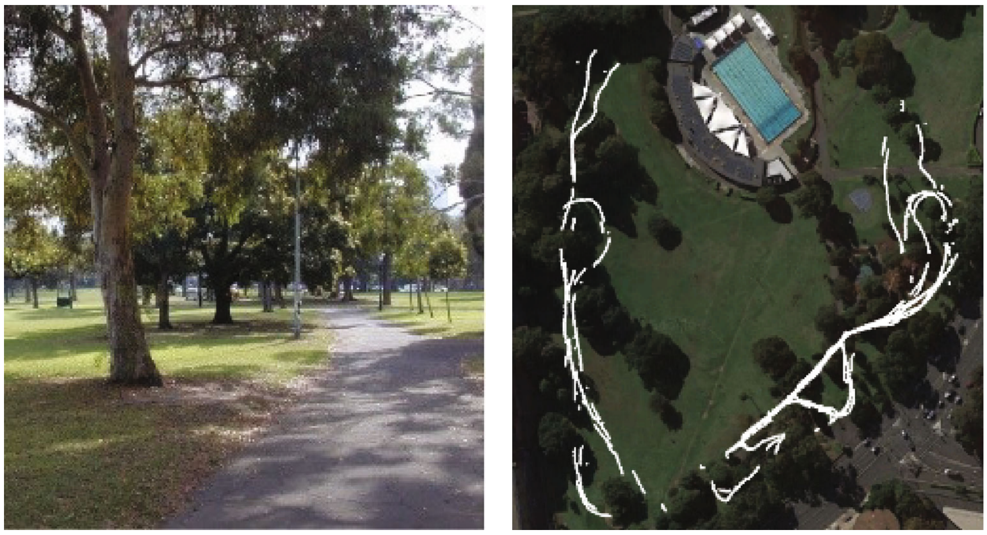

(a) Victoria Park (b) GPS trajectory

Figure 12: The experimental environment based on the Victoria Park dataset.

particles to estimate the system state. Based on the sufficient computing power in the cloud, the proposed algorithm uses 40 particles to estimate the system state. Figures 5(b)–5(d) show the experimental results of the three algorithms. The black solid line represents the motion trajectory, and the dotted line represents the estimated path by each algorithm. The black plus sign + represents the landmark position, and the circle sign ○ represents the estimated landmark position.

Estimation accuracy and execution efficiency are the most important indicators of SLAM algorithms. The comparison between the three algorithms is based on the two indicators. Estimation accuracy mainly includes the estimation accuracy for robot positions and the estimation accuracy for map. The higher the estimation accuracy, the closer the SLAM estimation is to the real state of the robot system. High estimation accuracy is beneficial to explore the unknown environment and navigate autonomously for robots. It is evident that the estimated trajectory and map by the proposed algorithm is closer to the actual system state. The error of the proposed algorithm in both path estimation and landmark position estimation is much smaller than that of the other two algorithms. The execution efficiency is the key to whether SLAM algorithms can be applied to autonomous navigation. Based on the cloud computing power, the average execution time of the single step in the proposed algorithm is the shortest execution time of 32 ms, followed by the FastSLAM 2.0 algorithm of 36 ms, and the UFastSLAM algorithm of 49 ms.

Because the control and observation noise are random, the results of each experiment are different. To obtain a more detailed and accurate comparison, 50 simulation experiments with the simulation environment of Figure 5(a) are carried out. Root mean square error (RMSE) is used to measure the deviation between the estimated value and the real value, and it can reflect well the SLAM accuracy. Table 1 shows the compared results based on RMSE.

