
# SwarmMap: Scaling Up Real-time Collaborative Visual SLAM at the Edge

Jingao Xu, Hao Cao, and Zheng Yang, *Tsinghua University;*
Longfei Shangguan, *University of Pittsburgh & Microsoft;*
Jialin Zhang, Xiaowu He, and Yunhao Liu, *Tsinghua University*

https://www.usenix.org/conference/nsdi22/presentation/xu

This paper is included in the Proceedings of the
19th USENIX Symposium on Networked Systems
Design and Implementation.

April 4–6, 2022 • Renton, WA, USA

978-1-939133-27-4

Open access to the Proceedings of the
19th USENIX Symposium on Networked
Systems Design and Implementation
is sponsored by

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

# SwarmMap: Scaling Up Real-time Collaborative Visual SLAM at the Edge

Jingao Xu[1*], Hao Cao[1*], Zheng Yang[1✉], Longfei Shangguan[2]
Jialin Zhang[1], Xiaowu He[1], Yunhao Liu[1]
[1]*School of Software, Tsinghua University* [2]*University of Pittsburgh & Microsoft*

## Abstract

The Edge-based Multi-agent visual SLAM plays a key role in emerging mobile applications such as search-and-rescue, inventory automation, and industrial inspection. This algorithm relies on a central node to maintain the global map and schedule agents to execute their individual tasks. However, as the number of agents continues growing, the operational overhead of the visual SLAM system such as data redundancy, bandwidth consumption, and localization errors also scale, which challenges the system scalability.

In this paper, we present the design and implementation of SwarmMap, a framework design that scales up collaborative visual SLAM service in edge offloading settings. At the core of SwarmMap are three simple yet effective system modules — a change log-based server-client synchronization mechanism, a priority-aware task scheduler, and a lean representation of the global map that work hand-in-hand to address the data explosion caused by the growing number of agents. We make SwarmMap compatible with the robotic operating system (ROS) and open-source it[1]. Existing visual SLAM applications could incorporate SwarmMap to enhance their performance and capacity in multi-agent scenarios. Comprehensive evaluations and a three-month case study at one of the world's largest oil fields demonstrate that SwarmMap can serve 2× more agents (>20 agents) than the state of the arts with the same resource overhead, meanwhile maintaining an average trajectory error of 38*cm*, outperforming existing works by >55%.

## 1 Introduction

Visual simultaneous localization and mapping (SLAM) systems take video streams from one or multiple cameras as input, reconstructing the 3D map of environment while simultaneously determining the position and orientation of cameras with respect to their surroundings [29, 34, 36]. With the size of the mapping area expanding rapidly, collaborative visual SLAM that involves multiple agents has been attracting growing interest from both academia and industry [25, 39, 40, 49]. For instance, Amazon, JD, and Alibaba have deployed dozens of picking and sorting robots in their logistics warehouses to save labor cost [45]; DJI and Amazon have also been developing drone grouping and swarming technology for urban modeling, express delivery, and industrial inspection [12]. In these scenarios, each agent has to conduct not only the localization but mapping tasks in real-time due to $(i)$ upper layer applications require the latest updated environment map to perform the subsequent maintenance and scheduling tasks, especially in those dynamic environments; and $(ii)$ since the two modules are tightly coupled, an agent also relies on a high-quality on-board map for a better localization performance and vice-versa [3, 47].

The SLAM agents profile the environment with their cameras, exchange data with each other, and execute vision tasks in real-time, with a significant computation overhead. The limited computation resource on the agent soon becomes the bottleneck, impairing system accuracy [3, 40, 47]. *Edge-offload* has emerged as a promising alternative due to the following two reasons. First, by offloading bulky tasks to edge devices, the agents only need to run light-weight and time-sensitive jobs locally, which effectively mitigates on-board resource shortage [3, 47]. Second, by fusing and further optimizing the visual map globally at a centralized edge device, map information that is originally unavailable to each other can be easily shared among agents [39, 40]. This will benefit collaborative missions such as collision avoidance and path planning.

Albeit inspiring, the growing number of agents brings new issues that challenge the scalability of edge-based real-time collaborative visual SLAM systems (§2.2):

• **Map synchronization stresses the network bandwidth**. Mobile agents like drones and robots heavily rely on wireless links to communicate with an edge device. However, wireless spectrum is a limited and overcrowded resource. Streaming large volumes of map data over wireless links will soon saturate the medium and cause significant delays.

• **FCFS-based job scheduling impairs the localization ac-**

✉Zheng Yang (hmilyyz@gmail.com) is the corresponding author. Jingao Xu and Hao Cao are co-primary authors.

[1]Code and data at https://github.com/MobiSense/SwarmMap.

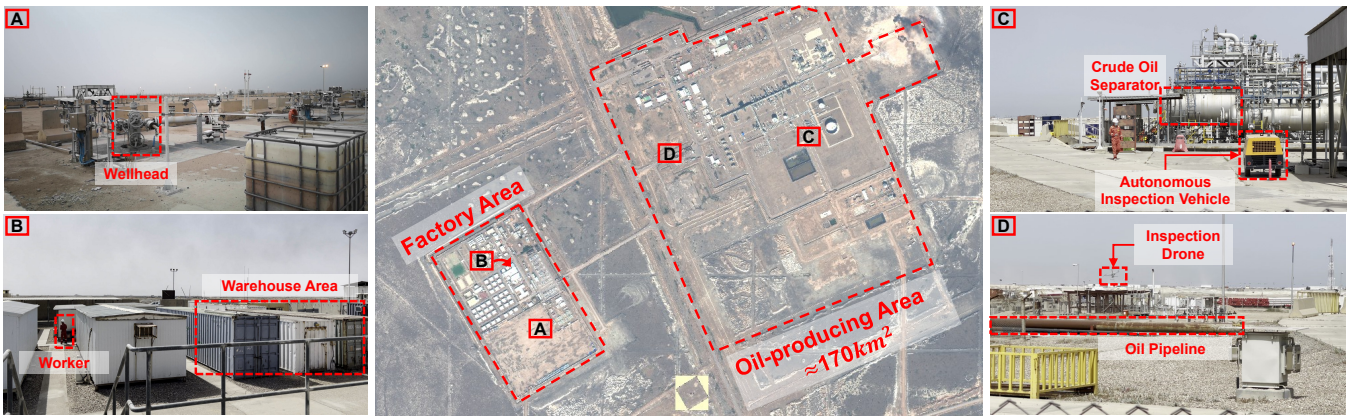



Figure 1: Industrial inspection is carried out by 10 drones and 2 autonomous vehicles in a large oil-field. These agents are coordinated by SwarmMap that runs on an Nvidia AGX Xavier edge server.

**curacy**. An edge device has to processes large volumes of requests from agents, which may cause significant delays to latecomers (i.e., those requests positioned in the tail of the queue). However, agents in different states are not equally sensitive to the queuing delay. The conventional first-come, first-served (FCFS) pipeline will exacerbate the localization error on those time-sensitive agents.

• **Map expansion exacerbates the memory footprint**. The size of the global visual map increases sharply with a growing number of agents, which is likely to exceed the limited memory capacity allocated to SLAM tasks by an edge node, causing memory overflow.

However, the current practice of edge-offload focuses primarily on computation-oriented task partitioning [3, 8, 23, 40, 47]. They fail to address the data explosion and its impact on transmission, scheduling, and storage. Hence these pioneer designs cannot scale with the sheer size of the real-time collaborative visual SLAM systems.

In this work, we present SwarmMap, a framework to scale up the real-time collaborative visual SLAM services at resource-constrained edge devices. SwarmMap does not innovate visual SLAM algorithms. Instead, it proposes functionality and resource abstractions of existing SLAM algorithms and provides additional system services to enhance system scalability. Hence, most variations of collaborative visual SLAM systems can take advantage of our design. With SwarmMap, the upper-layer user can outsource agent task scheduling and processing instead of understanding every detail of SLAM algorithms to manually adapt. SwarmMap contains three key plug-in modules, as described below.

First, we design a Map Information Tracker (*Mapit*) to maintain map data consistency between the agents and the edge while remarkably saving network bandwidth. Unlike existing methods that transfer bulky map data with each other [39, 40], *Mapit* records the operations associated with the map modification on the agent and transmits these operations to the edge. The edge node then follows these operations to update its local map. This allows the map synchronization between them at the minimum bandwidth consumption even compared with state-of-the-arts (e.g., CarMap [2]).

Second, we introduce a SLAM-specific task-aware scheduler (*STS*) that prioritizes requests based on the status of their producer (i.e., agent). The *STS* scheduler runs on both the agent and the edge. The agent *STS* evaluates agent status around the clock and updates this information with the edge through heartbeat packets. The edge *STS* designs a multi-level queue to ensure those urgent tasks will be processed timely.

Third, we propose a Map Backbone Profiling (*MBP*) technique to alleviate the storage overhead while retaining the mapping accuracy. This technique is based on an observation that the data quality among different agents' maps can be balanced by elements in co-visible areas. We propose a set of metrics to detect high-quality map elements and use them to offset those low-quality counterparts, thereby elevating the overall map quality. Applying model compression to this high-quality map allows us to remove large portions of redundant map data without sacrificing the map accuracy.

We evaluate SwarmMap on a testbed consisting of 4 Nvidia Jetson boards, 4 smartphones, 4 DJI RoboMasters, and 4 drones. Following the standard SLAM evaluation pipeline [2, 6, 28, 47], we further compare SwarmMap with two state-of-the-art (SOTA) edge-assisted multi-agent SLAM systems (CCM-SLAM [40] and Multi-UAV [39]) on three gold-standard SLAM datasets (TUM [11], KITTI [10], and EuRoC [9]) as well as a self-labeled dataset collected at a 22,927 sqft shopping mall. We also compare SwarmMap with CarMap [2] and Sum-Map [27] to evaluate each functional module in SwarmMap. Our head-to-head comparison shows that SwarmMap can serve 2× more agents than these SOTA systems with the same resource overhead, meanwhile maintaining an absolute trajectory error within 38$cm$ when serving 20 agents, outperforming these SOTA systems by >55%.

