Multi-Agent Mapping and Navigation of Unknown GPS-Denied Environments Using a Relative Navigation Framework

Jacob M. Olson[1], Nathan A. Toombs, Timothy W. McLain[2]

***Abstract*— When generating 3D maps with unmanned aerial vehicles (UAVs) in GPS-denied environments, it is important to correctly handle path planning, estimation, and mapping techniques. Because multirotor UAVs are limited in flight time, using multiple UAVs to map an environment collaboratively can significantly improve the mapping efficiency. This paper addresses the following key issues required to enable mapping with multiple agents: Combining a reactive path planner with an obstacle avoidance algorithm to handle navigation in complex environments. Estimating the relative and global states of a UAV separately with a relative navigation framework to allow for loop closures in the mapping process without causing the estimation to diverge. Adapting a graph-based simultaneous localization and mapping (graph-SLAM) technique for multiple UAVs flying simultaneously and merging their maps in real-time. We were able to use these strategies to generate dense maps in complex GPS-denied environments with multiple UAVs.**

I. Introduction

Many new applications are being explored to make use of dense 3D mapping technology, such as surveying an earthquake damaged building to find survivors and check for damage, infrastructure inspections, and generating as-built models for infrastructure. Mapping and navigating an environment such as these, where global positioning system (GPS) signals are degraded or entirely unavailable, are difficult. Often, these GPS-denied environments are inaccessible to ground robots due to rubble or structural damage. Environments like these lend themselves better to the use of unmanned aerial vehicles (UAVs) to carry out some or all of the mapping. When navigating indoor environments with a UAV, collision with any obstacles can be catastrophic and measures must be taken to avoid them. We use a combination of a high-level reactive path planner and a low-level obstacle avoidance filter to avoid collisions.

Most dense mapping approaches rely on high-quality GPS measurements to geotag data to enable map generation [1], [2]. These methods break down when GPS is not available because the global position must be estimated rather than measured. To overcome the lack of GPS, some form of simultaneous localization and mapping (SLAM) with visual odometry can be used. We use a form of graph-SLAM with a 3D camera and depth-enhanced visual odometry.

Because of the limited flight time of UAVs, mapping large areas with a single UAV can be inefficient. The mapping process can be streamlined by dividing the area between multiple UAVs and then, combining their individual maps into a single combined map. One recent approach to this problem by Michael et al. used ground robot and a UAV to collaboratively map an earthquake-damaged building. In this research, the UAV acted as an extension to the ground robot, flying into the difficult terrain to build onto the map started by the ground robot [3]. More recently, Mangelson et al. detailed a method to effectively merge maps collected from multiple robots acting separately after the mapping process is complete [4]. In our approach, we use multiple UAVs flying simultaneously to map an indoor GPS-denied environment. The method we propose also combines the maps from the UAVs into a single map in near real-time while the mapping process is ongoing.

The remainder of the paper is organized as follows: Section II presents the framework we use to navigate and map an environment along with background information about previous work that we build upon. Section III details the planning and control schemes used to successfully navigate an unknown environment. The method used to combine maps in near real-time is then explained in Section IV. Results showing and evaluating the generated maps are presented in Section V. Finally, conclusions are presented in Section VI.

II. Technical Approach

A. Problem Statement

For UAV-based map building to be successful, flight paths that produce high-quality loop closures and sufficient coverage of the environment are required. The framework presented in this section assumes that these high-quality paths will be supplied either by the user or by a high-level coverage path planner. Rather than focus on high-level path generation, the focus of this paper is first, to show that properly estimating UAV states allows for successful GPS-denied navigation, and second, to demonstrate how to streamline the mapping process by merging multiple maps into a single one. A high-level network diagram is shown in Fig. 1 that outlines the framework we use to successfully generate a single merged map from multiple UAVs in a GPS denied environment.

B. Sensors

Since we are operating in a GPS-denied environment, we are not able to rely on GPS measurements to give us global information about the UAVs locations. The sensors used by each UAV to estimate their states are a 3D color and depth (RGB-D) camera (Intel RealSense D435 [5]), a planar light detection and ranging (LiDAR) laser scanner, a single-beam LiDAR range finder, and an inertial measurement unit (IMU)

[1]The corresponding author can be contacted at jacobmo at byu.edu.

[2]All authors are with the Department of Mechanical Engineering, Brigham Young University, Provo, UT, 84602, USA.

