# Towards Collaborative Mapping and Exploration Using Multiple Micro Aerial Robots

**Sikang Liu, Kartik Mohta, Shaojie Shen and Vijay Kumar**

**Abstract** In this paper, we present a system for collaborative mapping and exploration with multiple quad rotor robots. The basic architecture and development of the algorithms for mapping and exploration validate our system with both simulation and real-world experiments. We utilize the 2.5-D structure of typical indoor environments and demonstrate the deployment of multiple autonomous quadrotors equipped with IMUs and laser scanners engaged in collaborative exploration. Estimation, control and planing algorithms are highly integrated in our system to achieve robust and efficient exploration behaviors.

**Keywords** Multi-robot · Mapping · Exploration · SLAM · Quadrotor

Robotics mapping is an attractive area of research with a lot of potentials for development. In particular, systems with micro aerial vehicles (MAVs) that are capable of autonomously mapping and exploring have significant impacts on the field of search and rescue [1]. Unlike to ground robots, aerial robots have superior mobility and they are not affected by complex terrains. These advantages make micro aerial robots as the ideal platform for search and rescue missions.

However, it is challenging to develop and deploy autonomous aerial robots for many reasons. First, because of the fast dynamics and limited payload for sensors, it is difficult to develop state estimators that can be used for real-time control. Second, limitations on energy and power density restrict the duration of the missions. Third, the infrastructure for running experiments with multiple autonomous aerial robots requires the tight integrations between communication, control and perception. Fortunately, it is possible to leverage the extensive literature on mapping and exploration with ground robots [2, 3], and the more recent literature on micro aerial vehicles [4] (IJRR reference).

Simultaneous Localization and Mapping (SLAM) algorithms address the problem of localizing a mobile robot while incrementally building a consistent map in an unknown environment [5]. It mainly consists of a front end and back end. In the front end, feature detection and scan matching are implemented to create constraints

S. Liu (✉) · K. Mohta · S. Shen · V. Kumar
GRASP Laboratory, University of Pennsylvania, Philadelphia, PA 19104, USA
e-mail: sikang@seas.upenn.edu

on incremental movements to estimate the robot position and orientation. In the back end, an optimization algorithm is used to recover the trajectory and the map over a set of robot actions and measurements. Multiple approaches have been used to solve SLAM problem. In particular, Smoothing and Mapping (SAM) is one method that can recover the history of poses (trajectory) and the map and is suited to real-time applications [6]. GTSAM is a C++ package including multiple SAM built-in functions. In this paper, we use this package as the back end to solve the SLAM problem.

The quadrotor is an ideal aerial robot platform for control and navigation. By carrying sensors like a laser and an IMU with an onboard computer, the quadrotor can be autonomously navigate indoor environments [7]. Graph-based SLAM algorithms have proved to be quite effective [8]. They are reliable and efficient in 2.5-D environments and relatively easy to use. Graph-based SLAM algorithms used with a laser-scanner and an IMU yields pose estimation. These estimates along with an Unscented Kalman Filter (UKF) [9] can be fused to get state estimation which in turn can be used for control. Traditional back-stepping PID controllers [10] or nonlinear controllers [11] can be used for control. We build on all these results to develop a robust architecture for autonomous flight for our quad rotors shown in Fig. 1.

The exploration problem involves the robot determining actions for navigation that allow efficient mapping of an unknown environment. Single robot exploration has been done in a GPS-denied indoor environment by multiple approaches [12, 13]. When it comes to multiple robots, the complexity of system and algorithms increase dramatically, and it's hard to implement in the real world especially for aerial vehicles. Some experiments related in collaborative mapping and exploration using multiple vehicles have been conducted in [14, 15], but none of them has done multi-robot mapping or exploration on system of multiple MAVs.

We mainly focus on constructing a reliable system for multiple MAVs mapping and exploring in an constraint indoor environment. We use the OmniMapper [16] and GTSAM [17] developed at Georgia Tech University for the back end and create our own front end. The main contributions of this paper however are the integration of these algorithms to solve the multi-quadrotor SLAM problem efficiently and the development of an exploration algorithm that allows autonomous mapping of unknown environments. We report simulation results with 3 quadrotors and experimental results with 2 quadrotors demonstrating the efficacy of the proposed approach.