**Real-world deployment**. We have developed a real-time collaborative visual SLAM system based on SwarmMap and deployed it in one of the world's largest oil-field (>170$km^2$) for industrial inspection (shown in Fig. 1). Our system con-

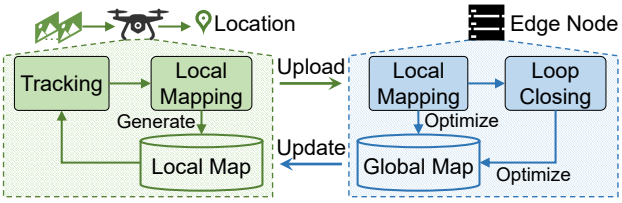


Figure 2: Workflow of existing edge-assisted SLAM [3, 47]

sists of twelve agents that communicate with an Nvidia Jetson AGX Xavier [46] edge node through Wi-Fi mesh networks. A three-month pilot study shows that SwarmMap achieves an average localization accuracy of 0.36$m$. The link throughput and RAM consumption are below 17MB/s and 26GB respectively, meeting inspection demands within the constraints of available resources.

In summary, this paper makes three contributions. First, we quantify the scalability challenges of deploying real-time collaborative visual SLAM at the edge to motivate framework support. Second, we design and implement SwarmMap as a framework to address the scalability issues spanning from communication, computation, to storage. As far as we are aware of, SwarmMap is the first system solution to scale up the collaborative visual SLAM in edge settings. Third, we deploy SwarmMap in a large oil field for industrial inspections. Our three-month pilot study demonstrates that SwarmMap makes a great process towards fortifying multi-agent collaborative visual SLAM to a fully practical system for wide deployment.

**Contribution to the community**. We implement SwarmMap as a software package of the robot operating system (ROS [26]), the dominating OS in the robotics field. We believe SwarmMap can provide a collection of tools for both academia and industry, and further enable fast prototyping of visual SLAM-based applications in multi-agent scenarios.

## 2 Background and Motivation

The data volume scales with the number of agents, and the need for framework support arises from the excessive bandwidth consumption and memory footprint caused by the data explosion. We discuss these in detail in this section.

### 2.1 Edge-assisted visual SLAM systems

The visual SLAM consists of multiple sub-tasks with diverse workloads. Edge-offload places those bulky tasks to an edge server, leaving an agent light-weight and time-sensitive jobs. The agent can thus run visual SLAM in real-time. We use ORB-SLAM2 [29], a top-ranked open-source visual SLAM system, to illustrate the SLAM operations under edge settings (refer to Fig. 2).

**Front-end.** Mobile agents run *Tracking* and part of the *Local Mapping* module locally. The *Tracking* module extracts 2D ORB feature points from each video frame and instantly estimates the pose of onboard camera(s) based on the geometry relationship between these feature points and the pre-constructed local map (i.e., a set of 3D map-points and keyframes[2] in which they appear). As the mobile agent moves, the *Local Mapping* module updates the local map timely.

**Back-end.** Due to high computation costs, the optimization part of the *Local Mapping* module is offloaded to the edge device, where the bundle adjustment (BA) algorithms [42] kick in to improve the pose and 3D location accuracy of those newly generated keyframes and map-points. The edge server also runs a *Loop Closing* module to detect repeated paths and leverage them to re-calibrate the global map.

**Data transfer in-between.** To improve the map accuracy, each agent periodically sends keyframes and map-points to the edge server for fine-grained optimization. The optimized visual map is then streamed to the clients.

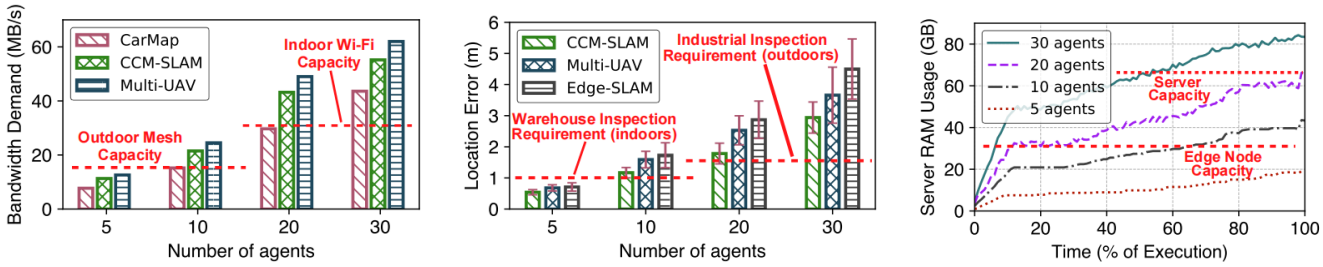
### 2.2 The scalability issues

As more agents get involved, running real-time collaborative visual SLAM on edge environment becomes increasingly complex, facing several challenges: $(i)$ the frequent data transfer between agents and edge is likely to saturate wireless links, causing significant delays; $(ii)$ the queueing delay on edge node exacerbates localization errors; $(iii)$ the data volume grows sharply, threatening the data storage at the edge node. We discuss these issues below.

**C1: Excessive bandwidth consumption**. The life-cycle of a collaborative visual SLAM system consists of cold-start and maintenance two sessions. In the cold-start session, the agents transfer all observed keyframes and map-points data to the edge server. The edge server then generates a global map of the entire space and optimizes the local map for each agent. Once the global map generation has been completed, the SLAM system enters the long-term maintenance session during which each agent regularly revisits each site and calibrates the mapping offset. However, since map elements are tightly coupled, a minor modification on a single map element will spread to many other elements. This will cause a significant amount of data transfer in the maintenance.

To reduce bandwidth consumption, recent works [2, 40] design compact map representations and transfer the difference before and after map element calibration (as opposed to transferring the entire calibrated map element [39, 47]). Although these systems can effectively reduce bandwidth consumption in the cold-start session, they encounter two issues in the maintenance session due to the frequent map updates: $(i)$ *extra computation overhead*. The acquisition of element-level differences requires pair-wise map feature comparison across the entire map. This will lead to extra computation workload pressure on resource-limited mobile agents; and $(ii)$ *limited data volume reduction*. Since a minor change on an element will spread to a batch of coupled elements. the volume of data to be transferred is still bulky.

[2]Keyframes are a subset of selected frames. Each keyframe stores the camera pose, the map-points it observed, and the co-visibility relationships with other keyframes.




(a) Excessive bandwidth consumption (b) Severe localization errors (c) Large data storage overhead

Figure 3: The scalability issues in multi-agent scenarios.

To validate our analysis, we measure the bandwidth requirement of three state-of-the-art (SOTA) systems in different number of agents settings. The results are shown in Fig. 3a. Compared with the vanilla Multi-UAV [39], we observe that CarMap [2] and CCM-SLAM [40] can effectively reduce the transmission workload during map synchronization. However, when serving more than ten agents, both systems still produce excessive wireless traffic that can easily go beyond the link capacity[3]; thus, significant system delays are expected.

**C2: Severe localization errors**. Under the edge settings, the localization accuracy of an agent highly depends on the quality of the local map which is optimized at the edge side. Typically, an agent needs to periodically (within 5s) send optimization requests to the edge server for every 3-5 newly generated keyframes [3,47]. As the number of agents scales, the concurrent requests from different agents block at the edge node's processing pipeline, resulting in excessive queuing delays. Consequently, some agents get their optimization tasks done untimely, causing severe localization errors. This situation is worsened by the fact that agents in different running states (e.g., flying speeds, self-tracking qualities) are not equally sensitive to the waiting delay. Recent multi-agent collaborative SLAM solutions focus on map fusion and optimization on edge or cloud servers, but ignore the task queuing issue for each agent. The conventional first-come, first-served (FCFS) scheduling will inevitably exacerbate the localization error on those task-sensitive agents (demonstrated in §5.3).

We measure the localization error (in $m$) of three related works in a different number of agents settings. The results are shown in Fig. 3b. Considering the accuracy requirement from a broad range of SLAM applications, we treat $1m$ and $1.5m$ as acceptable localization errors for indoor (warehouse inspection) and outdoor (anomaly detection) scenarios. Evidently, all these three systems fail to meet the localization requirement when serving more than 5 and 10 agents indoors and outdoors respectively, leaving room for improvements.

**C3: Large data storage overhead**. The global map maintained by the edge server contains large redundancy due to the following two reasons. First, to ensure the inspection efficacy, different agents will re-visit the same area at certain intervals, causing significant path duplication. Second, to complete the 3D map reconstruction, different agents have to share a co-visible area, resulting in bulky data redundancy. As the number of agents grows, the data redundancy increases sharply, and the data volume is likely to exceed the limited memory capacity of the edge node.

We set up an edge-based collaborative visual SLAM testbed using a commercial edge device Nvidia Jetson AGX Xavier (with 32GB RAM and costs $599) and measure its RAM usage in different numbers of agent settings. We repeat the measurement on a powerful server with 4× higher storage capacity (i.e., Dell PowerEdge T630 with 128GB RAM and costs $6,899) for comparison. The results are shown in Fig. 3c. In accordance with our analysis, as the system proceeds, the RAM usage increases rapidly and soon saturates the memory capacity of both the edge node and the high-end server. This limitation is worsened by the mismatch between the limited storage capacity of the edge node and the growing fidelity of video streams (i.e., 4K or 8K videos). Such high memory demand limits the maximum number of agents to five, which sets a strong barrier for the practical deployment of the edge-based collaborative visual SLAM system.

Due to the device heterogeneity (e.g., cameras on drones and robots may differ drastically in video resolution and frame rate) and diverse running status, the quality of maps provided by different agents may vary largely. An ideal map compression should remove those low-quality redundancy while retaining the high-quality counterpart. However, existing works ignore such difference when compressing the map data [27,32,43], resulting in degraded SLAM performance (details in §5.3).

## 2.3 SwarmMap: System goals

SwarmMap takes a solid step forward in solving these scalability issues. We list the system goals below.

**Goal 1: Functionality and resource abstraction**. SwarmMap should provide functionality and resource abstractions of existing SLAM algorithms. This allows any variation of map-point- and keyframe-based collaborative SLAM algorithms to take advantage of SwarmMap.

**Goal 2: Plug and play**. SwarmMap should be implemented as a plug-in module, exposing well-defined APIs to end-users for adaption. This avoids the deeply embedded manual code changes that may again challenge the system's scalability.

[3]The measurement shows the maximum throughput in an outdoor mesh and an indoor 2.4 GHz Wi-Fi network is 15MB/s and 30MB/s, respectively.

**Goal 3: Resource overhead reduction**. SwarmMap should effectively reduce the resource overhead spanning data storage, client-edge communication, and task scheduling while ensuring the precision and real-time performance.

## 3 Design

In this section, we first describe the high-level system architecture and then present each module design in SwarmMap.