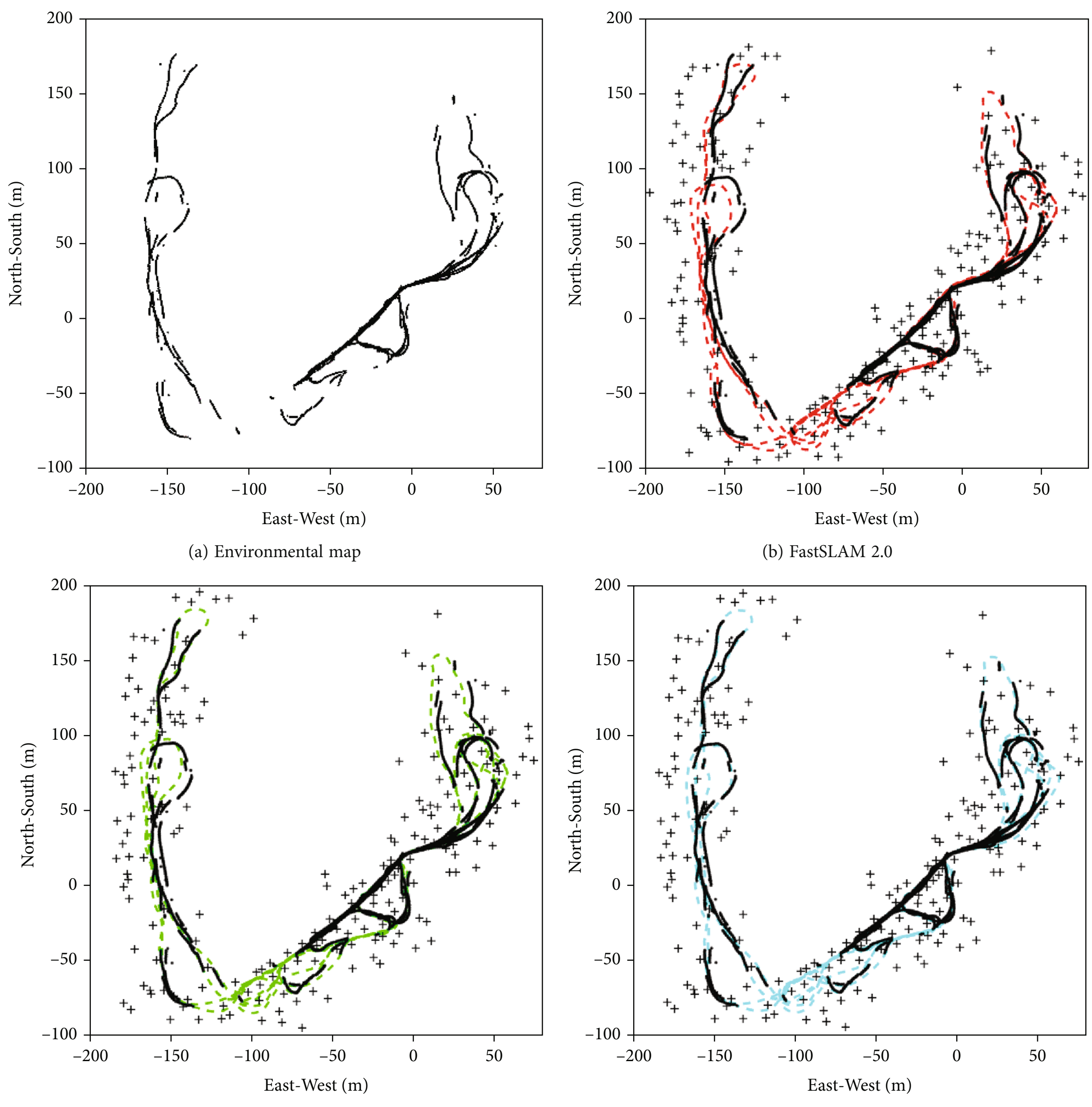
RMSE_P represents the RMSE on the robot position estimation, and RMSE_L represents the RMSE on the landmark estimation. The result shows that the proposed algorithm is more accurate than the other two algorithms in estimating the robot position and the landmark locations. The execution time is the accumulation of the single step execution time in each experiment. The execution time of the proposed algorithm is also shorter than that of the two other algorithms. For the limitation of experimental conditions, the total number of core processors in the cloud is smaller than the number of particles. If the total number of core processors is larger than the number of particles, the parallelism of the proposed algorithm will be greatly improved, and the performance will be further improved. At the same time, if the cloud resources are sufficient, more particles can be used to improve the accuracy.

