CCM-SLAM: Robust and efficient centralized collaborative monocular simultaneous localization and mapping for robotic teams

Author(s):
Schmuck, Patrik; Chli, Margarita

Publication date:
2019-06

Permanent link:
https://doi.org/10.3929/ethz-b-000313259

Originally published in:
Journal of Field Robotics 36(4), https://doi.org/10.1002/rob.21854

CCM-SLAM: Robust and Efficient Centralized Collaborative Monocular SLAM for Robotic Teams

Patrik Schmuck
Vision for Robotics Lab
ETH Zurich
8092 Zurich, Switzerland
pschmuck@ethz.ch

Margarita Chli
Vision for Robotics Lab
ETH Zurich
8092 Zurich, Switzerland
chlim@ethz.ch

Abstract

Robotic collaboration promises increased robustness and efficiency of missions with great potential in applications, such as search-and-rescue and agriculture. Multi-agent collaborative Simultaneous Localization And Mapping (SLAM) is right at the core of enabling collaboration, such that each agent can co-localize in and build a map of the workspace. The key challenges at the heart of this problem, however, lie with robust communication, efficient data management and effective sharing of information amongst the agents. To this end, here we present CCM-SLAM, a centralized, collaborative SLAM framework for robotic agents, each equipped with a monocular camera, a communication unit and a small processing board. With each agent able to run visual odometry onboard, CCM-SLAM ensures their autonomy as individuals, while a central server with potentially bigger computational capacity enables their collaboration by collecting all their experiences, merging and optimizing their maps or disseminating information back to them, where appropriate. An in-depth analysis on benchmarking datasets addresses the scalability and the robustness of CCM-SLAM to information loss and communication delays commonly occurring during real missions. This reveals that in the worst case of communication loss, collaboration is affected, but not the autonomy of the agents. Finally, the practicality of the proposed framework is demonstrated with real flights of three small aircraft equipped with different sensors and computational capabilities onboard and a standard laptop as the server, collaboratively estimating their poses and the scene on the fly.

Supplementary Material

Video — https://youtu.be/P3b7UiTlmbQ
Code — https://github.com/VIS4ROB-lab/ccm_slam

1 Introduction

Simultaneous Localization And Mapping (SLAM) constitutes one of the most fundamental problems in Robotics, since ego-motion estimation and map-building are key in enabling autonomous navigation. Excluding sensors that rely on external tracking, such as GPS, for the sake of generality, SLAM approaches often choose to rely only on onboard sensing systems. Aiming for sensors that are able to provide rich information about their environment, while exhibiting portability and low power, it is no surprise that vision-based

Figure 1: The UAV platforms used for our experiments: Two AscTec Neos with payload of 2 kg and one AscTec Hummingbird (middle) with payload of 200 g.

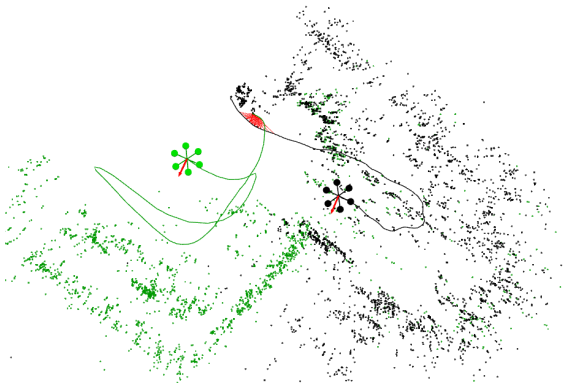


Figure 2: Two UAV agents collaboratively building a map using CCM-SLAM. Map points and trajectories for agent 1 and 2 are colored in black and green, respectively. Constraints across different trajectories in areas visited by both agents are indicated in red.

SLAM has become very popular. These properties render cameras particularly interesting onboard resource constraint platforms, such as small Unmanned Aerial Vehicles (UAVs) as the ones depicted in Figure 1, especially since early works in monocular SLAM (Eade and Drummond, 2006), (Davison et al., 2007) showed it is possible to perform SLAM with a single camera. With the state-of-the-art in SLAM having reached a certain maturity and robustness for single-robot scenarios (Engel et al., 2014), (Mur-Artal et al., 2015), multi-robot systems started to attract growing interest in robotics research. Involving multiple robots in a task promises to boost the efficiency of a mission (e.g. by dividing up a mapping task amongst the agents), increase the robustness of the system by sharing information, as well as enable robots to perform tasks not possible for a single robot, e.g. lifting heavy loads. Therefore, multi-agent systems promise great impact in a large variety of robotic applications, such as inspection of large industrial facilities and search-and-rescue scenarios.

While the biggest body of the literature focuses on SLAM with a single UAV, some works investigate either how information captured by multiple robotic agents can be merged to construct a single, global map, or the self-localization of agents in relation to each other on the go. However, both challenges simultaneously, i.e. collaborative localization *and* mapping from multiple robots, is only addressed by very few systems. It is only with such a collaborative SLAM functionality that we can use the full spectrum of possibilities in a multi-robot system, allowing sharing and re-use of information across the agents. This ability is key to collaborative robotic missions: all robots can be localized in a common map, which is constantly updated during the mission with the latest sensor measurements, to synchronize their activities and prevent collisions. Meanwhile, dealing with time delays, ensuring network connectivity and enabling transparent information access among all agents and the ground station pose great challenges in ensuring the consistency of collaborative SLAM.

In this spirit, here we propose **C**entralized **C**ollaborative **M**onocular SLAM (CCM-SLAM), a centralized collaborative SLAM system designed for multiple small UAVs or other robots, each equipped with a single camera and a small computer (CPU) onboard. These communicate with a ground station with potentially much more computational capabilities, as a result, any tasks that are computationally too expensive for the resource-limited agents (i.e. small UAVs here) are outsourced to the server, while ensuring that all tasks critical to the autonomy of each agent are still run onboard (e.g. visual odometry). Figure 2 shows a snapshot from a mission with two UAV agents participating in this framework. The information exchange between the agents and the server enables sharing of all information amongst all agents; for example, if Agent A visits a region that has already been visited Agent B, B's experiences are retrieved from the server and incorporated in A's onboard pose estimation process.

The main contributions of the proposed CCM-SLAM framework lie with combining the following key characteristics in one collaborative SLAM system:

- Efficient system architecture: the participating agents always preserve their autonomy running all navigation-critical tasks onboard, while all computationally expensive tasks on data management is pushed to the server.
- Robustness: a communication strategy is outlined, enabling CCM-SLAM to cope with limited bandwidth, network delays and message loss, most commonly occurring in a real setup.
- Information sharing: experiences collected by a single agent can be shared with any other, if they visit the same area.
- Scalability: efficient map management on the server detects and removes data with high redundancy without compromising the robustness of the estimation.
- Applicability: real-world flights are demonstrated and analysed with CCM-SLAM running onboard three UAVs with heterogeneous computational resources onboard and in a large-scale search-and-rescue environment.

With most existing literature demonstrating functionality on pre-recorded datasets simulating the agents using one PC and employing one-way communication (e.g. from the agents to the server), their practicality remains obscure, while the extent of collaboration that they allow within the team is restricted by design. Instead, CCM-SLAM addresses these by employing two-way transparent communication (i.e. informing the agents of others' experiences), and a thorough analysis on real, live experiments reveals the framework's strengths, weaknesses and practicality. This work builds on the centralized architecture proposed in Schmuck and Chli (2017), where a powerful proof of concept was presented, albeit demonstrated on pre-recorded datasets, avoiding to deal with issues arising during live estimation from communication over a wireless network. CCM-SLAM addresses these issues with an efficient and robust communication strategy that accounts for bandwidth constraints, demonstrated on real experiments, while it extends the server's map management capabilities to detect and remove redundancy without compromising robustness, boosting the scalability of this framework to more data from longer missions, larger scenes or more agents. In fact, CCM-SLAM is shown to run live with three UAVs flying simultaneously, while handling network problems, such as delays and message loss. To the best of our knowledge, this is the first time a collaborative SLAM system enabling communication both from the agents to the server and vice-versa is demonstrated *during a real-world flight* running onboard three UAVs.

2 Related Work

Following the emergence of real-time monocular SLAM techniques (Davison et al., 2007), (Klein and Murray, 2007), it was not long before first works started employing them onboard computationally constraint platforms such as UAVs (sfly, 2009). Strasdat et al. (2012) conducted an investigation of filter- and keyframe-based SLAM and concluded that the latter is overall more promising for robust and practically applicable SLAM systems. The field has been experiencing great progress since then in the robustness of the estimation processes and the size of the maps that can be managed (Engel et al., 2014), (Mur-Artal et al., 2015), resulting to some degree of maturity in single-robot SLAM paving the way to multi-robot directions. While ORB-SLAM (Mur-Artal et al., 2015) is a *feature-based* (or indirect) approach extracting salient regions in each camera frame, LSD-SLAM is a *direct* method, operating directly on image intensities. The recent work of Engel et al. (2017) shows remarkable performance with such a direct approach. However, operating directly on the image intensities complicates re-using existing map data when returning to a previously visited location, since the 3D map points cannot directly be matched to image points using a descriptor as in feature-based methods. Furthermore, a collaborative system using a direct method would require exchanging images between participants, therefore significantly increasing the network traffic compared to a feature-based approach where the exchanged image data can be reduced to the 2D feature keypoints extracted from the image. Therefore, indirect methods are currently more suited for multi-agent systems that target extensive exchange and re-use of data.

2.1 Multi-Robot Localization

Several works in the literature tackle the localization problem in a multi-robot scenario, with the aim to estimate the relative position between the robots, or the position of the robots in a map known in advance. The work in Martinelli et al. (2005) estimates the relative robot configuration based on mutual observations using an Extended Kalman Filter (EKF). Achtelik et al. (2011) take this idea a step further, showing a system where a flexible stereo rig is formed by two UAVs with monocular cameras and Inertial Measurement Unit (IMU). This system is then used to estimate the relative pose of the UAVs on a simple experimental setup. Piasco et al. (2016) adopt this approach and build a distributed stereo-vision system with multiple UAVs for collaborative localization, using an EKF scheme. The work of Luft et al. (2016) also employs an EKF to track correlations between robots in a decentralized multi-robot system. Oleynikova et al. (2015) present a visual-inertial system for a UAV and a ground-based robot that localize against the same map. The recent work of Leonardos and Daniilidis (2017) shows a theoretical approach for EKF-based decentralized multi-robot state estimation, which is tested in simulation and achieves less conservative estimates than using covariance intersection. While improving the localization estimates by using information from multiple robots, these collaborative localization systems do not fuse or share maps between the robots, and without the ability to build a common map, collaborative missions are very restricted.

2.2 Multi-Robot Mapping

In collaborative mapping the robotic agents participating in the mission contribute to one global map in order to jointly create a reconstruction of the environment. In Vidal-Calleja et al. (2011), the authors present such a collaborative mapping system for ground-based robots and UAVs. Each robot uses an EKF-scheme to build a local sub-map of its environment. The sub-maps are linked based on GPS measurements, co-observations or robot rendezvous. Other works, such as Google's "Project Tango" (Google Inc., 2014) and the work of Guo et al. (2016), do not directly target robotic application scenarios, but collect data with multiple hand-held platforms such as tablets or mobile phones to build a common map with the input from these devices. The recent work of Loianno et al. (2016) demonstrates a system that combines both UAVs and mobile phones. Using multiple UAVs with off-the-shelf smartphones as computation units, they present a system for map building in a cooperative manner and trajectory planning. Once collaborative SLAM established a collective estimate of the environment and the relative localization of the agents, systems such as Loianno et al. (2016) and other approaches such as Chung and Slotine (2009), Kushleyev et al. (2013) can be employed on top of it to guide the trajectories of the agents, even avoiding hazardous areas of the environment as demonstrated in Schwager et al. (2017).
Collaborative mapping offers the possibility to boost the efficiency of 3D reconstruction of structures and environments since information from multiple robots is merged into one global map. However, the map is usually not shared amongst the agents, and the position of the team members remains unknown to the agents, prohibiting collaboration amongst the agents going beyond pure scene reconstruction. Forster et al. (2013) show a collaborative SLAM system for UAVs building on top of a Structure From Motion pipeline. Onboard each agent runs a keyframe-based visual odometry system, sharing its keyframes with a central server. The server searches for correspondences between maps to merge them and performs global optimization on the maps. While this was probably the first collaborative SLAM system to address a multi-UAV setup, no information is sent back to the agents. As a result, the agents cannot profit from the collaborative estimation process and optimized map and receive information from other agents, e.g. when exploring an area already explored by another agent in the past. Generally, "collaborative SLAM" is used in a rather loose sense in literature. Since the map on the server is not used for pose tracking on the agent at any later stage, systems such as Forster et al. (2013) target basically collaborative mapping, not collaborative SLAM.