### 3.1 System overview

SwarmMap is a framework design to scale up collaborative visual SLAM service in edge offloading settings. To achieve this goal, we make the following layer-wise functionality and resource abstractions: $(i)$ *agent layer*, where each agent localizes itself and builds surrounding local maps in real-time; $(ii)$ *network layer*, which enables communications and data interactions between mobile and edge for map synchronization; and $(iii)$ *edge layer*, which fuses, optimizes, and maintains the global map. This layer-wise abstraction provides a clear view of map data transfer, processing, and storage in SLAMs.

**Key functional modules**. SwarmMap designs three plug-in modules to address the resource overhead and scheduling issues across these three layers.

• The *Mapit* (Map Information Tracker) module tracks system operations associated with map data calibration. It then transfers these operations to the peer(s) for map synchronization (§3.2).

• The *STS* (SLAM-specific Task Scheduling) module optimizes the batch request execution and manages the resource allocations among multiple agents (§3.3).

• The *MBP* (Map Backbone Profiling) module compresses the map data uploaded by individual agents while ensuring the overall mapping accuracy (§3.4).

**SwarmMap Architecture**. Fig. 4 shows the system architecture. SwarmMap shares similar edge-based architecture with previous works and provides extra system support on both the mobile agent and edge server side, as discussed below.

• On the mobile agent side, SwarmMap tracks the run-time status of each agent through a light-weight evaluation-based mechanism *STS* (mobile part). It then follows a dedicated information exchanging protocol *Mapit* to communicate and update map elements with the edge server.

• On the edge side, the edge node prioritizes the agents' requests by *STS* (edge part) based on their run-time status. It then takes into account the data quality of maps reported by individual agents and extracts a lean presentation of the overall map through a map backbone profiling algorithm (*MBP*). Finally, the optimized and compressed map backbones will be sent to each mobile agent by *Mapit*.

### 3.2 Mapit: Map Information Tracker

The inevitable frequent map data synchronization between clients and edge consumes large bandwidth in both cold-start and maintenance sessions, circumscribing the system capacity (i.e., the number of supported agents). Recent works (e.g., CarMap [2] and CCM-SLAM [40]) propose a compact map representation that greatly reduces the data transfer in the cold-start session. However, their effectiveness fails to translate to a sufficient reduction in the maintenance session (§2.2-C1). Therefore, in SwarmMap we focus on the data transfer reduction in the maintenance session.

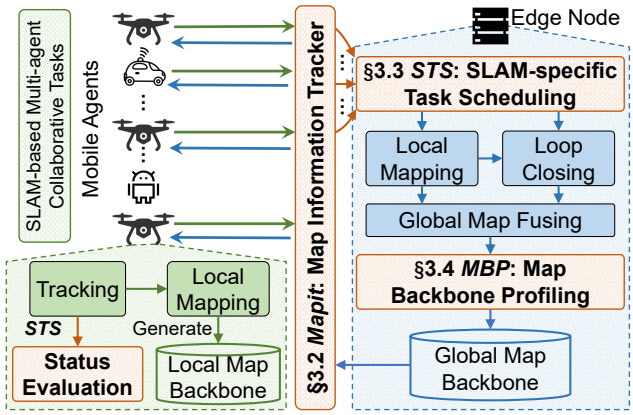



Figure 4: System architecture of SwarmMap. Compared with the conventional edge-based visual SLAM architecture, the added plug-in modules are highlighted in orange.

Our design is based on an observation that the map change on one side can be reproduced on the other side (e.g., agent vs. edge) by solely transferring the map change operations. This enables a light-weight map synchronization by avoiding transferring massive map-point data and the bulky geographical descriptors such as their spatial locations, features, observation relationships with keyframes [28]. Compared with the current practice, our design also achieves higher synchronization efficiency because it does not require a pair-wise map element comparison, which leads to extra computation workload pressure on resource-limited mobile agents.

To realize this basic idea, we design *Mapit*, a light-weight map information tracker to automate the operation tracking and reproducing on mobile and edge. *Mapit* runs as a daemon on both sides, monitoring the SLAM function calls and logging corresponding map operations (e.g., move a map-point by $2cm$). It then transfers this log to the agent (or the server), based on which the agent reproduces these operations locally. The map data are synchronized at the end.

The *Mapit* package periodically[4] synchronizes the map operation logs, and consists of five atomic operations: *add*, *aggregate*, *push*, *merge*, and *pull* (shown in Fig. 5).

① ***Mapit* add.** The atomic operation *add* registers a hook for each SLAM function call (listed in Table 3) and maintains a recording queue. Whenever an important function is called, an operation record containing its name, parameters, and influence on map elements is *added* to the operation queue.

[4]Similar to current practice [2, 3, 39], we empirically set the period to 2s.

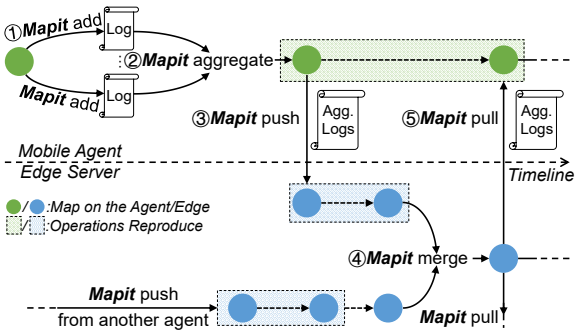



Figure 5: Workflow of the *Mapit*.

② ***Mapit* aggregate.** At the end of each period, *Mapit* aggregates the records in the operation queue to reduce their size. The intuition is that some removals or merges on certain types of functions will generate equivalent effects. For instance, if a function changes the location of a map-point and is marked as `overwritten`, we only need to focus on the latest record of it and ignore all the previous operations on the map-point. As for those marked as `stackable`, the implication is that records about modifying a same element can be merged by parameters. In this way, *Mapit* produces a minimal set containing necessary information.

③ ***Mapit* push.** After aggregating records, the atomic operation *push* on an agent sends the packed records to the edge server. By reproducing these operations, the map maintained on edge keeps synchronized with the ones on the client.

④ ***Mapit* merge.** On the edge server, the *merge* module periodically checks if there exists an overlap between the maps uploaded by individual agents and the global map. Once an overlap is detected, different maps will be coordinated and fused by the upper-layer SLAM algorithms (e.g., Sim3 optimization algorithm [28]). The map fusion process will operate and update some map elements, and hence the *merge* module also records these operations on the map elements in the same way as *add* and *aggregate*.

⑤ ***Mapit* pull.** The *pull* module can be treated as the reverse operation of *push*. It requests aggregated map modification logs generated by map optimization and *merge*, from the edge server to the agent. Additionally, if the global map has already been created (i.e., the whole system is in the maintenance session), *Mapit* will also transfer a set of closest map-points (e.g., associated with the next 5 keyframes) to the agent in the *pull* process. The benefit of this strategy is to enhance the agent's localization performance since these map-points with a high probability of appearing in the future would provide prior information for the *tracking* module on the agent side.

## 3.3 STS: SLAM-Specific Task Scheduling

As more agents get involved in SLAM systems, processing agents' requests (e.g., local map optimization) can cause excessive queuing delays. Since agents in different running states are not equally sensitive to the waiting delay, conventional FCFS scheduling may exacerbate localization errors on time-sensitive agents and hurt SLAM performance (§2.2-C2). To our best knowledge, there is still a lack of scheduling strategy tailor to multi-agent SLAM tasks.

To address this issue, we introduce *STS* – the first SLAM-Specific Task Scheduler that guides the edge to strategically prioritize requests. Specifically, *STS* divides agents into emergency and non-emergency groups based on the agents' status. It timely reorders the requests based on the following principles:

$(i)$ Prioritizing requests from agents in the emergency group.

$(ii)$ Among those non-emergency agents, *STS* prioritizes requests from agents that can provide higher information gain for global map construction or optimization.

The first principle aims to prevent each agent from losing self-tracking, and the second is for achieving a better overall global mapping performance. We propose a set of metrics to characterize the agent status and design a multi-level queue to schedule the requests from agents.

### 3.3.1 Agent Status Evaluation and Updating

**Agent side.** Each agent regularly updates its status with the edge by sending heartbeat packets. Since both environment and device dynamics may fluctuate violently during an agent's movement, the heartbeat interval should be shorter than the agent's request interval (i.e., 2s). In SwarmMap we expose the heartbeat setting (100ms by default) to end-users so that they can easily adapt to different environment settings. We define three variables that can fairly reflect an agent's status:

• *Tracking state*: a 1-bit Boolean value shows whether an agent is traceable or not. An agent's tracking state is set to LOST if its latest ORB feature maps cannot well match the local feature map. This variable is provided by the *tracking* module in many visual SLAM systems [29].

• *Velocity burst*: a 1-bit Boolean value shows whether an agent's speed changes abruptly or not. An abrupt change of velocity may result in motion blur in videos and make it hard for clients to extract visual features. In SwarmMap, we set the variable *Velocity burst* to True if the current moving speed is 20% greater than the averaged speed over the latest $N$ frames, where $N$ is a variable exposed to end-users. $N = 10$ by default.

• *Tracked map-points number*: an 8-bit variable represents the number of map-points observed by an agent. A larger number indicates the *tracking* module is running more stable.

**Server side.** Due to the heterogeneous device capability (e.g., cameras on different agents may differ in resolutions) and diversified trajectory, each agent contributes unequally to global map construction and optimization. SwarmMap prioritizes requests from those agents that can provide higher information gain for global map construction and optimization. To this end, we design the following two metrics to measure the information gain of each agent:

• *Map-point score* (MS) is defined as the average score of all map-points observed by an agent (the way to calculate the map-point score will be introduced in §3.4). A higher average

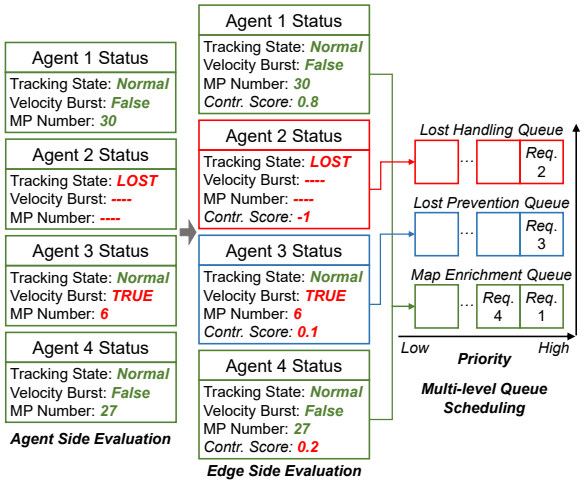



Figure 6: Workflow of the *STS* with an example.

score reflects that the current position is likely to have been visited before. On the contrary, a lower score indicates the agent is exploiting new or partially observed areas. Hence, *STS* prioritizes tasks with a lower map-point score.