## 1 Technical Approach

The control of a small team of robots can be done either in a centralized or a decentralized manner. Distributed data fusion from multi-robot system has been demonstrated in [18]. However, even for small teams, it is necessary to consider two constraints. First, the onboard computational resources can be limited, as it is in our case (an Atom 1.6 GHz processor). Second, it is not possible to transmit large amounts of information without latency over a wireless network. Thus, we use a centralized

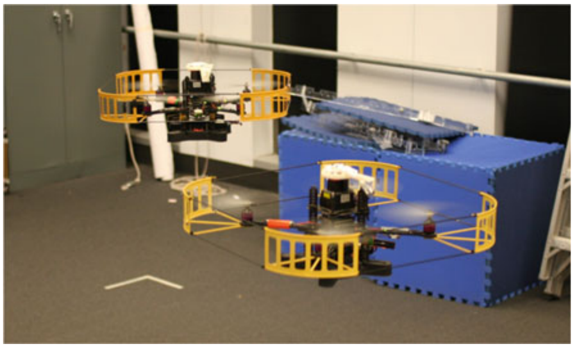
**Fig. 1** Our multi-quadrotor system. The base platform is an Ascending Technologies Pelican quadrotor equipped with a Hokuyo UTM-30LX laser scanner

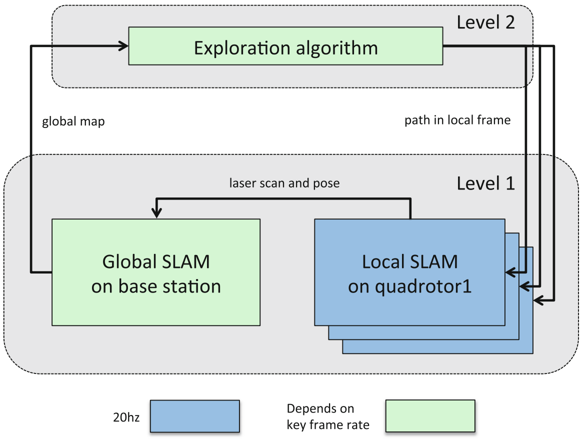

**Fig. 2** System hierarchy for multi-robot mapping and exploration

architecture in which a one base station (laptop) is connected to multiple clients (quadrotors). In this case, inter-process communication is established between robot and base station, but not between the robots themselves.

The software hierarchy (Fig. 2) has two levels. The first level is called collaborative SLAM, in which we mainly focus on solving the SLAM problem from information coming from all the robots. The second higher level is the planner for multi-robot exploration. We will present details of each level in the following sections.

## *1.1 Collaborative SLAM in Multi-robot Scale*

Collaborative SLAM includes SLAM algorithm running on all the devices. Each robot, or the client, runs independent onboard SLAM algorithm in which a occupancy grid map in robot frame will be created. This algorithm is a local SLAM solution without any loop-closure detection or correction. This is key to localization and generating state estimates for control.

Local SLAM is used to localize the robot and generate key frames. Key frames with corresponding laser scans are transmitted to the base station every 0.2 m or 0.1 rad.

The Global SLAM algorithm runs on the base station and merges the information obtained from all the robots. This involves the creation of a pose graph, detection of loop closures and the optimization of the pose graph.

**Local SLAM** The local SLAM algorithms assumes a 2.5-D indoor environment. We use the standard map correlation-based laser scan matching for local SLAM without considering loop closure. The details of this approach and implementation are described in [7]. Denote the local map of robot $k$ at time $i$ as ${}^{l}M_i^k$. When the pose estimator receives a new laser scan, it outputs the predicted position ${}^{l}x_{i+1}^k$ in local map frame. The pose state includes position in $x$, $y$ direction and yaw, denoted as ${}^{l}x^k = \{x, y, \theta\}$.

Our method differs from [7] in that our local SLAM algorithm generates two sets of key frames. The first set is based on the local time. Every 0.25 s, a key frame is established and the laser scan is used to build an occupancy grid map for localization. The second set is based on distance travelled. Key frames are generated every 0.2 m and 0.1 rad and sent to the base station to construct a global map. This allows us to minimize data rates for communication while ensuring adequate information is available for a good pose graph.

**Global SLAM** The Global SLAM algorithm runs on the base station. It takes as input key frames and laser scans from the the robots and generates a global map. Every key frame pose is added as a node to the pose graph, and the SAM algorithm is used to optimize the pose graph.

The construction of a global pose graph requires that constraints between independent Markov chains generated by individual robots are resolved correctly in [14]. The position of robot $k$ at time $i$ in global frame is denoted by node ${}^{g}x_i^k$ (Fig. 3). The edge between two adjacent nodes $\{{}^{g}x_{i-1}^k, {}^{g}x_i^k\}$ stands for corresponding odometry constraint, denoted as $e_{i-1}^k$, it's estimated from local SLAM. Knowing pose and odometry constraint at time $i-1$, we create the next pose ${}^{g}x_i^k$ as

$$ {}^{g}x_i^k = {}^{g}x_{i-1}^k + e_{i-1}^k \tag{1} $$

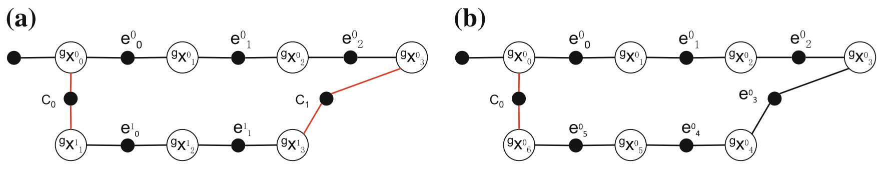

**Fig. 3** Pose-graph examples. **a** Pose-graph consists of two Markov chains, two constraints are added between robot 0 and 1, **b** loop-closure constraint in single Markov chain of robot 0