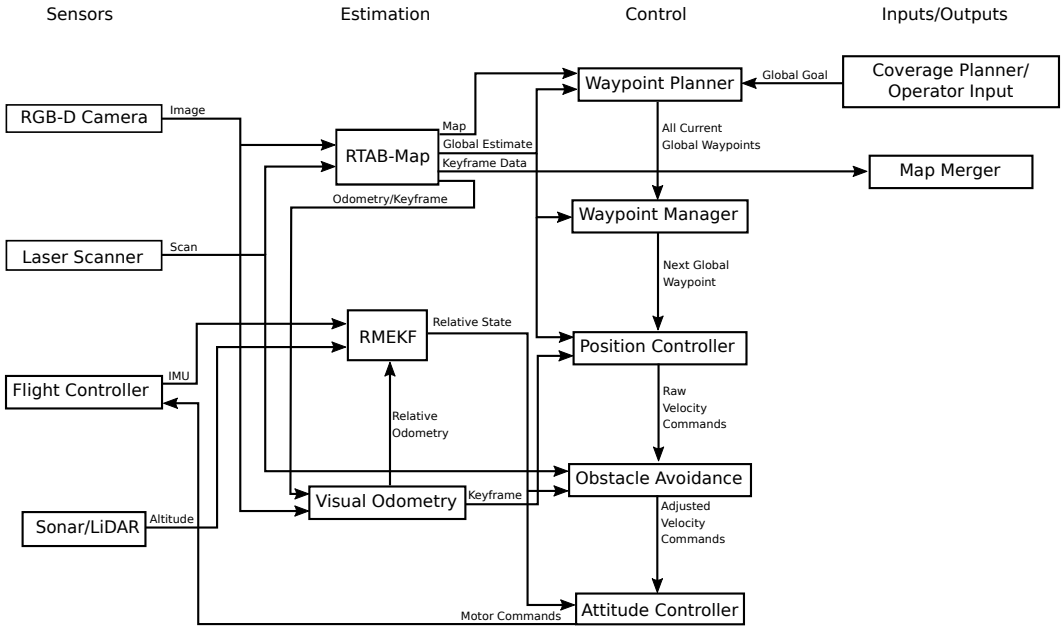

Fig. 1. The network diagram of the relative navigation framework proposed in this paper.

on the onboard flight controller. Using only these sensors and the flight computer, we estimate the states of the UAV with sufficient accuracy to control its attitude and position. The following section elaborates on how these sensors are used in the estimation.

C. Estimation

Estimation is the most critical element in enabling autonomous flight. Without adequate position and attitude estimation, autonomous navigation algorithms have difficulty. We use a graph-SLAM approach similar to that developed by Thrun et al. [6] to navigate and generate the maps. Loop closures, which occur when a robot sees the same objects from similar locations at different points in time, can cause estimators to diverge if they are not handled correctly. Every time a new loop closure occurs, the map and position estimate are re-optimized. If the new loop closure results in a large shift in the current position estimate, a naive estimator can diverge because of the discontinuity of the estimated global position. To avoid the issue of loop closures causing instability in the controller, we estimate the global and relative states of the UAV separately and do not rely on the global state estimate to control the attitude of the UAV.

We store the current global state estimates in a transformation tree as shown in Fig. 2. The $world$ frame is the inertial NED (north-east-down) frame with a fixed origin that does not change over time. The UAV's starting location with respect to the world is set as the map frame which is represented in the inertial NWU (north-west-up) orientation. The $base\_link$ frame represents the current estimated position of the UAV in the NED orientation with the $camera\_link$ frame representing the position in the NWU orientation. The $camera\_base\_link$ frame represents the current position of the camera in the camera frame.

The $odom$ frame is used to adjust for loop closures. When the flight begins, the $odom$ frame starts with zero offset from the map frame. Every time a loop closure is detected and the map is re-optimized, the transform between the map and $odom$ frames is adjusted to reflect the correction in the current global state estimate. This allows the transform between the $odom$ frame and the robot frames ($camera\_link$ and $base\_link$) to stay continuous when loop closures are detected even though the position estimate does not.

The $keyframe$, $keyframe\_world$ and $keyframe\_camera$ frames are used to track the relative visual odometry used by the relative estimator. More specifically, the relative visual odometry is stored in the tree as the transform between the $keyframe$ and $camera\_link$ frames in the NWU orientation. For convenience, we also keep track of the NED orientation with the $keyframe\_world$ frame and the camera orientation with the $keyframe\_camera$ frame.

The UAV's current waypoint is represented as the transform between the $world$ and $waypoint$ frames. This way, a loop closure does not shift the desired global position of the waypoint. The position controller operates on the error between the $waypoint$ and $base\_link$ frames. More detail regarding the uses and implementations of these transforms will be further explained in the following sections.

1) RTAB-Map: The real-time appearance-based mapping (RTAB-Map) package, developed by Labbe et al. [7], [8], [9], is a powerful open-source software library. It uses graph-SLAM with appearance-based loop closures to generate high-quality, dense 3D maps using only an RGB-D camera and a planar laser scanner. When coupled with the depth-enhanced visual odometry algorithm called RGBD-Odometry, which was developed in conjunction with RTAB-