• *Map elements generation speed* (MG) characterizes the number of unobserved map-points and keyframes uploaded by the latest *mapit push* operation. An agent with a higher map element generation speed contributes more to the edge's global map generation and optimization.

*STS* normalizes each metric and computes each agent's *contribution score* as normalized MG - normalized MS.

#### 3.3.2 Multi-level Queue Scheduling

On the edge side, *STS* designs three queues with different priorities to facilitate agent request scheduling.

• **Lost Handling Queue**. If an agent's tracking state is marked as LOST, *STS* will push its request into this queue.

• **Lost Prevention Queue**. If an agent has a velocity burst and merely tracks few map-points, it may become prone to LOST, and *STS* will push its request into this queue.

• **Map Enrichment Queue**. For those agents with stable running status (i.e., without the risk of losing self-tracking), *STS* will push their requests into this queue and sort them by their mapping contribution scores.

The lost handling queue owns the highest priority, followed by lost prevention queue and map enrichment queue. Upon the reception of an agent's request, *STS* inserts this request into one of these three queues based on the agent's tracking status and mapping contribution. The back-end SLAM algorithm pops requests from queues based on their priority.

We take Fig. 6 as an example to explain the job scheduling in SwarmMap. Suppose there are four agents in the system, with agent 2 in lost tracking status and agent 3 facing the velocity burst issue. *STS* will push agent 2 and 3's requests into the lost handling and prevention queue, respectively. The request from agent 1 and 4, two agents not in emergency states, will be pushed into the map enrichment queue. Since agent 1's mapping contribution score is higher than agent 4, the request from agent 1 will be put at the head of the queue. The edge processes these requests in the order of 2-3-1-4.

### 3.4 MBP: Map Backbone Profiling

The global map maintained by the edge node contains large redundancy (§2.2-C3). Due to the device heterogeneity (e.g., the onboard cameras may differ in resolution and frame rate) and diverse running status, the quality of maps contributed by different agents may vary largely. Existing map compression works [6, 13, 14] ignore such difference, resulting in information loss and hence degraded performance. The relevant works, CarMap and CCM-SLAM, design lean map representations to reduce the transmitted data volume for a faster map synchronization. However, they still need to reconstruct the huge global map through these compact representations on both mobile agent and edge node. Therefore, the memory footprint remains high when more agents are connected.

To address this issue, we introduce a map backbone profiling (*MBP*) algorithm. Unlike the current practice, we do not greedily remove redundant map elements in co-visible areas. Instead, we first leverage these redundant elements to generate a series of virtual keyframes and use them to improve those low-quality map segments. Once the overall quality of the global map got improved, we can thus compress the global map without compromising the mapping quality.

*MBP* first evaluates the quality and importance of each map element. It then $(i)$: finds high-quality map-points that could be leveraged to generate virtual keyframes; $(ii)$: searches for low-quality map segments that need to be improved; and $(iii)$: improves the overall map quality by inserting virtual keyframes to those low-quality map segments. Finally, *MBP* operates map compression on the balanced global map.

#### 3.4.1 Map Element Evaluation

Map-point evaluation has been extensively studied in related works [14]. The gold-standard metrics include the observing path length, maximum observing distance, maximum observing angle, and mean re-projection error. We borrow these metrics (detailed in §A.2) to evaluate a map-point and propose three new metrics to adapt to collaborative scenarios:

• **Observed number** represents the number of keyframes, in which the map-point is observed, across the entire global map. A higher score indicates multiple agents can observe a map point over a long period.

• **Update frequency** is defined as the total number of times the map-point was modified or updated by all agents in the last round of *Mapit push* operations. Map-points with high update frequency suggest a potential hot spot in a trajectory.

• **Moving velocity** records the speed of a mobile device when it generates the map element. A higher score indicates a potential blurriness that may influence the stability of the map-point. We take its negative value to evaluate the map-point score.

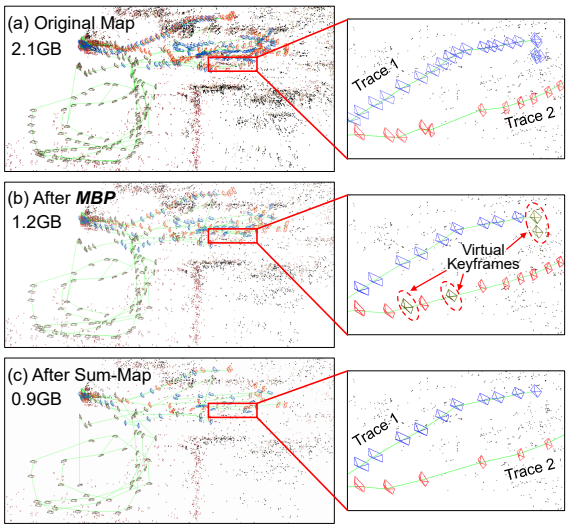



Figure 7: A running demo of *MBP*. The left column shows the map elements uploaded by different agents, while the right column presents partial zooming-in maps. The dotted keyframes in (b) are the synthetic virtual keyframes.

*MBP* normalizes each metric by its maximum value. We then define the score of a map-point as the sum of all normalized metrics values. The score of a keyframe is the sum of all observing map-point scores.

#### 3.4.2 Map Backbone Generation

The map backbone generation consists of two steps: virtual keyframe generation and map compression.

**Virtual keyframe generation.** The trajectory of an individual agent is first segmented with the awareness of where overlaps occur. The quality of each map segment is defined as the sum of all map element scores (i.e., scores of all keyframes and map-points) within it. For each map segment with low quality (e.g., its score is in the bottom 20%), *MBP* search for high-quality map-points in its neighborhood (i.e., within $60^\circ$ field-of-view of its keyframes) even though the original keyframes do not observe these map-points. Furthermore, *MBP* synthesizes virtual keyframes that could observe these high-quality map-points, and the pose (i.e., spatial location and orientation) of each keyframe can be calculated by the ICP algorithm [38] and optimized by BA [42]. Since the virtual keyframes only consider whether a map point is good enough regardless of which agent uploads it, they can supplement those low-quality segments.

**Map compression.** Once the quality of map segments is more balanced, *MBP* performs the similar map compression algorithm proposed by Sum-Map [27], eliminating redundancy by generating an enhanced minimum spanning tree across the global map. In addition, we introduce an extra optimization goal that guides the spanning tree to cover as many high-quality map elements as possible.

Fig. 7 compares the map compression performance of *MBP* and Sum-Map. Map elements from different agents are marked in red, blue, and brown in the figure. Although Sum-Map obviously reduces the map size, it neglects the map quality difference, making the compressed map of trace 2 too sparse and harming the SLAM performance. In contrast, with reducing the map size by nearly half, *MBP* inserts several virtual keyframes, balancing the map quality among different agents and ensuring mapping accuracy.

## 4 Implementation

We implement SwarmMap as an open-source package and make it compatible with ROS [26]. It contains 18,000 LOC (line of C++ code). SwarmMap is built upon ORB-SLAM2 [29], the top-ranked open-source SLAM algorithm that has been widely used by both research and industry communities. Our implementation avoids modifications on SLAM functions (e.g., tracking, local mapping, loop closing). This allows any variation of ORB-SLAM algorithms such as DynaSLAM [5], ORB-SLAM3 [7], as well as other map-point- and keyframe-based collaborative SLAM algorithms (e.g., Multi-UAV [39], C-ORB [22], CCM-SLAM [40]) to take advantage of SwarmMap (demonstrated in §A.5). Additionally, we also expose well-packaged APIs to facilitate users to modify some parameters (map synchronization period, status evaluation metrics, etc.) in SwarmMap according to specific upper-layer applications. A high-level abstraction of SwarmMap's implementation is detailed in §A.3.

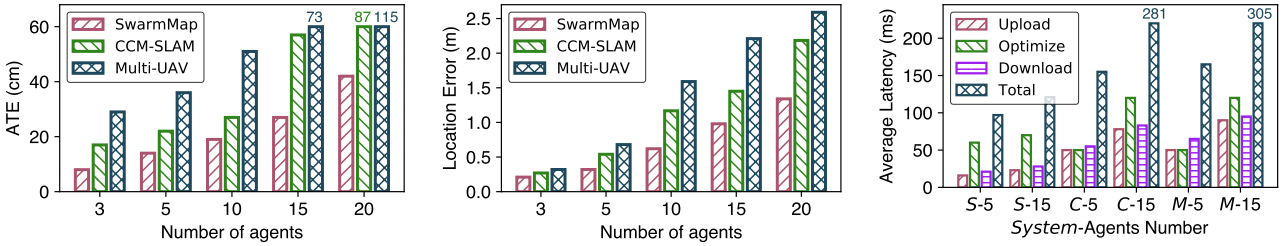
## 5 Evaluation

In this section, we first present the experimental methodology (§5.1), followed by the overall performance of SwarmMap compared against SOTA systems (§5.2). We then conduct an ablation study to understand each functional module in SwarmMap (§5.3). Further, we demonstrate the portability of SwarmMap by plugging it into baseline SLAM systems (§A.5).

### 5.1 Experimental Methodology

**Field studies**. We deploy 12 agents including 4 smartphones, 4 drones, and 4 mobile robots on a 22,927 sqft shopping mall. These agents collaboratively localize themselves and mapping the environment in real-time. The ground truth is obtained through the Kinect 360 RGB-D and Opti-Track [33] cameras. We also build a dataset using these video streams for trace-driven evaluation.

**Trace-driven evaluations**. Following the conventional visual SLAM evaluation methodology [2, 22, 40, 47], we also conduct comprehensive trace-driven evaluations based on public SLAM datasets (KITTI [10], EuRoC [9], and TUM [11]) and the handcrafted dataset mentioned above. The characterization of three public datasets is summarized in Table 4. In our evaluations, the movement speed of mobile agents various significantly, ranging from $0.5m/s$ (indoor DJI RoboMasters) to $15m/s$ (ourdoor vehicles), representing the status of devices in




(a) ATE comparison on public datasets. (b) Location error comparison on our dataset. (c) Map updating latency comparison

Figure 8: Overall performance comparison with a growing number of agents.

real world usage. Similar to the standard collaboration SLAM evaluation pipeline [19, 39, 40], we cut the video stream into overlapped segments and feed them to different agents to emulate the multi-agent scenario.

