Edge Robotics: Edge-Computing-Accelerated Multi-Robot Simultaneous Localization and Mapping

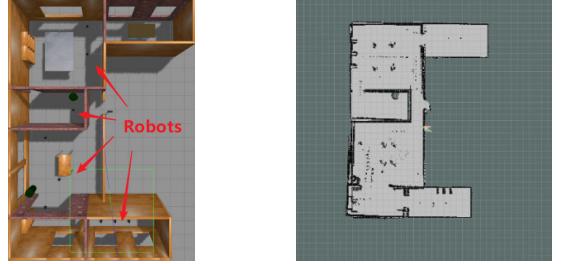Peng Huang†, Liekang Zeng†, Xu Chen, Ke Luo, Zhi Zhou, Shuai Yu

Abstract—With the wide penetration of smart robots in multifarious fields, Simultaneous Localization and Mapping (SLAM) technique in robotics has attracted growing attention in the community. Yet collaborating SLAM over multiple robots still remains challenging due to performance contradiction between the intensive graphics computation of SLAM and the limited computing capability of robots. While traditional solutions resort to the powerful cloud servers acting as an external computation provider, we show by real-world measurements that the significant communication overhead in data offloading prevents its practicability to real deployment. To tackle these challenges, this paper promotes the emerging edge computing paradigm into multi-robot SLAM and proposes RecSLAM, a multi-robot laser SLAM system that focuses on accelerating map construction process under the robot-edge-cloud architecture. In contrast to conventional multi-robot SLAM that generates graphic maps on robots and completely merges them on the cloud, RecSLAM develops a hierarchical map fusion technique that directs robots' raw data to edge servers for real-time fusion and then sends to the cloud for global merging. To optimize the overall pipeline, an efficient multi-robot SLAM collaborative processing framework is introduced to adaptively optimize robot-to-edge offloading tailored to heterogeneous edge resource conditions, meanwhile ensuring the workload balancing among the edge servers. Extensive evaluations show RecSLAM can achieve up to 39.31% processing latency reduction over the state-of-the-art. Besides, a proof-of-concept prototype is developed and deployed in real scenes to demonstrate its effectiveness.

Index Terms—Edge intelligence, multi-robot laser SLAM, edge offloading, distributed and parallel processing.

I. INTRODUCTION

AS a key enabling technology in robotics, Simultaneous Localization and Mapping (SLAM) is a computationally-intensive approach that targets at simultaneously constructing a graphic map and tracking the agent's location in an unknown environment [1]–[3]. It has been broadly used across a number of applications, opening a wide door to smart robots for understanding and interacting with environments wherever indoor, aerial, or underground. For example, in autonomous robotics, SLAM has been employed to process the surrounding information from laser scans to guide the self-contained navigation [4]–[7]. In disaster relief, snakelike robots apply SLAM to explore fragile buildings and carry out a rescue in human-unreachable places [8]–[10].

Generally, SLAM can be categorized into two classes according to the used sensors, namely visual SLAM [11] and laser SLAM [12]. Visual SLAM utilizes the images captured

† The two authors contributed equally to this work.

The authors are with the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, Guangdong, 510006 China, e-mail: {huangp57, zenglk3, luok7}@mail2.sysu.edu.cn, {chenxu35, zhouzhi9, yushuai}@mail.sysu.edu.cn. The corresponding author is Xu Chen.

(a) Example apartment scene with ten robots.

(b) The corresponding map generated by RecSLAM.

Fig. 1: An example multi-robot SLAM application at the edge, where the robots are navigated to walk in the apartment (a) and perform SLAM to construct the corresponding geometric indoor map (b).

by cameras to reckon geometric graphics. It strives to exploit the 2D visual input to construct 3D semantic models. Laser SLAM forks another technical path that extracts data from laser scans to shape a grid occupancy map. Particularly, laser data differs from visual images by containing depth-wise information and high-precision positions of objects. Benefited by such advantage in sources, laser SLAM is preferred and employed for a much wider spectrum of robotic applications for achieving much higher physical accuracy [13].

The widespread use of smart robots catalyzes SLAM to extend its deployment over multiple robots [14]–[16]. Specifically, a group of robots is deployed in a swarm manner, and they are committed to cooperating with each other in an unknown environment in order to construct the geometric mapping and determine their own locations through SLAM. An example scenario is to build the graphic maps of an apartment as shown in Fig. 1, where multiple robots are employed to perform SLAM simultaneously for global map construction. Another example is in mission-critical Search and Rescue (SAR) tasks [14], [15], where multiple robots are driven collaboratively to share map data with each other for rapid exploration in broken buildings. In these circumstances, traditional wisdom [17]–[19] to reconcile the distributed data sources calls for robot-cloud synergy, where robots run SLAM individually and upload their local map mutually to a centralized cloud server for global merging. Yet the two-tier architecture suffers from dual drawbacks. On the one hand, processing SLAM workloads locally on commodity robots typically demand plenty of latency and can even fail to complete under resource constraints, *e.g.* limited memory capacity. In our measurements on an ordinary robot (as will be shown in Sec.

II-B), running SLAM can force CPU workload at a high level consistently (always >83.60%), leading to excessive latency in on-device execution. Such significant and excessive latency impedes many multi-robot applications (e.g., disaster relief and high-resolution autonomous navigation) that generally demand real-time information fusion. On the other hand, transferring robots' local maps of a large volume to the remote cloud through the unreliable and delay-significant wide-area Internet connection not only incurs the communication bottleneck but raises users' concerns on security and privacy. Quantitatively, as we will show later, the data uploading time overhead can dominate the whole processing in common 4G, 5G and WiFi networks, exhibiting vulnerable performance.

To remedy these limitations, this paper intends to leverage the emerging edge computing [20]–[22] paradigm to perform multi-robot laser SLAM in low latency. Instead of relying on geographically distant data centers, edge computing concentrates on utilizing vicinal computing resources (e.g., 5G/WiFi edge severs) in physical proximity to end devices, and therefore remarkably shortens data communication distance, lowers offloading transmission delay, and allows the advanced quality of services [23], [24]. Nevertheless, enabling multi-robot SLAM with edge computing architecture is non-trivial with three-fold challenges. First, the spatially distributed nature of multi-robot deployment requires a unified flow for data collection, organization, and computation due to the versatile sources as input. For instance, to collect data from two physically isolated robots, the edge server needs to identify their source location, timestamp, and sensory format (e.g., raw data from whether laser scan or IMU sensors). Second, distributing the map fusion, which is traditionally centralized processed, to multiple edge servers desires a retrofit on handling SLAM maps, which is intractable under resource dynamics. Third, simultaneously orchestrating the computation and communication among multiple robots and multiple edge servers presents inherent complexity in formulation and optimization on SLAM execution and data transmission. Furthermore, the network dynamics among robots and edge servers further complicate the problem.

To tackle these challenges, this paper proposes RecSLAM, a system built upon the **R**obot-**e**dge-**c**loud architecture to enable multi-robot laser **SLAM** in low-latency services. RecSLAM's design is motivated by the observations on real-world measurements that 1) migrating SLAM workloads from robots to edge servers can effectively augment the robots' processing capability, and 2) preparative merging a subset of local maps at the edge can shrink the sizes of data to be uploaded to the cloud and therefore further reduces communication costs. By exploiting them, RecSLAM considers a hierarchical robot-edge-cloud pipeline, where each robot individually collects data and selectively transfers them to one of the edge servers. The edge servers commit to performing SLAM on their received robot data, merging the corresponding local maps, and uploading the preparative fusion results to a dedicated cloud server for global map fusion. An adaptive coordinator (i.e., task scheduler) is further developed to optimize the data flow between robots and edge servers, aiming at minimizing the end-to-end latency. We implement RecSLAM on both

simulation platforms and real robots. Simulation results show that our system can outperform the state-of-the-art solutions by up to 39.31% latency reduction. The proof-of-concept prototype in an indoor scene corroborates its feasibility and efficiency, demonstrating the promising advantage of edge-accelerated SLAM. In summary, we make the following key contributions:

- We conduct a fine-grained investigation on the processing costs of existing SLAM processing solutions. The robotics-based measurements reveal that the cloud offloading mechanism suffers from the considerable transmission latency in sensory data uploading, while the local execution falls short at efficiency due to the limited onboard computing resource.

- We propose RecSLAM, a collaborative SLAM system based on hierarchical robot-edge-cloud architecture to enable real-time SLAM serving. RecSLAM decouples the conventional SLAM pipeline to distribute them to multiple edge servers, where each robot can selectively offload their raw frames to one of the edge servers and the edge server can fuse multiple frames ahead of global merging. By extending the centralized computation to distributed and parallel processing, RecSLAM significantly improves the utilization of edge resources.

- We develop a novel graph based framework to optimize the collaborative SLAM processing between robots and edge servers. Specifically, we separate the workflow into two stages, namely robot data grouping and edge offloading. For robot data grouping, we abstract an undirected graph to describe the distributed robot data processing issue with workload-balancing among the edge servers, and apply an efficient balanced graph partitioning algorithm to make a balanced robot data grouping for offloaded processing at the edge. For edge offloading, we devise an efficient resource-aware collaborative processing strategy to adaptively offload the grouping data from the robots to the proper edge servers, in order to minimize the total processing latency.

- We implement and evaluate RecSLAM in both simulation and realistic deployment. The simulation on the Gazebo [25] platform demonstrates the effectiveness of RecSLAM and its collaborative processing algorithms, showing up to 39.31% latency speedup upon cloud offloading approach. The realistic prototype on three robots in an indoor experimental scene verifies its feasibility and validity in rendering efficient SLAM services for edge applications.

The rest of this paper is organized as follows. Section II characterizes the existing multi-robot SLAM solution. Section III, IV and V present the design, optimization and implementation of the proposed system. Section VI evaluates in terms of both simulation and prototype. Section VII reviews related works and Section VIII concludes.

II. BACKGROUND AND MOTIVATION

This section dives into multi-robot SLAM processing by characterizing the state-of-the-art cloud-based solution. We
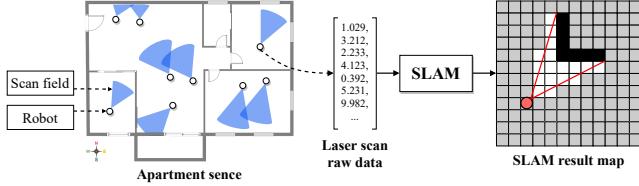
Fig. 2: The illustration of a typical SLAM computation. The example scenario covers several robots that run simultaneously to sketch an indoor map for the apartment, where each robot is represented by a dot and its scan field is abstracted as a sector. A robot walks intentionally to acquire the environment's laser scan data, in the form of a floating-point vector, and passes it to the local SLAM computation module to generate a geometric grid map.

first briefly introduce its workflow, and next break down the component-wise performance, which reveals the opportunities and challenges of edge computing.

A. Multi-Robot SLAM

Simultaneous Localization And Mapping (SLAM) is typically run at a robot with various sensors equipped, aiming at simultaneously localizing the robot itself and maintaining a graphic map of the environment [26]. Fig. 2 shows a typical scenarios of SLAM computation. The input to SLAM is the sensory data from robots in a form of floating-point vectors, which describe the scanning angle, laser distance, etc. The output is an occupancy grid map, where the black grids are obstacles, the whites represent available safe space and the grays mean the places that have not been detected or uncertain.

Multi-robot SLAM is an extended use case of single-robot SLAM, which manages a cluster of robots to perform SLAM for constructing a virtual graphic sense corresponding to reality [27]. Its core procedure beyond single-robot SLAM is the map fusion procedure that merges multiple grid maps to yield a global map of the targeted scene. Given that grid maps are collected from multiple distributed robots, state-of-the-art solutions [17]–[19] resort to the centralized cloud server to fulfill map fusion, as illustrated in Fig. 3. Particularly, it works in two stages. First, each robot collects sensory data and individually performs SLAM locally. Next, they upload their local SLAM result, i.e., grid maps, to the cloud, where all these maps are merged to acquire a global map. Specifically, two local maps that share an overlapping area will be fused, while those independent ones will be simply appended to constitute the global map. This global map can be used for downstream tasks such as navigation and furniture design.