2.3 Multi-Robot Centralized SLAM

The work in Zou and Tan (2013) presents CoSLAM, a collaborative monocular SLAM system with a focus on handling dynamic environments. Their approach takes image streams from multiple monocular cameras

as input and groups cameras with scene overlap using a place recognition module. Since a keyframe is captured by all cameras at the same time, CoSLAM requires the cameras to be synchronized, restricting the autonomy of the agents. Furthermore, this system runs its calculations on a Graphics Processing Unit (GPU), and initially requires all cameras to observe the same scene to initialize the collaborative SLAM estimation. These drawbacks render this system impractical to use on mobile robots.
Probably the most important work in the literature, Riazuelo et al. (2014), the authors present C^2TAM for two RGB-D cameras. C^2TAM runs PTAM (Klein and Murray, 2007) on the RGB-D image feeds for SLAM estimation, where only the position tracking is executed on the agent. Each agent creates keyframes and sends them to the server, that executes all mapping tasks. For further tracking steps, the server then sends the complete map back to each agent. Therefore, C^2TAM can be used with very limited computational resources. However the agent cannot operate autonomously without relying on the server, and the underlying assumption of being able to repeatedly send whole maps to the agent is doomed to become problematic in larger areas. Furthermore, PTAM was a seminal work in keyframe-based SLAM, though current state-of-the-art SLAM outperforms it (Mur-Artal et al., 2015), achieving more robust and precise estimates in larger areas. As a result, experiments with C^2TAM could only be run in a small office environment, leaving open questions on the practicality and consistency of the whole system.
Morrison et al. (2016) designed MOARSLAM, a collaborative SLAM system with the target of enabling multi-device mapping with hand-held devices. Each agent in this system runs "full" SLAM onboard (i.e. visual odometry, place recognition and global map optimization). The server acts as a central device for storing and sharing the agents' maps. The architecture of this system falls short of exploiting the full capacity and advantages that a collaborative system offers since the expensive optimization algorithms are still run on the agent's side. The server detects correspondences between maps but does not run a global optimization.
The system by Deutsch et al. (2016) presents a server back-end for collaborative SLAM that is able to combine different SLAM systems in a collaborative framework. To use this server module, the combined SLAM systems need to provide it with a pose graph including keyframe and an image linked to it, and absolute scale, thus excluding, amongst other systems, all pure monocular SLAM systems such as Davison et al. (2007), Klein and Murray (2007), Mur-Artal et al. (2015). The server back-end merges and optimizes the maps received by the agents if possible, yet each agent is only informed about updates in its local pose graph and does not receive information about the sub-maps of other agents on the server. Furthermore, here as in Morrison et al. (2016), a full SLAM system runs on each agent.
The recent work in Karrer et al. (2018) presents and extension of Schmuck and Chli (2017) to a visual-inertial sensor setup, which provides metric scale and gravity alignment for the SLAM estimate, and promises higher accuracy and robustness compared to monocular approaches due to the complementary nature of the sensors.

So far most collaborative SLAM experiments were performed using pre-recorded datasets. Forster et al. (2013) deployed their system on two UAVs flying in a small indoor environment, but as discussed in Section 2.2, no communication from the server to the agent takes place in this system. The multi-robot planner in Loianno et al. (2016) was as well demonstrated during real flight of three small aircraft. However, the system works in a sequential manner: When receiving information from the server to calculate trajectories, the UAV agents hover on the spot. Afterward, when trajectory calculation is finished for all agents, the UAVs start to move simultaneously. While this system is able to plan collision-free trajectories or multiple UAVs and collaboratively build a sparse map of the environment, no collaborative pose estimation takes place in this system. Furthermore, if no overlap in the field of view of the UAVs can be detected in the beginning to establish relative poses, the initial poses of the UAVs need to be known to the server.

2.4 Decentralized Architectures

A centralized architecture of a group of agents sharing information through a server is usually employed in the literature when it comes to systems applied practically in Robotics. However, some works tackle collaborative SLAM in a decentralized manner, where all agents exchange information directly amongst each other and perform all information fusion onboard, without having a central instance that coordinates the system. Cunningham et al. (2013) propose such a fully distributed SLAM system and evaluate it in

simulation. Each robotic agent performs full SLAM locally. The generated local map is summarized by marginalization and shared with the other agents. With this shared information, the agents can augment their local map with neighborhood information. Webster et al. (2013) introduced a decentralized algorithm in faulty environments using an information filter. Taking up the decentralized approach using a laser-inertial sensor suite, Dong et al. (2015) presented a decentralized approach for collaborative mapping using UAVs. Combining distributed architectures and object-based SLAM, the work in Choudhary et al. (2017) shows a decentralized SLAM system where co-localization of the participating robots is based on commonly observed pre-trained objects. The recent work in Cieslewski and Scaramuzza (2017) describes an efficient approach for decentralized place recognition, evaluated with real data, but in simulation. Furthermore, recent articles address the topic of decentralized collaborative SLAM, focusing on different parts of the problem, such as map overlap identification (Egodagamage and Tuceryan, 2017), efficient distributed loop closure (Giamou et al., 2018) and data exchange (Cieslewski et al., 2018), robustness (Zhang et al., 2018) or the architecture of the complete system (Chen et al., 2017). While all these works shine light onto different aspects of decentralized SLAM, only pre-recorded datasets are used for the experimental evaluation.

Distributed systems can be used in a wide range of applications and scale better to large numbers of agents (swarms), since not all data has to pass through a central point. Compared to centralized architectures, it becomes much more difficult to assure data consistency and avoid double-counting of information in a distributed system. Furthermore, decentralized systems fall short of the advantage to transfer computationally expensive algorithms that are not necessarily constraint to real-time operation to the server's side, such as map optimization (aka *bundle adjustment*) and *place recognition*. In a centralized system, the robotic agents can benefit from this feature enabling them to only dedicate their potentially limited resources to performing the most critical tasks, such as real-time visual odometry. Furthermore, when the robots operate in a large environment, a centralized system allows the robots to pass older experiences of the environment to the server, maintaining only a local map with limit size onboard.

The system proposed in this article overcomes the need for the agents to rely on the server as in Riazuelo et al. (2014), while allowing the agents to act autonomously without running a full SLAM system on every robot as in Morrison et al. (2016) and Deutsch et al. (2016). As opposed to Mohanarajah et al. (2015), Forster et al. (2013) and Deutsch et al. (2016), where the agent is either not informed about the global map on the server or only receives updates if the relevant part of the optimized map already exists on the agent, we consistently share data amongst the agents and augment the agents' local map with information from other agents. Unlike Riazuelo et al. (2014) and Deutsch et al. (2016), we use the same features throughout the whole system and do not need to extract features twice on the agent and the server. While different aspects of collaborative SLAM, such as robustness or scalability, are also investigated by other works in the literature, we combine all these aspects in one collaborative SLAM system and provide an in-depth analysis of their performance. This framework is no tied to UAVs, but uses them as a particularly hard case for collaborative SLAM as they have low onboard computational power and can move very fast, making it challenging for SLAM and potentially also for communication due to a higher probability of message loss. It does not assume a perfect wireless connection and shows an efficient communication design for collaborative SLAM dealing with network delays and connection loss, and a map management strategy that identifies and removes redundant parts of the map, thus ensuring applicability and scalability of the proposed system.

3 CCM-SLAM: System Overview

The architecture the CCM-SLAM framework, depicted in Figure 3, aims at efficiently off-loading computational intensive, but not real-time constraint parts of the system to a centralized ground station (the "server"), while keeping all modules necessary for basic autonomy on each agent. For this purpose, each agent runs real-time VO onboard to estimate its pose and a 3D map of its environment. Since the computational resources onboard the agent are assumed to be more limited than on the server, the *local map* maintained by this onboard VO is limited to the N closest keyframes in the vicinity of the agents. Instead,

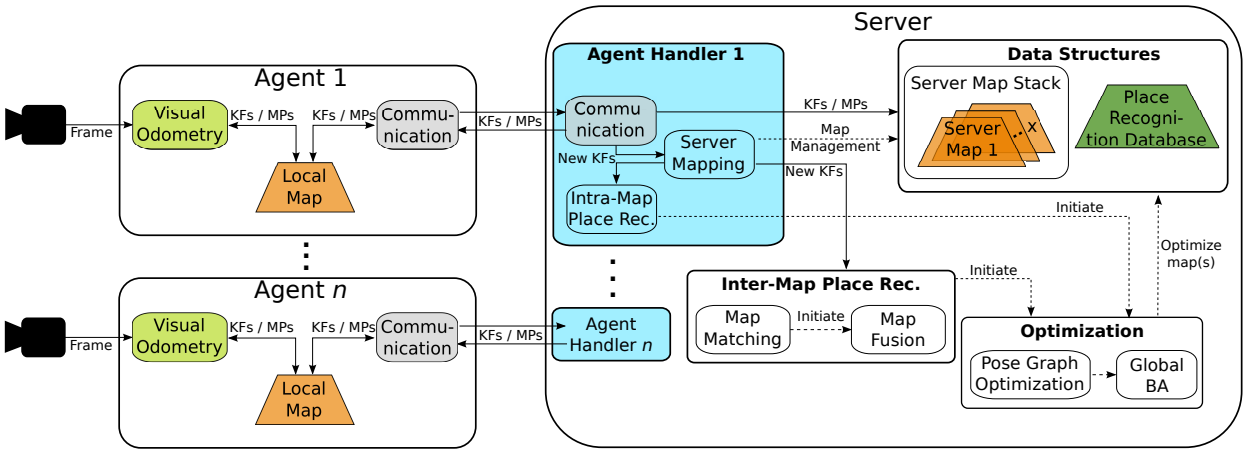


Figure 3: Overview of the CCM-SLAM system architecture. The robotic agent (e.g. a UAV) runs real-time *visual odometry* (VO) maintaining a *local map* of limited size N, and a *communication module* to exchange data (*keyframes* (KF) and *map points* (MP) and a reference KF KF_{ref}, representing its current position) with the server. The server is a ground station that executes non time-critical and computationally expensive processes: *map management*, *place recognition*, *map fusion*, and *global bundle adjustment* (BA). If all agents are recognized to have visited the same place(s), the *server map stack* contains the single, global map incorporating all agents' experiences.

the agent can use the server to offload information, which acts for the agents as a book-keeper that stores and maintains all experiences from all agents in the *server map stack*. When revisiting an already mapped location, the server provides the agents with these past experiences to augment their *local maps*. The choice of N depends primarily on the computational capacity of the agent since as shown in 5.4, a larger *local map* increases the timings of the onboard modules.

Furthermore, the server runs *place recognition*, global optimization (*bundle adjustment*) and *redundancy detection* to remove very similar or duplicate information arising from multiple visits at the same place by one or more agents. All maps use a local odometry frame, while information between the maps is exchanged in relative coordinates from one local odometry frame to the other, therefore the system never makes use of a fixed global reference frame. It should be noted that CCM-SLAM does not assume any prior information or configuration regarding the initial locations of the agents. All agents operate independently from each other until the *place recognition* module detects overlap between two maps and therefore allows to relate the measurements involved.