Figure 6 shows the RMSE comparisons based on time series between the three algorithms. The four subfigures show the RMSE comparisons in the x-axis, the y-axis, the heading, and the position. The RMSE of the proposed algorithm is less than that of the other two algorithms in each time period.

Figure 7 shows the RMSE comparison on landmark positions between the three algorithms after the loop is closed. It can be seen that the landmark estimation errors of the proposed algorithm are fewer than that of the other two algorithms.

Consistency means that the deviation between the estimated state and the true state should be kept at a roughly constant level. The normalized estimation error squared (NESS) [40] is often used in measuring the SLAM consistency over the N Monte Carlo runs and is defined as follows:

$$\varepsilon_t = (x_t - \hat{x}_t) P_t^{-1} (x_t - \hat{x}_t), \tag{42}$$



(a) Environmental map

(b) FastSLAM 2.0

(c) UFastSLAM

(d) The proposed algorithm

Figure 13: The comparison between the three algorithms based on the Victoria Park dataset.

where x_t and P_t are the estimated mean and the covariance of the robot pose, respectively. $\hat{x}_t$ is the real pose. Given N runs, the average NEES is computed as follows:

$$\bar{\varepsilon}_k = \frac{1}{n}\sum_{i=1}^{n} \varepsilon_{k,i}. \tag{43}$$

For the 3-dimensional vehicle pose and the 50 Monte Carlo runs, the two-sided 95% probability region is bounded by the interval [2.36, 3.72]. If $\bar{\varepsilon}_k$ exceeds the upper bound, the SLAM algorithm becomes optimistic [41]. Figure 8 shows the average NEES comparison between the three algorithms. It can be seen that the FastSLAM 2.0 algorithm becomes rapidly optimistic while the average NEES of the proposed algorithm maintains a low level for a long time.

In order to further verify the effectiveness of the proposed algorithm, a map with the sparse landmarks as shown in Figure 9 is constructed. 50 simulation experiments are carried out for this environment. Table 2 shows the compared results between the three algorithms based on RMSE. Comparing the experiment map of the Figure 5, the landmarks are decreased from 75 to 35. The decrease reduces the SLAM computation and estimation accuracy. Compared with the other two algorithms, the proposed algorithm has higher accuracy of the robot pose and landmark position estimation and less calculation time.

The experimental results show that the proposed algorithm has higher accuracy and less computation time, whether on the map with the dense landmarks or the sparse landmarks.

5.2. Experiments with the University Car Park and Victoria Park Dataset. The University Car Park and the Victoria Park dataset are collected by the Australian Centre for Field Robotics (ACFR) in Sydney. They are popular in the SLAM research community. A truck equipped with GPS, inertial, and laser sensors is tested. The motion model of the truck is shown in Figure 10.

The motion and the observation model are as follows:

$$x_t = g(x_{t-1}, u_t) = \begin{bmatrix} x_{x_{t-1}} + \Delta T V_t \left(\cos\left(\theta_{x_{t-1}} + \alpha_t\right) - \dfrac{\tan\left(\alpha_t\right)}{L}\left(a \sin\left(\theta_{x_{t-1}}\right) + b \cos\left(\theta_{x_{t-1}}\right)\right)\right) \\ y_{x_{t-1}} + \Delta T V_t \left(\sin\left(\theta_{x_{t-1}} + \alpha_t\right) - \dfrac{\tan\left(\alpha_t\right)}{L}\left(b \sin\left(\theta_{x_{t-1}}\right) - a \cos\left(\theta_{x_{t-1}}\right)\right)\right) \\ \theta_{x_{t-1}} + \dfrac{\Delta T V_t \tan\left(\alpha_t\right)}{L} \end{bmatrix} + \varepsilon, \tag{44}$$