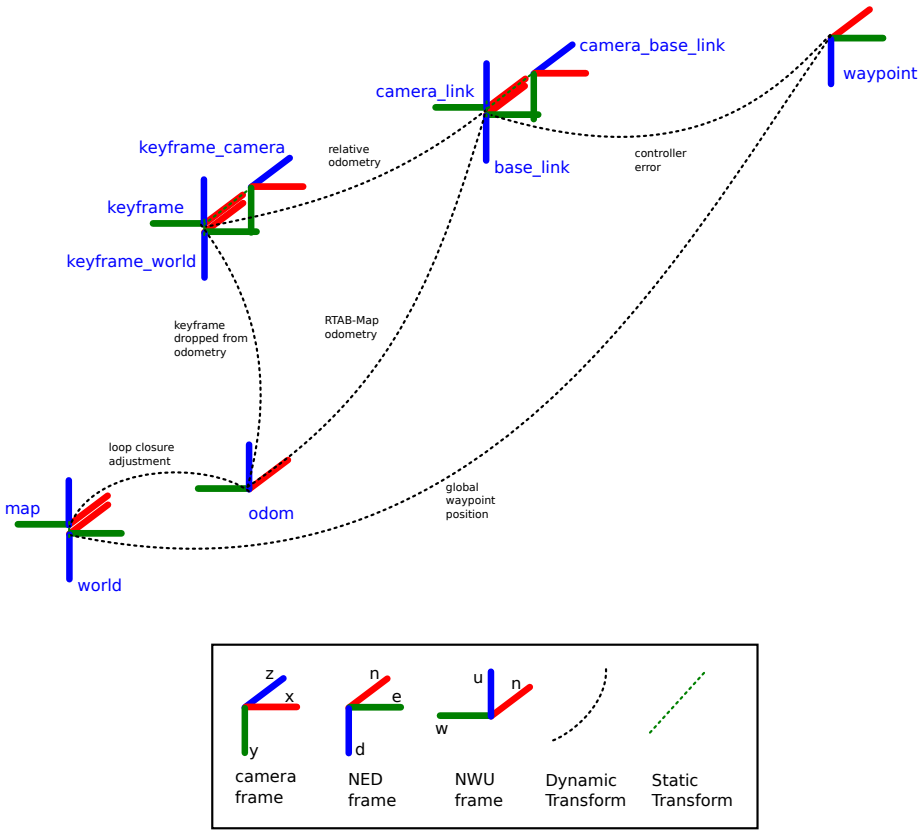

Fig. 2. The transformation tree of the reference frames used in estimation and control.

Map [9], the UAV's position with respect to the map that is being built can be accurately estimated at all times. The map-building algorithm uses a keyframe-based approach. Unlike the visual odometry algorithm, the map-building algorithm does not try to use the information from every camera frame to build the map and optimize the graph. Instead, it only uses camera information that has been saved periodically at a set rate. In our case, once every second the information from the current camera frame is sent to RTAB-Map and saved as a keyframe. Only these keyframes are used to build the map and optimize the graph. RTAB-Map is primarily designed for use with slow-moving ground robots that do not need high-rate state estimation to work. The only state estimation that is performed by RTAB-Map is from the visual odometry, which is limited by the frame rate of the camera. We found the estimation rate of RTAB-Map on our hardware to be in the 10-30 Hz range, which is not sufficient for autonomous navigation of a UAV. We use the state estimates from the RGBD-Odometry node as an input to the estimation for the relative framework.

Although the current functionality of RTAB-Map does allow for a single robot to combine maps from multiple sessions, it does not allow for multiple agents mapping simultaneously to combine the maps into a single one. This paper proposes a method to extend the functionality of RTAB-Map to combine the maps of multiple agents flying simultaneously into a single map in near real-time. The implementation of this method is detailed in Section IV.

RTAB-Map manages the map, $odom$, $base\_link$, $camera\_link$, and $camera\_base\_link$ frames in the transformation tree as previously described. Along with the frames, it handles all of the loop closures and graph optimization. We use the current position estimate produced by RTAB-Map for the position controller and waypoint manager. Because of the inevitable inaccuracies and the lower estimation rates of RTAB-Map, we do not use these estimates to perform attitude control on the UAVs. Rather, we use a relative navigation framework to estimate the relative state and control the attitude.

2) Visual Odometry: The odometry and keyframe information generated by RTAB-Map is used to produce a relative odometry message that is sent to the relative estimator. RTAB-Map produces a global visual odometry that provides a real-time estimate of the global position and orientation of the UAV. Each time a new keyframe is declared, a new node is added to the graph, and the visual odometry node shown in Fig 1 resets the transform between the $keyframe$ and $camera\_link$ frames to zero. The relative odometry only tracks the UAV's movement with respect to the last keyframe that was declared and stores the information as the transform between the $keyframe$ and $camera\_link$ frames. Because of the continual resetting of the keyframe transform, the

odometry used by the relative estimator is less susceptible to drift over time [10].

3) RMEKF: The relative navigation framework [11], [12], [13] has been shown to successfully estimate the UAV's relative state sufficient to autonomously navigate in GPS-denied environments that have previously been mapped. Thus far, however, it has not been extended to estimation and navigation in unknown and unmapped environments. This paper proposes a method to extend the functionality to these environments.

The relative multiplicative extended Kalman filter (RMEKF) is the heart of the relative framework. It uses the IMU measurements from the flight controller, the relative visual odometry explained previously, and the attitude measurement from the LiDAR single-beam range finder for the measurement updates. Then, using a multirotor dynamics model, the RMEKF accurately estimates the relative state of the UAV with respect to the previous keyframe. This estimate is used for obstacle avoidance and high-rate attitude control. The RMEKF makes no effort to estimate the global position of the UAV. As a result, corrections in the estimated global position from loop closures and drift in visual odometry do not cause the estimator to diverge. This enables the UAV to avoid stability issues in the velocity and attitude controllers.

D. Control

To successfully control a UAV using the relative navigation framework, the control must be segmented into different tiers to take advantage of both global and relative estimates. The following paragraphs explain the different tiers of the UAV control shown in Fig 1.

1) Waypoint Planner: The first stage of the control architecture is the waypoint planner. The inputs to the planner are the global goal, the current known obstacles, and the current global estimates. It outputs a path to the goal that avoids all known obstacles as set of waypoints. The detailed workings of the planner will be further explained in Section III.