The bidirectional communication between the server and the agents is established via a wireless network. The communication protocol is able to handle disturbances of the network such as delays and message loss. Both the server and the agents run a communication module that establishes the exchange of map information and monitors potential errors in this information exchange. In this centralized architecture, all communication across the agents goes via the server. In case of a complete loss of connection to the server, an agent cannot receive information about existing map information from the server, yet still runs VO with a limited map window onboard. Therefore, its autonomy is preserved at any time during its mission, independently of the connection to the server.

4 System Modules

The functionalities of the individual CCM-SLAM modules shown in Figure 3 are described in the following.

4.1 KeyFrame-based Visual Odometry

Incoming camera frames from the agent's camera sensor are processed by a keyframe-based *visual odometry* (VO). VO estimates the frame-to-frame motion of the camera by tracking scene landmarks inside the agent's

local map. These scene landmarks are stored as 3D *map point* (MPs) in the *local map*. The VO does not forward all incoming frames to the *local map*, but only a subset of the most representative frames, the *keyframes* (KFs). KFs and MPs in the *local map* are connected by edges as described in Section 4.2, forming a graph-based map. Within VO, tracking and mapping run in parallel in separate threads. The tracking thread estimates the frame-to-frame movement of the camera and decides whether a frame is transformed into a KF. The mapping thread processes any new KFs by establishing connections to other KFs in the *local map* and triangulating additional MPs using connected KFs. Each KF is assigned a globally unique ID (consisting of an ascending number and the ID of the agent that created it), to make each KF identifiable globally.
Conceptually, every VO system can be used as front-end for processing incoming frames on the agent as long as it is keyframe-based, according to Schmuck and Chli (2017). The VO of CCM-SLAM is implemented using the VO front-end of ORB-SLAM (Mur-Artal et al., 2015), since this is one of the best performing monocular SLAM systems currently available. Building on top of this VO system, both the *local map* structure and the *communication module* are integrated into this VO system. This allows CCM-SLAM to limit the *local map* to a fixed number of N KFs, ensuring real-time behavior onboard the computationally constraint agents.

4.2 Local Map

The *local map* L is a SLAM graph connecting consecutive KFs to their MPs as experienced by this agent. This map is essentially a representation of the agent's immediate vicinity. In addition, L contains covisibility information between the KFs in L, forming a covisibility graph. Two KFs are connected by an undirected covisibility edge if many MPs are observed by both KFs (here, we set the threshold to 15 MPs). The weight w of an edge connecting KF_a and KF_b is determined by the number of shared observations of these KFs. Furthermore, for pose graph optimization, a pose graph is deduced from L on demand.
Trimming the *local map* implements the limitation of L to N KFs. We define the reference KF that was last created by the VO as KF_{ref}, representing the current location of the agent. If the number of KFs in L exceeds N, *trimming* keeps KF_{ref} and the $N-1$ KFs most recently added to L, while removing older KFs. Before a KF is removed from the *local map*, the trimming algorithm verifies whether the server confirmed that this KF was received. This might not be the case due to communication interruptions. To prevent data loss, the *local map* can keep a buffer of up to B more KFs (here we set $B = N$), that is emptied as soon as an acknowledgment for the KFs in this buffer is received from the server. If this buffer is also filled up, so that there are in total $N + B$ KFs stored on the agent, the map trimming algorithm enforces this upper bound by removing data. Any KFs present in L that have been originally created by other agents are prioritized for removal, since information between agents is exchanged only via the server, consequently these KFs are guarantied to be present in the *server map stack*. This might result to removal of KFs with a strong covisibility connection to KF_{ref} and thus lead to a slightly less informative map of the environment, however this is less critical than the alternative of permanent information loss. If all of these KFs get removed and the number of KFs in the agent's buffer still exceeds B, *trimming* then prioritizes the removal of the oldest KFs from the buffer, even if these have not been acknowledged as received by the server.
The choice of N is primarily dependent on the computational power available on the agent, since as shown in Section 5.4 the timings of *tracking*, *mapping* and *communication* increase with higher N. This allows to adjust the system to the computational power available on the robotic platform used as agent. Besides having more information about the current environment onboard the agent, the advantage of a higher N is that L can hold more data, therefore in cases of communication interruptions, it later reaches the maximum map size and needs to erase data. Consequently, the system gets more robust and the risk of data loss drops with increasing N.

4.3 Server Map Stack

The *server map stack* contains all *server maps* currently associated to *agent handlers*. The *server maps* have the same graph structure as the *local maps* on the agents, yet without any restriction on the number of KFs in the map and as a result contain all past experiences of the respective agent. The information contained in

the *server map stack* as a whole corresponds to all information that was ever acquired by all agents during one mission. When initializing the system, one *server map* is created for each agent in the system, managed by its corresponding Handler. When two *server maps* are merged during the mission, both maps are removed from the Map Stack and replaced by one single new *server map* containing the information from both maps. This new map gets associated to all *agent handlers* that participated in the *map fusion*.

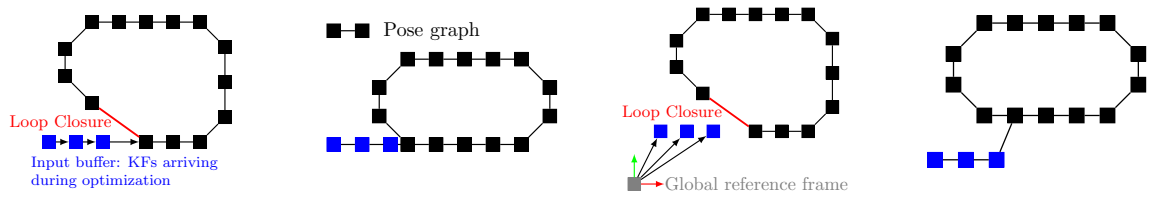
4.4 Agent Handler

The *agent handler* manages the corresponding agent's information arriving to the server. For each agent, one *agent handler* is instantiated, that initializes a *communication module* to communicate with that agent, initializes a new *server map* for this agent in the *server map stack* and creates a *map manager* and *intra-map place recognition* module for this *server map*. Each of these modules runs on a separate thread in parallel. Furthermore, it provides an interface to modify the dependencies of the modules of one agent, since when two *server maps* are merged, for all modules of all agents involved in the fusion the associated *server maps* need to be replaced by the new (merged) one. Additionally, each *agent handler* H_i, associated to Agent A_i, holds a Sim(3)-transformation ${}^{\mathrm{L_i}}\mathbf{S}_i^{\mathrm{M_j}} = \{{}^{\mathrm{L_i}}\mathbf{R}_i^{\mathrm{M_j}}, {}^{\mathrm{L_i}}\mathbf{t}_i^{\mathrm{M_j}}, {}^{\mathrm{L_i}}\mathrm{s}_i^{\mathrm{M_j}}\}$ that transforms data from the associated agent's *local map* L_i into the associated *server map* M_j from the *server map stack* on the server's side. Initially, the coordinate systems of the two maps on the agent's and the server's side coincide: rotation ${}^{\mathrm{L_i}}\mathbf{R}_i^{\mathrm{M_j}} = \mathbf{I}_{3\times3}$, translation ${}^{\mathrm{L_i}}\mathbf{t}_i^{\mathrm{M_j}} = \mathbf{0}_{3\times1}$, scale ${}^{\mathrm{L_i}}\mathrm{s}_i^{\mathrm{M_j}} = 1$. Following a fusion of M_j with M_k from the *server map stack* into a new *server map* M_m, the relative transformation and scale between L_i and the *server map* associated to Agent A_i needs to be updated to ${}^{\mathrm{L_i}}\mathbf{S}_i^{\mathrm{M_m}}$ as described in Section 4.8.

4.5 Communication Modules

The *communication modules* on the agent's and the server's sides establish the communication link between them. The ROS communication infrastructure (Quigley et al., 2009) is used for message passing over a wireless network. This framework does not guarantee real-time message passing, however this is not a requirement for the functionality of CCM-SLAM, since neither the server nor the agent expect information from the other side at a fixed rate. On the agent's side, the *communication module* keeps track of all changes in the map, i.e. any added and changed KFs and MPs, and converts this information into a message that is sent to the server. Since the map keeps changing constantly, the message publishing rate is limited to a maximum value. Every new message contains any changes since the last message. Furthermore, the maximum size of the message is also limited, to prevent packing of too large portions of the map into one message in cases when the connection to the server gets interrupted for several seconds. To prevent race conditions, the *local map* is not accessible for VO during message packing, therefore a big message size could block tracking for long periods of time. Instead, the information that needs to be sent to the server in such cases is split into several messages. In this implementation, the average time that L is blocked for VO is kept insignificant (about 0.4 ms). Furthermore, with every message, the agent informs the server about KF_{ref} from the *local map* that is closest to its current position.

For the *communication module* on the server's side, every message that the server sends to an agent contains the k KFs with the strongest covisibility edges to KF_{ref} and their observed MPs, to augment the *local map* with this data. This information is considered to be most valuable to support pose estimation onboard the agent, since VO uses the covisible neighbors of KF_{ref} to calculate the pose of an incoming camera frame. These k closest KFs are chosen based on their covisibility weight regardless of which agent they originated from. On the reception of KFs and MPs at the server's side, the *communication module* on the server transforms this incoming data from the coordinate frame of the agent's *local map* L_i to the coordinate frame of the associated *server map* M_j using the Sim(3)-transformation ${}^{\mathrm{L_i}}\mathbf{S}_i^{\mathrm{M_j}}$ stored by the *agent handler*, and vice versa outgoing messages. On the server's side, the message publishing is also limited to a maximum rate to limit the network bandwidth requirements and the computational effort of the agent required for processing received information. By adjusting the rate and the parameter k, the communication traffic can be adapted to the maximum bandwidth available in the network.



(a) Relative coordinates. (b) Correct alignment after optimization. (c) Absolute coordinates. (d) Incorrect alignment after optimization.

Figure 4: Sending coordinates relative to a previous KF (a,b) vs. sending absolute coordinates (c,d). KFs arriving during optimization (blue) will be aligned incorrectly after optimization if using absolute coordinates.

Any exchanges of KFs and MP position information between the server and the agents uses a relative coordinate scheme instead of absolute coordinates, illustrated in Figure 4. Exchanging absolute poses can lead to problems in case of scale drift on the agent or loop closure on the server: the KFs arriving during an optimization step will inevitably not be aligned with the resulting optimized map. During Global BA, the *server map* is locked, and incoming KFs are queued in an input buffer. After closing the loop, both ends of the loop are aligned, causing significant pose changes for parts of the map, as illustrated in Figure 4d. Since data in the input buffer is not corrected by the optimization step, the absolute coordinates of this data do not align with the optimized map anymore. Transforming incoming data with the calculated transformation from loop closure does not solve the problem, because this transformation is only applied locally to align the ends of the loop, and subsequently Global BA applies a different pose change to every KF in order to minimize the global error. Using relative coordinates however avoids this problem, as illustrated in Figure 4b: calculating the pose of newly arriving data using the pose of a KF already existing in the map implicitly propagates the optimization results to the new data. Therefore, each KF_i that is sent to the server encodes two relative poses: one pose p_{pred} relative to its predecessor KF_{i-1}, and a second pose p_{par} relative to the KF_{par} that has the strongest covisibility connection[1] to KF_i. KF_{par} is named the *parent* of KF_i in the covisibility graph. Usually, p_{pred} is used to register KF_i in the *server map*, while p_{par} is only used if KF_{i-1} is not available in the *server map*. This allows CCM-SLAM to compensate to some extent for the loss of data – it does not necessarily need KF_{i-1} to register KF_i in the *server map*, so KF_i can be processed even if the message containing KF_{i-1} gets lost. For the transfer of data from the server to the agents, the server encodes all KF poses and MP positions relative to KF_{ref}. When the agents' *communication modules* receive KFs and MPs that are already known and registered in the *local map*, the KFs and the MPs are updated according to the message since these are already connected in the *local map*. If the incoming KFs and MPs are not present in the *local map*, these connections are established. KFs and MPs sent from the agents to the server enter the *server map* accordingly. Any communication between the agents to the server can be distinguished as two types of messages: "new data" or "updates". For new data, i.e. newly created KFs and MPs, an agent sends the whole data structure including 2D feature keypoints extracted from the image, their feature descriptors, and the associated 3D MPs (for KFs) or connected KFs (for MPs). For MPs, this results in a message of around 200 Bytes, including the IDs of the associated KFs and a 36-Byte feature descriptor representing the image keypoint of this MP. For each new KF, the message has to encode all 20-Byte 2D image keypoints and their associated 36-Byte feature descriptors, which results in a message of around 55 kB together with other information contained in the message (computed assuming that the maximum of 1000 features in this KF). However, the 2D keypoints and their descriptors for each KF are computed only once and do not change, therefore we send this information only once to avoid unnecessary network traffic. Hence, for all following changes, we only need to send the update message to the server, containing the new poses of the KFs (and the new positions of each MP). These messages only have a size of 148 Bytes for KFs and 52 Bytes for MPs. For message passing from the agent to the server, we employ an optimistic approach: if a message is sent to the server, we assume that it arrives successfully unless evidence is given that this is not the case. The KF in a message sent to the server are always sorted beginning with the smallest ID, and also processed by the server in this order. In any message sent to the