$$V_t = \frac{V_{et}}{1 - (H/L) \times \tan\left(\alpha_t\right)}, \tag{45}$$

$$z_t\left(m_j\right) = h(x_t) = \begin{bmatrix} l_t\left(m_j\right) \\ \beta_t\left(m_j\right) \end{bmatrix} = \begin{bmatrix} \sqrt{\left(x_{m_j} - x_{x_t}\right)^2 + \left(y_{m_j} - y_{x_t}\right)^2} \\ \arctan\left(\dfrac{y_{m_j} - y_{x_t}}{x_{m_j} - x_{x_t}}\right) - \theta_{x_t} + \dfrac{\pi}{2} \end{bmatrix} + \delta, \tag{46}$$

where V_t is the velocity of the center of the axle, V_{et} is the velocity of the back left wheel, ΔT is the sampling interval, and αt is the steering angle.

5.2.1. The University Car Park Dataset. The University Car Park dataset [42] is used to compare the FastSLAM 2.0, the UFastSLAM, and the proposed algorithm. The experimental environment is shown in Figure 11(a), and the results of the three algorithms are shown in Figures 11(b)–11(d). The estimated trajectory by the proposed algorithm is the most consistent with the GPS trajectory. It indicates that the estimation accuracy of the proposed algorithm is better than that of the UFastSLAM and the FastSLAM 2.0 algorithms.

Table 3 shows the estimation accuracy comparison between the three algorithms where $d_{\mathrm{r-ave}}$ is the average distance error for the robot position estimation and $d_{\mathrm{l-ave}}$ is the average distance error for the landmark position estimation. As shown in Table 3, the estimated error of the proposed algorithm is less than that of the FastSLAM 2.0 and the UFastSLAM whether on the robot position estimation or the landmark position estimation. The execution time of the proposed algorithm is also less than that of the UFastSLAM and the FastSLAM 2.0. The single step execution time of the proposed algorithm is 29 ms, which is less than the sampling interval. It can meet the real-time requirements.

5.2.2. The Victoria Park Dataset. The Victoria Park dataset [43] is used to compare the three algorithms. The experimental environment is a $300 \times 300\,\mathrm{m}^2$ large-scale environment, as shown in Figure 12. The dataset records the sensor data obtained by manually driving the intelligent vehicle for approximately 3.5 kilometers in 1545 seconds. The vehicle collects 61945 frames of inertial navigation data, 7249 frames of lidar data, and 4461 frames of GPS data.

The main obstacle in the environment is trees. For the low installation position of lidar, the trunk is used as the landmark. Due to the shelter of trees, the GPS data is discontinuous. The positions of these natural landmarks are not measured, and the true position of landmarks cannot be marked on the map. Figure 13 shows the experimental results.

The black line represents the GPS data, the dotted line represents the path estimated by each algorithm, and the plus sign + is the estimated landmark position. For the discontinuity of the GPS data and lack of the natural landmarks position data, the difference between the estimated position and the real position is not calculated. As shown in Figure 13, the estimated path by the proposed algorithm is more consistent with the GPS trajectory. For the linearization error of the FastSLAM 2.0 algorithm, the estimated path does not match the GPS trajectory. Because the cumulative error of the UFastSLAM and the proposed algorithm is small, the difference between the estimated path and GPS trajectory is less than that of the FastSLAM 2.0 algorithm. The execution time of the proposed algorithm is 674 seconds, which is less than the 1029 seconds of the UFastSLAM algorithm and the 952 seconds of the FastSLAM 2.0 algorithm.

6. Conclusions

The emergence of edge-cloud collaboration provides a new solution to the SLAM problem. The distributed and parallel stream service on the cloud extends the computing capacity of mobile robots and improves the accuracy of SLAM with more particles. The edge builds the local map to meet the real-time requirements. The cloud changed the traditional serial sampling and resampling process to distributed parallel execution, and the cloud computing power has been used to improve the execution efficiency. Edge-cloud collaborative computing can compensate for the lack of compute ability of mobile robots and enhance their environmental adaptability.