Multi-robot SLAM has been widely adopted in a broad range of scenarios. For example, some indoor mapping services employ multiple robots to scan the house and perform multi-robot SLAM to construct a 3D graphic model that shapes the indoor environment [15]. In some geographical mapping applications, many robots are run to scan landforms and build topographic maps [17].
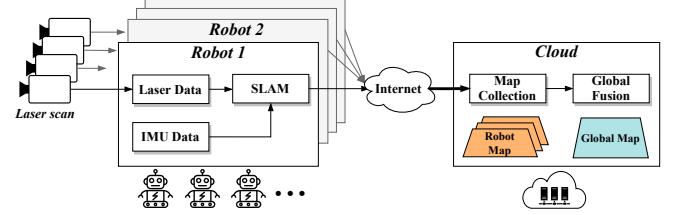

Fig. 3: The pipeline of the traditional cloud-based multi-robot SLAM solution, where the robots performs SLAM individually and upload their data via Internet to a centralized cloud server for global map fusion.
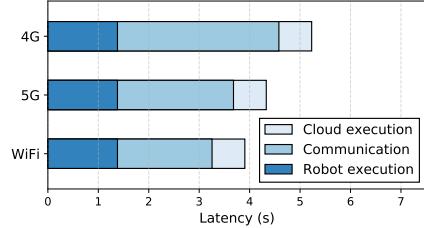

Fig. 4: The latency breakdown of a cloud-based multi-robot SLAM solution. The communication latency contributes a major portion out of the total and is sensitive the networking condition.

B. Performance Implications of State-of-The-Art

State-of-the-art cloud-based solutions highly rely on the Internet to gather local frames from distributed robots, which makes it sensitive to the vulnerable network. To make a clearer understanding of how the network conditions impact, we explicitly examine the costs of a typical cloud serving process. Specifically, we deploy a multi-robot SLAM prototype using three Turtlebot3 and a cloud server, aiming at measuring the duration from laser scan input at robots to a global map obtained on the cloud. The Turtlebot is equipped with Raspberry Pi as the processing core, which has 1.4GHz ARM Cortex-A53 CPUs, 1GB LPDDR2 without GPU. The cloud server is with Intel Xeon CPU E5-2678, deployed at the available region that robots locate. The robots contact the cloud via three channels: 3G, 4G, and WiFi, all are under commercial operation networks.

Fig. 4 visualizes the measurements, where the latency breaks down in robot computing, frames uploading, and cloud merging. We can witness that the total latency is highly sensitive to the channel switching with an increase from about 3.83s (WiFi) to 5.36s (5G). Looking closer, we observe that such cost fluctuation comes from the communication side, which dramatically varies under different network conditions. Concretely, the uploading latency takes 3.13ms, 2.35ms, 1.88ms for 4G, 5G, and WiFi, respectively.

Another expensive workload revealed is the local computing cost. To compute SLAM at the robot consumes around 1.51s, occupying a percentage of 49.74% in total latency under WiFi channel. To inspect the cause behind that, we log the background load level on robots and plot the trace as in Fig. 5. As shown in the figure, the trajectory of the load always
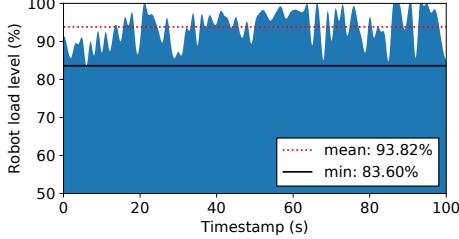
Fig. 5: The monitored background load level during SLAM runtime on the robot. The load curve consistently lies at a high level with a minimum of 83.60% and an average of 93.82%, indicating the robot is overloaded.

goes on top of 80%. It records an average of 93.82% and a minimum of 83.60%, sometimes even reaching 100.00%, indicating that the robot is overloaded and can not sufficiently support the local SLAM computation.

In summary, the existing cloud-based solution suffers from dual bottlenecks. One is the inherent communication bottleneck from the remote Internet transmission that stresses the local frame uploading process between robots and the cloud. The other is the resource-hungry SLAM computation tasks, conflicting with the limited computing capability of the robots. The integration of these two knobs makes the cloud serving fall short in rendering low latency multi-robot SLAM, desiring a holistic retrofit on the execution pipeline.

C. Opportunities and Challenges with Edge Computing

To tackle the knobs of cloud-based approaches in both data offloading and robot computing, we promote edge computing to enable multi-robot SLAM processing. Edge computing sinks computing power to the physical proximity to the robots and thereby provides a potential assist to greatly reduce the computation stress for robots while significantly lowering the transmission latency compared to cloud offloading paradigms [21].

With edge computing, there are opportunities to redesign the multi-robot map fusion pipeline to accomplish both efficient SLAM computation and low latency transmission. Specifically, we can utilize the distributed edge servers to take over the workload from both robots and the cloud, acting as agents to resolve the processing bottlenecks smoothly.

Nevertheless, different from the cloud data center that provides unified, powerful, and closed resource access, edge servers exhibit a loosely coupled and uneven nature: they are usually distributed, heterogeneous, and dynamic. To orchestrate the serving between robots, edge servers, and cloud, the communication protocol should be carefully designed considering data transmissions in robot-to-edge, edge-to-edge, and edge-to-cloud. Besides, the workload balance among the multiple edge servers, as another impacted aspect related to the overall performance, should also be taken into account given the network dynamics.

III. RECSLAM SYSTEM DESIGN

In this section, we introduce the design details of Rec-SLAM, a multi-robot laser SLAM system that leverages hierarchical robot-edge-cloud architecture to accelerate multi-robot map construction. We present the design of RecSLAM by explaining its modular details following the data flow.

A. Overview

The environment we envisaged to deploy RecSLAM spans multiple robots with laser scans and multiple edge servers in proximity. The robots can walk randomly or along with a preset route, and therefore their sensory data are time-varying and may be overlapped in some moments. The edge servers can accept data from robots simultaneously and promise to be available during an epoch of execution.

RecSLAM works upon robot-edge-cloud architecture and designs specific modules in each tier. Fig. 6 shows a high-level view of RecSLAM. In the beginning, each robot captures environmental perceptions via its installed sensors and acquires the raw sensory data. In RecSLAM, we mainly consider two kinds of typical data: laser data from the laser scans and acceleration data from the Inertial Measurement Unit (IMU). Instead of running SLAM locally in a traditional mechanism, we immediately pack the sensory data and offload it to a certain edge server. When it accepts a data package, the edge server launches the SLAM execution and obtains a map corresponding to it. For clarity, we identify this map as *local map* or *robot map*, indicating a direct mapping to a robot's raw data. For an edge server that serves multiple robots, a sequence of the local maps are generated and will be merged in situ to an *edge map*. This merging process is referred to as *preparative fusion*, as it is a preparatory action towards *global fusion* in the cloud. All the edge maps from distributed edge servers are finally merged in a centralized cloud to calculate the global map.

B. Robots: Data Collection and Packing

As discussed in Section II-B, one of the performance bottlenecks is the excessive computing latency on robots, which comes from the contradiction between resource-hungry SLAM computation and resource-constrained on board processors. To alleviate this, we propose to reserve only the lightweight data collection procedure on robots. Specifically, in our implementation, the sensory data to be collected are laser data and IMU data. The former is a structure named sensor_msgs::LaserScan, including timestamp, range data, angular distance and so on, while the latter contains information for coordinates' transformation. Though our current deployment only takes these two types of sensory data into account, it is convenient to accommodate additional formats in RecSLAM.

To offload the SLAM computation, we pack these raw data and transfer them to edge servers through Robot Operating System (ROS). Particularly, each robot establishes a connection and offloads its data to exactly one edge server, while each edge server can serve multiple robots at the same time. Here we assume that these connections are relatively well and stable, which can be relevant [28], [29] provided that they are within the same LAN. Therefore, the mobility of robots will not influence the data transmission and the
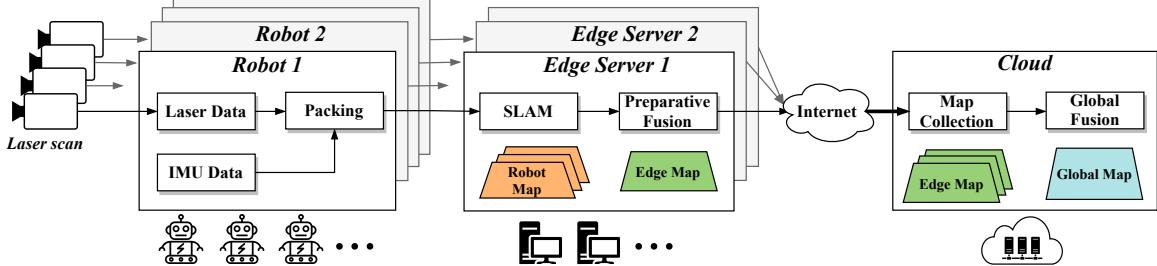
Fig. 6: RecSLAM architecture overview. Each robot individually collects its sensory data and directly packs and transfers to a dedicated edge server. Each edge server runs SLAM using the received data and performs a preparative fusion to aggregate an edge map with its owned robot maps. All the edge maps will be sent to a centralized cloud for global map fusion.

consequent procedures. We note that in realistic deployment the robot-edge communication may be fluctuated and even fail, which raises robustness issues in multi-robot SLAM system. To address that, we can leverage multi-access mobile edge computing [30]–[32] to enable partial offloading, where a robot can transfer its data to multiple edge servers to ensure service availability and improve system performance.

C. Edge Servers: Preparative Fusion for Edge Maps

The robots can access a set of edge servers in proximity via 5G/WiFi connections, and the edge servers, take charge of the SLAM computation and assisting the map fusion. Each edge server accepts raw data from multiple robots nearby and sends its result to a cloud.

The SLAM computation is inherently tightly coupled. If it is forced to split, the explosive complexity would lead to additional overhead in handling the graphic algorithms. Therefore, we treat the SLAM computation as a whole by encapsulating it as a function module for processing the raw data from a robot. In this case, the input to SLAM is collected from robots, and the output will be passed to map fusion. In our prototype, we develop the SLAM module with GMapping [33], a popular lightweight open-source SLAM implementation. For further acceleration, we manage a pool of execution instances to enable parallel and asynchronous serving, i.e., launch a separated process to compute SLAM whenever a robot's raw data arrive.

The results of these processing threads are called robot maps, which will be merged via a fusion module. We refer to this module as preparative fusion, in contrast to the global fusion on the cloud. Particularly, the preparative fusion targets at the robot maps on the edge server and perform fusion by first checking the overlapping degree of robot maps and next merging those overlapping ones. We design to separate such a preparatory process from the global fusion for two reasons. On the one hand, by sinking the fusion workload to the edge servers, we can shrink the computing latency by avoiding excessive delay-significant transmissions to the remote cloud. On the other hand, fusing the robot maps on an edge server can effectively deduplicate the redundancy across overlapping robot maps and thus reduce the data size of edge maps, for which the communication overhead between edge servers and cloud is saved.

D. Cloud: Aggregated Fusion for Global Map

The cloud is responsible to collect all edge maps from edge servers. We designate the cloud as the destination of final merging because a large number of downstream applications of SLAM are deployed on the cloud. However, if some service needs to deploy a SLAM-based task on end devices, we can expediently assign an edge server to finish global fusion and push the result to the specific device.

In the robot-edge-cloud architecture of RecSLAM, each edge server runs its SLAM and preparative fusion individually as long as it receives the robot data, which implies a parallel and distributed processing paradigm among the edge servers. To optimize the performance in such workflow, we need to carefully decide on a robot data processing assignment strategy with a target of balanced workload distribution among the edge servers to avoid the straggler effect. Nevertheless, this problem is non-trivial in its two-fold challenges. First, the overlapping degrees (i.e., information redundancy) across the robot maps (wrt. the robot's sensory data) vary greatly among the robots given the mobility and randomness of the robots' walk, while they can largely impact the computing latency of SLAM on edge servers and the data size of the fused edge maps. Both these two metrics are key performance contributors towards total execution time. Second, the network conditions between the robots and cloud can be heterogeneous. This requires the robot data processing offloaded from the robots to the edge servers to be aware of the diverse network connections. To overcome these challenges, we design an efficient optimization framework for robot-edge collaborative processing in the next section.

IV. ROBOT-EDGE COLLABORATION OPTIMIZATION ALGORITHMS

This section concentrates on scheduling the data flow between robots and edge servers in order to minimize end-to-end multi-robot SLAM latency. We first formulate the cost optimization problem in terms of the latency expired by robots, edge servers and cloud server. Next, we address it via a two-step workflow, i.e., balanced robot data grouping and network-aware edge offloading.