[1]Highest edge weight in the covisibility graph described as in 4.2

agent, the server includes information on which KFs it has already processed. Therefore, when the agent receives information that KF_i was processed, without any acknowledgment that KF_{i-1} was received, it becomes evident that this data got lost. In this case, all data for KF_{i-1}, including 2D image keypoints and descriptors, is sent again to the server. The same approach applies for the MPs. This distinction between "new data" and "updates" is crucial for the ability of CCM-SLAM to run smoothly onboard all robots using a wireless connection. As an indication, the mean traffic of 0.37 MB/s achieved in Figure 6a soared to around 10 MB/s before introducing this scheme. Communicating from the server to the agents, we do not need to verify whether the messages arrive, since the information relative to the current location of the agent is continuously sent, and therefore re-sending an old message does not make sense since the agent's location probably changes in the meantime until the loss of the message is detected.

In case of a loss of connection to the server, the agent will not receive any more data to augment its *local map* or acknowledgment that data arrived at the server. Therefore, it will fill the *local map* up to the maximum permitted size and then begin erasing data from it to respect the upper limit. Hence, the system onboard the agent falls back to a VO with a limited local map window in cases of connection loss, which still provides all necessary information for an agent to move autonomously through the environment. As shown in Section 5.3, disturbed communication increases the network traffic and could also in the worst case lead to situations where the *local map* (on the agent) and the corresponding *server map* cannot be connected any more. The system will again fall back to pure VO mode. A delay in the communication does not increase the network traffic, but it affects the degree of collaboration between the server and the agents. If we assume a network of delay δ ms, the current location of the agent KF_{ref} known to the server will always be δ ms behind the real location of the agent. The server then packs the KFs in the vicinity of KF_{ref} into a message and sends it to the agent, which again arrives with a delay of δ ms. Therefore, with increasing delay, the information from the server will be further away from the current agent position when it arrives and therefore, less useful to the agent.

4.6 Server Mapping

The *server mapping* module has three main responsibilities. First, it forwards newly arrived KFs to the KF Database, *intra-map place recognition* and the *map matching* module. The second task is to establish connections between new KFs and MPs and the exiting pose graph of the corresponding *server map*. The third task of the *server mapping* module is redundancy detection for the KFs in the *server map* (**KF Rejection**). When one distinct location is visited multiple times by one or more agents during a mission, the map contains several KFs from this location. While some of these KFs, such as those from different viewpoints, add more information the system, some others might encode almost the same information and therefore it is not necessary to keep all of these similar measurements. Since map size grows quickly even in medium-sized environments, such as the one shown in Section 5.7, this redundancy detection is important to keep a manageable map. This enables a collaborative system to work in large-scale environments, because the size of the map affects the timings of most processes on the server such as *place recognition*, map queries or BA. This *KF rejection* scheme randomly picks a KF_i from the *server map*, iterates through all neighboring KF_j of KF_i in the covisibility graph, and, similar to Mur-Artal et al. (2015), checks their observed MPs. If for a pre-defined threshold θ % of all MPs of KF_j are observed by at least 3 other KFs, KF_j is considered redundant and removed from the *server map*. Since the connections between the KFs in the map might change during the mission, CCM-SLAM uses this probabilistic scheme instead of sequentially checking only new KFs for redundancy with other KFs in the map The *KF rejection* scheme is performed when the system's capacity is not fully used by BA, and in return the BA time is reduced and the *server map* becomes accessible sooner again. Furthermore, the rest of the server's modules (*mapping*, *place recognition*) benefit from smaller *server maps*, therefore this *KF rejection* procedure is indispensable for large-scale missions with many agents in large areas.

4.7 Place Recognition and Keyframe Database

The *place recognition* system detects overlap between locations in the *server map stack* using a *KF database*. An important feature of this *place recognition* system is that it is able to match measurement from multiple monocular camera sensors with different camera parameters. This makes CCM-SLAM a versatile system since it can be used with heterogeneous robotic agents equipped with different cameras.

The KF Database is incrementally built at runtime, using all acquired KFs from all agents. Implementing an inverted file index using DBOW2 (Gálvez-López and Tardos, 2012) it permits for efficient look-up queries for a new KF with respect to past KFs with the same features. Extracting the agent's ID from the unique ID of each KF, the database can be queried for only a subset of KFs belonging to specific maps.

For every new KF that arrives at the server, two types of place recognition queries are performed using the *KF database*: **Intra-Map Place Recognition** and **Map Matching**. *Intra-map place recognition* detects previously visited locations inside one *server map*. Detecting such a trajectory overlap allows to add new constraints to the pose graph, that can be used in the optimization step to improve the overall map accuracy. A successful query of *intra-map place recognition* is followed by BA (Section 4.9). *Map matching* detects overlap between two *server maps*, which allows to calculate a Sim(3)-transformation between these maps and to add constraints between them. If *map matching* successfully detects overlap between two maps, this match is forwarded to the *map fusion* module.

4.8 Map Fusion

The *map matching* module sends a pair of matching KFs $(\mathrm{KF}_q, \mathrm{KF}_m)$ belonging to maps M_q and M_m, respectively, for *map fusion* together with a vector of matching MPs from both maps and the Sim(3)-transformation ${}^{\mathrm{M_m}}\mathbf{S}^{\mathrm{M_q}}$. *Map fusion* initializes then a new, third *server map* M_f to fuse information from all agents that contributed to M_q and M_m. The coordinate frame of M_f is adopted from M_m, therefore all KFs (and MPs) in M_m are entered directly into M_f, while data from M_q is transformed to the coordinate frame of M_f. During the transfer from M_q and M_m to M_f, the associated *server map* of each KF and MP is changed to M_f, since all map data holds a pointer to the *server map* it belongs to. Any matching MP pairs between M_q and M_m identified during *map matching* are merged into one MP in M_f, yielding new constraints between the KFs of M_q and M_m. Global BA is then performed on M_f, and finally, all pointers to either M_q or M_m are changed to M_f using the interface provided by the *agent handlers* before eliminating M_q and M_m completely from the *server map stack*. Furthermore, the Sim(3)-transformation ${}^{\mathrm{L_i}}\mathbf{S}_i^{\mathrm{M_q}}$ of all *agent handlers* A_i associated to M_q before the map fusion is corrected to ${}^{\mathrm{L_i}}\mathbf{S}_i^{\mathrm{M_f}} = {}^{\mathrm{L_i}}\mathbf{S}_i^{\mathrm{M_q}} \cdot ({}^{\mathrm{M_m}}\mathbf{S}^{\mathrm{M_q}})^{-1}$

4.9 Global Bundle Adjustment

Whenever *place recognition* successfully detects a match between two *server maps*, global optimization follows on the new *server map*. Global optimization is also triggered when a loop is detected within a particular *server map*. Global Bundle Adjustment (BA) optimizes that respective *server map* by minimizing the reprojection error for all KFs and MPs, in order to improve its accuracy of the graph and reduce scale drift. Since CCM-SLAM is designed to allow the agents operate in large environments, this 7 Degrees-of-Freedom (DoF) optimization of the pose graph is only performed on the server, because as shown in Section 5.6, an optimization step takes several seconds for large maps. Since other modules cannot access the map during an optimization step, the communication module stores any incoming data in buffers and processes them when the optimization is finished. For our implementation we use the Levenberg-Marquardt algorithm of g2o (Kümmerle et al., 2011) for global BA. Before starting global BA, pose-graph optimization is performed on the map using a subgraph of the covisibility graph incorporating only the strong edges (edge weight $w \geq 100$) termed essential graph (Mur-Artal et al., 2015), since this significantly improves the timings and accuracy of the optimization as shown in Mur-Artal et al. (2015)

5 Experimental Results

5.1 Setup

Table 1: Hardware setup for the evaluation of CCM-SLAM

Platform	**Type**	**Characteristics**	**Sensor**
Server	ThinkPad T460s	2.60 Ghz × 4, 20 GB RAM	—
Agent 1	AscTec Neo UAV	Intel NUC 5i7RYH 3.1 GHz × 4, 8 GB RAM	Realsense R200 RGB-D camera (only the RGB image is used here)
Agent 2	AscTec Neo UAV	Intel NUC 5i7RYH 3.1 GHz × 4, 8 GB RAM	Bluefox grayscale camera (2.8 mm focal length)
Agent 3 (datasets)	Intel NUC 7i7BNH	3.5 GHz × 4, 32 GB RAM	—
Agent 3 (flight)	AscTec Hummingbird UAV	AscTec Atomboard V3 (1.91 Ghz × 4, 4 GB RAM)	Bluefox grayscale camera (2.4 mm focal length)
Router	TL-WR802N USB Router		

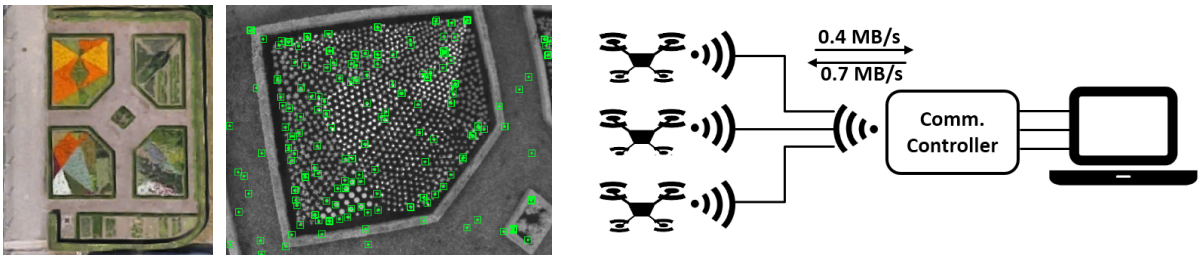
For all experiments presented in this section, the infrastructure listed in Table 1 was used. For the real-world flight with 3 UAVs, the third agent used for the evaluation on datasets is replaced by a smaller UAV with less computational capabilities to demonstrate the adaptability of CCM-SLAM to different platforms. Namely, we use the AscTec Hummingbird listed in Table 1. For the analysis of CCM-SLAM, we use both pre-recorded dataset of the same area run on each agent simultaneously as well as demonstrate the whole framework in real experiments where monocular data captured simultaneously from all UAVs is processed online. For the analysis on pre-recorded dataset, the three agents and the server are connected via a wireless network, so that real communication between the server and the agents takes place. Then the recorded datasets described in Section 5.2 are processed onboard the UAVs. Using the same input data renders evaluations across different runs more comparable, with this setup being nearly equivalent to a real-world flight. Furthermore, this setup allows us to influence the quality of the network connection, and the datasets used provide ground truth for our evaluation. To evaluate the behavior of the communication in case of a disturbed connection, the network is artificially disturbed with the setup described in Section 5.3. Finally, in Section 5.8 and Section 5.9, we show two real-world experiments to prove the ability of CCM-SLAM to work in a real-world scenarios.