Based on the simulation environments and two public datasets, the proposed algorithm, FastSLAM 2.0, and UFast-SLAM are compared. Experimental results show that the proposed algorithm has the highest estimation accuracy and the lowest RMSE compared with the other two algorithms. The experimental results also prove that the proposed algorithm has the fastest execution time, which can ensure the real-time performance of SLAM. For SLAM algorithms based on particle filter, the accuracy can be improved by increasing the particle number. Increasing the processor core number in the cloud can enable the proposed algorithm to use more particles to improve the accuracy without increasing the computing time. The accuracy and efficiency of the proposed algorithm can be further improved while the cloud computing power is increased. Compared with the other two algorithms, the proposed algorithm transfers the heavy computation from robots to the cloud, and it can enhance the environmental adaptability of mobile robots. The proposed algorithm not only ensures the real-time performance but also improves the accuracy and efficiency by the edge-cloud collaborative architecture. Based on the low-cost edge and the performance improvement, the proposed algorithm has the high utility and promotion value.

The proposed algorithm combines power of the edge and the cloud to improve the reliability and efficiency. However, several aspects need to be studied further. (1) The proposed algorithm can be regarded as a hybrid SLAM algorithm. The particle number and the transmission delay in the cloud are important factors affecting the fusion results. The further study is to optimize the proposed algorithm by testing the edge-cloud fusion effect under different network environments and different particle numbers. (2) In experiments, the test of single robot edge-cloud fusion is completed. In the future, the experiments of multirobot edge-cloud fusion will be completed. The resource scheduling and task optimization under the sudden situation of big streaming data will be the research focus.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no competing interests.

Acknowledgments

This work was financially supported by the China Postdoctoral Science Foundation (2019M651844), Jiangsu Postdoctoral Science Foundation (2018K035C), Excellent Scientific and Technological Innovation Team of Jiangsu Higher Education Institutions of China (Maritime big data team), Six Talent Peaks Project in Jiangsu Province (XYDXX-149), and Young Academic Leaders for QingLan Project of the Jiangsu Higher Education Institutions of China.

References

[1] C. Kolhatkar and K. Wagle, “Review of SLAM algorithms for indoor mobile robot with LIDAR and RGB-D camera technology,” *Innovations in Electrical and Electronic Engineering*, vol. 661, pp. 397–409, 2021.

[2] C. Feng-Kui, Y. Zhuang, Y. Fei, Y. Qi-Feng, and W. Wei, “Long-term autonomous environment adaptation of mobile robots: state-of-the-art methods and prospects,” *Acta Automatica Sinica*, vol. 66, no. 2, pp. 205–221, 2020.

[3] J. Manyika, M. Chui, J. Bughin, R. Dobbs, and P. Peter, *Disruptive technologies: Advances that will transform life, business, and the global economy*, McKinsey Global Institute, 2013.

[4] G. P. S. Citi, *Disruptive innovations VII: ten more things to stop and think about*, Citi, 2020.

[5] J. Kuffner, “Cloud-enabled humanoid robots,” in *Proceedings of the 10th IEEE-RAS International Conference on Humanoid Robot*, pp. 19–23, Nashville, TN, USA, 2010.

[6] K. Goldberg, “Robots and the return to collaborative intelligence,” *Nature Machine Intelligence*, vol. 1, no. 1, pp. 2–4, 2019.

[7] S. Olimpiya and D. Prithviraj, “A comprehensive survey of recent trends in cloud robotics architectures and applications,” *Robotics*, vol. 7, no. 3, pp. 25–47, 2018.

[8] T. Z. Lv, J. Zhang, J. Zhang, and Y. Chen, “A path planning algorithm for mobile robot based on edge-cloud collaborative computing,” *International Journal of System Assurance Engineering and Management*, vol. 13, pp. 594–604, 2022.

[9] R. Bouziane, L. S. Terrissa, S. Ayad, J. F. Brethe, and O. Kazar, “A web services based solution for the NAO robot in cloud robotics environment,” in *Proceedings of the 4th International Conference on Control, Decision and Information Technologies*, pp. 809–814, Barcelona, Spain, 2017.