Notice the global map frame is different from local map frame, but we can add the relative odometry equivalently.

Except odometry constraints from local SLAM, there're two extra constraints can be created, namely, constraint between robots and loop-closure constraint. Essentially, the types of these two constraints are the same.

One source of constraint comes from laser scans comparison between different nodes of different robots (the black dot connected by red lines in Fig. 3a). Denote it as $c_p$, $p = 0, 1, 2 \ldots$. It is generated through following way: for robot $k$, we apply a searching area around its current pose ${}^g x_i^k$, if the closest pose in global map comes from robot $j$, $j \neq i$, and this pose ${}^g x_p^j$ satisfies

$$\|{}^g x_i^k - {}^g x_p^j\|_t < r_t, \quad \|{}^g x_i^k - {}^g x_p^j\|_\theta < r_\theta \tag{2}$$

Here $\| \bullet \|_t$ and $\| \bullet \|_\theta$ means translation and rotation between two poses. $r_t$ denotes the euclidean distance threshold and $r_\theta$ indicates the yaw threshold. Normally, we set it as $r_t = 10\,\text{m}$, $r_\theta = \frac{\pi}{2}$. Then we try scan matching on two scans corresponding to these two poses. If these two scans match successfully, we can get the constraint $c_p$ and add it into pose graph between the corresponding nodes.

The other type of constraint is derived from loop-closure detection in similar way: when a new pose ${}^g x_i^k$ added into the current pose-graph, we check the region around it with respect to a certain threshold $\{r_t, r_\theta\}$. If the closest pose within the region comes from robot itself, as ${}^g x_p^k$, $p < i$, we check the difference in timestamp as $d_t = i - p$. If $d_t > d_{thresh}$, which means the robot traveled for a while and might form a loop, we try scan matching between these two laser scans and add output constraint in to pose-graph if there is a valid match (Fig. 3b).

The scan matching algorithm we employ in global SLAM is called canonical scan matching (CSM) which is a fast version of iterative closest point (ICP) algorithm. The strategy is depicted in paper [19]. CSM is vulnerable to noise since the matching uses laser scans directly without creating a probabilistic map. In order to increase rate of matching and achieve correct constraint, filtering out the floor in laser scan is quite necessary.

Optimization process starts when the pose-graph is updated. Batch optimizing is the fundamental way to solve a non-linear maximization problem as stated in [20]. However, it requires considerable computation to solve this equation and this requirement increases dramatically with the number of nodes in pose-graph. A novel linearizing algorithm based on batch optimizing is proposed aiming at decreasing the computational demand, called incremental smoothing and mapping (iSAM). We use the iSAM2.0 as the solver of SLAM back-end, due to our experimental comparison, it's 10-times faster than vanilla batch method. iSAM2.0 is ready-to-use in GTSAM library, it takes pose graph as input and calculates optimized pose-graph. We use OmniMapper to construct pose graph from multiple local SLAM and links pose graph to iSAM2.0 solver. The algorithm detail is referred in [21].

## 1.2 Autonomous Navigation and Exploration

We now present modules that enable autonomous exploration of the environment based on the incrementally updated global map built by base station. All modules in the exploration subsystem runs in response to new maps from the global SLAM.

**Frontier detection** The algorithm for autonomous exploration relies on the detection of frontier points in the current 2-D occupancy grid map.

We define frontier points as unoccupied points that have half of their neighbors in free space and the other half in unknown space.

To be specific, for any point in free space, we check all its neighbors within radius $r$ (denoted as the red circle in Fig. 4). If half of points inside the circle is in free space and the other half is in unknown space, and there is no occupied point within this circle, we identify the center point as a frontier point. Figure 4a illustrates a typical frontier point, while Fig. 4b, c are examples of invalid frontier points.

In most cases, the frontier detector gives all the adjacent frontier points along the frontier edge (Fig. 5a). Grouping of frontier points is necessary to avoid decreases in algorithm efficiency due to excessive number of frontier points along the same frontier edge. We combine frontier points along the same frontier edge into a single point for exploration (Fig. 5b).