2) Waypoint Manager: After receiving the waypoints, the waypoint manager selects the appropriate waypoint for the UAV and sends the global location to the position controller. The waypoint manager monitors the position and heading error between the current estimated position and the current waypoint. When the error decreases below a user-defined threshold value, the next waypoint is sent to the position controller.

3) Position Controller: The position controller drives the error between the current estimated position and heading of the UAV and the next waypoint to zero. Since it operates in the error space of the UAV rather than the state space, sudden shifts in the UAV's position estimate caused by loop closures have minimal effect on the controller. When these shifts happen, the controller is able to quickly adjust to continue controlling the error to zero. The position controller outputs a velocity command.

4) Obstacle Avoidance: Before passing the velocity command into the attitude controller, it is filtered through an obstacle avoidance algorithm. This algorithm uses the current relative estimates from the RMEKF and current obstacles detected by the planar laser scanner to modify the input velocity command. It uses a cushioned extended-periphery avoidance (CEPA) technique [14] to alter the velocity command when necessary by pushing the UAV away from obstacles while changing the incoming velocity command as little as possible. Because the position controller does not manage the trajectory of the UAV, it often causes overshoot after reaching waypoints. This overshoot is suppressed by the CEPA filter if it causes the UAV to fly near obstacles. The obstacle avoidance node then sends the modified velocity command to the attitude controller.

5) Attitude Controller: The attitude controller is a conventional PID controller that runs on the autopilot. It takes the linear velocity from the position controller as the input commands, performs the attitude control loop, and outputs the raw motor commands to the electronic speed controllers (ESCs).

E. Inputs/Outputs

The input for each agent in the system is the desired high-level path either from operator input or from a high-level path planner. As each agent maps the environment, keyframe data consisting of the color and depth images and the feature descriptions from the images is sent to the map merger each time a new keyframe is declared. The map merger will be further explored in Section IV.

III. Planning

When designing the reactive planner, we explored using both node-based optimal algorithms and sampling based algorithms [15]. Node-based (or heuristic search) algorithms like A* work well to find optimal paths around obstacles [16]. The downside to these planners is that in order to find a path, they need to exhaustively search the design area. This makes them less efficient to use when the map is large and constantly changing. They also require the environment to be discretized, which can result in the planner not finding paths to the goal when a path does exist. This tends to happen in complex maps when the environment is coarsely discretized. Sampling-based algorithms, like rapidly-exploring random trees (RRT) [17] are effective for planning in real-time with dynamic obstacles. RRT randomly searches the full, non-discretized map for feasible paths rather than requiring an exhaustive search. Although RRT does not guarantee optimal paths, it is much less computationally intensive than A* each time a new path is planned. More recently, improvements to RRT have been explored such as RRT* [18], which generates asymptotically optimal paths by modifying the search tree. Since our planner is used as a form of exploration of unknown environments, we chose to use RRT. This allows the flight path to be more random and encourage more exploration of the map while flying paths.

We used a form of RRT with a path smoothing approach similar to that proposed [19] to improve and simplify the resultant paths. Since our planner uses a dense 2D grid map of all currently known obstacles, some adjustments are

needed to efficiently plan and re-plan when new obstacles are discovered in the current path.

A. Global Goal Following with Relative Estimation

As mentioned earlier, the relative estimator critical to keeping the UAV airborne only estimates the UAV's relative state with respect to the last keyframe and makes no attempt to estimate the global position of the UAV. By doing so, the global estimate of the position does not have to be continuous and is able to slide and adjust with loop closure corrections without affecting the estimator. The path planner generates a global path from the current position to the goal. The global paths do not adjust with loop closures and the map is continually changing as the UAV flies. For both of these reasons, obstacles can appear in the path at any time. To avoid these obstacles, the planner dynamically re-plans any time an obstacle is detected in the path.

B. Reactive Path Planning

Fig. 3 shows the process of how the dynamic path planner works in an example scenario. When the UAV starts, little is known about the environment. The only obstacles in the map are the ones that are within line-of-sight of the laser scanner when the flight begins. A path is planned to the goal that avoids the obstacles that are initially detected. As the UAV flies, the obstacle map is continuously updated with new obstacles and the path is constantly being checked for collisions. If collisions are found, a new path is planned to the goal from the current location that avoids the newly discovered obstacles. This process continues until the agent successfully reaches the goal. Since the path is updated any time a potential collision is detected, no prior knowledge of the environment is required to begin flying.

The planner also includes a buffer around each detected obstacle to avoid planning paths that would cause the UAV to fly too close to the obstacles. The buffer size is set by the user, but is always at least half the width of the UAV. This ensures that every waypoint is sufficiently far from obstacles that it can be reached without any part of the UAV touching an obstacle.

Since this planner is used in conjunction with the CEPA obstacle avoidance node explained in II-D.4, the UAV can effectively navigate through complex maps and avoid obstacles in emergency situations. The planner enables complex navigation that would not be possible with only obstacle avoidance. The obstacle avoidance node filters velocity commands from the position controller that would inadvertently cause the UAV to collide with obstacles when either a new obstacle is detected that is not in the current map, or when the position controller causes overshoot.

1) Efficient Collision Detection: Most implementations of RRT are not designed to deal with extremely large number of obstacles. The obstacles we use are from a 2D grid map generated by RTAB-Map based on the planar LiDAR scanner returns. Consequently, rather than having just a few large obstacles, there are many small obstacles. Using a standard obstacle detection check with each propagation of the RRT would be extremely inefficient. To maximize the efficiency during planning, we use a strategy that allows the planner to ignore most obstacles in the map when performing collision checks. An example of how this works is illustrated in Fig. 4.