[10] P. Li, B. DeRose, J. Mahler, J. A. Ojea, A. K. Tanwani, and K. Goldberg, “Dex-net as a service (DNAAS): a cloud-based robust robot grasp planning system,” in *Proceedings of the 14th IEEE International Conference on Automation Science and Engineering*, pp. 1420–1427, Munich, German, 2018.

[11] P. Pandey, D. Pompili, and J. G. Yi, “Dynamic collaboration between networked robots and clouds in resource-constrained environments,” *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 471–480, 2015.

[12] R. Arumugam, V. R. Enti, L. Bingbing et al., “DAvinCi: a cloud computing framework for service robots,” in *Proceedings of the 2010 IEEE International Conference on Robotics and Automation*, pp. 3084–3089, Anchorage, AK, USA, 2010.

[13] P. Zhang, H. Wang, B. Ding, and S. Shang, “Cloud-based framework for scalable and real-time multi-robot SLAM,” in *Proceedings of the 2018 IEEE International Conference on*

Web Services (ICWS), pp. 147–154, San Francisco, CA, USA, 2018.

[14] S. Kamburugamuve, H. J. He, and G. Fox, "Cloud-based parallel implementation of SLAM for mobile robots," in *Proceedings of the International Conference on Internet of things and Cloud Computing*, pp. 1–7, Cambridge, UK, 2016.

[15] S. Kamburugamuve, L. Christiansen, and G. Fox, "A framework for real time processing of sensor data in the cloud," *Journal of Sensors*, vol. 2015, no. 2, Article ID 468047, 11 pages, 2015.

[16] L. Riazuelo, J. Civera, and J. M. M. Montiel, "C2TAM: a cloud framework for cooperative tracking and mapping," *Robotics & Autonomous Systems*, vol. 62, no. 4, pp. 401–413, 2014.

[17] B. Xu and J. Bian, "A cloud robotic application platform design based on the microservices architecture," in *Proceedings of the 2020 International Conference on Control, Robotics and Intelligent System*, pp. 13–18, New York, NY, USA, 2020.

[18] Y. Liu, H. Zhang, and C. Huang, "A novel RGB-D SLAM algorithm based on cloud robotics," *Sensors*, vol. 19, no. 23, p. 5288, 2019.

[19] N. N. Yatim and B. Norlida, "Particle filter in simultaneous localization and mapping (SLAM) using differential drive mobile robot," *Jurnal Teknologi*, vol. 77, no. 20, pp. 91–97, 2015.

[20] E. Duymaz, A. E. Ouz, and H. Temelta, "Exact flow of particles using for state estimations in unmanned aerial systems navigation," *PLoS One*, vol. 15, no. 4, pp. 1–27, 2020.

[21] Y. B. Shen and Z. P. Jiao, "A novel self-positioning based on feature map creation and laser location method for RBPF-SLAM," *Journal of Robotics*, vol. 2021, Article ID 9988916, 11 pages, 2021.

[22] S. H. Park and S. Y. Yi, "Least-square matching for mobile robot SLAM based on line-segment model," *International Journal of Control, Automation and Systems*, vol. 17, no. 11, pp. 2961–2968, 2019.

[23] L. Duggan, J. Dowzard, J. Katupitiya, and K. C. Chan, "A rapid deployment big data computing platform for cloud robotics," *International Journal of Computer Networks and Communications*, vol. 9, no. 6, pp. 77–88, 2017.

[24] T. Z. Lv, C. X. Zhao, and H. F. Zhang, "An improved FastSLAM algorithm based on revised genetic resampling and SR-UPF," *International Journal of Automation and Computing*, vol. 15, no. 3, pp. 325–334, 2018.

[25] J. H. Kong, X. Mao, and S. Li, "BDS/GPS dual systems positioning based on the modified SR-UKF algorithm," *Sensors*, vol. 16, no. 5, pp. 1–15, 2016.