5.2 Datasets

For the evaluation of CCM-SLAM we use the publicly available EuRoC dataset (Burri et al., 2016) and our own Irchel dataset, with their most important characteristics briefly summarized in Table 2. Both datasets provide accurate ground-truth position data from a Leica Total Station. The EuRoC dataset contains video sequences captured using an AscTec FireFly UAV with a forward looking camera flying through an industrial environment repeatedly. The sequences are each processed on a separate agent simultaneously, while communicating with the server online.

Table 2: Datasets used for the evaluation of CCM-SLAM.

Dataset	Path	Time	Camera view	Environment
MH01 (`Machine Hall 01`)	80 m	3 min	Front	Industrial, indoor
MH02 (`Machine Hall 02`)	70 m	2:30 min	Front	Industrial, indoor
MH03 (`Machine Hall 03`)	130 m	2:10 min	Front	Industrial, indoor
Irchel-1	140 m	2 min	Downward	Garden, outdoor
Irchel-2	130 m	2 min	Downward	Garden, outdoor
Irchel-3	160 m	2 min	Downward	Garden, outdoor



(a) Top view from Google Earth of the test area.

(b) View from UAV with scene landmarks for tracking superimposed.

(c) Network setup to evaluate network connection disturbances, introducing delays and message losses via the *communication controller*. The data rates shown are for a perfect, undisturbed connection.

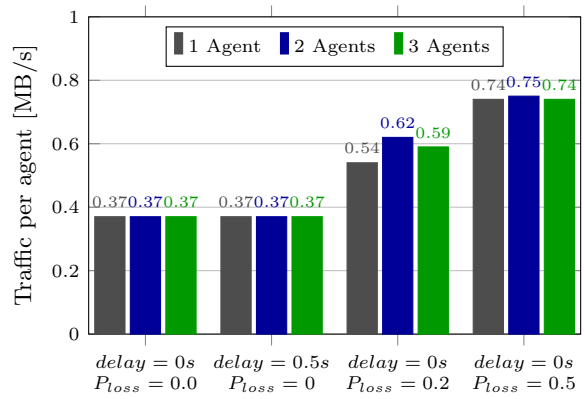
Figure 5: Experimental environment from the Irchel dataset (a,b) and setup to evaluate the robustness of the communication module (c). The Irchel dataset contains several sequences captured during flights of an AscTec Neo UAV with a downward looking camera over an outdoor garden area.

Our Irchel dataset was used for the evaluation in Schmuck and Chli (2017) and contains several sequences captured during flights of an AscTec Neo UAV with a downward looking camera over an outdoor garden area illustrated in Figure 5. Here, we use the sequences from this dataset listed in Table 2 to reproduce three agents in simultaneous flights over the same area.

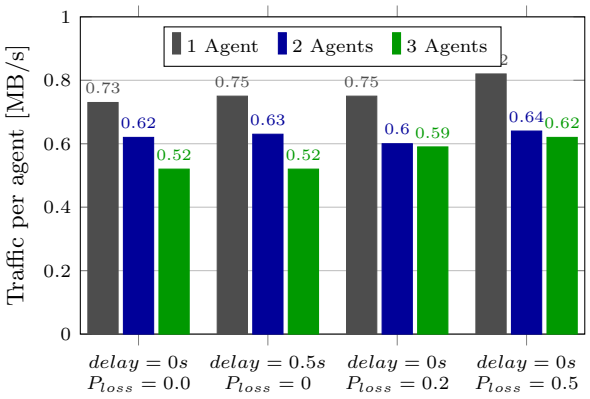
5.3 Bandwidth Requirements

As explained in Section 4, CCM-SLAM is able to deal with network problems commonly occurring in real situations, such as delays and message loss, arising due to the large distance between the agents and the server or due to too many agents in the network, for example. To evaluate this behavior, we use the setup presented in Figure 5c to model disturbances in the communication. We place an additional module, called the *communication controller*, in the interface between the agents and the server, which are connected via a wireless connection. All messages pass through this *communication controller*, which disturbs communication by dropping incoming messages with a probability P_{loss}, and forwards them to the recipient with a probability $1 - P_{loss}$. Furthermore, it holds back the message for a specified delay, before it forwards the message. With this setup, we can model network connections of bad quality and assess the influence of this network effects on the system. For these experiments, the agents and the server publish messages at a maximum rate of 2 Hz, and the number of KFs the server sends to the agent is limited to $k = 5$ for every message. Since messages sent from the server to the agents are usually larger, as shown in Figure 6, the server publishes at a lower rate than the agents.

Figure 6a shows the average network traffic from an agent to the server for each agent in a one-, two- and three-agent setup. With a perfect network connection (i.e no delay, no message loss), the average network traffic is around 0.4 MB/s. As shown in Figure 6a, network delays of up to 0.5 s do not change this value. This is expected (as explained in Section 4) since small delays do not lead to a need of sending more messages. A disturbed communication with a probability of message loss $P_{loss} = 0.2$ and $P_{loss} = 0.5$ indeed increases the network traffic since data needs to be sent repeatedly when it gets lost. The experiments show that $P_{loss} = 0.2$ can be handled well by the system, since the network traffic increases only slightly, and even if a message gets lost, the recipient usually quickly receives the lost information again. Yet a message loss with $P_{loss} = 0.5$ is critical for the system. In addition to the double network traffic in this situation, with small *local maps*, this high probability of message loss can lead to situations where an agent reaches the limit of its permitted local map size and has to erase information before it can be received by the server, resulting to permanent loss of this information. Figure 6a also shows that the number of agents does not significantly influence the traffic from each single agent to the server, since the network traffic caused by a single agent is almost equal with one, two and three participating agents, therefore meaning that the network traffic induced by sending data from the agents to the server scales linearly with the number of participating agents. This comes from the fact that the local map



(a) Network traffic from each single **agent to the server**



(b) Network traffic from **server to the each agent**

Figure 6: The effect of the connection disturbance to the network traffic from each agent to the server and vice versa for a setup with 1, 2 and 3 participating agents. All values are averaged over 5 runs for each experiment using the EuRoC dataset.

size is kept constant in the number of included KFs, regardless of whether one or more agent contributed KFs.

The impact of the network disturbance to the traffic from the server to each agent is illustrated in Figure 6b. As expected, since the server always sends the closest[2] k KFs to the last known position of the agent, this traffic stays almost constant, independently of connection quality. The slight decrease of the network traffic with the number of agents arises from the fact that in a multi-agent scenario, more loops in the *server maps* can be closed, resulting in more periods where the *server maps* are locked and no information is transmitted to the agents because Global BA is performing map optimization.

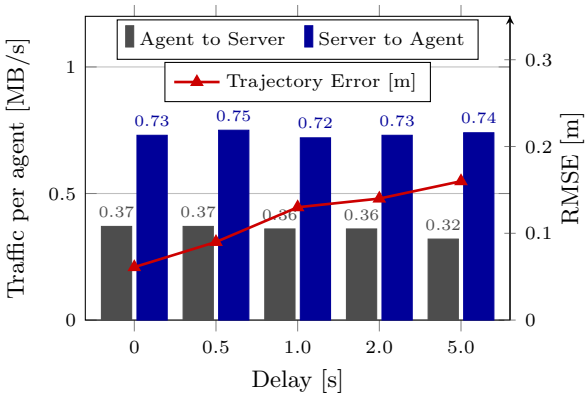
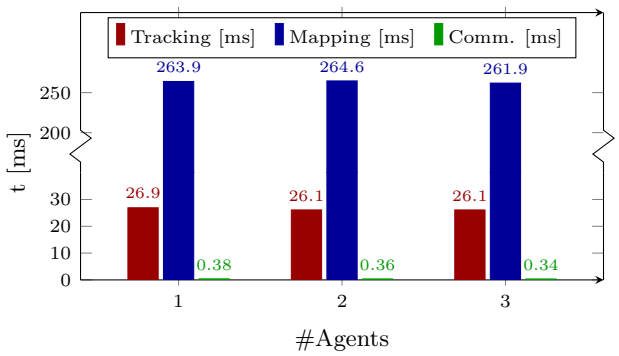


Figure 7: The influence of network delays enforced by the *communication controller* on the network traffic per agent and the trajectory error of the *server map*, using MH01.

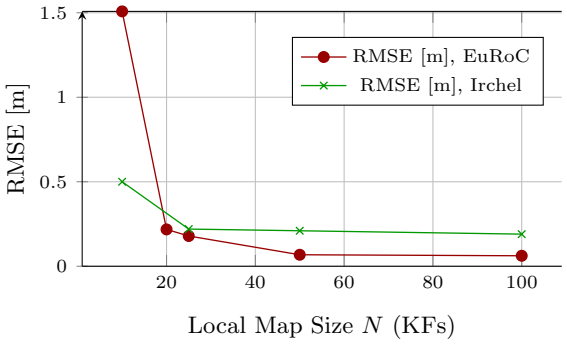
Figure 7 illustrates the influence of network delays on the system, using the MH01 sequence. Since Figure 6a showed that there is no significant impact caused by the number of agents, this experiment is executed in a single-agent setup. While delays have no significant influence on the network traffic, the trajectory error of the *server map* grows with larger delays. As explained in Section 4.5, larger delays result in a larger difference between the actual position of the agent and the position of the agent as it is known to the server. Therefore, the server sends map data close to a past location to the agent, that gets less useful for the agent the larger the delay is.

These experiments attest to the resilience of the proposed system to realistic disturbances in the communication between the server and the participating agents. The number of agents that can be used without compromising the collaboration through the CCM-SLAM framework depends mainly on the bandwidth of the network connection and the computation power available on the server. However, as aforementioned, even when the number of agents is so large that it completely overloads the network traffic, the agents are still ensured to run VO onboard undisturbed, by design.

[2]Closest in the covisibility graph described as in 4.2, i.e. the KFs that share most features with a specific KF



(a) Timings of the agent's side modules with a growing number of agents in the system, measured on Agent 1. All values are averaged over 5 runs using the EuRoC dataset.



(b) The Root Mean Squared Error (RMSE) of the trajectory of one agent with varying *local map* size. Values are averaged over 3 runs using MH01 and Irchel-1.

Figure 8: Timings of the different agent side modules (a) and influence of the local map size N on the accuracy of CCM-SLAM (b).

5.4 Real-Time Behavior & Local Map Size

Table 3 shows the timings for VO tracking, VO mapping and communication on the agent's side for different sizes N of the *local map*, using one agent. The time for tracking is the time to process one frame, while mapping and communication times sum up the time used for one iteration of the module. Mapping is executed once for every KF, while the *communication module* runs at tracking frequency (i.e. at 20 Hz, matching the camera's frame rate). The time for mapping increases with *local map* size since the map that needs to be maintained grows. Tracking and communication times also increase slightly, because firstly, obtaining data from the map needs more time with larger maps, and secondly, with increased mapping time tracking needs to wait longer to obtain map access.

Table 3: Timings in ms (mean $\pm$ standard deviation) for all modules running onboard one agent, while varying the *local map* size N in KFs. All values are averaged over 3 runs for each experiment using MH01 and Irchel-1. The last column reports the values when running CCM-SLAM's VO front-end with unbounded map size.

Module	**CCM-SLAM**				**VO only**
	$\mu \pm \sigma$ $[ms]$				
	N = 10	**N = 20**	**N = 50**	**N = 100**	**N = ∞**
Irchel-1					
Tracking	23.3 ± 5.4	26.6 ± 6.7	27.9 ± 7.2	28.2 ± 7.6	31.9 ± 7.4
Mapping	82.1 ± 16.7	187.2 ± 62.9	208.0 ± 85.2	232.0 ± 105.0	235.2 ± 102.7
Comm.	0.26 ± 1.7	0.34 ± 1.5	0.38 ± 1.6	0.49 ± 1.7	–
MH01					
Tracking	22.9 ± 4.4	25.2 ± 5.2	26.9 ± 5.9	9.8 ± 7.4	34.9 ± 11.1
Mapping	87.7 ± 19.9	206.9 ± 47.4	273.9 ± 95.2	330.2 ± 186.8	401.2 ± 346.2
Comm.	0.29 ± 1.8	0.37 ± 1.4	0.38 ± 1.7	0.5 ± 1.9	–

Figure 8a illustrates how the timings of the agent's modules change with a growing number of agents, for a *local map* size of $N = 50$ KFs. The timings stay in the same range and do not exhibit any strong effects with increasing number of agents, confirming good scalability of this framework. While the timings for tracking and mapping depend on the underlying VO system, the communication time is low enough to allow for real-time tracking with the current implementation.