**Goal assignment** Usually, there are multiple frontier points in the global map during exploration. We assign a feasible goal to each robot such that the following cost function is minimized:

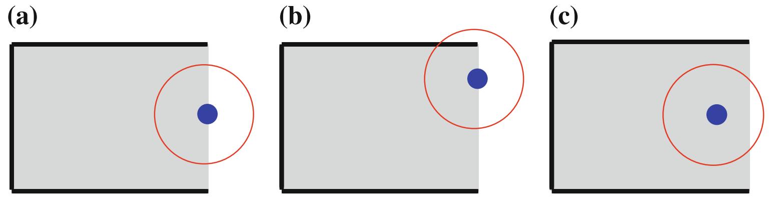

**Fig. 4** Samples of valid and invalid frontier points. **a** Shows a typical frontier point. The *circle* in (**b**) includes occupied point, and the number of points in unknown space is insufficient in (**c**). Unknown, unoccupied and occupied spaces are *color coded* in *white*, *gray* and *black* respectively. **a** Valid, **b** invalid, **c** invalid

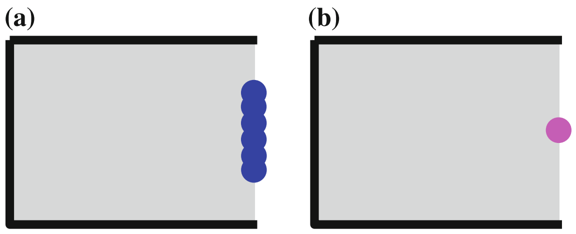

**Fig. 5** Grouping, *blue dots* are raw frontier points, *purple* one is the corresponding grouped point. **a** Adjacent frontier points along a edge, **b** grouped frontier point

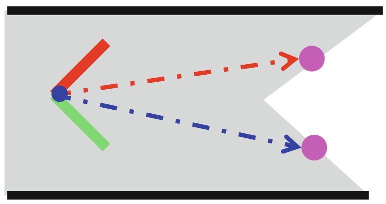

**Fig. 6** The costs of two paths (*red* and *blue*) are equivalent if $w_3 = 0$, the robot could frequently jump from one path to the other while moving

$$c({}^g x_i^k, {}^g f_{ij}) = w_1 \| {}^g x_i^k - {}^g f_{ij} \|_t + w_2 \| {}^g x_i^k - {}^g f_{ij} \|_\theta + w_3 \| {}^g f_{ij} - {}^g f^k \|_t \tag{3}$$

where ${}^g f_{ij}$ is the $j$-th frontier point in current timestamp global coordinates. ${}^g f^k$ indicates the last goal assigned to robot $k$. $w_i, {}_{i=1,2,3}$ is the weighting factor for each cost terms. First two terms in RHS of Eq. (3) are the displacement in distance and orientation between current robot location and $j$-th frontier. The third term is used for anchoring the goal with respect to previous goal to avoid frequent switching of goals with similar costs (Fig. 6). Goals may be cleared before the robot reaches it due to the sensor measurement range. In such case, a pending goal is assigned to the robot it exists, otherwise the exploration is halted until a newer map with more frontiers is received.

**Path planning** Since current poses of all robots, goal positions, as well as the global map are available, it's feasible to plan the collision-avoidance path in occupancy grid map by RRT*. RRT* [22] is employed due to its fast planning speed and its convergence property to the optimal solution. An open source implementation of RRT* is available in the `ompl` library [23]. We plan with free space assumption and consider both unknown and free space as the collision-free. The output from RRT* is not guaranteed to be optimal, however, if we set a reasonable region around robot as "no-collision-zone" to keep the robot path away from obstacles, RRT* is able to produce valid collision-free paths for our MAV platforms.

As the robot moves along the planned path and observes new environments, the path may be invalidated due to newly observed obstacles intersecting with the path. In such case, a replan will be initiated.

**Strategy for multi-robot coordination** If there are more than one robot exploring in same environment simultaneously, the exploration steps will be more complicated. For instance, assigning the same goal to multiple robots is dangerous, aerial vehicles can crash easily when they move close to each other due to the interference from turbulence generated by propellers; when each robot plan the path, the planner should take consider of the position of other robots such that the path won't collide with any of them. Several rules need to be integrated such that a reliable exploration algorithm can be achieved for multi-robot system.

- In step of goal assignment, if a target is chosen as the current goal for robot $k$, this goal will be removed from the queue of frontier points;
- To avoid collision between robots, when planning path for robot $k$, set the region with a certain radius around other robots' locations be occupied.