**Edge Setup**. Most of the previous works cannot be deployed on a resource-constrained edge node to support large numbers of agents because they consume a considerable amount of network bandwidth and edge computational resources (§2). We thus use a powerful server, which is equipped with an Intel(R) Xeon(R) CPU E5-2620v4 of 2.10GHz main frequency and 64GB RAM running Ubuntu 18.04, to explore the capacity of these systems and compare them with SwarmMap. The agents communicate with the server through 2.4 GHz and 5 GHz Wi-Fi links in the shopping mall and our laboratory. The maximally achievable link throughput measured with `iperf3` is 27.4MB/s and 46.1MB/s, respectively.

**Metrics.** We use *absolute trajectory error* (ATE, in *cm*) to evaluate SLAM accuracy on the three public datasets while adopting *location error* (in *m*) to evaluate the positioning accuracy in field studies and our handcrafted shopping mall dataset. ATE is a golden metric for evaluating the tracking performance of SLAM algorithms [11]. Since ATE pre-calibrates the generated trace with the ground-truth trajectories before measuring the absolute errors, it achieves fewer errors than the actual location errors. To evaluate system overhead, we count the *bandwidth demand* (in MB/s) of all participants in the system (defined as the sum of the average volume of data transferred per second by all agents). Similar to previous works [40, 47], we store the global map in RAM rather than SSD during system operation for faster map recall and update. We hence record the *RAM usage* (in GB) on the edge server to measure the memory consumption.

**Map updating latency.** Similar to previous works such as Edge-SLAM [3] and CCM-SLAM [40], SwarmMap adopts the same edge-assisted architecture where the *tracking* task is running locally on the agents. This allows an agent to localize itself in real-time (i.e., >30 fps with camera rate). We thus take the *map updating latency* (in *ms*)–the delay until the agent gets the latest optimized map from the server–as the metric to evaluate the real-time performance of map updating in SwarmMap. Map updating latency takes into account both the map synchronization and optimization latency.

## 5.2 Overall Performance Comparison

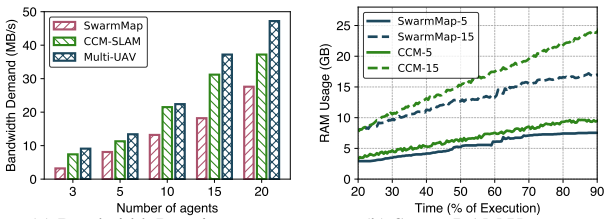
We first compare SwarmMap with CCM-SLAM [40] and Multi-UAV [39], two most relevant SOTA edge-based multi-agent SLAM systems, to evaluate the overall performance.

### 5.2.1 Accuracy Comparison

We first evaluate the average ATE and location error in a different number of agent settings. The results are depicted in Fig. 8a and Fig. 8b. As seen, SwarmMap achieves the best tracking and localization performance in all scenarios. Compared with related works, SwarmMap reduces ATE by $>$ 30%, 20%, 20%, 50%, 55% for scales with 3, 5, 10, 15, 20 agents, respectively. The location errors are also significantly degraded by >40% when serving more than 10 agents. On the other hand, the performance of CCM-SLAM and Multi-UAV degrades remarkably with the growing number of agents. (i.e., the ATE and location errors expand 3$\times$ and 7$\times$ respectively from 3 to 20 agents). When serving more than 10 agents in the shopping mall, they fail to guarantee that the average location error of each client is within 1.5*m*, which is typically the localization precision requirement for indoor drones [48]. In contrast, SwarmMap can still bound ATE and location error within 40*cm* and 1.4*m* even serving 20 agents. Generally speaking, above delightful results come from the fact that the localization performance of each agent highly depends on the quality of the on-board maintained local map [3, 47], and the three modules (*Mapit*, *STS*, and *MBP*) in SwarmMap exactly enable each agent to acquire an optimized local map in time.

### 5.2.2 Map Updating Latency Comparison

We further examine the end-to-end latency of each agent from uploading map segments to eventually obtaining the optimized map from the edge node. To save space in the figure, we denote SwarmMap, CCM-SLAM, and Multi-UAV as *S*, *C*, and *M*, respectively. Fig. 8c shows the averaged latency on map uploading, optimizing, and downloading of each system in different number of agent settings. As seen, the total latency of SwarmMap is around 95*ms* and 105*ms* for 5 and 15 agents respectively, outperforming baselines by $>$ 40% and 65%. The majority part of the latency reduction comes from the data uploading and downloading because *Mapit* reduces the amount of data transfers to a large extent. On the other hand,




(a) Bandwidth Requirement (b) Server RAM Usage

Figure 9: Resource overhead comparison with a growing number of agents.

the processing latency also drops around 10% when serving 15 agents as *STS* module reduces the averaged queuing delay.

#### 5.2.3 Resource Overhead Comparison

**Bandwidth Demand**. We then measure the bandwidth demand of these three systems. As depicted in Fig. 9a, on average, SwarmMap reduces $>$ 35%, 20%, 30%, 25%, 20% of network bandwidth requirement when serving 3, 5, 10, 15, 20 mobile agents compared with existing works. Said differently, SwarmMap could serve more agents with the same wireless link throughput. For instance, under 27.4MB/s shopping mall bandwidth limitation, SwarmMap can support more than 20 agents while existing works merely around 10.

**RAM Usage**. We stitch the 00-05 trajectories on the KITTI dataset to generate a trajectory with 16.2*km* length and conduct a 30min experiment to measure the RAM usage. As shown in Fig. 9b, compared to CCM-SLAM, SwarmMap saves an average memory overhead of 2GB and 6GB when serving 5 and 15 agents, respectively, and the map size becomes stable under an upper bound (as seen, 15GB when serving 15 agents) once the whole scene is well mapped. Unlike CCM-SLAM which requires transmission of a large volume of map elements, SwarmMap leverages *Mapit* and significantly reduces the bandwidth demand for map synchronization. In addition, the *MBP* module prunes the size of the global map maintained and optimized on the server, thus reducing the system overhead on computational resources.

Generally speaking, SwarmMap aims to scale the collaborative SLAM service with the same resource overhead at the edge. SwarmMap will achieve a better performance with more computational resources are allocated and advanced resource management technologies (e.g., swap or virtual memory) are leveraged on edge, which are left as future works.

### 5.3 Ablation Study

We then conduct an ablation study to understand the effectiveness of each module in SwarmMap.

**Performance of *Mapit***. We compare *Mapit* with CarMap [2], CCM-SLAM [40], and benchmark (e.g., edgeSLAM [47] and Edge-SLAM [3] that directly transmit the entire map without feature compression). Table 1 records the average data interaction speed (i.e., the average amount of map data uploaded and downloaded by each agent per second) of them

Table 1: Transmitted data volume comparison.

| Solution | Average Data Interaction Speed (MB/s) | | | |
|---|---|---|---|---|
| | TUM | KITTI | EuRoc | Shopping Mall |
| *Mapit* | **1.3** | 1.1 | 1.3 | **1.4** |
| CarMap | 1.9 | **0.9** | **1.2** | 1.8 |
| CCM-SLAM | 3.2 | 1.9 | 2.2 | 2.9 |
| Benchmark | 5.2 | 4.3 | 4.7 | 4.9 |

Table 2: Map compression performance comparison.

| Solution | KITTI 02 | | KITTI 05 | |
|---|---|---|---|---|
| | Map Size (GB) | ATE (cm) | Map Size (GB) | ATE (cm) |
| *MBP* | 3.1 | **7.6** | 1.9 | **6.4** |
| Sum-Map | 2.8 | 10.7 | 1.8 | 9.3 |
| Benchmark | 5.2 | 7.4 | 4.1 | 5.8 |

on different datasets. As seen, *Mapit* saves nearly two times the bandwidth compared to CCM-SLAM and benchmark on all datasets. *Mapit* performs slightly worse than CarMap on KITTI and EuRoc datasets, where the operating environments are relatively large (e.g., broad city roads). In these scenarios, the agents spend most of their time in the cold-start session during which they continuously transfer the newly generated map elements. In contrast, on TUM and our shopping mall datasets, the SLAM system completes the environment profiling quickly and soon enters the maintenance session during which *Mapit* eliminates map data transfer and saves the bandwidth by adopting the strategy of transmitting only records of map modifications rather than the modifications themselves.

**Performance of *STS***. We evaluate *STS* by counting the average tracking lost percentage (i.e., proportion of video frames, with which agents fail to track themselves, in all video frames) of SwarmMap with (w/) and without (w/o) *STS*. As depicted in Fig. 10, despite the increasing service scale, SwarmMap (w/ *STS*) maintains a stable service quality, and the lost percentage is within 4% in all scenarios. In contrast, the lost percentage of CCM-SLAM as well as SwarmMap (w/o *STS*) increases rapidly, and the average lost percentage is at least 8% when serving more than 10 agents, which may lead to a terrible self-tracking and environmental mapping performance. Generally speaking, the *STS* strategy enables SwarmMap to prioritize tasks depending on the agent emergence states and prevent most agents from losing self-tracking.

**Performance of *MBP***. We finally compare *MBP* with a map compression algorithm Sum-Map [27]. Specifically, we evaluate the map size after compression by their approaches and, equally important, the localization accuracy of each agent using the compressed map for self-tracking. The results are recorded in Table 2. We conduct experiments on the KITTI 02 and 05 trajectories because of the large map redundancy in them. The benchmark (only store the global map without compressing it) shows the size of the original map and the ATE by using it. As seen, *MBP* reduces the original map size by almost half. Although the map compression ratio of *MBP* is a little smaller than that of Sum-Map, *MBP* barely sacrifices the accuracy of the global map.

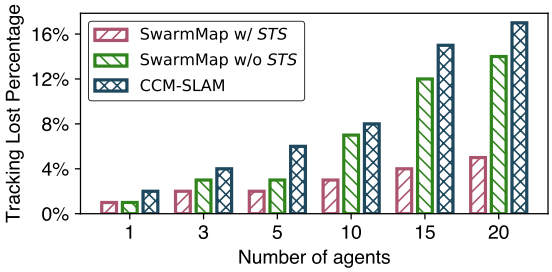


Figure 10: Tracking stability comparison.

## 6 Oil-field Case Study

Based on SwarmMap, we have developed a real-time collaborative visual SLAM system and deployed it in a large oil-field ($> 170km^2$) for industrial inspection. The details about the deployment setups can be found in §A.6. We conduct a three-month pilot study (from June 2021 to August 2021) and summarize our main findings regarding the SLAM accuracy and system overhead.