Figure 8b shows the influence of the size of the *local map* on the trajectory position error. With a smaller *local map*, the VO is missing information that would be needed for the VO system to improve the pose estimate of the agent. With growing map size, more information is available and the robustness to network problems

grows, and therefore the error decreases. Since for $N = 50$ KFs most information about the immediate vicinity of the current location of the agent is included the *local map*, the transition to $N = 100$ KFs only has a slight effect.

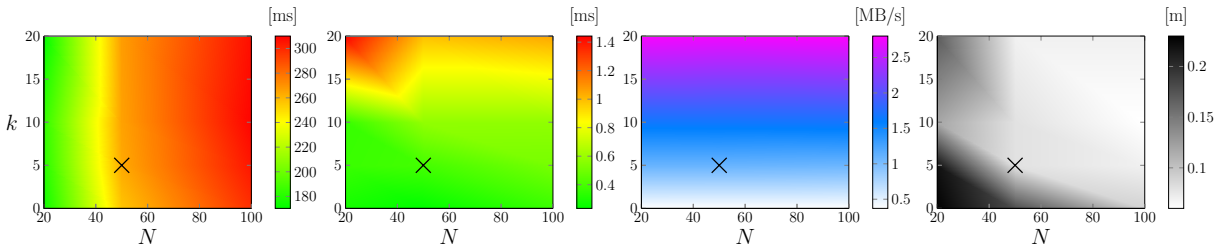
5.5 Parameter Configuration

As shown in the previous sections, the timings of the processes running onboard an agent and the network traffic can be controlled by altering the parameters k (number of KFs sent from server to agent to augment *local map*) and N (size of the *local map*). Figure 9 illustrates the influence of these parameters on timings, network traffic and trajectory error. The parameters chosen for subsequent experiments are marked by an "$\times$".

Figures 9a and b show the influence of the parameter setup on the timings of the *mapping* and *communication* module on the agent, per run of the module. Mapping time increases with higher N, since the *mapping module* has to manage a larger *local map*. k, however, does not have a significant influence on mapping time. Timings for the *communication module* show the opposite effect: the influence of N is small – a larger *local map* means querying the *local map* and sending more data to the server, yet these operations are not time-consuming. The influence of k, however, is much stronger. With more KFs arriving from the server, it is more likely that received KFs and MPs are not already included in the *local map*. Then new KFs and MPs need to be constructed, which is more time-consuming than other communication operations. The communication time reaches its peak for a small N and high k. A parameter setup like this is not reasonable for practical applications since for $k = N$ every message from the server already contains the maximum number of KFs permitted in the *local map*, possibly replacing the whole *local map* at once.
The network traffic, shown in Figure 9c, increases with higher k, as more data gets sent to the agent. Figure 9d shows the trajectory error in the *server map*. For smaller *local maps*, increasing k significantly improves the accuracy of the *server map*, since the data from the server support the agent's pose estimation to reduce error and drift. For higher N, the *local map* encodes more information about the immediate vicinity of the agent, therefore the influence of k is smaller.
Following this analysis, for all subsequent experiments, $N = 50$ KFs and $k = 5$ KFs are used. These are chosen to suit the limited bandwidth of the wireless router used and provide a good trade-off between the timings of each agent's onboard modules and the accuracy of the trajectory estimate.

5.6 Keyframe Rejection

Aiming to study the effect of *KF rejection* explained in Section 4.6 on the *server maps* maintained by CCM-SLAM, Table 4 shows the effect of the *KF rejection* on the server map size and the resulting time for BA, where the effect of the reduced map size is visible most prominently since its complexity is cubic in the number of KFs included in the optimization. To evaluate this, two agents are started at the same time and



(a) Mapping time (b) Communication time (c) Average network traffic (d) Server Map RMSE

Figure 9: The influence of the *local map* N (KFs), and KFs send from the server to an agent k on the VO mapping time in (a), the communication between the agent and the server in (b), the average network traffic in (c) and the RMSE in the *server map* in (d)

any *map fusion* is halted until 80s after start, using different culling thresholds θ. This threshold θ means that a KF_k is considered redundant and removed from the *server map* if θ % of all MPs of KF_k are observed by at least 3 other KFs. The reduced number of KFs directly translates into a faster BA. For this map size and a threshold of $\theta = 98$ %, the savings achieved by *KF rejection* already outweigh the additional time spend for redundancy detection.

θ	BA [s]	KFs	Rej. KFs	Time for Rej. [s]
100%	19.8	446	0 [0%]	0
98%	16.2	396	58 [11.2%]	2.4
96%	9.0	356	95 [20.2%]	1.8

Table 4: The effect of *KF rejection* in a *server map*, depending on the strictness of parameter θ (at $\theta = 100\%$ no *KF rejection* takes place). All values are averaged over 5 runs for each experiment, using 2 UAVs on the EuRoC dataset.

θ	KFs	Rej. KFs	Rej. Rate	RMSE [m]
100%	388	0	0%	0.057
98%	358	59	14%	0.061
96%	310	91	23%	0.486

Table 5: The effect of *KF rejection* on the RMSE of the *server map* using two UAVs on the EuRoC dataset. All values are averaged over 5 runs for each experiment.

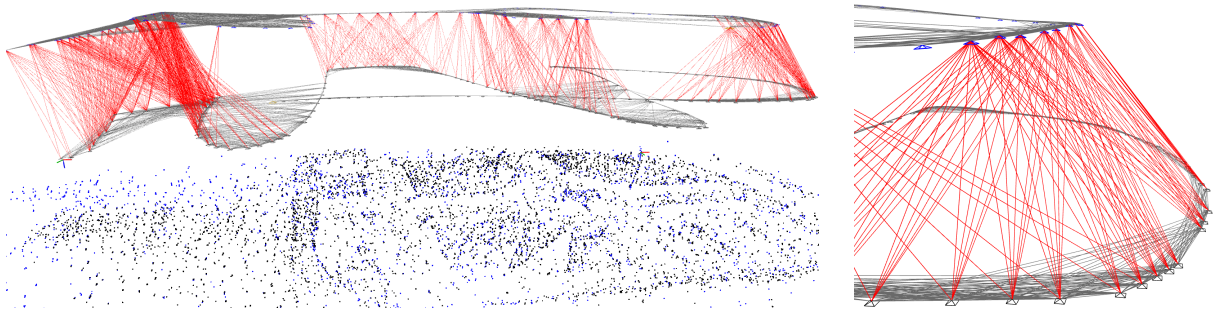
However, removing information from the system can also affect the resulting accuracy of the map estimate, as shown in Table 5. Furthermore, a high *KF rejection* rate can reduce the number of successful *place recognition* queries that can be detected. Evidently, KF redundancy detection is an important feature to keep the map manageable, especially when revisiting areas. Yet the KF rejection should not be too restrictive since too much information is removed from the system. For the rest of the experiments, a threshold of $\theta = 98$ % is used.

As shown in Table 4, Global BA takes several seconds for a map of around 400 KFs. Since the timing of Global BA grows cubic in the number of KFs, for a server with the computational power of a standard notebook, such as the Thinkpad T460s used in the experiments presented in this article, handling up to around 1000 KFs can be reasonable, which translates into three to four agents flying over a medium-sized area, such as the ones in the datasets used.

5.7 Collaborative SLAM

After analyzing the communication and scalability of CCM-SLAM, the accuracy of the estimates are compared to ground-truth position data from a Leica Total Station on both datasets. Figure 10 shows the resulting map (KFs & covisibility graph) for two agents on the Irchel dataset. We display those covisibility edges with weight $w \geq 50$, i.e. edges between two KFs that both observe at least 50 identical MPs. This illustrates that CCM-SLAM does not only align two maps from two agents, but connects the information collected by the two agents throughout the mission. This is key to collaboration, since the data associations across KFs of different agents allow an agent to include and connect information from other agents in its own covisibility graph, and also allow BA to use these additional links between the KFs to produce a collaborative map estimate that is better than when using the maps from the single agents individually. Figure 11 shows the map resulting from a three-agent scenario with 738 KFs in total, using the three EuRoC sequences.

Table 6 shows a comparison of the accuracy of estimations against the position ground truth from a Leica Total Station on the Irchel and EuRoC datasets. The trajectory RMSE for each agent is reported in a single-agent scenario, followed by the error of the collaborative map estimate from the multi-agent scenario. The results show that in absolute values, the collaborative estimates exhibit comparable trajectory error. However, in the multi-agent scenario, the scene and the trajectories estimated are overall much larger. Therefore, Table 6 additionally reports the trajectory error relative to the length of the trajectory, revealing the accuracy improvements resulting from robotic collaboration. These results show that compared to single-agent SLAM, with collaborative SLAM using CCM-SLAM the scene is mapped and the individual agents are localized with at least similar or even better accuracy, but faster, since the estimation effort can be shared amongst multiple agents. Furthermore, Table 6 shows a comparison of the collaborative estimate of CCM-SLAM to open-source VINS-mono (Qin et al., 2018), which provides a multi-session functionality allowing to estimate joint trajectories by running datasets sequentially. While the purely monocular CCM-SLAM exhibits comparable performance to the monocular-inertial VINS on the EuRoC sequences (MH1,MH2 and



(a) The final, common *server map* constructed by two UAVs, 311 KFs. (b) Close-up from a loop closure area.

Figure 10: The final, common *server map* created by tow UAVs agents on the Irchel dataset. The KFs and MPs of the agents are colored in black and blue, respectively. KFs with more than 50 covisible MPs ($w \geq 50$) are connected with an edge. Edges between KFs from the same agent are colored gray, while edges between KFs from different agents are colored red.

Table 6: Comparison of the trajectory error for single-agent and collaborative SLAM. The last column shows the results of VINS-mono (Qin et al., 2018), which allows estimating joint trajectories using its multi-session functionality and sequentially running the datasets. Results are averaged over 5 runs. The collaborative RMSE is calculated by comparing the entire joint trajectory over all agents to ground truth.

Dataset	Single Agent RMSE [m]			CCM-SLAM	VINS-mono
	Agent 1	Agent 2	Agent 3	Col. RMSE [m]	Col. RMSE [m]
Irchel (1,2,3)	0.21 (0.15%)	0.22 (0.17%)	0.21 (0.13%)	0.21 (**0.06%**)	0.36 (0.11%)
EuRoC (MH1,MH2,MH3)	0.061 (0.076%)	0.081 (0.116%)	0.048 (0.04%)	0.077 (**0.03%**)	0.074 (**0.03%**)

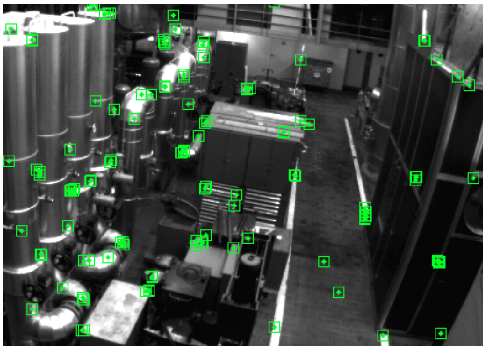
MH3), CCM-SLAM outperforms VINS on the Irchel dataset. Despite that VINS-mono incorporates inertial information additionally to the monocular cues, in our experience, its performance fluctuates a lot across runs. The results shown in Table 6 are leveraged over five runs. As visible in Figure 10, the trajectories in the Irchel sequences are mostly at different altitude levels, while the three EuRoc trajectories in Figure 11b are more interleaved. Judging from our experiments, CCM-SLAM estimates more accurately the altitude offset between these trajectories than VINS-mono, resulting in a more precise collaborative estimate on the Irchel dataset, while this effect does not occur with the nested EuRoC trajectories. Finally, we evaluated the online pose estimation of VO to examine the effect of shared data on the tracking of the agent, with the results shown in Table 7. We run CCM-SLAM with two agents using for the agent the datasets indicated in Table 7, without sharing data amongst the participant ('Individual Agents') and with data exchange ('Collaborative Agents'). For each incoming frame, we store the pose estimate calculated by *tracking*, i.e. without any global optimization. At the end of the experiment, we globally align all poses to ground truth, and calculate the error to estimate the accuracy of VO, and add up the errors for Agent 1 and Agent 2 to obtain an accuracy measure incorporating all participants. The results reported in Table 7 show that through collaboration, the participating agents can improve the accuracy of their onboard VO estimates *during* the mission.