As the RRT branches propagate, before each candidate node is added to the tree, an obstacle collision check is done only on the obstacles within range of the new node. The obstacles are determined to be in range according to algorithm 1.

Algorithm 1 Obstacle Range Filter

Require: $candidate\_branch$, $obstacle\_list$, r_{buf}
  for all $obstacles$ in $obstacle\_list$ **do**
    if $x_{obs} + r_{buf} \leq \min(x_1, x_2)$ **then**
      continue
    else if $x_{obs} - r_{buf} \geq \max(x_1, x_2)$ **then**
      continue
    else if $y_{obs} + r_{buf} \leq \min(y_1, y_2)$ **then**
      continue
    else if $y_{obs} - r_{buf} \geq \max(y_1, y_2)$ **then**
      continue
    else
      $filtered\_obstacles \leftarrow obstacle$
    end if
  end for
  return $filtered\_obstacles$

The points (x_1, y_1) and (x_2, y_2) are the endpoints of the candidate branch, (x_{obs}, y_{obs}) is the location of the obstacle being checked, and r_{buf} is the buffer radius for each obstacle. The green bounding box in Fig. 4 shows which obstacles would be included in the $filtered\_obstacles$ after this check. The perpendicular distance between the candidate line (shown in red) and each remaining obstacle is found by

$$d = \frac{|\Delta y\ x_{obs} - \Delta x\ y_{obs} + \Delta s|}{\sqrt{\Delta y^2 + \Delta x^2}}, \tag{1}$$

where

$$\Delta x = x_2 - x_1 \tag{2}$$

$$\Delta y = y_2 - y_1 \tag{3}$$

$$\Delta s = x_2\ y_1 - x_1\ y_2. \tag{4}$$

If the distance d is less than the buffer radius, a collision is detected and the candidate is rejected. By only checking for collisions with obstacles within the bounding box, nearly all obstacles are ignored for each step of propagation. This significantly improves performance of the RRT planner and allows it to plan in real-time and dynamically update the path whenever needed. The collision detection for path smoothing and path checking works the same way, with (x_1, y_1) and (x_2, y_2) being the endpoints of each path segment.

IV. Map Merging

The map merging process we use is able to generate a combined map from multiple agents on a base-station computer in near real-time as the UAVs are mapping the

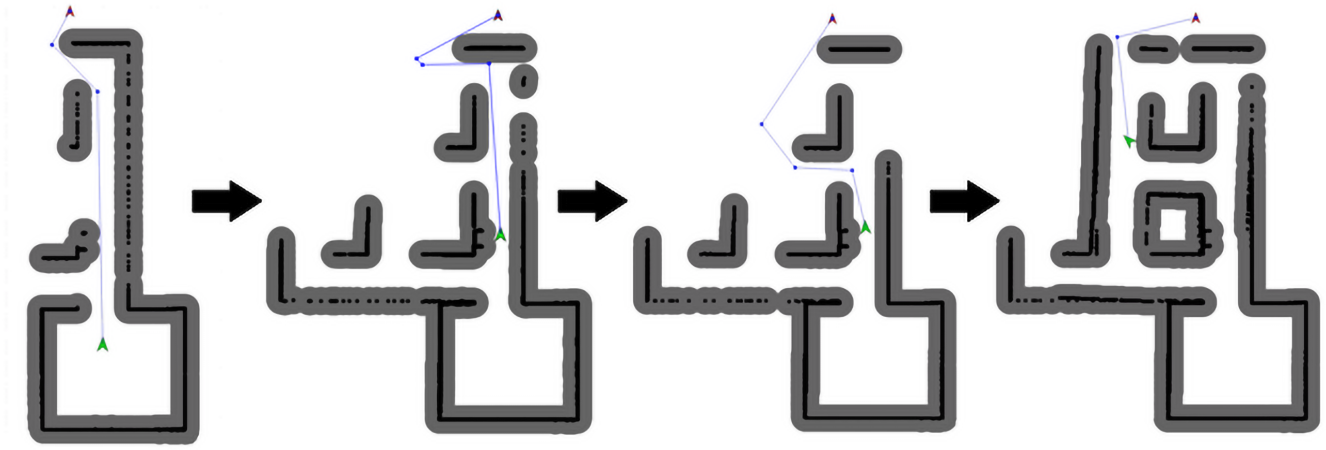

Fig. 3. An example of how the reactive path planner works as the UAV flies the planned path. The current estimated position is marked by the green arrow, the current goal position is marked by the red arrow, and the current path is marked with blue lines. Detected obstacles and safety buffers are represented with black and grey respectively.