A. Problem Formulation

We consider a cluster of R robots, where each robot r can walk freely or with the user-defined routes, and pushes its

(a) The computing time of the map fusion procedure in multi-robot SLAM with varying number of input maps.

(b) The data size of two input maps and the corresponding merged map with varying overlapping degrees.

(c) Illustration of the map fusion procedure with two example input maps. The blue area indicates the overlapping part of the inputs.
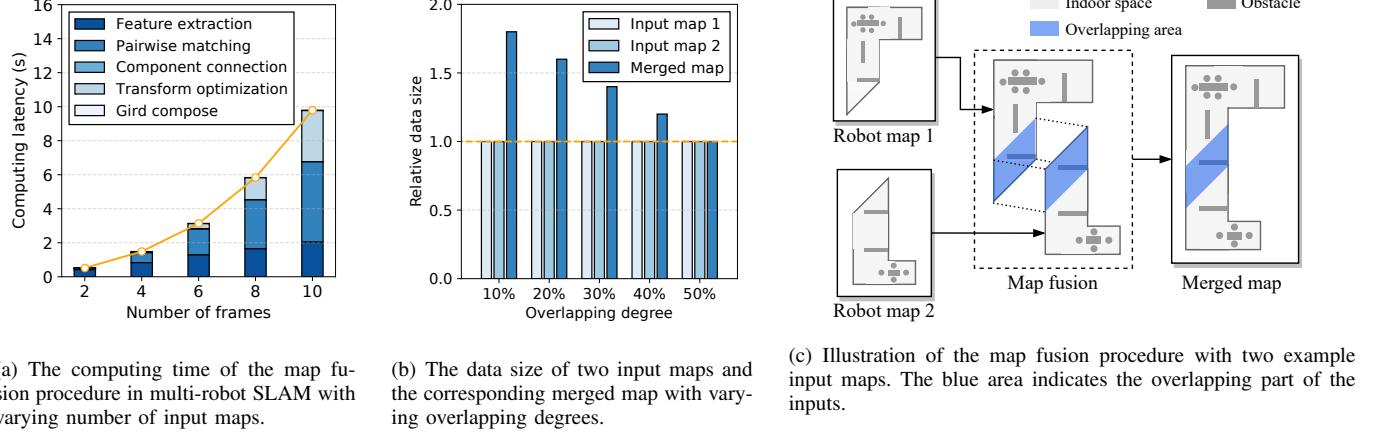
Fig. 7: Analysis of the map fusion procedure. The subfigure (a) shows the computing time of map fusion procedure with respect to the five functions that it applies: feature extraction, pairwise matching, component connection, transform optimization, and grid composing. The subfigure (b) observes that as the overlapping degree increases, the data size of the merged map decreases. Note that the two inputs are of equal data size, and the overlapping degree is defined by the percentage of the overlapping area out of the total areas. The subfigure (c) illustrates a map fusion procedure with two robot maps, where it first checks the overlapping area of the inputs via their feature information, and next connects these overlapping parts and constructs a merged map. The larger the overlapping area, the smaller the data size the merged map has.

sensory data to a specific edge server e among the E edge servers. To obtain a robot r's robot map processed with the assist of an edge server e, it counts the time for data packing, transferring and frame transformation on e:

$$t_r = t_r^{\text{pack}} + t_{r \to e}^{\text{trans}} + t_e^{\text{frame}}. \tag{1}$$

Moving forward the data flow, calculating the edge map on an edge server e needs to wait until all its owned robot maps ready and fuse them in situ, which takes the time:

$$t_e = \max_{r \in e} t_r + t_e^{\text{fuse}}, \tag{2}$$

where $r \in e$ means the robots whose data are assigned to the edge server e, t_e^{fuse} represents the fusion latency in edge server e. From a cloud view, it strives to aggregate all the edge maps from edge servers to reckon the global map, which takes the time of receiving all edge maps and performing global fusion:

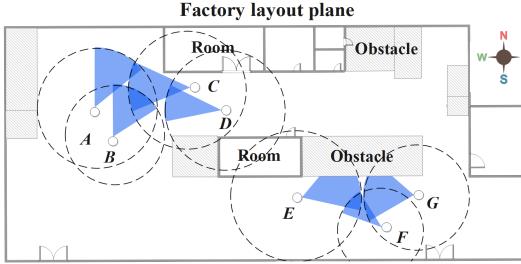$$t_c = \max_e (t_e + t_{e \to c}^{\text{trans}}) + t_c^{\text{fuse}}. \tag{3}$$

Essentially, t_c is the total execution time that characterizes the duration from raw data collection to global map completion, which is exactly the optimizing objectives of RecSLAM. t_c^{fuse} represents the fusion latency in cloud. Therefore, we can derive the execution latency optimization problem as:

$$\min \quad t_c, \tag{4}$$
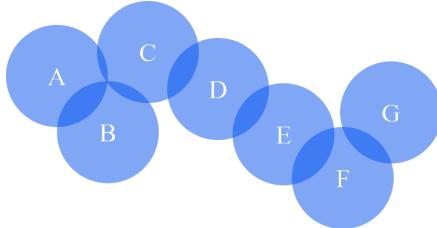$$\text{s.t.} \quad (2), (3).$$

Discussion. The formulation provides an abstracted form of the problem with various details hidden. Among them, we mainly consider two factors. One is the computation side, where we expect the assignment from robots to edge servers is relatively even such that the effect of parallel processing is maximized. The rationale behind can be seen in Fig. 7(a), where the total computing time of map fusion grows

quadratically as the number of frames linearly increases. In Fig. 7(a), we further break down map fusion procedure into five successive functions: *feature extraction* extracts a frame' edge features, and next *pairwise matching* performs feature points matching between different data frames; *component connection* and *transformation optimization* functions intend to solve the pose, and finally *grid compose* is performed to fuse the frames. We observe from function-wise latency that this trend primarily comes from the pairwise matching function, which performs $k(k-1)/2$ times of pair-wise checking given k frames. Therefore, if excessive robot data is assigned to a few edge servers, the imbalance workload distribution will lead to a much higher completion time of the total process. Reflecting on Equation (3), it is for minimizing $\max_e (t_e + t_{e \to c}^{\text{trans}})$.
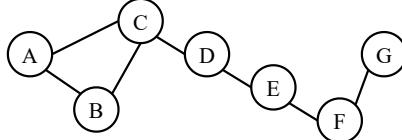
The other factor is the communication side that focuses on the data size of edge maps. This metric, as we observe in measurements, is highly sensitive to the overlapping degree, i.e., the ratio of the overlapping area to the total area of input maps. To illustrate that, Fig. 7(b) plots the data size of the corresponding edge map with varying overlapping degree and Fig. 7(c) depicts how map fusion works. In Fig. 7(b), we take two robot maps of equal data size as input, and control the content of them to adjust the overlapping area. The result reports that as the overlapping degree increases, the edge map's data size shrinks. Particularly, a 50% overlapping degree of two inputs indicates they are fully overlapped, and thus the obtained edge map is exactly one of the inputs and has the same data size of a single robot map. Fig. 7(c) explains this impact visually, where the blue area indicates the overlapping part. The larger the overlapping area, the higher the overlapping degree, and thus the smaller data size the edge map has. This observation drives us to assign robot maps with as higher overlapping degree as possible together, so as to yield smaller edge maps and save the transmission cost between

(a) An example smart factory scene with multiple robots, where some robots, e.g., robot A and B, have overlapping scan fields.

(b) The abstracted robot maps based on the robots' scan fields in (a). Each circle represents a robot map and the overlapping area means the two robots' scan fields have intersection (i.e., data redundancy between the robot maps).

(c) The corresponding graph of the maps in (b). A vertex is a robot map and a link is the overlapping relation between robot maps, with the overlapping degree as its weight.

Fig. 8: An example abstraction flow of the overlapping graph. For the robots in the factory scene (a), we first abstract robot maps in (b) and then derive the overlapping graph in (c).

edge servers and the cloud.

Overall, accommodating both computation and communication aspects to optimize the problem (4) is inherently complicated. Essentially, it is hard to solve the optimal solution fast due to its combinatorial nature. To achieve efficient solving, we intend to decouple it into two sub-problems by separately accounting for robot data grouping and robot data offloading. On the one hand, considering the characteristics of map fusion performed by SLAM, robot data grouping for workload-balancing needs to ensure that the number of nodes in each group tends to be uniform, while the in-group overlapping degree is as large as possible, so as to avoid the straggler effect due to imbalanced processing workload assignment and reduce the total size of fused edge maps. On the other hand, bandwidth conditions of the edge servers may be different, so network-aware offloading optimization for efficient robot-to-edge data transmission and computation is also desired.

B. Balanced Robot Data Grouping

The robot data grouping targets at dividing the V robots' SLAM sensory data into N groups (i.e., N edge servers) with the objective of evenness. Here evenness means the number of

---

Algorithm 1 Balanced robot data grouping algorithm

Input: The overlapping graph $\mathcal{G}$ (with V vertices),
The number of edge servers N,
The neighboring function $\mathcal{N}_C(v)$ that returns the neighbor vertices of v in a set C
Output: Grouping result $\langle P_1, P_2, \cdots, P_N \rangle$
1: Initialize an empty buffer set R and an candidate set C
2: Assign all vertices in $\mathcal{G}$ to C, set exit $flag = 0$
3: Initialize empty grouping result $\langle P_1, P_2, \cdots, P_N \rangle$
4: **repeat**
5:   **if** R is empty **then**
6:     **for** $i = 1, 2, \cdots, N-1$ **do**
7:       Select vertex $v_{\min}^{\text{neigh}}$ from C such that $v_{\min}^{\text{neigh}}$ has the least number of neighbors
8:       $P_i \leftarrow P_i + \{v_{\min}^{\text{neigh}}\}$
9:       $C \leftarrow C - \{v_{\min}^{\text{neigh}}\}$
10:       $R \leftarrow R + C \cap \mathcal{N}_C(v_{\min}^{\text{neigh}}) - \{v_{\min}^{\text{neigh}}\}$
11:       **if** $|C| \leq \frac{V}{N}$ **then**
12:         set $flag = 1$ and break
13:       **end if**
14:     **end for**
15:   **else**
16:     **for** $i = 1, 2, \cdots, N-1$ **do**
17:       Select vertex $v_{\max}^{\text{gain}}$ from C such that $v_{\max}^{\text{gain}}$ has the maximum gain using Equation (5)
18:       $P_i \leftarrow P_i + \{v_{\max}^{\text{gain}}\}$
19:       $C \leftarrow C - \{v_{\max}^{\text{gain}}\}$
20:       $R \leftarrow R + C \cap \mathcal{N}_C(v_{\max}^{\text{gain}}) - \{v_{\max}^{\text{gain}}\}$
21:       **if** $|C| \leq \frac{V}{N}$ **then**
22:         set $flag = 1$ and break
23:       **end if**
24:     **end for**
25:   **end if**
26: **until** $flag = 1$
27: $P_N \leftarrow C$
28: Apply Algorithm 2 to optimize the grouping result
29: **return** $\langle P_1, P_2, \cdots, P_N \rangle$

---

robots across groups should be approximately equal, in order to achieve balanced processing workloads among the edge servers later on. Besides, for minimizing the size of edge maps after fusion, the robot data within the same group is desired to yield robot maps with a higher overlapping degree as much as possible. We accomplish these goals by firstly abstracting an overlapping graph to shape the relations between robot maps.

Graph abstraction. Fig. 8 illustrates how we abstract an overlapping graph. Given the robots in Fig. 8(a)'s factory layout plane, we can collect their position and localization information in a bottom-up manner: the robots report their meta sensory data to the nearby edge servers and the edges push these data to the cloud to infer the global overlapping information. In many well-planned deployments, this procedure can be even accomplished offline as long as we know the robots' scanning routes prior to runtime. With the information, we can therefore represent the robots' corresponding maps in Fig. 8(b), where each one reflects the area that a robot's

scanning field covers. It is represented by circle, because the robot can scan the surrounding 360-degree during the movement. Let a pair of maps with an overlapping area share a link, we abstract a graph in Fig. 8(c), named as the overlapping graph. A vertex in such a graph is a robot map, whereas a link quantifies the overlapping degree between two robot maps. Particularly, two vertices are not connected if there is no overlap between the two robot maps. We denote the overlapping degree as $w(u, v)$, which represents the weight of the link between vertex u and v.