[26] W. Zhou, X. Yin, Z. Cao, and J. Shao, "Two measures for enhancing data association performance in SLAM," *Journal of Sensors*, vol. 2014, Article ID 326820, 8 pages, 2014.

[27] R. Havangi, "Intelligent FastSLAM: an intelligent factorized solution to simultaneous localization and mapping," *International Journal of Humanoid Robotics*, vol. 14, no. 1, p. 1650026, 2017.

[28] A. Panah, H. Motameni, and A. Ebrahimnejad, "An efficient computational hybrid filter to the SLAM problem for an autonomous wheeled mobile robot," *International Journal of Control, Automation and Systems*, vol. 19, no. 10, pp. 3533–3542, 2021.

[29] M. Henein, J. Zhang, R. Mahony, and V. Ila, "Dynamic SLAM: the need for speed," in *Proceedings of the 2020 IEEE International Conference on Robotics and Automation*, pp. 2123–2129, Paris, France, 2020.

[30] A. Filatov, A. Filatov, K. Krinkin, B. Chen, and D. Molodan, "2D SLAM quality evaluation methods," in *Proceedings of the 21st Conference of Open Innovations Association*, pp. 120–126, Helsinki, Finland, 2017.

[31] T. Z. Lv and M. Y. Feng, "An improved FastSLAM 2.0 algorithm based on FC&ASD-PSO," *Robotica*, vol. 35, no. 9, pp. 1795–1815, 2017.

[32] T. Bailey, J. Nieto, and E. Nebot, "Consistency of the Fast SLAM algorithm," in *Proceedings of the 2006 IEEE International Conference on Robotics and Automation*, pp. 424–429, Orlando, FL, USA, 2006.

[33] P. Norgren and R. Skjetne, "A multibeam-based SLAM algorithm for iceberg mapping using AUVs," *IEEE Access*, vol. 6, pp. 26318–26337, 2018.

[34] M. Tang, Z. Chen, and F. Yin, "An improved H-infinity unscented FastSLAM with adaptive genetic resampling," *Robotics and Autonomous Systems*, vol. 134, article 103661, 2020.

[35] Y. Zhang, S. F. Wang, and J. C. Li, "Improved particle filtering techniques based on generalized interactive genetic algorithm," *Journal of Systems Engineering and Electronics*, vol. 27, no. 1, pp. 242–250, 2016.

[36] P. Giordani and R. Kohn, "Adaptive independent Metropolis-Hastings by fast estimation of mixtures of normal," *Journal of Computational and Graphical Statistics*, vol. 19, no. 2, pp. 23–259, 2010.

[37] M. Montemerlo and S. Thrun, *Fast SLAM: A Scalable Method for the Simultaneous Localization and Mapping Problem in Robotics*, Springer, 2007.

[38] C. Kim, H. Kim, and W. K. Chung, "Exactly Rao-Blackwellized unscented particle filters for SLAM," in *Proceedings of the 2011 IEEE International Conference on Robotics and Automation*, pp. 3589–3594, Shanghai, China, 2011.

[39] S. Das, R. Kumari, and S. D. Kumar, "A review on applications of simultaneous localization and mapping method in autonomous vehicles," in *Proceedings of the Advances in Interdisciplinary Engineering*, pp. 367–375, Singapore, 2021.

[40] W. Youn and M. Hyun, "Robust interacting multiple model with modeling uncertainties for maneuvering target tracking," *IEEE Access*, vol. 2019, no. 7, pp. 65427–65443, 2019.

[41] J. S. Liu, R. Chen, and T. Logvinenko, "A theoretical framework for sequential importance sampling with resampling," in *Proceedings of the Sequential Monte Carlo in Practice*, pp. 225–2456, New York, NY, USA, 2001.

[42] E. Nebot, "ACFR university car park," https://www-personal.acfr.usyd.edu.au/nebot/car_park.htm.

[43] E. Nebot, "ACFR Victoria park," https://www-personal.acfr.usyd.edu.au/nebot/victoria_park.htm/.