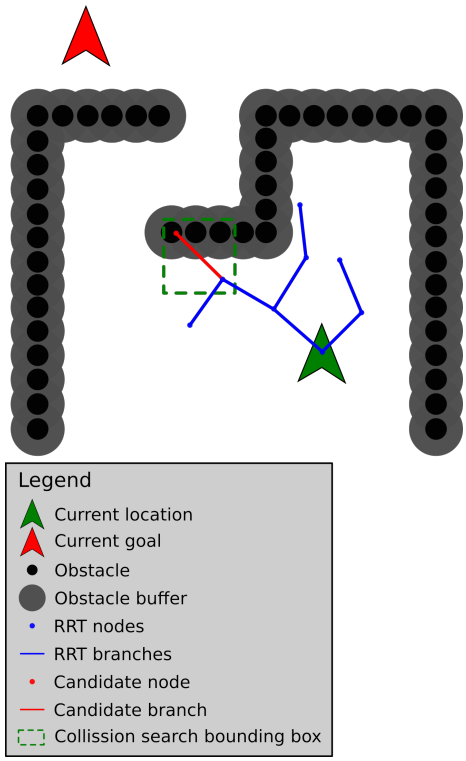

Fig. 4. Example of one step of collision check with the RRT planner.

environment. Fig. 5 shows the network diagram of the process of merging the maps. To merge the maps in near real-time, each time a new keyframe is initialized, its data is stored in the RGB-D Cache database which is hosted on the base-station computer. A 3D color and depth (XYZRGB) pointcloud is generated using the color and depth information from the RGB-D cameras and the features are generated from either SIFT/SURF or ORB using OpenCV.

Once the database has been initialized, the maps are periodically merged using functions from an instance of RTAB-Map running on the base-station computer. This merging process functions similarly to how individual maps are generated for each UAV [7], [8], [9]. The first step is to search for loop closures in the feature descriptions from each keyframe using a bag-of-words approach. Rather than only look for loop closures from the keyframes of a single agent, this process looks for loop closures from all keyframes from all agents. Each time a new loop closure is found, a new edge is added to combined map graph with an estimated transformation between keyframes. After finding all loop closures with the current dataset, the graph is optimized using the pose graph optimizer built into RTAB-Map. The optimized pose graph is then sent to the map assembler along with the XYZRGB pointcloud from each keyframe where the pointclouds are combined according to the optimized graph edges. This generates a single map with all keyframes that can be connected into a single graph. This map is then processed to reduce noise and filter out the ceiling to make the map more understandable to the operator.

Since the base-station computer is merging the maps, the algorithm is able to run in real-time. The larger the map grows, the longer it takes to re-optimize the combined map. This causes the updated map to lag behind real-time when the map is large.

V. Results and Discussion

A. Simulation

We successfully navigated and mapped a simulation environment autonomously in ROS Gazebo [20] with multiple agents and combined the maps. The simulation environment was designed to be as close to the real world and hardware as possible. We used a software-in-the-loop (SIL) version of ROSflight [21], an open-source autopilot library built with ROS, to mimic the flight controller. We did not use any ground-truth information in the simulation; all measurements were from simulated sensors with noise characteristics similar to hardware. Fig. 6 shows the simulation setup used to

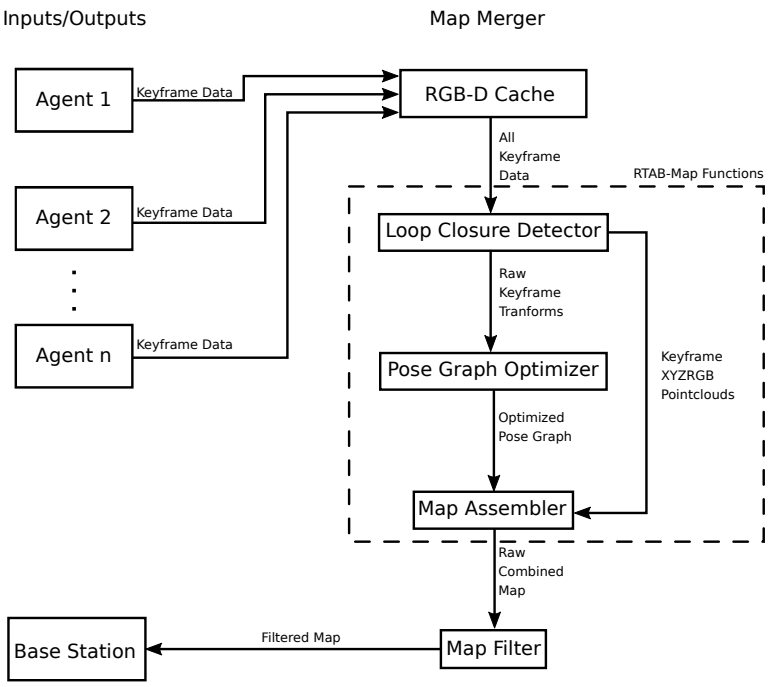

Fig. 5. The network diagram for the multi-agent map merging node proposed in this section.

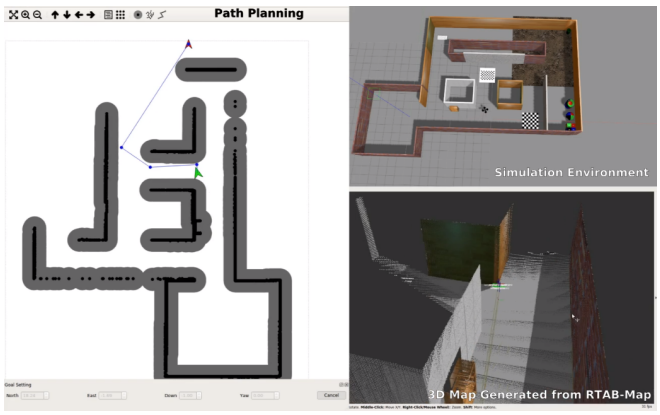

Fig. 6. Setup used for the simulation results. The reactive planner is shown on the left, the simulation world is shown on the top right, and the current map is shown on the bottom right.

map the environment.