**Transformation between different coordinate frames** The control process is running onboard (in local map frame), but we plan path in global map frame. It requires for transformation between local frame and global frame before sending a new path from base station to robot. Assume we have a path planned in global map, and the waypoint generated is denoted as ${}^{g}p_i^k$, where $i = 1, 2 \ldots n$, $n$ is the total number of waypoints in this path. The corresponding waypoint in local map is ${}^{l}p_i^k$. We can get the rotation matrix ${}^{l}R_g$ and translation vector ${}^{l}T_g$ from global coordinate into local coordinate from initial waypoints in two coordinates, and apply Eq. 4 for waypoints transformation. Notice that the initial waypoint is always the current pose of robot when starting planning in both coordinates.

$${}^{l}p_i^k = {}^{l}R_g {}^{g}p_i^k + {}^{l}T_g \tag{4}$$

## 2 Experimental Results and Analysis

In this section, we present simulation and experimental results demonstrating the working of the mapping and exploration algorithms. We developed the system in steps, the first being testing the collaborative SLAM system offline using log files recorded by the carrying the robots. Then, the higher level exploration algorithm was implemented in simulation with up to 3 quadrotor models. The simulation included a laser simulator based on ray casting and a quadrotor dynamics simulator in order to have an accurate simulation of the real system. Once we were confident about the algorithm working in simulation, we implemented it for a single quadrotor and extensively tested it in different indoor environments in order to check its reliability. Convinced that the reliability was high enough, we added one more robot in order to build the exploration system with two quadrotors. Due to our assumption about the star topology of communications and limited range of wireless, we were not able to test the multi-robot system in large indoor spaces.

### *2.1 Test on Collaborative SLAM and Simulation for Exploration*

Our exploration algorithm requires a good map to find frontiers and generate valid paths for each robot. Distortion or loop-closure errors in the map often lead to inappropriate paths. We ran tests of the collaborative SLAM using log files to examine the quality of mapping. The quality of created maps and speed of optimizing pose graph were considered as the two important criteria for the algorithm. Log files from three robots carried around the same area have been used to create a global map as shown in Fig. 7. The initial positions of robots are close to each other so that odometry constraints can be added in the pose graph between the initial poses of the robots

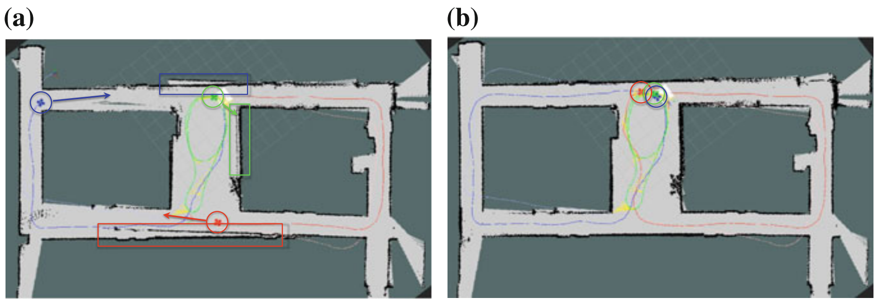

**Fig. 7** Test on collaborative SLAM using log files, iSAM2.0 is used in this test. **a** Loop-closure error results in the creation of spurious walls (inside *rectangulars*), **b** map has been corrected after finding loop-closure constraints

to not have disjoint Markov chains. The onboard local SLAM generates key frames after robot has travelled at distance of 0.2 m or rotated by 0.1 rad. This led to more than 500 nodes in the final pose graph (Fig. 7a), and iSAM2.0 is able to optimize it in 2 ms, while a batch optimization approach consumes more than 600 ms, both providing a similar quality output map. Thus, it was clear that in order to do real-time exploration, iSAM2.0 would serve as a good pose graph backend. From Fig. 7 we can see that our collaborative SLAM algorithm works well for merging maps from multiple robots.

The core task of exploration is the feasibility of finding frontiers and exploring those frontiers. We create a 2-D occupancy grid map from a schematic and generate 2.5-D environment from this prior map, laser simulation is used to provide laser scans similar to real-world estimation. Based on this architecture, we tested our algorithms in simulation with three robots. Figure 8 demonstrates exploration and map merging in a B-shape map. Both red and blue robots run a S curve, and these two curves cover the whole map such that the third robot (green) seems to be unnecessary (Fig. 8c). On the contrary, during exploration, even though two S-shape trajectories share common part of explored regions, the robots face opposite directions in these regions such that scan matching doesn't provide any odometry constraints between them (Fig. 8a). The third robot passes through this region and creates odometry constraint between itself and other robots. Consequently, even though there is a lack of constraints between the red and blue robots' poses, the constraints between green and red, and green and blue robots are able to recover the map and correct accumulated errors (Fig. 8b).