Balanced grouping algorithm. Using the overlapping graph model, we observe that the robot map grouping problem can be approximated to a graph clustering variant. Concretely, we can operate on the overlapping graph to decide a balanced grouping with maximal in-group overlapping degree[1]. To accomplish this, we develop a balanced grouping solution in Algorithm 1 on the basis of the balanced graph partitioning method [34], [35]. The key idea is first to generate a feasible grouping plan and next use Tabu searching for further optimization. Particularly, the way that Algorithm 1 finds an initial grouping is to iteratively put the vertex with maximum performance gain to a group such that for each group the in-group overlapping degree is maximized, and the Tabu searching optimization is to avoid a local optimum result and will be described in Algorithm 2 detailedly.

The input of Algorithm 1 are an undirected graph $\mathcal{G}$ (with V vertices), the number of groups (i.e., the number of edge servers) N, and a neighboring function $\mathcal{N}(v)$ that returns the neighbor vertices of v. The output is the grouping result $\langle P_1, P_2, \cdots, P_N \rangle$, where P_i is a set that includes the vertices assigned to i-th partition.

At the beginning of Algorithm 1, we initialize an empty buffer set R and an empty candidate set C, and immediately add all vertices in the overlapping graph $\mathcal{G}$ to C. We also initialize empty grouping result $\langle P_1, P_2, \cdots, P_N \rangle$, and then goes into an iterating process. For each iteration, we first check whether the buffer R is empty: if empty, it means there is no suitable candidate related to vertices in the current group P_i, thus we add the vertex $v_{\min}^{\text{neigh}}$ with the least number of neighbors from the candidate set C, and put it to P_i (Line 7-9). The vertices associating with $v_{\min}^{\text{neigh}}$, obtained by $\mathcal{N}(v)$, will be added to R for the coming assignment (Line 10). Whenever a node is deleted from C, we check whether $|C| \leq \frac{V}{N}$. If the condition is satisfied, which means the difference of nodes' number in any two groups is less than 2, it will set the exit $flag$ as true and jump out of the loop (Line 11-13). It should be noted that these neighboring vertices $\mathcal{N}(v)$ have never been visited, i.e., they have not been assigned to a specific group. If R is not empty, there remains vertices in the buffer, and we will select the vertex $v_{\max}^{\text{gain}}$ with the maximum gain from candidates C and move it to P_i. The gain is estimated using Equation (5): given a vertex v in C, we reckon the sum of $w(u, v)$ in P_i and C, respectively, and calculate their difference. Such difference values actually measures the weight change brought by the

---

[1]Particularly, in-group overlapping degree means the geometrical sum of $w(u, v)$ within the same group, while out-group degree is the sum of $w(u, v)$ between two specific groups.

Algorithm 2 Tabu-searching-based optimization

Input: The overlapping graph $\mathcal{G}$,
    Initial grouping $\langle P_1, P_2, \cdots, P_N \rangle$,
    The maximum iterations times κ,
    The maximum tabu list length τ
Output: Optimized grouping result $\langle P_1^*, P_2^*, \cdots, P_N^* \rangle$
1: Initialize an empty tabu list T of vertex pairs, an empty list G of obtained grouping results, and an empty list S of vertex pairs to be swapped
2: $\langle P_1^*, P_2^*, \cdots, P_N^* \rangle \leftarrow \langle P_1, P_2, \cdots, P_N \rangle$
3: **repeat**
4:   **for** vertices $v, u \in \mathcal{G}$ **do**
5:     **if** v and u are not in the same group **then**
6:       Add (v, u) to S
7:       Swap the locations of v and u to obtain a new grouping $\langle P_1', P_2', \cdots, P_N' \rangle$
8:       Add $\langle P_1', P_2', \cdots, P_N' \rangle$ to G
9:     **end if**
10:   **end for**
11:   $k \leftarrow \arg\min\limits_{i, g_i \in G} \{f(g_i)\}$ using Equation (6)
12:   **if** $f(G_k) < f(\langle P_1^*, P_2^*, \cdots, P_N^* \rangle)$ **then**
13:     **if** $S_k \in T$ **then**
14:       Remove S_k from T
15:     **end if**
16:     $\langle P_1^*, P_2^*, \cdots, P_N^* \rangle \leftarrow G_k$
17:   **end if**
18:   **if** $S_k \notin T$ **then**
19:     Add S_k to T
20:   **end if**
21:   **if** $|T| > \tau$ **then**
22:     Remove the first element of T
23:   **end if**
24:   $\langle P_1, P_2, \cdots, P_N \rangle \leftarrow G_k$
25: **until** Iterating for κ times
26: **return** $\langle P_1^*, P_2^*, \cdots, P_N^* \rangle$

---

vertex migration. The greater the difference, the greater the possibility of the point being selected.

$$\text{gain}(v) = \sum_{u \in P_i} w(u, v) - \sum_{u \in C \setminus \{v\}} w(u, v). \qquad (5)$$

The principle behind exploiting the gain formula is to make each in-group overlapping degree as large as possible, while the out-group is as small as possible. Again, we assign $v_{\max}^{\text{gain}}$ to the current group P_i and remove it from C, and the neighboring vertices $\mathcal{N}(v_{\max}^{\text{gain}})$ will be added to the buffer R (Line 17-20). In this step, we still need to check whether $|C| \leq \frac{V}{N}$ is met (Line 21-23). The above iterating processing continues until exit $flag$ is true, which indicates that all the vertices have been allocated in a balanced manner. The remaining vertices in C will be directly assigned to N-th group P_N, and therefore we obtain a feasible initial grouping result (Line 26).

At the same time, $\mathcal{N}(v)$ will be added to R. Finally, when all the vertices are visited, we can get an appropriate grouping result (Line 27). However, this result is based on heuristic

assignment and may terminate at a local optimum. To avoid such circumstances, we further apply a Tabu searching to seek a better grouping result (Line 28), as described in Algorithm 2.

Tabu-searching-based optimization. The key idea of Algorithm 2 is to use a tabu list to save the local optimal solution, to avoid falling into the same result. The input of the algorithm is the overlapping graph $\mathcal{G}$, the initial grouping result $\langle P_1, P_2, \cdots, P_N \rangle$, the maximum iterations times κ and the maximum tabu list length τ. The output is an optimized grouping result $\langle P_1^*, P_2^*, \cdots, P_N^* \rangle$.

Algorithm 2 begins with initializing an empty tabu list T with length τ, an empty list G of obtained grouping results, and an empty list S of vertex pairs to be swapped. We instantiate the expected grouping $\langle P_1^*, P_2^*, \cdots, P_N^* \rangle$ with the initial grouping $\langle P_1, P_2, \cdots, P_N \rangle$, and goes into an iterating process. For each iteration, we first construct all possible swapping pairs (Line 4-10): find vertices v and u that are not in the same group, add them to S, apply a swapping to obtain an updated grouping $\langle P_1', P_2', \cdots, P_N' \rangle$, and put the updated grouping to G. With these swapped candidates, we then expect to filter those that improve the quality of grouping. Specifically, we define a fitness function $f(\cdot)$ in Equation (6):

$$f(\langle P_1, P_2, \cdots, P_N \rangle) = \sum_{u \in P_i, v \in P_j, i \neq j} w(u, v). \quad (6)$$

The *fitness* defines the suitability of the grouping by calculating the sum of weighted edges between different groups in Equation (6). It is designed for weighing the sum of edges in an overlapping graph, and we can therefore use it in our optimization on workload grouping. The smaller the fitness value, the better the grouping result. In Algorithm 2, we find the swapping vertex pair, noted with index k, such that it has the smallest fitness within all candidates in G (Line 11). If this vertex pair S_k has smaller fitness than the current optimized counterpart's ($\langle P_1^*, P_2^*, \cdots, P_N^* \rangle$), we will update this optimized result (if it is exactly in the tabu list T, just remove it from T). Otherwise, S_k is accepted and will be added S_k to the tabu list (Line 18-20), which means we will accept the local optimal solution and prevent it from being visited again, unless the amnesty rules are met. If the tabu list T is overlength than the preset maximum length τ, we will remove the first element, the one that is the most out-of-date, from T, based on the first-in-first-out principle. The current grouping will also be updated will k-th grouping. The above iterations continue until reaching the maximum κ times, and the final optimized grouping $\langle P_1^*, P_2^*, \cdots, P_N^* \rangle$ will be returned.

Complexity analysis. Assuming an overlapping graph has V vertices and L links and we need to divide N groups, the time complexity of the balanced grouping algorithm is $O(V + L)$. In the worst case, each partition has to traverse V/N vertices. The operation of selecting vertices is regarded as $O(1)$, because this part can be optimized by maintaining the tree on the data structure. Moreover, the optimization requires traversing the edges, so the total complexity is $O(V/N + L) = O(V + L)$.

Algorithm 3 Resource-aware edge offloading algorithm

Input: The grouping result $\langle P_1, P_2, \cdots, P_N \rangle$,
    The edge servers list $\mathcal{E} = \langle 1, 2, \cdots, N \rangle$,
    Bandwidth B_j between edge server j and the cloud,
    Latency prediction model $L_j(P_i)$ that returns the computing latency of P_i on edge server j,
    The function $\mathcal{D}(P_i)$ that returns the output data size of partition P_i

Output: Offloading plan $\langle \theta_1, \theta_2, \cdots, \theta_N \rangle$

1: **for** $i = 1, 2, \cdots, N$ **do**
2:     $j \leftarrow \arg\min_{j \in \mathcal{E}}(L_j(P_i) + \frac{\mathcal{D}(P_i)}{B_j})$
3:     $\theta_i \leftarrow j$
4:     Remove edge server j from $\mathcal{E}$
5: **end for**
6: **return** $\langle \theta_1, \theta_2, \cdots, \theta_N \rangle$

C. Resource-Aware Edge Offloading

After robot data grouping, we have several groups of robot data and need to decide their placement to the proper edge servers for edge map fusions to minimize the total collaborative processing time. Since the data transmission from the robots and edge servers can be a bottleneck and the network conditions of the edge servers can be different, the decision needs to tailor for the available bandwidth. Moreover, the computing capacity of the edge servers can also be heterogeneous. Therefore, we propose a resource-aware edge offloading strategy in Algorithm 3.

Algorithm 3 works in a greedy fashion and the input is the robot data grouping result $\langle P_1, P_2, \cdots, P_N \rangle$ from Algorithm 1, the edge servers list $\mathcal{E} = \langle 1, 2, \cdots, N \rangle$, and B_j that records the bandwidths between edge server j and the cloud. Besides, two functions are also employed: the latency prediction function $L_j(P_i)$ that returns the computing latency of P_i on edge server j, the function $\mathcal{D}(P_i)$ that returns the data size of partition P_i. The output is the offloading strategy $\langle \theta_1, \theta_2, \cdots, \theta_N \rangle$, where θ_i logs the edge server that grouping i will be placed. In Algorithm 3, we iteratively select a group P_i and pick an edge server j such that the robot-to-edge offloading (including transmission and computation) latency is minimized (Line 2). Particularly, we use $L_j(P_i)$ to estimate the computing latency on the edge server and $\frac{\mathcal{D}(P_i)}{B_j}$ to calculate the transmission time. Since this latency simultaneously tackles computing aspect, i.e., $L_j(P_i)$, and the bandwidth B_j, it is aware of the available resource in both computation and communication, and can support heterogeneous cases. The selected edge server j that minimizes the latency will be assigned to θ_i and be removed from the available edge server list $\mathcal{E}$. We carry out this process until all edge servers are assigned with a group, and return the final offloading plan $\langle \theta_1, \theta_2, \cdots, \theta_N \rangle$.