Table 7: Comparison of tracking accuracy with and without collaboration amongst agents. The cumulative tracking RMSE adds up from the tracking RMSE of both agents and is calculated using the online pose estimates for each frame after it was processed by *tracking*. All values are averaged over 3 runs for each experiment.

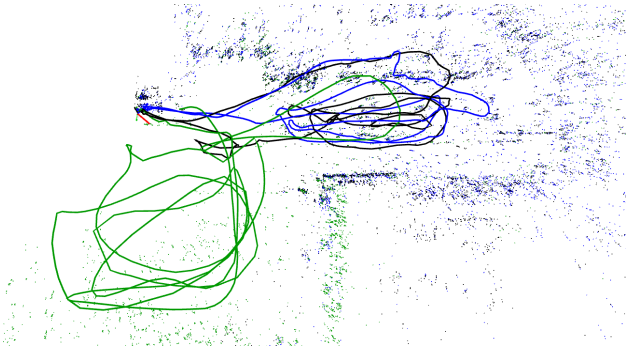
Dataset		Cumulative Tracking RMSE [m]	
Agent 1	Agent 2	Individual Agents	Collaborative Agents
MH01	MH02	0.296	**0.265**
MH02	MH03	0.327	**0.272**

5.8 Real-world Flight with Three UAVs

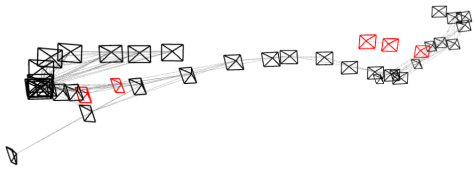
In order to attest for to the practicality of CCM-SLAM in a real-world scenario, simultaneous flights are performed with three UAVs running CCM-SLAM during flight, with the results illustrated in Figure 12. For



(a) View from one agent, with the VO scene landmarks for tracking superimposed.



(b) 3D view of the final joint KF trajectories in the *server map*



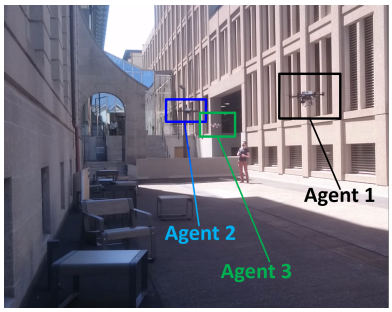
(c) *Local map* from Agent 1, with newly received KFs from the server to augment the *local map* colored in red.

Figure 11: The *server map* (joint KF trajectories and MPs) created by three UAV agents on EuRoC sequences, colored in black, blue and green to distinguish the three agents. The illustration shows only strong covisibility edges of weight $w \geq 100$.

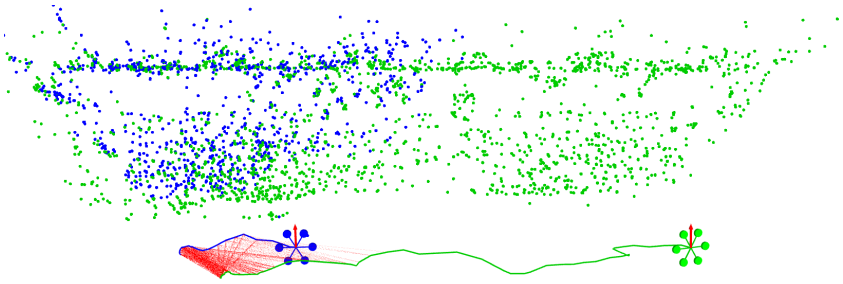
this experiment, the agent side's modules were run *onboard the agents during simultaneous flight*, continuously communicating with the ground station using a wireless connection. The experiment took place in an urban environment of approximately 30m $\times$ 7m, shown in Figure 12a. In addition to Figure 12, the video[3] accompanying this article shows the experiment described in this section.

Initially, one *server map* is initialized for each agent. Then overlap between the *server maps* of Agent 2 and 3 is detected, resulting in a new *server map* incorporating the experiences of both agents, shown in Figure 12b. Subsequently, when overlap is detected with the *server map* of Agent 1, all *server maps* are merged to one final *server map* containing the KFs and MPs of all agents, shown in Figure 12c directly after the fusion. From then on, all UAVs are localized in the same map, continuing their onboard estimations and collaboration, as shown in Figure 12d at the end of the experiment. In addition to the two *map fusion* steps, *intra-map place recognition* was able to detect two loops during this experiment.

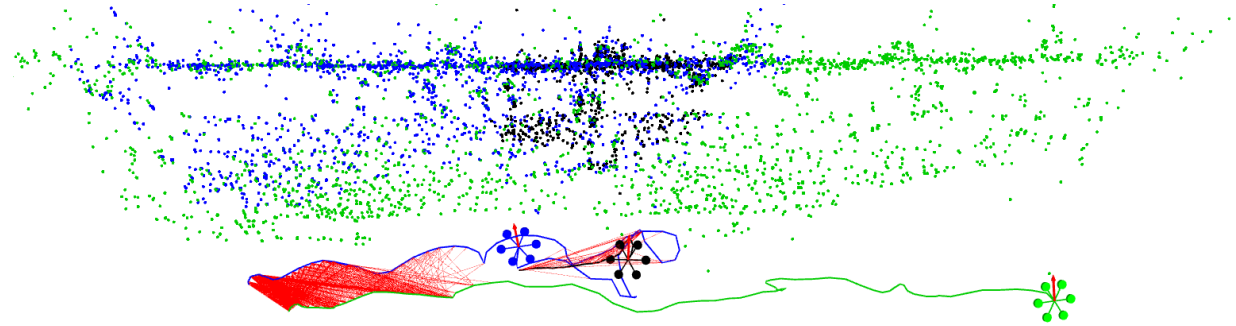
[3] `https://tinyurl.com/y8rdugwy`



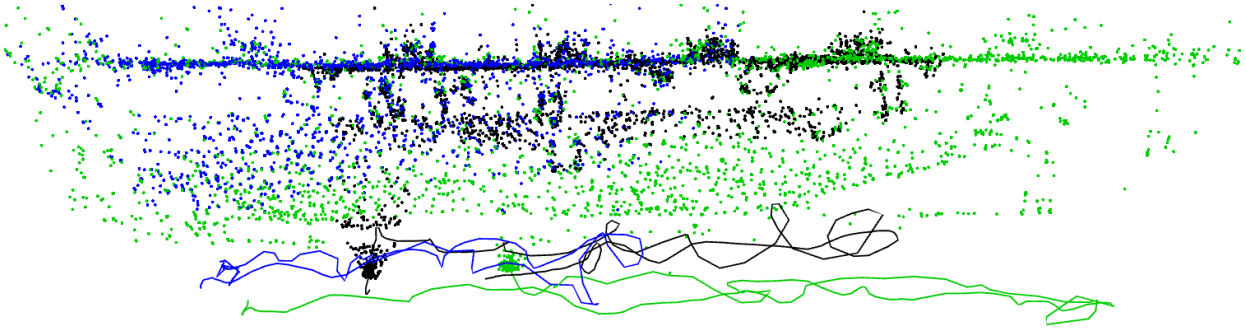
(a) Scene view.



(b) *Server map* (trajectories and MPs) of Agent 2 and 3 after *map fusion*.



(c) *Server map* with the trajectories and MPs from of Agent 1, 2 and 3 directly after *map fusion*. All three agents are localized in one *server map*.



(d) Final *Server map* with the trajectories and MPs from Agent 1, 2 and 3, at the end of the experiment. All agents contributed with their newly created KFs to this single *server map*. Constraints between trajectories and UAV bodies are left our for clarity.

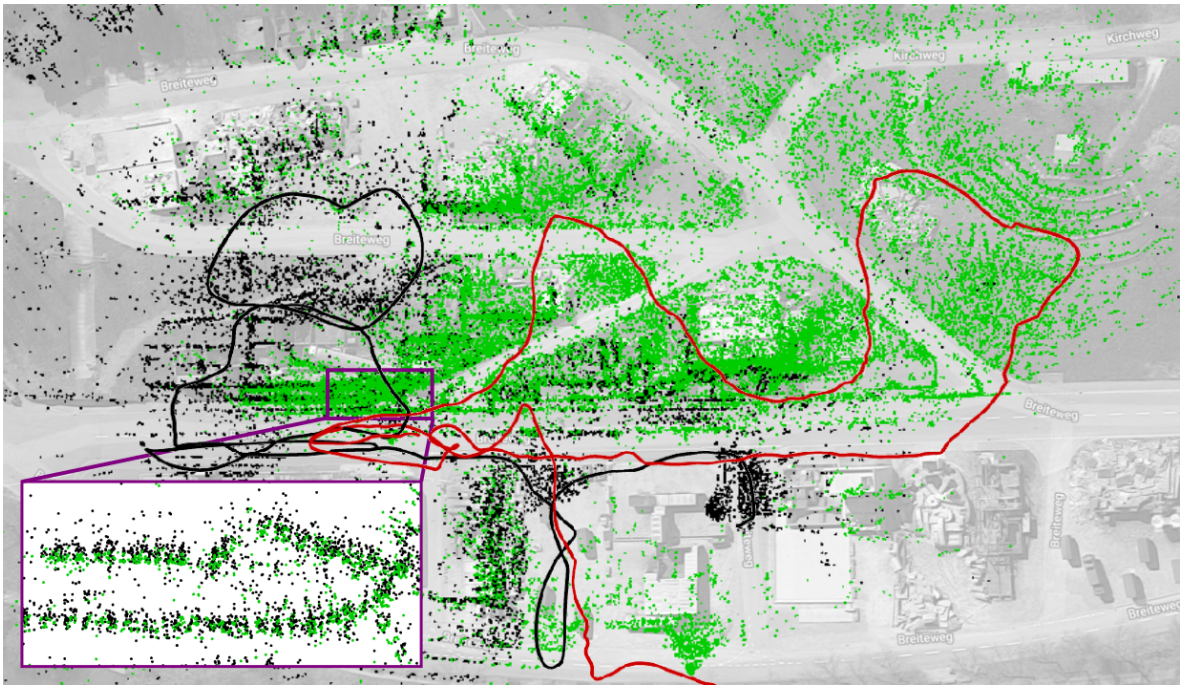
Figure 12: Real-world experiment with three UAV agents simultaneously flying, running CCM-SLAM onboard during flight. Different colors encode data (trajectories and MPs) contributed from different agent. Red lines indicate constraints between different trajectories from *map fusion* and *loop closure*.

5.9 Collaborative SLAM in a Search-and-Rescue Scenario

After confirming the practicality of CCM-SLAM in a real-world experiment (Section 5.8), in this section we aim to test its applicability to larger scenes by deploying two UAVs running CCM-SLAM in a search-and-rescue scenario. We use the training village for rescue operations depicted in Figure 13a, where we use an area spanning approximately 200m $\times$ 100m for our experiment, containing several collapsed building and other disaster scenarios. For this experiment, Agent 1 covers the left side of this area, while Agent 2 explores the right side. Again, both agents start individually exploring their environment. When overlap between the *Server maps* from both agents is detected, these maps are merged into one single *Server map* where both agents contribute to, exploring the area from then on as a team. This eventually results in the trajectories and the collaborative map shown in Figure 13c. Figure 13b shows a view from Agent 2 on the scene.

(a) 3D scene view of the search-and-rescue training village used for this experiment.