## 2.2 *Single Quadrotor Exploration*

After extensively testing the algorithms in simulations, we moved to a real robot and built a system for autonomous exploration with a single quadrotor. The quadrotor sends key-frame laser scan and pose from local SLAM to base station (laptop), the

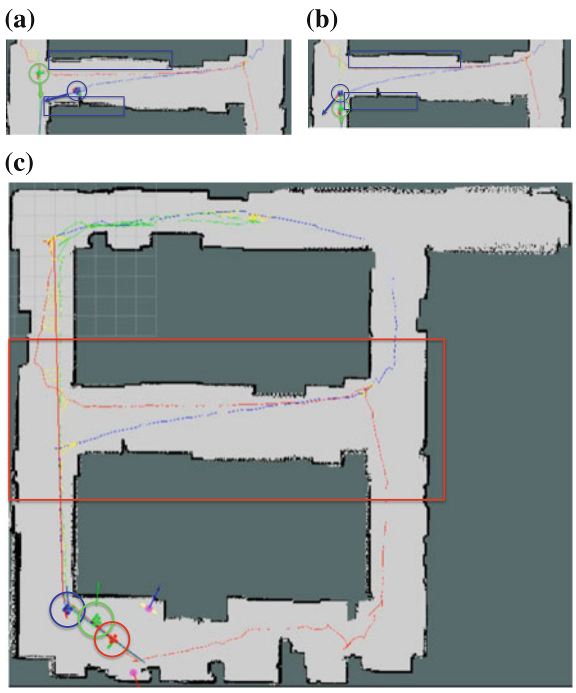

**Fig. 8** Simulation of 3 robots exploring in an environment created from a prior 2-D map. Figure (**a**) and Fig. (**b**) shows the region inside *red rectangular* in Fig. (**c**). **a** Accumulated error results in spurious walls when multiple robots explore same area with large bearing differences, **b** odometry constraints between *green* and *blue* robots correct the map and spurious wall disappears, **c** after exploration done, we get a map which is almost the same as the prior map

planning result is sent back to robot after every iteration of exploration process. As all the computations in order to localize and control the robot are running onboard the robot, even if the robot loses connectivity with the base station, the robot is still able to control itself, though it won't get any new assigned waypoints to follow.

We tested the single robot exploration in different cases and Fig. 9 shows the exploration process in a relatively large environment.

Now the problem is, how to evaluate the quality of autonomous exploration. Intuitively, we may consider the benchmark as the human controlled exploration. The speed comparing autonomous to manual exploring is a important index to show the quality of our algorithm. Figure 10 shows the experiments of single robot manually exploration and autonomous exploration. The total distances in $x - y$ plane travelled by robots in two cases are similar. Ideally, if we fly robot in the same velocity, the time consumed in these two trials should be similar. In manually fly, maneuver can control quadrotor fly at arbitrary speed; the maximum velocity of autonomous navigation depends largely on the system, in general it's slower than former. Hence even with the same travelled distance, the time for completely exploration in a certain area of these two cases varies. In our experiment, we manually fly the quadrotor at the same speed as autonomous fly, the time to complete exploration of these two experiments are almost the same (Table 1).

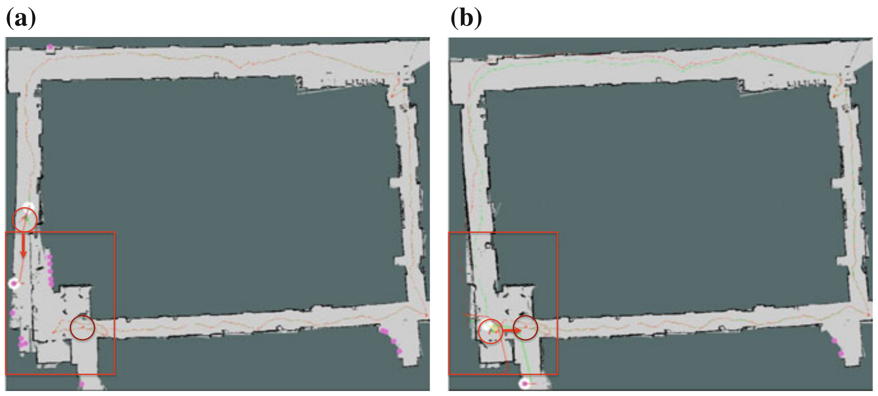

**Fig. 9** Single quadrotor exploration in a large environment (about 30 × 25 m). *Dark red circle* indicates the start position, *bright red circle* indicates the current robot position in global frame. **a** When quadrotor comes back to original position, the loop-closure error is significant (*red rectangular*), **b** Loop-closure constraint is generated from front-end, the map is corrected