Complexity analysis. In our implementation, we use an offline profiling method to obtain the estimation of computing time and output data size, which will be detailed in Section V. Using the profiling result, the overhead of calculating the computing/transmission time of each server is $O(1)$, the complexity of Algorithm 3 is $O(N)$, which is linear and fast.
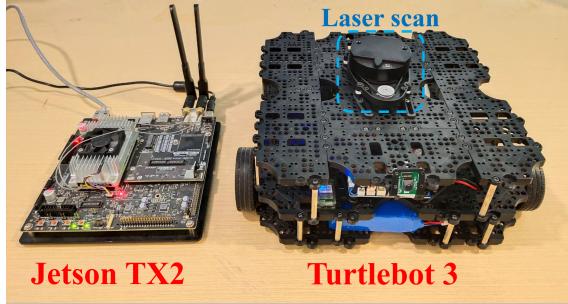
Fig. 9: Our prototype employs Turtlebot 3 as the robot and Jetson TX2 as the edge server. The Turtlebot is equipped with a RPLIDAR A1 as the laser scan and a Raspberry Pi as the processing module.
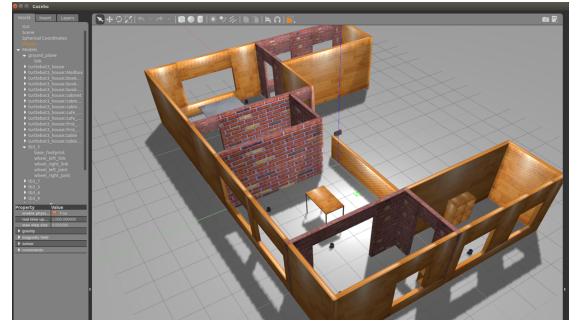

Fig. 10: The simulated scene of Fig. 1(a) in the Gazebo [25] simulator, where robots and edge servers are placed inside to emulate multi-robot SLAM applications in physical edge scenarios.

V. Implementation

We have implemented RecSLAM in both realistic prototype and simulation platform. The proof-of-concept prototype is built with Jetson TX2 and Turtlebot 3 as shown in Fig. 9, while the simulations are carried on Gazebo [25], a high-precision simulator of robotics emulations. Fig. 10 shows the graphical interface of the scene in Fig. 1(a).

SLAM integration. We adopt GMapping as the SLAM kernel on edge servers. However, our system can support different SLAM toolkits as long as the robot can provide the necessary data. Each time a robot offloads data to the edge, the edge server will create a dedicated GMapping instance for the robot to calculate the local map with a specific topic. For example, when two robots offload data to an edge server at the same time, the topics of them are tb4_0 and tb4_2 respectively. At this time, there will be two GMapping instances on the edge that subscribe the data with tb4_0/ and tb4_2/ namespace, and then publish the occupancy grid map with tb4_0/map and tb4_2/map topic respectively.

To leverage the proposed scheduling algorithms over GMapping, we implement them in a centralized manner. Specifically, we select one of the available edge servers as the coordinator to be responsible for collecting robots' maps information, estimating the overlapping degrees, and running scheduling algorithms. According to the scheduling result, the coordinator will route the robot data to corresponding edge servers, and orchestrate them to collaboratively run map fusion tasks. The algorithms can also be implemented distributionally, where a tailored mechanism is desired for global information sharing and consensus management, and we leave it as a future work.

Robot-edge communication. The communications among robots, edge servers, and the coordinator are conducted via ROS[2]. They all communicate using the publisher-subscriber model. In such a mechanism, the subscriber node will first register on the node master, and then the node master will search among the registered publishers, find the matching publisher, and finally the publisher and the subscriber will establish a communication channel.

The communications involve multiple subscribing topics. In RecSLAM, the data format released by the robot cluster is tbn_x, where x is the specific robot number. The data format of the GMapping instance subscripter in edge is tbn_x, and the data with topic tbn_x/map will be published after processing. The Map Fusion module receives coon_x/map and publishes the data with edgem_x/map at the same time.

For coordinator, it will receive tbn_x/map's data. The workload grouping algorithm in the coordinator will split the robots into groups, and then the offloading algorithm will generate allocation decisions and publish the data with the topic coon_x/map. The communication mechanism based on ROS ensures the reliability of communication between robot and edge.

Parameter profiling. The coordinator derives the overlapping degree matrix using a modified package called weight_cal. It is based on multirobot_map_merge, a generic package in ROS. It can read multiple maps in a batch, extract their features, perform matching checking, and outputs a rough overlapping degree matrix. To run it in a lightweight style, we invoke this package periodically, i.e., calculates and saves the overlapping degree matrix of the robot maps in a preset regular frequency.

To estimate the time of preparative fusion t_e^{fuse} at the edge server, we employ an offline profiling mechanism that trains regression models for performance prediction. In our deployment, we carry out profiling on Turtlebot 3 and record the computing latency of map fusion with respect to the number of data frames. Fig. 11 illustrates the regression result. An interesting observation of the figure is that the computation latency rises in a quadratic trend as the frame number increases from 2 to 12, This is because the computation of merging requires pairwise matching of data frames, and the number of matches has an exponential relationship with the number of frames.

For the transmission latency between robots, edge servers and the cloud, there exists many network bandwidth measure tools available, which enables the real-time transmission time estimation based on the data size of the input frames. In

---

[2]ROS is an operating system specifically designed for manipulating robotics. It provides a loosely coupled distributed communication framework and rich libraries of various functional packages, making it convenient to carry out robotic system development.
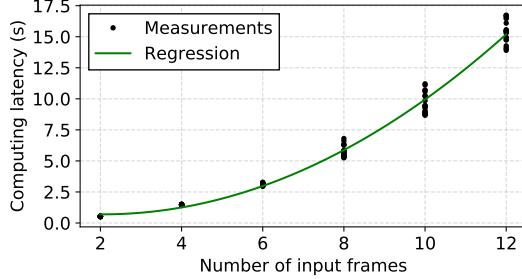
Fig. 11: The regression result of SLAM computing latency on the Jetson TX2. The dots are the measurements and the curve is the regression.

RecSLAM, we utilize the principle of tolerance and exclusion formula to estimate the output size of preparative fusion. Since the number of robot maps is relatively small, the estimation error of the overlapping area is within a tolerable range.

VI. EVALUATION

This section evaluates RecSLAM in both a simulator platform and a prototype deployment. Particularly, we focus on the feasibility of global map construction and the end-to-end execution time of running a multi-robot SLAM process.

A. Functional Verification

We first conduct functional verification of RecSLAM based on a small-scale realistic prototype. Specifically, we mainly focus on collecting realistic robotics measurements and examining the effectiveness of the hierarchical map fusion procedure, and thus providing necessary data to facilitate subsequent larger-scale simulation experiments. We use the TurtleBot3 Waffle Pi mobile robot for laser data collection and the Jetson TX2 as the edge server. The TurtleBots are all installed with Raspberry Pi 3b+ as their processing module, which has 1.4GHz ARM Cortex-A53 CPUs and 1GB LPDDR2 memory. The Jetson TX2 is equipped with NVIDIA Denver and ARM Cortex-A57 CPUs, an NVIDIA Pascal GPU with 256 CUDA-cores, and 8 GB LPDDR4 memory. All the robots and the edge server are connected via the 802.11ac wireless signal and in the same LAN. The cloud server employs an instance (8vCPU, 16GB memory, Ubuntu 16.04) that locates at the same available zone with the robots and their physical distance is about 200km. The testbed scene is built in a $6 \times 5.5 m^2$ conference room, where we manually add walls and obstacles, as shown in Fig. 12. The linear velocity of the robots is $0.20 m/s$, and the angular velocity is set to $0.20 - 0.80 m/s^2$ during the runtime. Although RecSLAM's design allows robots to walk freely, we have set determined routes for robots in advance for the sake of simplicity and experimental repeatability. As the robots move around, the equipped laser sensors perceive the surrounding environment at a fixed frequency and continue collecting sensory data.

Fig. 12 presents the mapping results of our prototype. Particularly, Fig. 12(b), (c) and (d) visualize the robot maps that are computed based on the robots' sensory data, where

we can see that the robots pave three different routes on the left, center and right of the scene, respectively. Overlapping areas can be observed from these robot maps, e.g., robot map 1 in (b) and robot map 2 in (c) share the same area at their bottom. Fig. 12(e) shows the obtained edge map that is fused from the robot maps, where the complete scene is sketched and the obstacles' details are reserved.

To further demonstrate RecSLAM's effectiveness, we make additional simulation experiments and build several virtual scenes in Gazebo [25], a popular platform[3] that provides high-precision and near-physical experimental environments. Specifically, we design typical apartment indoor scenes in Gazebo's virtual environments and place robots and edge servers to emulate multi-robot scenarios at the network edge. In all scenes, we distribute and preset routes for robots in advance, and drive them to walk around the apartment. During the runtime of multi-robot SLAM, we record the robots' sensory data and calculate the maps by continuously replaying the collected data. To fully reflect the specifications in our prototype, we configure the same motion parameters, such as velocity and scanning field, and replay the robots' movement for all simulations.

The first scene, as shown in Fig. 1(a), is the main scene for our simulation experiments, which is a $13 \times 18 m^2$ apartment scene with three edge servers (and a varying number of robots on demand). Fig. 1(b) is exactly the global map generated by RecSLAM. We also built another two scenes in Fig. 13 and Fig. 14 to examine the effectiveness in constructing the whole map. The scene in Fig. 13(a) is in a $4 \times 4 m^2$ square and is designed to emulate a symmetrical layout. The subfigures Fig. 13(b)-(e) depicts the robot maps constructed using the laser scan data from the four robots, while the subfigures (f) and (g) are the corresponding edge maps and (h) is the global map in the cloud. Specifically, (f) is obtained by fusing (b) and (c), and (g) is from (d) and (e). The last scene is in a $5 \times 5 m^2$ room as shown in Fig. 14(a), which exhibits an asymmetrical layout. In contrast to the neat robot maps in Fig. 13, the maps in Fig. 14 shows relatively irregular styles due to the scene's asymmetry. Nevertheless, it can be seen from both scenes that the global map after map fusion matches the corresponding scenes with respect to the whole layout and the obstacle positions.

Remark that the global mapping results from RecSLAM and the cloud approach are the same since both of them reserve complete raw scan data from robots and run the same SLAM module. Particularly, RecSLAM improves multi-robot SLAM beyond traditional cloud approach by mitigating the workload of SLAM and fusion to edge servers and achieves much lower execution latency without sacrificing map accuracy.

B. Performance Evaluation and Comparison

Due to the limitations of the robotics hardware and physical space in our laboratory, we focus on verifying the functional feasibility of RecSLAM's multi-robot map fusion. To fully evaluate multi-robot SLAM performance with more robots and

---

[3]Gazebo [25] is a robotics simulator that offers the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments, and has been widely used in the community.

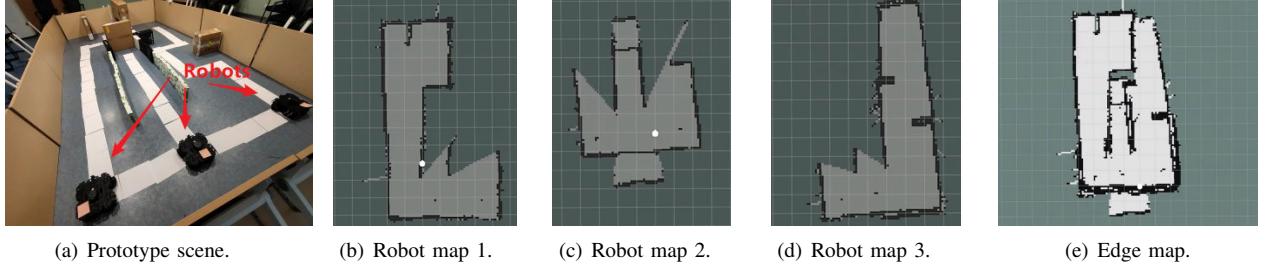(a) Prototype scene.　　(b) Robot map 1.　　(c) Robot map 2.　　(d) Robot map 3.　　(e) Edge map.

Fig. 12: Proof-of-concept prototype experiments with three robots and one edge server in a self-built real scene. Due to the limitations in robotics hardware and physical space, the testbed in (a) is not built perfectly and gaps exist between walls, for which the corresponding global map in (e) has a rough boundary. Despite that, the global map still shows its feasibility by sketching the complete scene and reversing the obstacles' details, which verifies the practicality of RecSLAM.