**SLAM Accuracy.** We calculate the average location error of each agent during inspections and present these results in Fig. 11 and Fig. 12. Note that we cannot directly obtain ground-truth in the same way as in the experiment (e.g., deploy expensive Lidar or Opti-Track cameras), hence we collect the video frames captured by all agents and run the multi-agent ORB-SLAM3 offline afterward without considering the system latency. On this basis, we take the difference between the real-time localization performance of SwarmMap and offline processed results as the location error. As shown in Fig. 11, the average location error is $19.3cm$ and $29.1cm$ in indoor and outdoor scenarios, respectively, satisfying the task requirement ($1m$ and $1.5m$ for indoor and outdoor inspections). Fig. 12 further illustrates the performance of each agent, and we find that two outdoor inspection drones (agents 9 and 10) suffer from a higher location error (up to $1m$). The reason behind it is that these two drones are carrying out oil pipeline inspection at the border of the oil field; they fly faster (e.g., $> 5m/s$) and far away from the edge server (e.g., $15km$). Therefore, they may experience certain delays due to the data forwarding through multi-hop mesh networks. Such a transmission delay may set a barrier for the drones to obtain the optimized map in time, causing localization errors. Nevertheless, the worst localization error of these two drones still satisfies the localization requirement in the outdoor scenario.

**Latency**. We measure each agent's onboard localization latency (the delay on estimating its own location from an input image) and map updating latency. The results are depicted in Fig. 13. We observe that each agent could localize itself in a real-time manner (i.e., the localization delay is within 35ms, typically the camera inter-frame interval). The average map updating delay is around 100ms. Although agent 9 suffers from a higher map updating delay (an average of 191ms) due to multi-hop data forwarding, it can still localize itself in real-time by leveraging its local map data.

**Bandwidth demand**. We record the total bandwidth demand for indoor (4 agents) and outdoor (8 agents) inspection tasks. Fig. 14 shows a snapshot over a span of 175 minutes. We find there is a drop in bandwidth demand at 45min and 75min, respectively. This is because the SLAM system enters the maintenance session at these two time points. Thanks to *Mapit*, the transferred data volume in the maintenance session is significantly reduced, with 4MB/s for indoor and 11MB/s for outdoor inspections. Additionally, due to the relatively higher flight speed and map updating delay for outdoor drones, the edge server needs to frequently transmit updated maps to them in *Mapit pull* to prevent them from losing self-tracking, which results in the outdoor bandwidth demand fluctuates more dramatically than indoor ones.

**RAM Usage**. We further record the edge's RAM usage when executing the indoor and outdoor inspection tasks. As shown in Fig. 15, the maximum RAM usage in the indoor and outdoor scenarios is around 20GB and 12GB, both of which are well below the capability (32GB) of the edge node.

**On-board CPU Usage**. We also record the CPU occupancy rate of SwarmMap task (mobile part) on agent 1 (indoor drone) and 6 (outdoor drone) and plot these results in Fig. 16. The CPU usage of the outdoor drone is in the range of 20%-35%, while the indoor drone is 22%-43% during the 210 minutes of inspections. Due to the high dynamics of the indoor environment, the agent has to frequently update the local map although the whole area is well-mapped, which takes up more CPU resources than outdoor environments. Note that SLAM is an underlying algorithm that provides an agent with location and environmental information, and SwarmMap still leaves more than 50% CPU computational resources for each agent to perform upper-layer applications (e.g., context-aware interaction, object detection, or segmentation).

## 7 Related work

We review the most related works in this section.

**Visual SLAM**. One of the most fundamental algorithms in robotics has been a topic of research in robotics and mobile systems for several decades [6]. It consists of the concurrent construction of a surrounding environment and the state estimation of the robot moving within it. Typically, systems use monocular cameras [15, 20], stereo cameras [29], or RGB-D cameras [31]. Some of the more well-known visual SLAM examples include RGBD-SLAM [16], RTAB-Map [21], and ORB-SLAM [7,28,29]. Although SwarmMap is implemented on the top of ORB-SLAM2 [29], it can be easily ported to other map point-based visual SLAM like S-PTAM [34]. Other multi-map merging or optimization algorithms leveraged in recent work like ORB-SLAM3 [7], can also be integrated into SwarmMap. Our platform can also be applied to some feature/map point-based multi-sensor SLAM systems like VI-ORB [30], VINS [35], mmWave SLAM [24, 44].

**Edge-assisted Real-time SLAM.** Recent studies [2, 3, 8, 23, 40, 47] speed up the computation-intensive tasks on agents by task partition and offloading workload to an edge server.

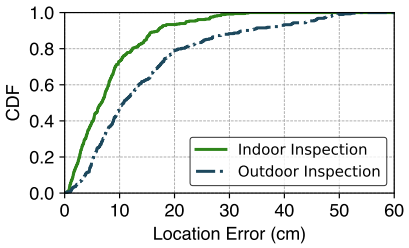



Figure 11: Different scenarios accuracy.

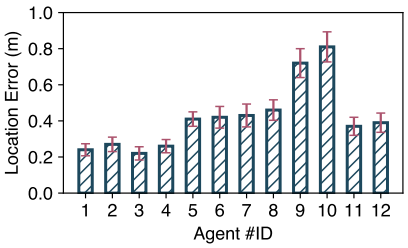



Figure 12: Accuracy of different agents.

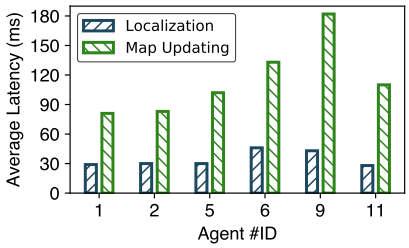



Figure 13: Latency measurement.

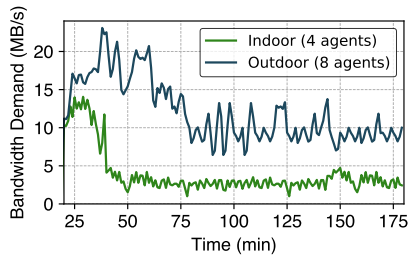



Figure 14: Bandwidth consumption.

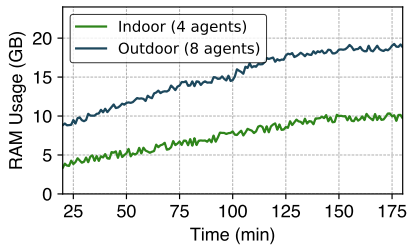



Figure 15: Edge server RAM usage.

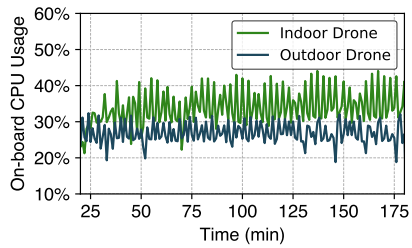



Figure 16: Mobile CPU occupation.

Therein, edgeSLAM [47] and Edge-SLAM [3] enable mobile agents to run visual SLAM in real-time. They split the original ORB-SLAM2 architecture and offload the *local mapping* and *loop closure* tasks to an edge server. CarMap [2] leverages the map constructed by crowdsourced agents and designs a near real-time map update framework between client and cloud. Muti-UAV [39] and CCM-SLAM [40] leverage a central server with potentially larger computational capacity to merge and optimize maps constructed by different agents, while each agent maintains partial local maps for tracking. However, as the number of serving agents scales, these works face severe scalability issues including excessive bandwidth consumption, severe localization errors, and large data storage. SwarmMap is the first work that solves these scalability issues based on the same edge settings.

**Multi-agent Collaborative SLAM.** Collaborative SLAM has been explored recently [6]. C2TAM [37], C-ORB [22], and CVI-SLAM [19] present collaborative SLAM frameworks based on PTAM [20], ORB-SLAM2 [29], and VI-ORB [7] respectively. CSfM [17] also proposes a framework to coordinate maps upload from different agents. In general, the system goals of these works and SwarmMap are orthogonal: above systems mainly focus on map fusion, optimization, and segmentation to generate a high-quality global map of the environment, ignoring the real-time performance of each agent and the entire system. In contrast, SwarmMap aims at solving the scalability issues and support each agent for real-time tracking, mapping, and map updating. Inspired by current efforts, we could integrate some map merging, optimizing, and even compressing algorithms proposed by recent works [6, 13, 14, 27, 49] into SwarmMap for a better SLAM performance, which are left as future works.

## 8 Discussion

We briefly discuss limitations and future work in this section.

**The capacity of SwarmMap**. Although SwarmMap significantly reduces the bandwidth consumption and memory overhead for collaborative visual SLAM systems, such resource consumption still grows linearly with the number of the agents, which still fundamentally limits the system capacity. The way to make the resource consumption grow sub-linearly [18] with respect to the number of agents worth further research. On the other hand, the current *Mapit* design merely focuses on reducing bandwidth consumption in the maintenance session. Serving the system throughput the entire life-cycle with *Mapit* could potentially save more bandwidth.

**Map optimization algorithms integration.** SwarmMap provides a basic map transmission and management platform for multi-agent SLAM. To date, SLAM map optimization is still a trending topic in the robotics field. Integrating existing advanced technologies (e.g., map compression, fusion, and semantic recognition) into SwarmMap for a better system performance is an ongoing work. Furthermore, efficient map data sharing not only between mobile and edge, but among different agents could also benefit upper layer applications.

## 9 Conclusions

We have presented the design and implementation SwarmMap, a framework to support real-time collaborative visual SLAM at edge devices. SwarmMap proposes functionality and resource abstractions of SLAM systems and provides three light-weight system services to address the communication, storage, and scheduling issues in edge-based scenarios. We implement SwarmMap as a software package on the ROS platform so that most variations of visual SLAM systems can directly benefit from it. Extensive evaluations and a three-month pilot study demonstrate its superior performance.

## Acknowledgements

We thank the MobiSense group, the anonymous reviewers and our shepherd, Ramesh Govindan, for their insightful comments. This work is supported in part by the NSFC under grant 61832010, 61972131.

## References


[1] Numba GPU Acceleration. https://numba.pydata.org/.

[2] Fawad Ahmad, Hang Qiu, Ray Eells, Fan Bai, and Ramesh Govindan. Carmap: Fast 3d feature map updates for automobiles. In *Proceedings of the USENIX NSDI*, 2020.

[3] Ali J Ben Ali, Zakieh Sadat Hashemifar, and Karthik Dantu. Edge-slam: edge-assisted visual simultaneous localization and mapping. In *Proceedings of the ACM Mobisys*, 2020.