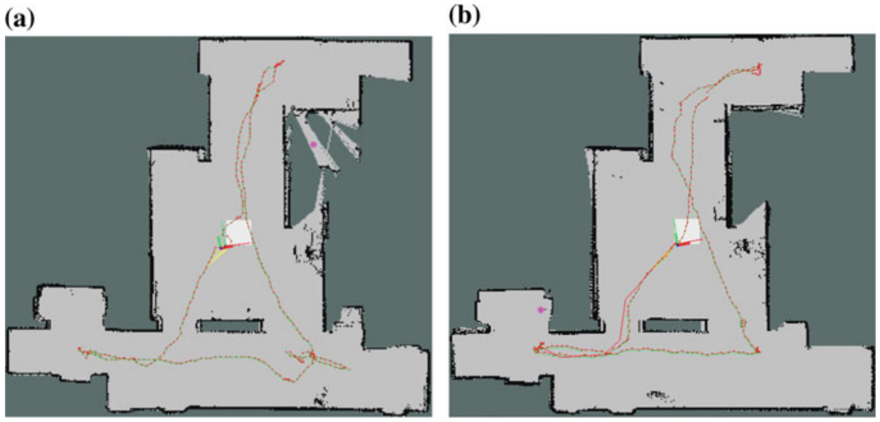

**Fig. 10** Comparison between manually and autonomously exploration in constraint indoor environment (15 × 15 m). **a** Manually control, **b** autonomous exploration

**Table 1** Exploration time for different methods

| Experiment | Manually control | Single-quad | Two-quad |
|---|---|---|---|
| Time (s) | 105 | 101 | 81 |

## *2.3 Two Quadrotors Exploration*

As a consistent to last section, we extend the system of single robot exploration into two. According to Sect. 1.2, several rules for avoiding collision and allocating goals are activated in order to guarantee a robust and reliable multi-robot maneuver system.

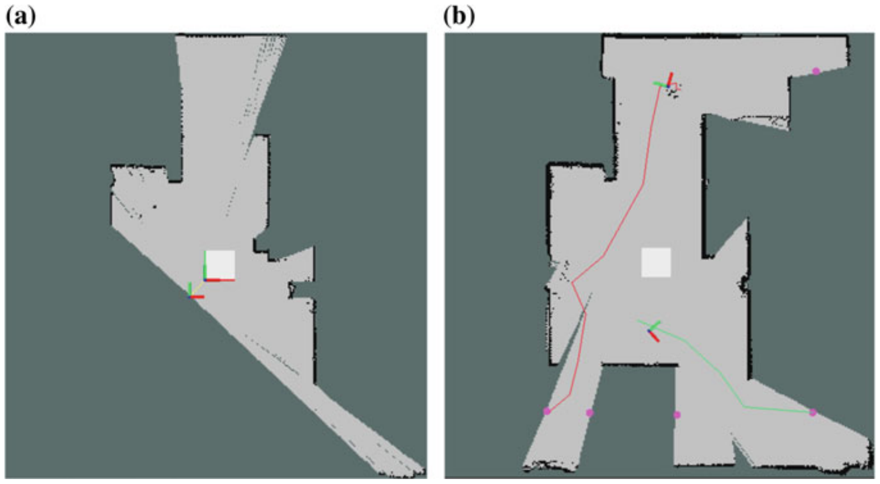

**Fig. 11** Two quadrotors exploration. **a** Initial positions of two robots, odometry constraint can be detected if they're close enough, **b** while exploring, the path is guaranteed to avoid crossing region around the other robot

Considering the initial constraint between the initial poses of both robots, we must not place robots in arbitrary positions, CSM would fail to generate the odometry constraint if the distance and angular difference between robots is too large. Instead, we start the two robots close to each other and face the same direction to ensure the initial constraint between robots (yellow lines in Fig. 11a) between robots generated successfully. Then we take off the robot one by one, start the exploration algorithm. The time consumed for exploration using two quadrotors in same area is 20 s shorter than single quadrotor (Table 1), it's straightforward to explain this improvement that the exploration algorithm gives similar total distance for exploration, but each robot just need to travel half of the total distance.

## 3 Conclusion and Future Work

We have presented experiments that evaluate collaborative mapping and exploration algorithm for multiple MAVs. We have shown the advantages of collaborative SLAM and demonstrated autonomous exploration through both simulation and experiments. To our knowledge, this is the first experimental realization of autonomous, multiple quadrotors exploration.

There are many extensions to this work. First, we don't explicitly consider communication constraints. The advantages of a central base station for global SLAM are offset by the disadvantages of possible disruption to communication and the inability to scale to large numbers of robots and large environments. Second, we do not explicitly sense or model the possibility of collisions. While we do plan collision-free paths, it is also necessary for the robots to detect each other in order to guarantee safety, particularly in environments where aerodynamic interactions can be quite complex. In addition, 2.5-D assumption is not always held, the ideal system requires SLAM and exploration in 3D in clustered indoor environment. By utilization of 3-D sensors, we can develop this technology based on similar framework proposed in this paper. All these limitations indicate future directions for our work.