Fig. 7 shows the results from mapping the simulated environment with two agents and combining the maps. Neither agent saw everything in the combined map, but the maps were successfully merged together into a single map with all features from each individual map.

Flying in simulation helped validate the reactive planner and obstacle avoidance. It was also helpful to debug and sort out the communication architecture of the relative navigation framework and control schemes. The area where the simulation falls short is with the computer vision applications such as visual odometry and loop-closure detection. Gazebo is excellent at simulating realistic physics and dynamics, but the environments are significantly less detailed than the real world. This makes designing a simulation world more difficult. If there is too little detail added to the world, the visual odometry algorithms often fail or perform poorly. If too much repetitive detail is used, RTAB-Map finds too many false loop closures and the mapping fails. The simulated world developed and used to obtain results as shown in Figs. 6 and 7 is able to minimize these issues, but still failed to produce results on par with a real world test. There were only a few locations in the simulated world with enough unique detail that loop closures were reliably detected. In contrast, a real-world environment has enough detail that loop closures can be detected nearly everywhere except for large blank walls and glass. After proving the setup in a high-fidelity simulation, we moved to hardware to more thoroughly test the computer vision aspects of the approach

B. Hardware

The extension of RTAB-Map to enable real-time map merging of multiple UAVs flying simultaneously is robust enough to handle several vehicles at the same time. We began testing with only merging two flight paths and were able to expand it up to four simultaneous flight paths. We were able to successfully combine maps generated from multiple UAVs in near real time with manual flight. Fig. 8 shows an example of a map built from an indoor environment with both hallways and large rooms. The generated map is sufficiently dense to show detail for a human user to interpret and get actionable information. This map was built from manually flying a single UAV four times and recording the data, then playing back all data simultaneously and merging the map in real time as shown in Fig. 9. By the end of the map merging process with this data, the combined map generation

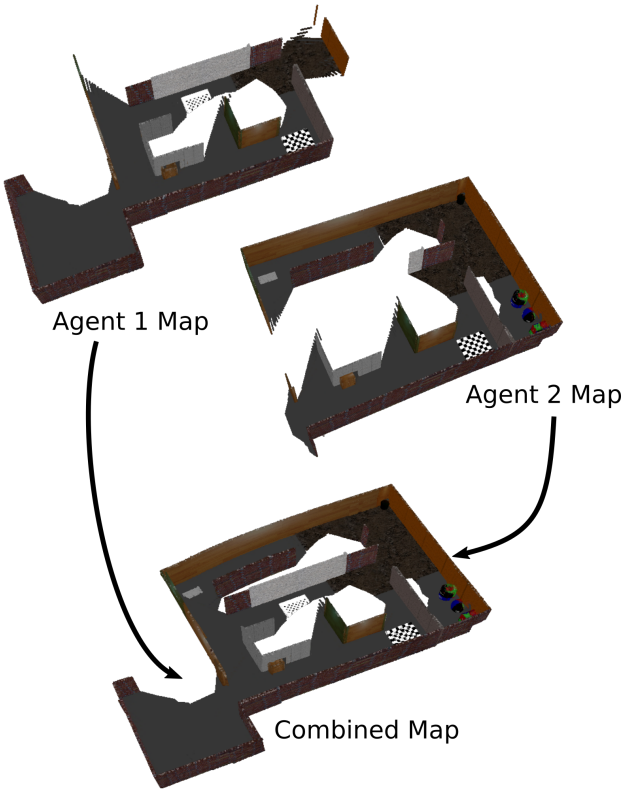

Fig. 7. Map generated from combined maps in the simulated environment.

was lagging behind real-time by about one minute. The map information is stored in the random-access memory (RAM), this map uses approximately 3GB of memory. The size of the area mapped is approximately 40 meters wide and 60 meters long, with each of the four flights lasting anywhere from three to five minutes. There is theoretically no limit to the number of vehicles that could be added and flown simultaneously, but as more vehicles are added, the merging process lags further behind real time, more memory is needed, and loop closure detection gets more complex.

VI. Conclusions

Using UAVs to generate dense 3D maps of GPS-denied environments requires a careful choice of planning, estimation, and mapping techniques to be successful. Using the combination of a reactive path planner and a CEPA obstacle avoidance velocity filter allows for navigation and exploration through complex GPS-denied environments. Estimating relative and global states separately allows for the necessary decoupling of position and attitude controllers to fly autonomously without the use of GPS. Using multiple UAVs to collaboratively map an area improves mapping efficiency and, when handled correctly, still allows the map building to occur in near real-time. Future work includes streamlining the map merging process to allow for full real-time map generation, and connecting the UAVs to a high-level coverage path planner to allow for fully autonomous flight without human guidance.

VII. Acknowledgements

Thanks to Mathieu Labbe for being responsive to answering questions on the RTAB-Map forum and helping with developing the map merging node. This research was funded by The National Institute of Standards and Technology (NIST) under award number 70NANB17H211.

References

[1] S. Siebert and J. Teizer, "Mobile 3D mapping for surveying earthwork projects using an unmanned aerial vehicle (UAV) system," *Automation in construction*, vol. 41, pp. 1–14, 2014.