(b) View from Agent 2 on the scene.



(c) Topdown view of the final *Server map* collaboratively built by Agent 1 and 2, with trajectories, overlayed with the view of the area from Google Maps. The purple inlet enlarges the collaborative point cloud at the location of the derailed train, with the correctly aligned 3D points from Agent 1 and 2 in this area, indicating a good quality of the collaborative estimates.

Figure 13: Application of CCM-SLAM in a search-and-rescue scenario, attesting to its applicability in real-world large-scale environments. Agent 1 (black) explores the left side of the area, while Agent 2 (green MPs, trajectory in red) covers the right side of the village, with overlap between the two trajectories along the main street in the middle and at the structures at the bottom of the area.

Since accurate ground truth is not available in this environment, we cannot assess the error of the estimation quantitatively. However, examining the aligned CCM-SLAM-map with the corresponding view from Google Maps, e.g. the area of the derailed train (enlarged in the purple inlet in Figure 13c), reveals that the CCM-SLAM map points of both agents align well, indicating a good quality of the collaborative estimates.
Besides attesting to the applicability of CCM-SLAM in one of the target application scenarios, this experiment also shows that CCM-SLAM is able to handle an area of this size (1336 KFs, 46413 MPs) and with a standard Wi-Fi router (TP-LINK Archer C7 used here), collaborative SLAM can still run reliably ensuring real-time operation onboard each agent. For much larger scenarios, it would probably be necessary to improve the network infrastructure to cover the area with a more powerful router, as well as with and a more powerful *Server* that is able to process larger amounts of data during the mission in order to ensure that the agents have access to the collaborative estimates at runtime.

6 Conclusion

In this article, we present CCM-SLAM, a novel and powerful framework for collaborative SLAM for a centralized server and multiple agents, such as small UAVs or other robots, each equipped with a monocular camera and a small processing unit. The robotic agents start off the mission in a configuration previously unknown to each other and to the server, while running real-time visual odometry onboard to preserve their autonomy throughout the mission. The potentially more powerful central server collects the estimates computed onboard all agents acting as a book-keeper of everyone's experiences, while executing more time-consuming, but non time-critical tasks, such as searching for loop closures and overlap across agents' maps, and performing redundancy detection and global optimization when necessary. The server informs the agents of any updates of their onboard *local map* (e.g. following global optimization) and provides each agent with additional information (i.e. poses and landmarks) in its current operational vicinity if available, for example, collected by the same agent in the past or other agents. While existing works most often permit communication from the agents to the server only, this two-way communication in CCM-SLAM is key in enabling better and more consistent onboard estimates. Successfully addressing the challenge of robust handling of common communication disturbances, CCM-SLAM goes the step further to truly exploit the presence of multiple agents within the area of interest during the mission.

A thorough experimental analysis on benchmarking and new datasets reveals CCM-SLAM's powerful ability in dealing with common network problems, such as delays and message loss and its adaptability to the computational resources onboard each agent and the available bandwidth of the communication network. A comparison of CCM-SLAM's estimates confirms that collaboration amongst participating agents improves the accuracy of the final global trajectory estimates as well as the online pose estimates onboard the agents during runtime. Moreover, the scalability of this framework in the number of agents is studied, confirming that the agent's real-time capabilities are not compromised with more agents. As expected, every new agent adds more burden to the network traffic, therefore, bandwidth is the only limiting factor for scalability on top of the server's computational capabilities. Making use of the insights of this analysis in a setup with three UAVs of heterogeneous onboard computational resources, we demonstrate the power of CCM-SLAM in real experiments, with all UAVs flying over the same area, while exchanging information via a standard laptop acting as the server. It is only with such a robust and efficient framework that collaborative localization and mapping can be performed, which is the first step for further collaborative tasks for the robotic team.

Future directions will focus on studying the value of each bit of information to be shared amongst the robotic team to provide insights towards more agile manipulation of information and good trade-offs between efficiency and accuracy in the estimates of the collaborative system.

Acknowledgments

This research was supported by the Swiss National Science Foundation (SNSF, Agreement no. PP00P2157585) and NCCR Robotics. The authors want to thank Armasuisse for providing access to the search-and-rescue training village for our experiments.

References

Achtelik, M. W., Weiss, S., Chli, M., Dellaert, F., and Siegwart, R. (2011). Collaborative stereo. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Burri, M., Nikolic, J., Gohl, P., Schneider, T., Rehder, J., Omari, S., Achtelik, M. W., and Siegwart, R. (2016). The EuRoC micro aerial vehicle datasets. *International Journal of Robotics Research (IJRR)*, 35(10):1157–1163.

Chen, X., Lu, H., Xiao, J., and Zhang, H. (2017). Distributed monocular multi-robot slam. In *Proceedings*

of IEEE International Conference on CYBER Technology in Automation, control, and intelligent systems (IEEE-CYBER).

Choudhary, S., Carlone, L., Nieto, C., Rogers, J., Christensen, H. I., and Dellaert, F. (2017). Distributed mapping with privacy and communication constraints: Lightweight algorithms and object-based models. *International Journal of Robotics Research (IJRR)*, 36(12):1286–1311.

Chung, S.-J. and Slotine, J.-J. E. (2009). Cooperative robot control and concurrent synchronization of Lagrangian systems. *IEEE Transactions on Robotics (T-RO)*, 25(3):686–700.

Cieslewski, T., Choudhary, S., and Scaramuzza, D. (2018). Data-efficient decentralized visual slam. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA).*

Cieslewski, T. and Scaramuzza, D. (2017). Efficient decentralized visual place recognition using a distributed inverted index. *IEEE Robotics and Automation Letters (RA-L)*, 2(2):640–647.

Cunningham, A., Indelman, V., and Dellaert, F. (2013). DDF-SAM 2.0: Consistent distributed smoothing and mapping. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA).*

Davison, A. J., Reid, I. D., Molton, N. D., and Stasse, O. (2007). MonoSLAM: Real-time single camera SLAM. *IEEE Transactions on Pattern Analysis and Machine Intelligence (T-PAMI)*, 29(6):1052–1067.

Deutsch, I., Liu, M., and Siegwart, R. (2016). A Framework for Multi-Robot Pose Graph SLAM. In *IEEE International Conference on Real-time Computing and Robotics (RCAR).*

Dong, J., Nelson, E., Indelman, V., Michael, N., and Dellaert, F. (2015). Distributed real-time cooperative localization and mapping using an uncertainty-aware expectation maximization approach. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 5807–5814. IEEE.

Eade, E. and Drummond, T. (2006). Scalable monocular SLAM. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 469–476. IEEE.

Egodagamage, R. and Tuceryan, M. (2017). Distributed Monocular SLAM for Indoor Map Building. *Journal of Sensors*, 2017.

Engel, J., Koltun, V., and Cremers, D. (2017). Direct sparse odometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence (T-PAMI).*

Engel, J., Schöps, T., and Cremers, D. (2014). LSD-SLAM: Large-Scale Direct Monocular SLAM. In *Proceedings of the European Conference on Computer Vision (ECCV).*

Forster, C., Lynen, S., Kneip, L., and Scaramuzza, D. (2013). Collaborative monocular SLAM with multiple micro aerial vehicles. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS).*

Gálvez-López, D. and Tardos, J. D. (2012). Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics (T-RO)*, 28(5):1188–1197.

Giamou, M., Khosoussi, K., and How, J. P. (2018). Talk resource-efficiently to me: Optimal communication planning for distributed slam front-ends. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA).*

Google Inc. (2014). Project Tango. url http://www.google.com/atap/projecttango.

Guo, C. X., Sartipi, K., DuToit, R. C., Georgiou, G., Li, R., OLeary, J., Nerurkar, E. D., Hesch, J. A., and Roumeliotis, S. I. (2016). Large-scale cooperative 3D visual-inertial mapping in a Manhattan world. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA).*

Karrer, M., Schmuck, P., and Chli, M. (2018). CVI-SLAM - Collaborative Visual-Inertial SLAM. *IEEE Robotics and Automation Letters (RA-L)*, 3(4):2762–2769.

Klein, G. and Murray, D. (2007). Parallel tracking and mapping for small AR workspaces. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 225–234. IEEE.

Kümmerle, R., Grisetti, G., Strasdat, H., Konolige, K., and Burgard, W. (2011). g2o: A general framework for graph optimization. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA).*

Kushleyev, A., Mellinger, D., Powers, C., and Kumar, V. (2013). Towards a swarm of agile micro quadrotors. *Autonomous Robots*, 35(4):287–300.

Leonardos, S. and Daniilidis, K. (2017). A game-theoretic approach to robust fusion and kalman filtering under unknown correlations. In *American Control Conference (ACC), 2017*, pages 2568–2573. IEEE.

Loianno, G., Mulgaonkar, Y., Brunner, C., Ahuja, D., Ramanandan, A., Chari, M., Diaz, S., and Kumar, V. (2016). A swarm of flying smartphones. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*, pages 1681–1688. IEEE.

Luft, L., Schubert, T., Roumeliotis, S. I., and Burgard, W. (2016). Recursive Decentralized Collaborative Localization for Sparsely Communicating Robots. In *Proceedings of Robotics: Science and Systems (RSS)*.

Martinelli, A., Pont, F., and Siegwart, R. (2005). Multi-robot localization using relative observations. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Mohanarajah, G., Usenko, V., Singh, M., D'Andrea, R., and Waibel, M. (2015). Cloud-based collaborative 3D mapping in real-time with low-cost robots. *IEEE Transactions on Automation Science and Engineering*, 12(2):423–431.

Morrison, J. G., Gálvez-López, D., and Sibley, G. (2016). MOARSLAM: Multiple Operator Augmented RSLAM. In *Distributed Autonomous Robotic Systems*, volume 112, pages 119–132.

Mur-Artal, R., Montiel, J., and Tardós, J. D. (2015). ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics (T-RO)*, 31(5):1147–1163.

Oleynikova, H., Burri, M., Lynen, S., and Siegwart, R. (2015). Real-time visual-inertial localization for aerial and ground robots. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*, pages 3079–3085. IEEE.

Piasco, N., Marzat, J., and Sanfourche, M. (2016). Collaborative localization and formation flying using distributed stereo-vision. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Qin, T., Li, P., and Shen, S. (2018). VINS-mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics (T-RO)*, 34(4):1004–1020.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). ROS: an open-source Robot Operating System. In *ICRA workshop on open source software*.

Riazuelo, L., Civera, J., and Montiel, J. (2014). C2TAM: A cloud framework for cooperative tracking and mapping. *Robotics and Autonomous Systems (RAS)*, 62(4):401–413.

Schmuck, P. and Chli, M. (2017). Multi-UAV Collaborative Monocular SLAM. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Schwager, M., Dames, P., Rus, D., and Kumar, V. (2017). A multi-robot control policy for information gathering in the presence of unknown hazards. In *Robotics Research*, pages 455–472. Springer.

sfly (2009). Swarm of micro flying robots (sFly). `http://www.sfly.org`.

Strasdat, H., Montiel, J. M., and Davison, A. J. (2012). Visual SLAM: why filter? *Image and Vision Computing*, 30(2):65–77.

Vidal-Calleja, T. A., Berger, C., Solà, J., and Lacroix, S. (2011). Large scale multiple robot visual mapping with heterogeneous landmarks in semi-structured terrain. *Robotics and Autonomous Systems (RAS)*, 59(9):654–674.

Webster, S. E., Walls, J. M., Whitcomb, L. L., and Eustice, R. M. (2013). Decentralized extended information filter for single-beacon cooperative acoustic navigation: Theory and experiments. *IEEE Transactions on Robotics (T-RO)*, 29(4):957–974.

Zhang, H., Chen, X., Lu, H., and Xiao, J. (2018). Distributed and collaborative monocular simultaneous localization and mapping for multi-robot systems in large-scale environments. *International Journal of Advanced Robotic Systems*, 15(3):1729881418780178.

Zou, D. and Tan, P. (2013). CoSLAM: Collaborative visual SLAM in dynamic environments. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 35(2):354–366.