**Acknowledgments** We gratefully acknowledge support from ARL Micro Autonomous Systems and Technology Collaborative Technology Alliance Grant no. W911NF-08-2-0004.

## References

1. Shen, S., Mulgaonkar, Y., Michael, N., Kumar, V.: Multi-sensor fusion for robust autonomous flight in indoor and outdoor environments with a rotorcraft mav. In: 2014 IEEE International Conference on Robotics and automation (ICRA). IEEE (2014)
2. Howard, A.: Multi-robot simultaneous localization and mapping using particle filters. Int. J. Robot. Res. **25**(12), 1243–1256 (2006)
3. Thrun, S., Liu, Y.: Multi-robot slam with sparse extended information filers. In: Robotics Research, pp. 254–266. Springer (2005)
4. Shen, S., Michael, N., Kumar, V.: Stochastic differential equation-based exploration algorithm for autonomous indoor 3d exploration with a micro-aerial vehicle. I. J. Robot. Res. **31**(12), 1431–1444 (2012)
5. Durrant-Whyte, H., Bailey, T.: Simultaneous localization and mapping: Part I. IEEE Robot. Autom. Mag. **13**(2), 99–110 (2006)
6. Dellaert, F., Kaess, M.: Square root sam: simultaneous localization and mapping via square root information smoothing. Int. J. Robot. Res. **25**(12), 1181–1203 (2006)
7. Shen, S., Michael, N., Kumar, V.: Autonomous multi-floor indoor navigation with a computationally constrained mav. In: 2011 IEEE International Conference on Robotics and automation (ICRA), pp. 20–25. IEEE (2011)
8. Stachniss, C., Kretzschmar, H.: Pose graph compression for laser-based slam. In: International Symposium of Robotics Research (ISRR) (2011)
9. Van Der Merwe, R., Wan, E.A.: Sigma-point kalman filters for integrated navigation. In: Institute of Navigation (ION) (2004)
10. Michael, N., Mellinger, D., Lindsey, Q., Kumar, V.: The grasp multiple micro-uav testbed. IEEE Robot. Autom. Mag. **17**(3), 56–65 (2010)
11. Lee, T., Leoky, M., McClamroch, N.H.: Geometric tracking control of a quadrotor uav on se (3). In: 2010 49th IEEE Conference on Decision and Control (CDC), pp. 5420–5425. IEEE (2010)
12. Shen, S., Michael, N., Kumar, V.: Autonomous indoor 3d exploration with a micro-aerial vehicle. In: 2012 IEEE International Conference on Robotics and Automation (ICRA), May 2012, pp. 9–15
13. Bachrach, A., He, R., Roy, N.: Autonomous flight in unknown indoor environments. Int. J. Micro Air Veh. **1**(4), 217–228 (2009)
14. Kim, B., Kaess, M., Fletcher, L., Leonard, J., Bachrach, A., Roy, N., Teller, S.: Multiple relative pose graphs for robust cooperative mapping. In: 2010 IEEE International Conference on Robotics and Automation (ICRA), pp. 3185–3192. IEEE (2010)
15. Rogers, J.G., Nieto-Granda, C., Christensen, H.I.: Coordination strategies for multi-robot exploration and mapping (2012)

16. Trevor, A.J.B., Rogers III J.G., Christensen, H.I.: Omnimapper: a modular multimodal mapping framework. In: 2014 IEEE International Conference on Robotics and Automation (ICRA). IEEE (2014)
17. Dellaert, F.: Factor graphs and gtsam: a hands-on introduction (2012)
18. Cunningham, A., Paluri, M., Dellaert, F.: Ddf-sam: fully distributed slam using constrained factor graphs. In: 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 3025–3030. IEEE (2010)
19. Censi, A.: An icp variant using a point-to-line metric. In: IEEE International Conference on Robotics and Automation, ICRA 2008, pp. 19–25. IEEE (2008)
20. Kaess, M., Ranganathan, A., Dellaert, F.: isam: Incremental smoothing and mapping. IEEE Trans. Robot. **24**(6), 1365–1378 (2008)
21. Kaess, M., Johannsson, H., Roberts, R., Ila, V., Leonard, J.J., Dellaert, F.: isam2: Incremental smoothing and mapping using the bayes tree. Int. J. Robot. Res. **31**(2), 216–235 (2012)
22. Karaman, S., Frazzoli, E.: Sampling-based algorithms for optimal motion planning. Int. J. Robot. Res. **30**(7), 846–894 (2011)
23. Sucan, I., Moll, M., Kavraki, E.: The open motion planning library. IEEE Robot. Autom. Mag. **19**(4), 72–82 (2012)