(a) Scene.　　(b) Robot map 1.　(c) Robot map 2.　(d) Robot map 3.　(e) Robot map 4.　(f) Edge map 1.　(g) Edge map 2.　(h) Global map.
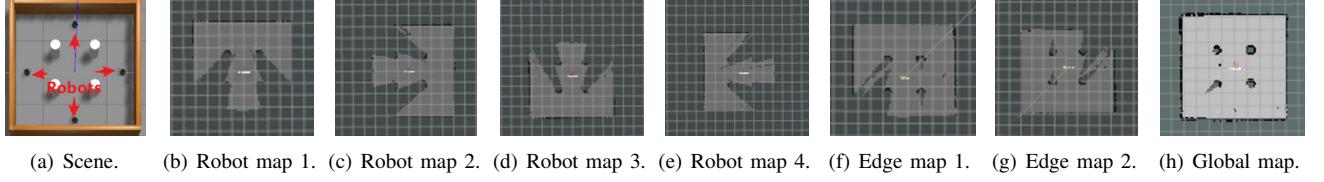
Fig. 13: Simulation results with four robots and two edge servers in a symmetrical apartment scene. The edge servers are not drawn in (a) for simplicity. The robots walk according to preset routes and pass their sensory data to edge servers for multi-robot SLAM computation. The complete global map demonstrates RecSLAM's effectiveness in merging multi-robot sources.
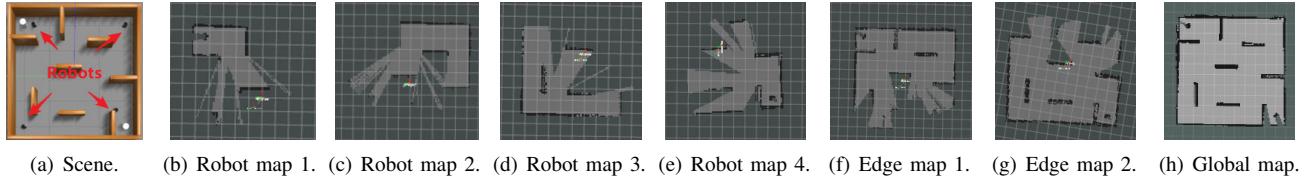

(a) Scene.　　(b) Robot map 1.　(c) Robot map 2.　(d) Robot map 3.　(e) Robot map 4.　(f) Edge map 1.　(g) Edge map 2.　(h) Global map.

Fig. 14: Simulation results with four robots and two edge servers in an asymmetrical apartment scene. The edge servers are not drawn in (a) for simplicity. RecSLAM effectively renders the complete mapping again, and note that the flaws in the global map mean the area that the robots have not scanned.

more edge servers, as well as in larger scenes, we conduct large-scale simulation experiments in Gazebo to demonstrate RecSLAM's effectiveness and efficiency.

Algorithm runtime. RecSLAM consists of two parts, namely the grouping algorithm in Algorithm 1 and the Tabu-searching-based optimization algorithm in Algorithm 2. In order to explore the algorithm's applicability in different situations, we inspect the performance of RecSLAM by breaking down the latency of grouping and Tabu searching in Fig. 15. It should be noted that the undirected graphs generated in the test scenario before are all sparse matrices. Here we use randomly generated matrices for testing.

See Fig. 15, as the robot number increases, the total scheduling overhead climbs. In our scenario, tabu search always contributes the largest cost in the entire algorithm operation since the number of robots is small (< 20). But in dense scenarios, when the number of robots increases, we can find that the grouping overhead becomes larger, even exceeding tabu search. Because grouping is also a heuristic algorithm,

it is greatly affected by the number of machines, but the algorithm runtime is still very small.

Since Tabu search overhead accounts for a relatively large amount of the algorithm runtime in our scenario, we further explored the impact of Tabu search though it is small. It can be seen from Fig. 16 that with the number of robots from 10 to 50, Tabu search can obviously find a better solution, making the weights sum of the cross-group edges smaller.

Performance comparison. Our first experiment targets at comparing RecSLAM with the cloud serving approach. Fig. 17 shows their latency results with varying robot numbers, where RecSLAM enjoys lower latency across all settings. Concretely, RecSLAM reduces 39.31%, 26.50% and 12.96% latency with 6, 8 and 10 robots, respectively. The result demonstrates the advantages came from edge servers' assists. Specifically, the cloud approach lets robots compute SLAM locally and collects all robot maps in the cloud for centralized aggregation, where the workload on robots is always at a high level (Fig. 5). In contrast, RecSLAM forces to transfer the SLAM workload
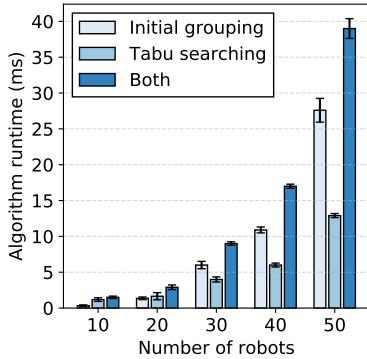
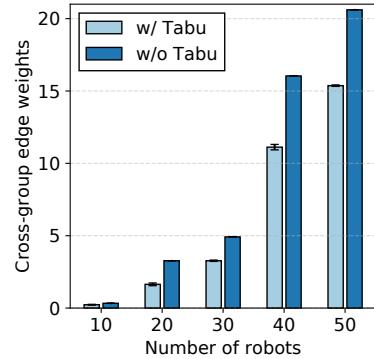Fig. 15: The algorithm runtime of different components in RecSLAM with varying number of simulated robots.

Fig. 16: The optimization effect of RecSLAM's grouping algorithm with and without Tabu optimization. Lower cross-group edge weights is better.
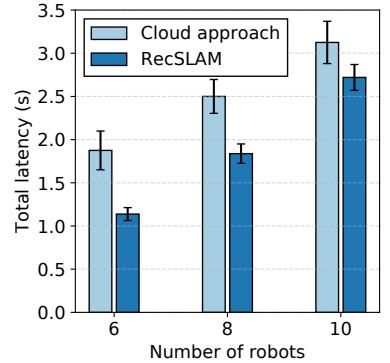

Fig. 17: Total execution time of RecSLAM and the state-of-the-art cloud approach under WiFi environment.
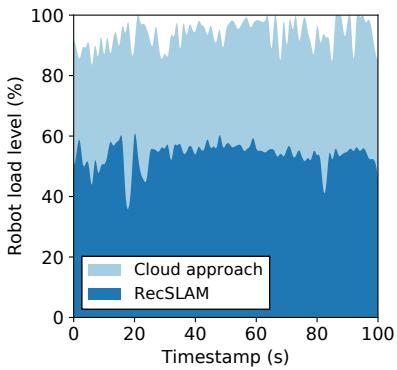

Fig. 18: The recorded load level of the robot under the cloud approach and RecSLAM.
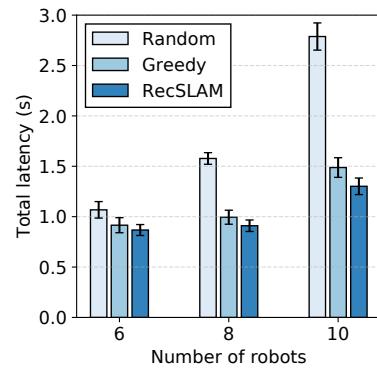

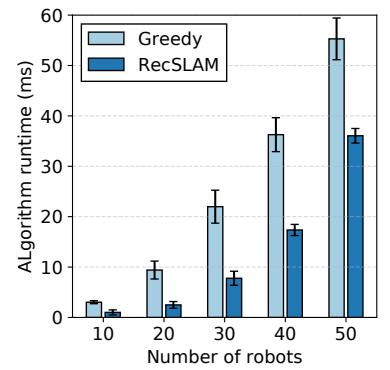Fig. 19: Total execution time of RecSLAM and ablated scheduling policies with varying number of robots.

Fig. 20: The algorithm runtime of RecSLAM and the greedy counterpart with varying number of robots.

and separate part of the fusion tasks to edge servers, therefore alleviating the computing stress on both sides. As we measure during the RecSLAM runtime, the load level in robots records a mean of 53.91% and a minimum of 36.18%, significantly reducing the robot workload as shown in Fig. 18.

To assess the effectiveness of RecSLAM's optimizing algorithms, we compare it with two heuristic counterparts and visualize the achieved latency results in Fig. 19, where Rec-SLAM refers to the corresponding scheduling policy mentioned above. The random algorithm randomly distributes the robots' data to edge servers directly, recording the highest execution time among the three approaches. Particularly, when serving 10 robots, it takes 2.79s, which is 2.15× higher than RecSLAM. The greedy algorithm assigns the robot-edge data flow by first randomly initializing the grouping and next greedily selecting and swapping the robots in different groups. For example, given r robots and n edge servers ($r \geq n$), it first selects any r robots to respectively distribute to the n groups. For the rest operations, we orderly select and then swap robots in different groups such that the overlapping degree with the edge server's existing robot maps is maximized. Such a mechanism partially utilizes the overlapping information, but still lacks a global determination on overall overlapping

degrees. As a result, it yields better partitioning than the random algorithm but is slower than RecSLAM. RecSLAM considers the relations between robots' data by constructing an overlapping graph and scheduling the data flow in a global view. Fig. 19 shows that it reduces up to 53.41% and 12.16% latency over the random and greedy counterparts.

We further examine the algorithm runtime of RecSLAM and the greedy algorithm in Fig. 20 by simulating multiple robots in Gazebo [25] with numbers from 10 to 50. In the figure, we observe that the RecSLAM consumes a smaller overhead than Greedy with an average of 58.71% and up to 12.16% lower latency. Particularly, it takes less than 1ms for 10 robots and only 36ms for 50 robots, indicating that RecSLAM runs lightweight even with a large number of robots.

VII. RELATED WORK

To the best of our knowledge, RecSLAM is the first one that optimizes multi-robot laser SLAM with the assist of edge computing. Only a few works are close to RecSLAM, and we list the most relative ones in Table I.

Multi-robot SLAM processing. The research on SLAM has experienced decades [38], [39]. With years of development and practice, a number of single-robot SLAM algorithms

TABLE I: Comparing RecSLAM with relative edge-assisted SLAM frameworks.

	Multi-robot support	Multi-edge deployment	Task decomposition	Bandwidth adaptability	Laser SLAM
CCM-SLAM [36]	✓				
EdgeSLAM [37]			✓	✓	
Edge-SLAM [28]			✓		
RecSLAM (Ours)	✓	✓	✓	✓	✓

[40]–[47] has been proposed and deployed in real world. For example, ORBSLAM2 [44] can utilize a variety of sensors to collect image data and estimate pose during movement to achieve positioning and mapping, and has been used in underwater scenarios [48], [49] and autonomous driving [50]. Cartographer [46] is a graph-optimized laser SLAM framework, which has been deployed on Google's street view services.

While these single-robot SLAM systems have been widely used, multi-robot SLAM remains to be intractable and still under exploration. A considerable part of the works focuses on the algorithm layer, including multi-sensor fusion [51]–[55] and deep learning assistance [56]–[58], etc. CCM-SLAM [36] proposes a data-sharing mechanism between multiple robots to solve communication problems. CoSLAM [59] proposes a collaborative SLAM system with multiple moving cameras in a possibly dynamic environment. CorbSLAM [60] also uses a centralized method to detect the overlapping area of multiple local maps through DBoW method and uses global optimization through bundling adjustment to fuse these maps. However, a majority of these works implicitly rely on centralized servers, ignoring the practical scenarios of edge deployment.

Edge-assisted SLAM computation. The emerging edge computing [20], [21], [61] is a distributed computing paradigm that sinks the workload from the remote cloud to the servers in physically approximate to the end devices, and therefore significantly shortens the transmission distance and reduces the communication overhead. In recent years, the community has been explored using edge computing to assist SLAM computation. Dey et al. [29] apply a particle filtering algorithm to an edge-cloud architecture and propose a dynamic offloading strategy. However, it only accounts for a single robot scenario without considering transmission bottlenecks and realistic prototype implementation. EdgeSLAM [37] propose an edge-assisted mobile semantic visual SLAM framework, which can perform visual SLAM in real-time. However, incorporating semantic segmentation brings great limitations to the system's scalability. Besides, it does not take multiple edge servers into consideration. Edge-SLAM [28] decouples SLAM from a modularity perspective, with the main focus on resource usage on mobile devices. However, it only involves a single robot and an edge node, and cannot be directly extended to the communication between multi-robot and multi-edge. While these systems intend to optimize SLAM computation from edge computing perspectives, they all focus on the single-robot scenario with peer-to-peer robot-edge synergy, ignoring the emerging multi-robot scenarios. In contrast, our proposed system tackles a more complex SLAM deployment with the mutual presence of both multiple robots and multiple edge servers. Besides developing a collaborative execution framework between robots and edge servers, we further design a coordinating policy to enable high-performance multi-robot SLAM map fusion.

VIII. CONCLUSION

This paper proposes RecSLAM, a multi-robot laser SLAM system under hierarchical robot-edge-cloud architecture. RecSLAM introduces edge servers to embrace the workload of SLAM computation and map fusion, which remarkably reduces the robot computing stress and saves the communication overhead between edge and cloud. To optimize the overall performance, an efficient collaboration framework is further developed to orchestrate the data flow between robots and edge servers tailored to the heterogeneous network conditions. Extensive evaluations with both simulation and real prototype implementation demonstrate the effectiveness and efficiency of RecSLAM over existing solutions.