[4] ArduPilot. https://ardupilot.org/ardupilot/.

[5] Berta Bescos, José M Fácil, Javier Civera, and José Neira. Dynaslam: Tracking, mapping, and inpainting in dynamic scenes. *IEEE Robotics and Automation Letters*, 3(4):4076–4083, 2018.

[6] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John J Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on robotics*, 2016.

[7] Carlos Campos, Richard Elvira, Juan J Gómez Rodríguez, José MM Montiel, and Juan D Tardós. Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam. *IEEE Transactions on Robotics*, 2021.

[8] Tiffany Yu-Han Chen, Lenin Ravindranath, Shuo Deng, Paramvir Bahl, and Hari Balakrishnan. Glimpse: Continuous, real-time object recognition on mobile devices. In *Proceedings of the ACM Sensys*, 2015.

[9] EuRoC Dataset. https://projects.asl.ethz.ch/datasets/.

[10] KITTI Dataset. http://www.cvlibs.net/datasets/kitti/eval_odometry.php.

[11] TUM Dataset. https://vision.in.tum.de/data/datasets/rgbd-dataset/tools.

[12] DJI drones for industrial inspection. https://www.dji.com/products/industrial.

[13] M. Dymczyk, S. Lynen, M. Bosse, and R. Siegwart. Keep it brief: Scalable creation of compressed localization maps. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2536–2542, 2015.

[14] Marcin Dymczyk, Thomas Schneider, Igor Gilitschenski, Roland Siegwart, and Elena Stumm. Erasing bad memories: Agent-side summarization for long-term mapping. In *Proceedings of the IEEE IROS*, 2016.

[15] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *Proceedings of the Springer ECCV*, 2014.

[16] Nikolas Engelhard, Felix Endres, Jürgen Hess, Jürgen Sturm, and Wolfram Burgard. Real-time 3d visual slam with a hand-held rgb-d camera. In *Proceedings of the RGB-D Workshop on 3D Perception in Robotics at the European Robotics Forum, Vasteras, Sweden*, volume 180, pages 1–15, 2011.

[17] Christian Forster, Simon Lynen, Laurent Kneip, and Davide Scaramuzza. Collaborative monocular slam with multiple micro aerial vehicles. In *Proceedings of the IEEE IROS*, 2013.

[18] Samvit Jain, Xun Zhang, Yuhao Zhou, Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, Paramvir Bahl, and Joseph Gonzalez. Spatula: Efficient cross-camera video analytics on large camera networks. In *Proceedings of the IEEE/ACM SEC*, 2020.

[19] Marco Karrer, Patrik Schmuck, and Margarita Chli. Cvi-slam—collaborative visual-inertial slam. *IEEE Robotics and Automation Letters*, 3(4):2762–2769, 2018.

[20] Georg Klein and David Murray. Parallel tracking and mapping for small ar workspaces. In *Proceedings of the IEEE ISMAR*, 2007.

[21] Mathieu Labbé and François 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*, 2019.

[22] Fu Li, Shaowu Yang, Xiaodong Yi, and Xuejun Yang. Corb-slam: a collaborative visual slam system for multiple robots. In *International Conference on Collaborative Computing: Networking, Applications and Worksharing*. Springer, 2017.

[23] Luyang Liu, Hongyu Li, and Marco Gruteser. Edge assisted real-time object detection for mobile augmented reality. In *Proceedings of the ACM Mobicom*, 2019.

[24] Chris Xiaoxuan Lu, Stefano Rosa, Peijun Zhao, Bing Wang, Changhao Chen, John A Stankovic, Niki Trigoni, and Andrew Markham. See through smoke: robust indoor mapping with low-cost mmwave radar. In *Proceedings of the ACM MobiSys*, 2020.

[25] Yunfei Ma, Nicholas Selby, and Fadel Adib. Drone relays for battery-free networks. In *Proceedings of the ACM Sigcomm*, 2017.

[26] ROS Melodic. https://wiki.ros.org/melodic.

[27] Peter Mühlfellner, Mathias Bürki, Michael Bosse, Wojciech Derendarz, Roland Philippsen, and Paul Furgale. Summary maps for lifelong visual localization. *Journal of Field Robotics*, 33(5):561–590, 2016.

[28] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.

[29] Raul Mur-Artal and Juan D Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017.

[30] Raúl Mur-Artal and Juan D Tardós. Visual-inertial monocular slam with map reuse. *IEEE Robotics and Automation Letters*, 2(2):796–803, 2017.

[31] Richard A Newcombe, Steven J Lovegrove, and Andrew J Davison. Dtam: Dense tracking and mapping in real-time. In *Proceedings of the IEEE ICCV*, 2011.

[32] Van Opdenbosch et al. *Data Compression for Collaborative Visual SLAM*. PhD thesis, Technische Universität München, 2019.

[33] Opti-Track. https://optitrack.com/.

[34] Taihú Pire, Thomas Fischer, Gastón Castro, Pablo De Cristóforis, Javier Civera, and Julio Jacobo Berlles. S-ptam: Stereo parallel tracking and mapping. *Robotics and Autonomous Systems*, 93:27–42, 2017.

[35] Tong Qin, Peiliang Li, and Shaojie Shen. Vins-mono: A robust and versatile monocular visual-inertial state estimator. *Proceedings of the IEEE Transactions on Robotics*, 2018.

[36] Hang Qiu, Fawad Ahmad, Fan Bai, Marco Gruteser, and Ramesh Govindan. Avr: Augmented vehicular reality. In *Proceedings of the ACM MobiSys*, 2018.

[37] Luis Riazuelo, Javier Civera, and JM Martınez Montiel. C2tam: A cloud framework for cooperative tracking and mapping. *Robotics and Autonomous Systems*, 62(4):401–413, 2014.

[38] Szymon Rusinkiewicz and Marc Levoy. Efficient variants of the icp algorithm. In *Proceedings of the IEEE 3-D digital imaging and modeling*, 2001.

[39] Patrik Schmuck and Margarita Chli. Multi-uav collaborative monocular slam. In *Proceedings of the IEEE ICRA*, 2017.

[40] Patrik Schmuck and Margarita Chli. Ccm-slam: Robust and efficient centralized collaborative monocular simultaneous localization and mapping for robotic teams. *Journal of Field Robotics*, 36(4):763–781, 2019.

[41] CUDA Toolkit. https://developer.nvidia.com/cuda-toolkit.

[42] Bill Triggs, Philip F McLauchlan, Richard I Hartley, and Andrew W Fitzgibbon. Bundle adjustment—a modern synthesis. In *Proceedings of the Springer International workshop on vision algorithms*, 1999.

[43] Dominik Van Opdenbosch and Eckehard Steinbach. Collaborative visual slam using compressed feature exchange. *IEEE Robotics and Automation Letters*, 4(1):57–64, 2018.

[44] Teng Wei, Anfu Zhou, and Xinyu Zhang. Facilitating robust 60 ghz network deployment by sensing ambient reflectors. In *Proceedings of the USENIX NSDI*, 2017.

[45] Amazon Warehouse with Robots. https://www.wired.com/story/amazon-warehouse-robots/.

[46] Nvidia Jetson AGX Xavier. https://developer.nvidia.com/embedded/jetson-agx-xavier-developer-kit.

[47] Jingao Xu, Hao Cao, Danyang Li, Kehong Huang, Chen Qian, Longfei Shangguan, and Zheng Yang. Edge assisted mobile semantic visual slam. In *Proceedings of the IEEE INFOCOM*, 2020.

[48] Shengkai Zhang, Wei Wang, and Tao Jiang. Wi-fi-inertial indoor pose estimation for microaerial vehicles. *Transactions on Industrial Electronics*, 68(5):4331–4340, 2020.

[49] Danping Zou, Ping Tan, and Wenxian Yu. Collaborative visual slam for multiple agents: A brief survey. *Virtual Reality & Intelligent Hardware*, 1(5):461–482, 2019.

# A Appendix

## A.1 Functions Registered in *Mapit*

In the *Mapit add* module, we dig the insights about how map elements get changed and find these changes mainly caused by certain important SLAM functions, a fraction of which is listed in Table 3. Thus, modifications that happened to the map can be recorded as calling history of these functions. For certain functions shown in the table, some removal and compression on the records will not harm data consistency. For instance, if a function is marked as `overwritten`, it indicates that its only effective change on a map element is the latest one i.e., changing the pose of a map point. As for those

Table 3: Functions that could change the map element (only some fundamental functions are listed)

| Target | Function | Type | Description |
|---|---|---|---|
| KeyFrame | `SetPose` | overwritten | set the pose of the keyframe |
| KeyFrame | `AddMapPoint` | unique | add a map point to the keyframe |
| KeyFrame | `EraseMapPointMatch` | unique | remove a map point from the keyframe |
| KeyFrame | `SetBadFlag` | unique | mark the keyframe bad and delete it |
| MapPoint | `SetWorldPos` | overwritten | set map point position in the world coordinate |
| MapPoint | `AddObservation` | unique | add a keyframe that observes the map point |
| MapPoint | `EraseObservation` | unique | remove a keyframe from observations |
| MapPoint | `SetBadFlag` | unique | mark the map point bad and delete it |
| MapPoint | `IncreaseVisible` | stackable | increase the count that map point is observed |
| MapPoint | `IncreaseFound` | stackable | increase the count that map point is matched |
| MapPoint | `SetLastTrackedTime` | overwritten | set the last tracked time of the map point |
| MapPoint | `UpdateNormalAndDepth` | overwritten | update the normal vector and depth of the map point |
| Map | `Clear` | overwritten | clear the current map |
| Map | `AddLoopClosing` | unique | add a keyframe to loop closing queue |

Table 4: Dataset Description

| Dataset Label | Trajectory Sequence | Total Time (min) | Total Path (m) | Total Frames | Environment |
|---|---|---|---|---|---|
| T-M (TUM Medium & Easy) | fr2_desk | 1.66 | 18.88 | 2965 | office |
| | fr3_long_office_household | 1.45 | 21.46 | 2585 | |
| T-D (TUM Difficult) | fr2_large_with_loop | 2.88 | 39.11 | 5182 | industrial hall |
| | fr2_large_no_loop | 1.87 | 10.93 | 3359 | |
| K-M (KITTI Medium & Easy) | 00 / 05 | 7.57 / 4.79 | 3724.18 / 2205.58 | 4541 / 2761 | city road |
| K-D (KITTI Difficult) | 02 / 04 | 7.77 | 5067.23 / 393.65 | 4661 / 271 | city road |
| E-M (EuRoC Medium & Easy) | MH_01 / MH_02 | 2.47 / 2.50 | 68.52 / 73.50 | 3682 / 3040 | machine hall |
| E-D (EuRoC Difficult) | MH_04 / MH_05 | 1.65 / 1.85 | 91.70 / 97.59 | 2033 / 2273 | machine hall |
| Shopping Mall (Our Dataset) | N/A | 15 | 314.2 | 24,365 | shopping mall |

marked `stackable`, the implication is that records about modifying the same element can be merged by parameters and still yield the same effect.