REFERENCES

[1] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Transactions on robotics*, vol. 32, no. 6, pp. 1309–1332, 2016.

[2] B. Huang, J. Zhao, and J. Liu, "A survey of simultaneous localization and mapping," *arXiv preprint arXiv:1909.05214*, 2019.

[3] S. Huang and G. Dissanayake, "A critique of current developments in simultaneous localization and mapping," *International Journal of Advanced Robotic Systems*, vol. 13, no. 5, p. 1729881416669482, 2016.

[4] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine *et al.*, "Scalability in perception for autonomous driving: Waymo open dataset," in *Proceedings of the IEEE/CVF CVPR*, 2020, pp. 2446–2454.

[5] A. Buyval, A. Gabdullin, R. Mustafin, and I. Shimchik, "Realtime vehicle and pedestrian tracking for didi udacity self-driving car challenge," in *2018 IEEE ICRA*. IEEE, 2018, pp. 2064–2069.

[6] H. Fan, F. Zhu, C. Liu, L. Zhang, L. Zhuang, D. Li, W. Zhu, J. Hu, H. Li, and Q. Kong, "Baidu apollo em motion planner," *arXiv preprint arXiv:1807.08048*, 2018.

[7] C. Badue, R. Guidolini, R. V. Carneiro, P. Azevedo, V. B. Cardoso, A. Forechi, L. Jesus, R. Berriel, T. M. Paixao, F. Mutz *et al.*, "Self-driving cars: A survey," *Expert Systems with Applications*, vol. 165, p. 113816, 2021.

[8] D. Rückert and M. Stamminger, "Snake-slam: Efficient global visual inertial slam using decoupled nonlinear optimization," in *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2021, pp. 219–228.

[9] X. Xiao and R. Murphy, "A review on snake robot testbeds in granular and restricted maneuverability spaces," *Robotics and Autonomous Systems*, vol. 110, pp. 160–172, 2018.

[10] H. Wang, C. Zhang, Y. Song, B. Pang, and G. Zhang, "Three-dimensional reconstruction based on visual slam of mobile robot in search and rescue disaster scenarios," *Robotica*, vol. 38, no. 2, pp. 350–373, 2020.

[11] T. Taketomi, H. Uchiyama, and S. Ikeda, "Visual slam algorithms: a survey from 2010 to 2016," *IPSJ Transactions on Computer Vision and Applications*, vol. 9, no. 1, pp. 1–11, 2017.

[12] D. M. Cole and P. M. Newman, "Using laser range data for 3d slam in outdoor environments," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* IEEE, 2006, pp. 1556–1563.

[13] C. Wang, C. Wen, Y. Dai, S. Yu, and M. Liu, "Urban 3d modeling with mobile laser scanning: a review," *Virtual Reality & Intelligent Hardware*, vol. 2, no. 3, pp. 175–212, 2020.

[14] S. Saeedi, M. Trentini, M. Seto, and H. Li, "Multiple-robot simultaneous localization and mapping: A review," *Journal of Field Robotics*, vol. 33, no. 1, pp. 3–46, 2016.

[15] R. Dubé, A. Gawel, H. Sommer, J. Nieto, R. Siegwart, and C. Cadena, "An online multi-robot slam system for 3d lidars," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 1004–1011.

[16] Y. Rizk, M. Awad, and E. W. Tunstel, "Cooperative heterogeneous multi-robot systems: A survey," *ACM Computing Surveys (CSUR)*, vol. 52, no. 2, pp. 1–31, 2019.

[17] P. Zhang, H. Wang, B. Ding, and S. Shang, "Cloud-based framework for scalable and real-time multi-robot slam," in *2018 IEEE ICWS*. IEEE, 2018, pp. 147–154.

[18] V. Sarker, J. P. Queralta, T. Gia, H. Tenhunen, and T. Westerlund, "Offloading slam for indoor mobile robots with edge-fog-cloud computing," in *2019 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT)*. IEEE, 2019, pp. 1–6.

[19] P. Yun, J. Jiao, and M. Liu, "Towards a cloud robotics platform for distributed visual slam," in *International Conference on Computer Vision Systems*. Springer, 2017, pp. 3–15.

[20] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[21] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.

[22] S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar, and A. Y. Zomaya, "Edge intelligence: The confluence of edge computing and artificial intelligence," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 7457–7469, 2020.

[23] L. Zeng, X. Chen, Z. Zhou, L. Yang, and J. Zhang, "Coedge: Cooperative dnn inference with adaptive workload partitioning over heterogeneous edge devices," *IEEE/ACM Transactions on Networking*, vol. 29, no. 2, pp. 595–608, 2020.

[24] T. Ouyang, X. Chen, L. Zeng, and Z. Zhou, "Cost-aware edge resource probing for infrastructure-free edge computing: From optimal stopping to layered learning," in *2019 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2019, pp. 380–391.

[25] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, vol. 3. IEEE, 2004, pp. 2149–2154.

[26] B. Schreck, C. Victorri-Vigneau, M. Guerlais, E. Laforgue, and M. Grall-Bronnec, "Slam practice: A review of the literature," *European Addiction Research*, vol. 27, no. 3, pp. 161–178, 2021.

[27] A. Gautam and S. Mohan, "A review of research in multi-robot systems," in *2012 IEEE 7th international conference on industrial and information systems (ICIIS)*. IEEE, 2012, pp. 1–5.

[28] A. J. Ben Ali, Z. S. Hashemifar, and K. Dantu, "Edge-slam: edge-assisted visual simultaneous localization and mapping," in *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, 2020, pp. 325–337.

[29] S. Dey and A. Mukherjee, "Robotic slam: a review from fog computing and mobile edge computing perspective," in *Adjunct Proceedings of the 13th International Conference on Mobile and Ubiquitous Systems: Computing Networking and Services*, 2016, pp. 153–158.

[30] Y. Wu, K. Ni, C. Zhang, L. P. Qian, and D. H. Tsang, "Noma-assisted multi-access mobile edge computing: A joint optimization of computation offloading and time allocation," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 12, pp. 12 244–12 258, 2018.

[31] Y. Wu, L. P. Qian, K. Ni, C. Zhang, and X. Shen, "Delay-minimization nonorthogonal multiple access enabled multi-user mobile edge computation offloading," *IEEE Journal of Selected Topics in Signal Processing*, vol. 13, no. 3, pp. 392–407, 2019.

[32] Y. Wu, L. P. Qian, H. Mao, X. Yang, H. Zhou, and X. Shen, "Optimal power allocation and scheduling for non-orthogonal multiple access relay-assisted networks," *IEEE Transactions on Mobile Computing*, vol. 17, no. 11, pp. 2591–2606, 2018.

[33] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE transactions on Robotics*, vol. 23, no. 1, pp. 34–46, 2007.

[34] J. Wu, G. Jiang, L. Zheng, and S. Zhou, "Algorithms for balanced graph bi-partitioning," in *HPCC, CSS, ICESS*. IEEE, 2014, pp. 185–188.

[35] K. Andreev and H. Racke, "Balanced graph partitioning," *Theory of Computing Systems*, vol. 39, no. 6, pp. 929–939, 2006.

[36] P. Schmuck and M. Chli, "Ccm-slam: Robust and efficient centralized collaborative monocular simultaneous localization and mapping for robotic teams," *Journal of Field Robotics*, vol. 36, no. 4, pp. 763–781, 2019.

[37] J. Xu, H. Cao, D. Li, K. Huang, C. Qian, L. Shangguan, and Z. Yang, "Edge assisted mobile semantic visual slam," in *INFOCOM*. IEEE, 2020, pp. 1828–1837.

[38] R. C. Smith and P. Cheeseman, "On the representation and estimation of spatial uncertainty," *The international journal of Robotics Research*, vol. 5, no. 4, pp. 56–68, 1986.

[39] R. Smith, M. Self, and P. Cheeseman, "Estimating uncertain spatial relationships in robotics," in *Autonomous robot vehicles*. Springer, 1990, pp. 167–193.

[40] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "Monoslam: Real-time single camera slam," *IEEE transactions on pattern analysis and machine intelligence*, vol. 29, no. 6, pp. 1052–1067, 2007.

[41] G. Klein and D. Murray, "Parallel tracking and mapping for small ar workspaces," in *2007 6th IEEE and ACM international symposium on mixed and augmented reality*. IEEE, 2007, pp. 225–234.

[42] T. Qin, P. Li, and S. Shen, "Vins-mono: A robust and versatile monocular visual-inertial state estimator," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 1004–1020, 2018.

[43] M. Labbé and F. Michaud, "Rtab-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation," *Journal of Field Robotics*, vol. 36, no. 2, pp. 416–446, 2019.

[44] R. Mur-Artal and J. D. Tardós, "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras," *IEEE transactions on robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[45] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "Orb-slam: a versatile and accurate monocular slam system," *IEEE transactions on robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[46] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2d lidar slam," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 1271–1278.

[47] G. Grisettiyz, C. Stachniss, and W. Burgard, "Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling," in *Proceedings of the 2005 IEEE international conference on robotics and automation*. IEEE, 2005, pp. 2432–2437.

[48] E. Vargas, R. Scona, J. S. Willners, T. Luczynski, Y. Cao, S. Wang, and Y. R. Petillot, "Robust underwater visual slam fusing acoustic sensing," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 2140–2146.

[49] F. Hidalgo, C. Kahlefendt, and T. Bräunl, "Monocular orb-slam application in underwater scenarios," in *2018 OCEANS-MTS/IEEE Kobe Techno-Oceans (OTO)*. IEEE, 2018, pp. 1–4.

[50] F. Nobis, O. Papanikolaou, J. Betz, and M. Lienkamp, "Persistent map saving for visual localization for autonomous vehicles: An orb-slam 2 extension," in *2020 Fifteenth International Conference on Ecological Vehicles and Renewable Energies (EVER)*. IEEE, 2020, pp. 1–9.

[51] T. Du, Y. H. Zeng, J. Yang, C. Z. Tian, and P. F. Bai, "Multi-sensor fusion slam approach for the mobile robot with a bio-inspired polarised skylight sensor," *IET Radar, Sonar & Navigation*, vol. 14, no. 12, pp. 1950–1957, 2020.

[52] W. Shao, S. Vijayarangan, C. Li, and G. Kantor, "Stereo visual inertial lidar simultaneous localization and mapping," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 370–377.

[53] T.-M. Nguyen, S. Yuan, M. Cao, T. H. Nguyen, and L. Xie, "Viral slam: Tightly coupled camera-imu-uwb-lidar slam," *arXiv preprint arXiv:2105.03296*, 2021.

[54] M. Palieri, B. Morrell, A. Thakur, K. Ebadi, J. Nash, A. Chatterjee, C. Kanellakis, L. Carlone, C. Guaragnella, and A.-a. Agha-Mohammadi, "Locus: A multi-sensor lidar-centric solution for high-precision odometry and 3d mapping in real-time," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 421–428, 2020.

[55] X. Zuo, Y. Yang, P. Geneva, J. Lv, Y. Liu, G. Huang, and M. Pollefeys, "Lic-fusion 2.0: Lidar-inertial-camera odometry with sliding-window plane-feature tracking," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 5112–5119.

[56] R. Kang, J. Shi, X. Li, Y. Liu, and X. Liu, "Df-slam: A deep-learning enhanced visual slam system based on deep local features," *arXiv preprint arXiv:1901.07223*, 2019.

[57] S. Wang, R. Clark, H. Wen, and N. Trigoni, "Deepvo: Towards end-to-end visual odometry with deep recurrent convolutional neural networks," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 2043–2050.

[58] Y. Almalioglu, M. R. U. Saputra, P. P. de Gusmao, A. Markham, and N. Trigoni, "Ganvo: Unsupervised deep monocular visual odometry and depth estimation with generative adversarial networks," in *2019 International conference on robotics and automation (ICRA)*. IEEE, 2019, pp. 5474–5480.

[59] D. Zou and P. Tan, "Coslam: Collaborative visual slam in dynamic environments," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 2, pp. 354–366, 2012.

[60] F. Li, S. Yang, X. Yi, and X. Yang, "Corb-slam: a collaborative visual slam system for multiple robots," in *International Conference on Collaborative Computing: Networking, Applications and Worksharing*. Springer, 2017, pp. 480–490.