## A.2 Map-point Evaluation Metrics

A typical SLAM map consists of two types of elements, map points and keyframes. Map points represent discrete 3D landmarks in the global coordinate, and keyframes are selected frames indicating poses and positions of the corresponding camera (as illustrated in Fig. 18 with corresponding notations in Table 5). EBM [14] introduces several features based on local geometry information; we list four important metrics to evaluate a map-point we used in *MBP*:

• **Observing Path Length**. The distance traveled while observing the map-point and is obtained as

$$\phi_d^i = \sum_{j \in \mathcal{S}^i} \|\mathbf{t}_{\mathrm{G}}^{j+1} - \mathbf{t}_{\mathrm{G}}^{j}\|_2.$$

• **Maximum Observing Distance**. The distance traveled between two most distant keyframes on a track, and each of them observes the map-point. Its computation requires maximization over all keyframes observing the same map-point, i.e.,

$$\phi_\delta^i = \max_{j,k \in \mathcal{S}^i} \|\mathbf{t}_{\mathrm{G}}{}^{j} - \mathbf{t}_{\mathrm{G}}{}^{k}\|_2.$$

• **Maximum Observing Angle**. The maximum angle between two keyframes that could observe the map-point and is obtained as

$$\phi_a^i = \max_{j,k \in \mathcal{S}^i} \arccos(\mathbf{r}_{\mathrm{G}}^{j,i} \cdot \mathbf{r}_{\mathrm{G}}^{k,i}).$$

• **Mean Re-projection Error**. Apart from the map-point track geometry, it is also worth considering the consistency of the map in the map-point's locality. EBM calculate the average re-projection error of each map-point to represent the mapping stability, i.e.,

$$\phi_p^i = \frac{\sum_{j \in \mathcal{S}^i} \|m_{i,j} - m'_{i,j}\|_2}{|\mathcal{S}^i|}.$$

## A.3 SwarmMap Abstraction

Fig. 17 shows the high-level abstraction of SwarmMap's implementation. The *MBP* module assists the map fusion and optimization unit to eliminate the data redundancy in the global map. The *STS* module replaces those handcrafted request handlers in conventional SLAM implementations [19, 39, 40] and thus alleviates the end users' development overhead. Finally, we replace the communication unit and map handlers with a unified *Mapit* module. Such a layered implementation decouples SwarmMap's functional modules, allowing the end-users to turn on/off each module as they need. It also avoids the deeply embedded manual code changes (e.g., defining handlers) that again challenge the system scalability.

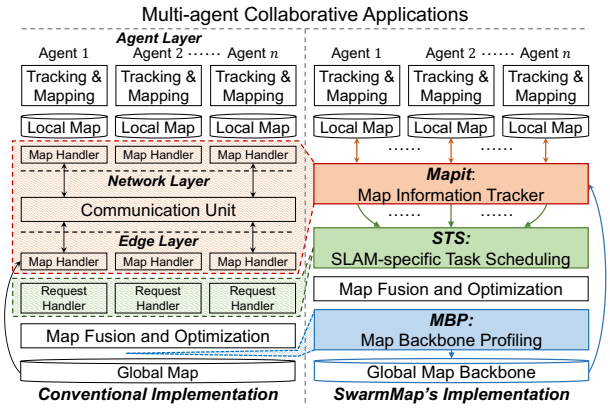



Figure 17: High-level abstraction of SwarmMap's implementation. The arrow shows the data flow.

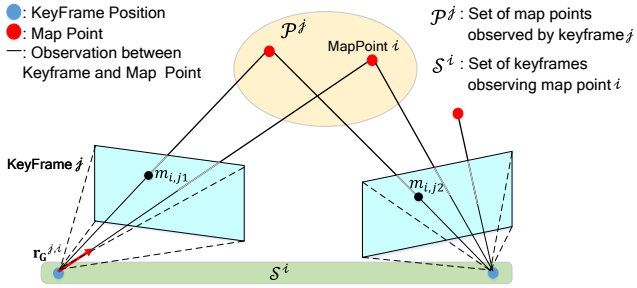



Figure 18: Observation connection between keyframes and map-points.

## A.4 Experimental Dataset Description

We list the public datasets, the trajectories we used, and our handcrafted shopping mall dataset in Table 4. We select representative trajectories with different difficulty levels (in terms of environmental dynamics, path length, feature point sparsity, ambient light intensity, etc..) in TUM, KITTI, and EuRoc datasets, respectively.

## A.5 Plug-and-play

We demonstrate the portability of SwarmMap by integrating each of its components into two different SLAM systems. We add *STS*, *Mapit*, and *MBP* to ORB-SLAM3 [7], the latest follow-up of the ORB-SLAM system, and measure the accuracy gain brought by each module. Fig. 20 shows the results. As seen, all these three modules contribute to localization accuracy. When serving 5 agents, *STS*, *Mapit*, and *MBP* decrease location errors by around 50% and 30%, and 10%, respectively. The contribution of each component also grows with an increasing number of agents.

We also integrate SwarmMap into our baselines CCM-SLAM, Multi-UAV, and ORB-SLAM3 (abbreviated as *C*, *M*, and *O*, respectively) to explore the location error reduction. As depicted in Fig. 21, the location error of CCM-SLAM, Multi-UAV, and ORB-SLAM3 decreases by 13.4%, 12.2%, and 16.7% respectively in 5 agents settings. When serving 15 agents, the error decreases further to 17.2%, 31.3%, and 29.6%.

**Remarks**. These results show that most existing works in multi-agent scenarios (especially scenarios with more agents) can directly benefit from SwarmMap. It is worth mentioning that we do not re-design or modify the code structure of these existing works for integration. We merely provide a wrapper to hook up these systems and SwarmMap (i.e., call the API defined in SwarmMap).

Table 5: Notation Description

| Notation | Description |
|---|---|
| $\mathbf{X}_{\mathtt{G}}{}^{i}$ | position of map point $i$ in global coordinate $\mathtt{G}$ |
| $\mathbf{t}_{\mathtt{G}}{}^{j}$ | position of keyframe $j$ in global coordinate $\mathtt{G}$ |
| $\mathcal{S}^{i}$ | set of all keyframes observing map point $i$ |
| $\mathbf{r}_{\mathtt{G}}{}^{j,i}$ | unit-length observing vector starting from the observing keyframe $j$ to map point $i$ in global coordinate $\mathtt{G}$ |
| $\mathcal{P}^{j}$ | set of all map points observed by keyframe $j$ |
| $\mathcal{M}^{i}$ | set of all agents observing map point $i$ |
| $t_c^i, t_l^i$ | creation and last tracked time for map point $i$ |

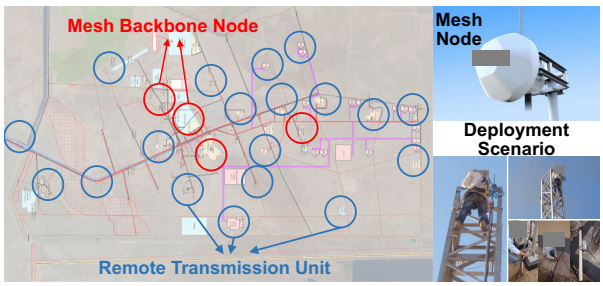



Figure 19: Mesh network deployment in the Oil-field.

## A.6 Case Study Setups

Our system consists of 12 mobile agents to perform daily inspection tasks both indoors and outdoors. These agents communicate with an Nvidia Jetson AGX Xavier edge node through Wi-Fi mesh networks, as shown in Fig. 19.

**Inspection agents**. We have deployed 12 mobile agents to perform daily inspection tasks, including 4 DJI Inspire drones (Agent #ID 1-4, equipped with 2K cameras) for indoor warehouse inspection as well as 6 DJI Inspire (#ID 5-10) and 2 inspection vehicles (#ID 11-12, equipped with 1080P cameras) for outdoor oil-field inspection. For drones, we integrate the mobile part of SwarmMap into ArduPilot [4], a widely-used open source drone development platform. The output localization and mapping results are streamed to the Ardupilot Mega controller through a Micro-USB port for supporting upper-layer applications (e.g., real-time drone flight control, abnormal events detection). The two inspection vehicles are equipped with Nvidia Jetson TX1 as their computing units.

**Edge server**. We implement the edge side of SwarmMap on an Nvidia Jetson AGX Xavier edge node with a 32GB 256-Bit LPDDR4x RAM, a 16-core ARM v8.2 64-bit CPU, and a

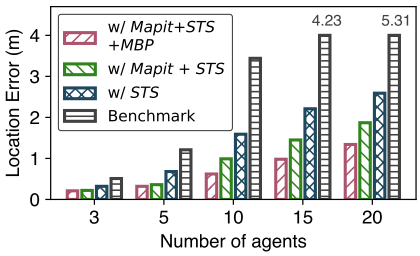



Figure 20: Performance of each module.

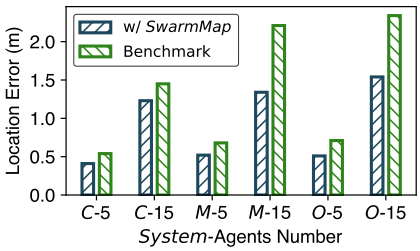



Figure 21: Performance gains.

512-core Volta GPU. We also turn on the GPU acceleration by Numba [1] and CUDA [41] to speed up the back-end global map optimization procedure. The power consumption of the edge node is below 30W, which is less than the available power supply in the industrial scenario.

**Wireless Network**. The 4 indoor inspection drones communicate with the edge node via 2.4 GHz WiFi, while the 8 outdoor inspection agents communicate through a mesh network. In order to make the mesh network cover the whole $170km^2$ outdoor oil-field (the west-east distance is around $30km$), 24 communication nodes, including 4 mesh backbone nodes and associated 20 remote transmission units (RTU) are deployed (shown in Fig. 19). The maximum throughput measured by `iperf3` in the outdoor mesh and indoor WiFi network is 14.3MB/s and 26.8MB/s, respectively.