[61] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

REFERENCES

[1] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Transactions on robotics*, vol. 32, no. 6, pp. 1309–1332, 2016.

[2] B. Huang, J. Zhao, and J. Liu, "A survey of simultaneous localization and mapping," *arXiv preprint arXiv:1909.05214*, 2019.

[3] S. Huang and G. Dissanayake, "A critique of current developments in simultaneous localization and mapping," *International Journal of Advanced Robotic Systems*, vol. 13, no. 5, p. 1729881416669482, 2016.

[4] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine *et al.*, "Scalability in perception for autonomous driving: Waymo open dataset," in *Proceedings of the IEEE/CVF CVPR*, 2020, pp. 2446–2454.

[5] A. Buyval, A. Gabdullin, R. Mustafin, and I. Shimchik, "Realtime vehicle and pedestrian tracking for didi udacity self-driving car challenge," in *2018 IEEE ICRA*. IEEE, 2018, pp. 2064–2069.

[6] H. Fan, F. Zhu, C. Liu, L. Zhang, L. Zhuang, D. Li, W. Zhu, J. Hu, H. Li, and Q. Kong, "Baidu apollo em motion planner," *arXiv preprint arXiv:1807.08048*, 2018.

[7] C. Badue, R. Guidolini, R. V. Carneiro, P. Azevedo, V. B. Cardoso, A. Forechi, L. Jesus, R. Berriel, T. M. Paixao, F. Mutz *et al.*, "Self-driving cars: A survey," *Expert Systems with Applications*, vol. 165, p. 113816, 2021.

[8] D. Rückert and M. Stamminger, "Snake-slam: Efficient global visual inertial slam using decoupled nonlinear optimization," in *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2021, pp. 219–228.

[9] X. Xiao and R. Murphy, "A review on snake robot testbeds in granular and restricted maneuverability spaces," *Robotics and Autonomous Systems*, vol. 110, pp. 160–172, 2018.

[10] H. Wang, C. Zhang, Y. Song, B. Pang, and G. Zhang, "Three-dimensional reconstruction based on visual slam of mobile robot in search and rescue disaster scenarios," *Robotica*, vol. 38, no. 2, pp. 350–373, 2020.

[11] T. Taketomi, H. Uchiyama, and S. Ikeda, "Visual slam algorithms: a survey from 2010 to 2016," *IPSJ Transactions on Computer Vision and Applications*, vol. 9, no. 1, pp. 1–11, 2017.

[12] D. M. Cole and P. M. Newman, "Using laser range data for 3d slam in outdoor environments," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* IEEE, 2006, pp. 1556–1563.

[13] C. Wang, C. Wen, Y. Dai, S. Yu, and M. Liu, "Urban 3d modeling with mobile laser scanning: a review," *Virtual Reality & Intelligent Hardware*, vol. 2, no. 3, pp. 175–212, 2020.

[14] S. Saeedi, M. Trentini, M. Seto, and H. Li, "Multiple-robot simultaneous localization and mapping: A review," *Journal of Field Robotics*, vol. 33, no. 1, pp. 3–46, 2016.

[15] R. Dubé, A. Gawel, H. Sommer, J. Nieto, R. Siegwart, and C. Cadena, "An online multi-robot slam system for 3d lidars," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 1004–1011.

[16] Y. Rizk, M. Awad, and E. W. Tunsel, "Cooperative heterogeneous multi-robot systems: A survey," *ACM Computing Surveys (CSUR)*, vol. 52, no. 2, pp. 1–31, 2019.

[17] P. Zhang, H. Wang, B. Ding, and S. Shang, "Cloud-based framework for scalable and real-time multi-robot slam," in *2018 IEEE ICWS*. IEEE, 2018, pp. 147–154.

[18] V. Sarker, J. P. Queralta, T. Gia, H. Tenhunen, and T. Westerlund, "Offloading slam for indoor mobile robots with edge-fog-cloud computing," in *2019 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT)*. IEEE, 2019, pp. 1–6.

[19] P. Yun, J. Jiao, and M. Liu, "Towards a cloud robotics platform for distributed visual slam," in *International Conference on Computer Vision Systems*. Springer, 2017, pp. 3–15.

[20] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[21] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.

[22] S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar, and A. Y. Zomaya, "Edge intelligence: The confluence of edge computing and artificial intelligence," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 7457–7469, 2020.

[23] L. Zeng, X. Chen, Z. Zhou, L. Yang, and J. Zhang, "Coedge: Cooperative dnn inference with adaptive workload partitioning over heterogeneous edge devices," *IEEE/ACM Transactions on Networking*, vol. 29, no. 2, pp. 595–608, 2020.

[24] T. Ouyang, X. Chen, L. Zeng, and Z. Zhou, "Cost-aware edge resource probing for infrastructure-free edge computing: From optimal stopping to layered learning," in *2019 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2019, pp. 380–391.

[25] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, vol. 3. IEEE, 2004, pp. 2149–2154.

[26] B. Schreck, C. Victorri-Vigneau, M. Guerlais, E. Laforgue, and M. Grall-Bronnec, "Slam practice: A review of the literature," *European Addiction Research*, vol. 27, no. 3, pp. 161–178, 2021.

[27] A. Gautam and S. Mohan, "A review of research in multi-robot systems," in *2012 IEEE 7th international conference on industrial and information systems (ICIIS)*. IEEE, 2012, pp. 1–5.

[28] A. J. Ben Ali, Z. S. Hashemifar, and K. Dantu, "Edge-slam: edge-assisted visual simultaneous localization and mapping," in *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, 2020, pp. 325–337.

[29] S. Dey and A. Mukherjee, "Robotic slam: a review from fog computing and mobile edge computing perspective," in *Adjunct Proceedings of the 13th International Conference on Mobile and Ubiquitous Systems: Computing Networking and Services*, 2016, pp. 153–158.

[30] Y. Wu, K. Ni, C. Zhang, L. P. Qian, and D. H. Tsang, "Noma-assisted multi-access mobile edge computing: A joint optimization of computation offloading and time allocation," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 12, pp. 12 244–12 258, 2018.

[31] Y. Wu, L. P. Qian, K. Ni, C. Zhang, and X. Shen, "Delay-minimization nonorthogonal multiple access enabled multi-user mobile edge computation offloading," *IEEE Journal of Selected Topics in Signal Processing*, vol. 13, no. 3, pp. 392–407, 2019.

[32] Y. Wu, L. P. Qian, H. Mao, X. Yang, H. Zhou, and X. Shen, "Optimal power allocation and scheduling for non-orthogonal multiple access relay-assisted networks," *IEEE Transactions on Mobile Computing*, vol. 17, no. 11, pp. 2591–2606, 2018.

[33] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE transactions on Robotics*, vol. 23, no. 1, pp. 34–46, 2007.

[34] J. Wu, G. Jiang, L. Zheng, and S. Zhou, "Algorithms for balanced graph bi-partitioning," in *HPCC, CSS, ICESS*. IEEE, 2014, pp. 185–188.

[35] K. Andreev and H. Racke, "Balanced graph partitioning," *Theory of Computing Systems*, vol. 39, no. 6, pp. 929–939, 2006.

[36] P. Schmuck and M. Chli, "Ccm-slam: Robust and efficient centralized collaborative monocular simultaneous localization and mapping for robotic teams," *Journal of Field Robotics*, vol. 36, no. 4, pp. 763–781, 2019.

[37] J. Xu, H. Cao, D. Li, K. Huang, C. Qian, L. Shangguan, and Z. Yang, "Edge assisted mobile semantic visual slam," in *INFOCOM*. IEEE, 2020, pp. 1828–1837.

[38] R. C. Smith and P. Cheeseman, "On the representation and estimation of spatial uncertainty," *The international journal of Robotics Research*, vol. 5, no. 4, pp. 56–68, 1986.

[39] R. Smith, M. Self, and P. Cheeseman, "Estimating uncertain spatial relationships in robotics," in *Autonomous robot vehicles*. Springer, 1990, pp. 167–193.

[40] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "Monoslam: Real-time single camera slam," *IEEE transactions on pattern analysis and machine intelligence*, vol. 29, no. 6, pp. 1052–1067, 2007.

[41] G. Klein and D. Murray, "Parallel tracking and mapping for small ar workspaces," in *2007 6th IEEE and ACM international symposium on mixed and augmented reality*. IEEE, 2007, pp. 225–234.

[42] T. Qin, P. Li, and S. Shen, "Vins-mono: A robust and versatile monocular visual-inertial state estimator," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 1004–1020, 2018.

[43] M. Labbé and F. Michaud, "Rtab-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation," *Journal of Field Robotics*, vol. 36, no. 2, pp. 416–446, 2019.

[44] R. Mur-Artal and J. D. Tardós, "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras," *IEEE transactions on robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[45] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "Orb-slam: a versatile and accurate monocular slam system," *IEEE transactions on robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[46] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2d lidar slam," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 1271–1278.

[47] G. Grisettiyz, C. Stachniss, and W. Burgard, "Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling," in *Proceedings of the 2005 IEEE international conference on robotics and automation*. IEEE, 2005, pp. 2432–2437.

[48] E. Vargas, R. Scona, J. S. Willners, T. Luczynski, Y. Cao, S. Wang, and Y. R. Petillot, "Robust underwater visual slam fusing acoustic sensing," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 2140–2146.

[49] F. Hidalgo, C. Kahlefendt, and T. Bräunl, "Monocular orb-slam application in underwater scenarios," in *2018 OCEANS-MTS/IEEE Kobe Techno-Oceans (OTO)*. IEEE, 2018, pp. 1–4.

[50] F. Nobis, O. Papanikolaou, J. Betz, and M. Lienkamp, "Persistent map saving for visual localization for autonomous vehicles: An orb-slam 2 extension," in *2020 Fifteenth International Conference on Ecological Vehicles and Renewable Energies (EVER)*. IEEE, 2020, pp. 1–9.

[51] T. Du, Y. H. Zeng, J. Yang, C. Z. Tian, and P. F. Bai, "Multi-sensor fusion slam approach for the mobile robot with a bio-inspired polarised skylight sensor," *IET Radar, Sonar & Navigation*, vol. 14, no. 12, pp. 1950–1957, 2020.

[52] W. Shao, S. Vijayarangan, C. Li, and G. Kantor, "Stereo visual inertial lidar simultaneous localization and mapping," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 370–377.

[53] T.-M. Nguyen, S. Yuan, M. Cao, T. H. Nguyen, and L. Xie, "Viral slam: Tightly coupled camera-imu-uwb-lidar slam," *arXiv preprint arXiv:2105.03296*, 2021.

[54] M. Palieri, B. Morrell, A. Thakur, K. Ebadi, J. Nash, A. Chatterjee, C. Kanellakis, L. Carlone, C. Guaragnella, and A.-a. Agha-Mohammadi, "Locus: A multi-sensor lidar-centric solution for high-precision odometry and 3d mapping in real-time," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 421–428, 2020.

[55] X. Zuo, Y. Yang, P. Geneva, J. Lv, Y. Liu, G. Huang, and M. Pollefeys, "Lic-fusion 2.0: Lidar-inertial-camera odometry with sliding-window plane-feature tracking," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 5112–5119.

[56] R. Kang, J. Shi, X. Li, Y. Liu, and X. Liu, "Df-slam: A deep-learning enhanced visual slam system based on deep local features," *arXiv preprint arXiv:1901.07223*, 2019.

[57] S. Wang, R. Clark, H. Wen, and N. Trigoni, "Deepvo: Towards end-to-end visual odometry with deep recurrent convolutional neural networks," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 2043–2050.

[58] Y. Almalioglu, M. R. U. Saputra, P. P. de Gusmao, A. Markham, and N. Trigoni, "Ganvo: Unsupervised deep monocular visual odometry and depth estimation with generative adversarial networks," in *2019 International conference on robotics and automation (ICRA)*. IEEE, 2019, pp. 5474–5480.

[59] D. Zou and P. Tan, "Coslam: Collaborative visual slam in dynamic environments," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 2, pp. 354–366, 2012.

[60] F. Li, S. Yang, X. Yi, and X. Yang, "Corb-slam: a collaborative visual slam system for multiple robots," in *International Conference on Collaborative Computing: Networking, Applications and Worksharing*. Springer, 2017, pp. 480–490.

[61] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.