[2] R. Martin, I. Rojas, K. Franke, and J. Hedengren, "Evolutionary view planning for optimized UAV terrain modeling in a simulated environment," *Remote Sensing*, vol. 8, no. 1, p. 26, 2016.

[3] N. Michael, S. Shen, K. Mohta, Y. Mulgaonkar, V. Kumar, K. Nagatani, Y. Okada, S. Kiribayashi, K. Otake, K. Yoshida, *et al.*, "Collaborative mapping of an earthquake-damaged building via ground and aerial robots," *Journal of Field Robotics*, vol. 29, no. 5, pp. 832–841, 2012.

[4] J. G. Mangelson, D. Dominic, R. M. Eustice, and R. Vasudevan, "Pairwise consistent measurement set maximization for robust multi-robot map merging," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 2916–2923.

[5] Intel, "Intel® RealSense depth camera D435 - Intel® RealSense depth cameras." [Online]. Available: https://click.intel.com/intelr-realsensetm-depth-camera-d435.html

[6] S. Thrun and M. Montemerlo, "The graph SLAM algorithm with applications to large-scale mapping of urban structures," *The International Journal of Robotics Research*, vol. 25, no. 5-6, pp. 403–429, 2006.

[7] M. Labbé and F. Michaud, "Memory management for real-time appearance-based loop closure detection," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2011, pp. 1271–1276.

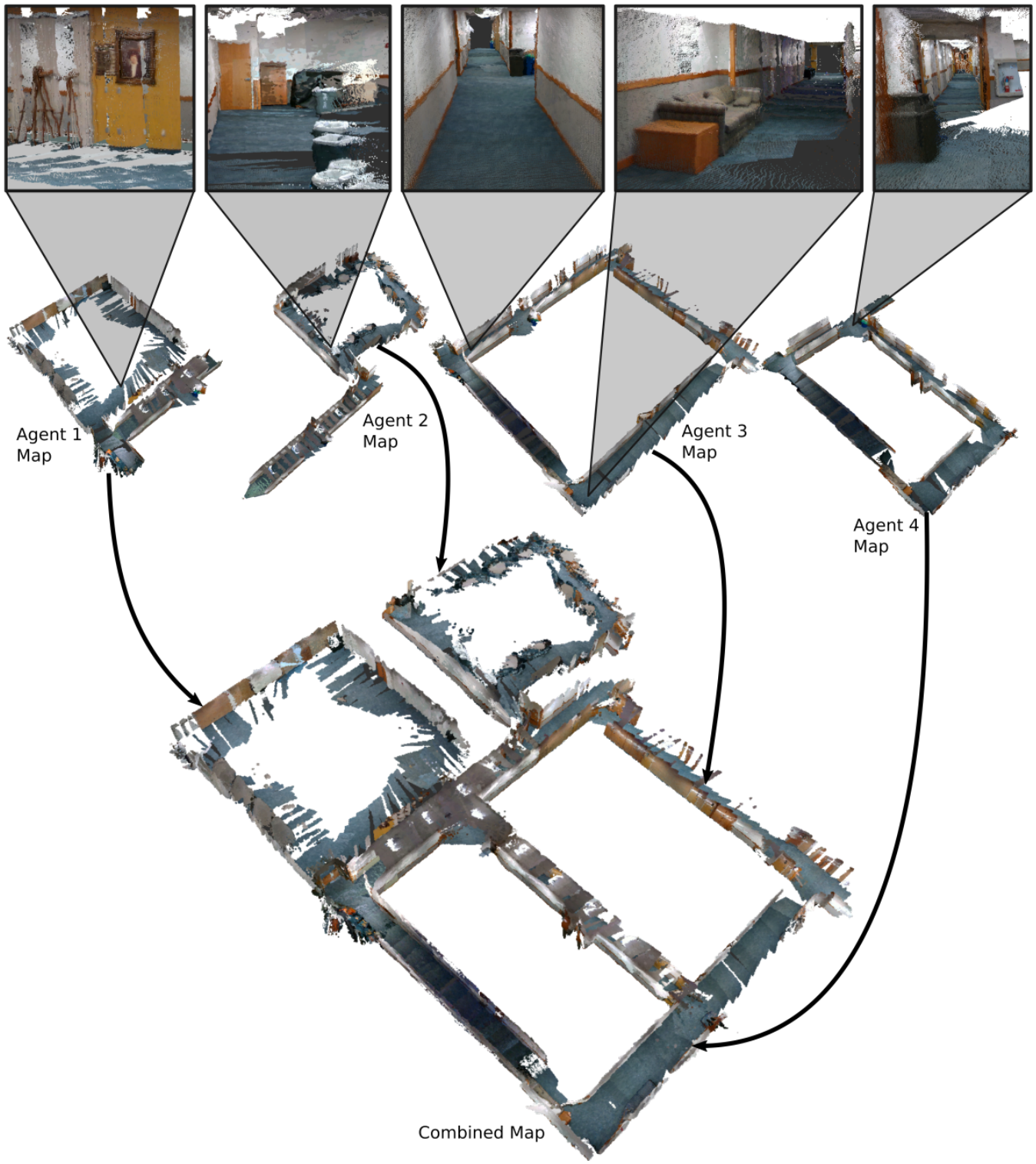

Fig. 8. Example of hardware results of merging maps from four agents into a single map in an indoor environment. Above the individual agent maps are examples of the detail in the pointclouds when zoomed in.

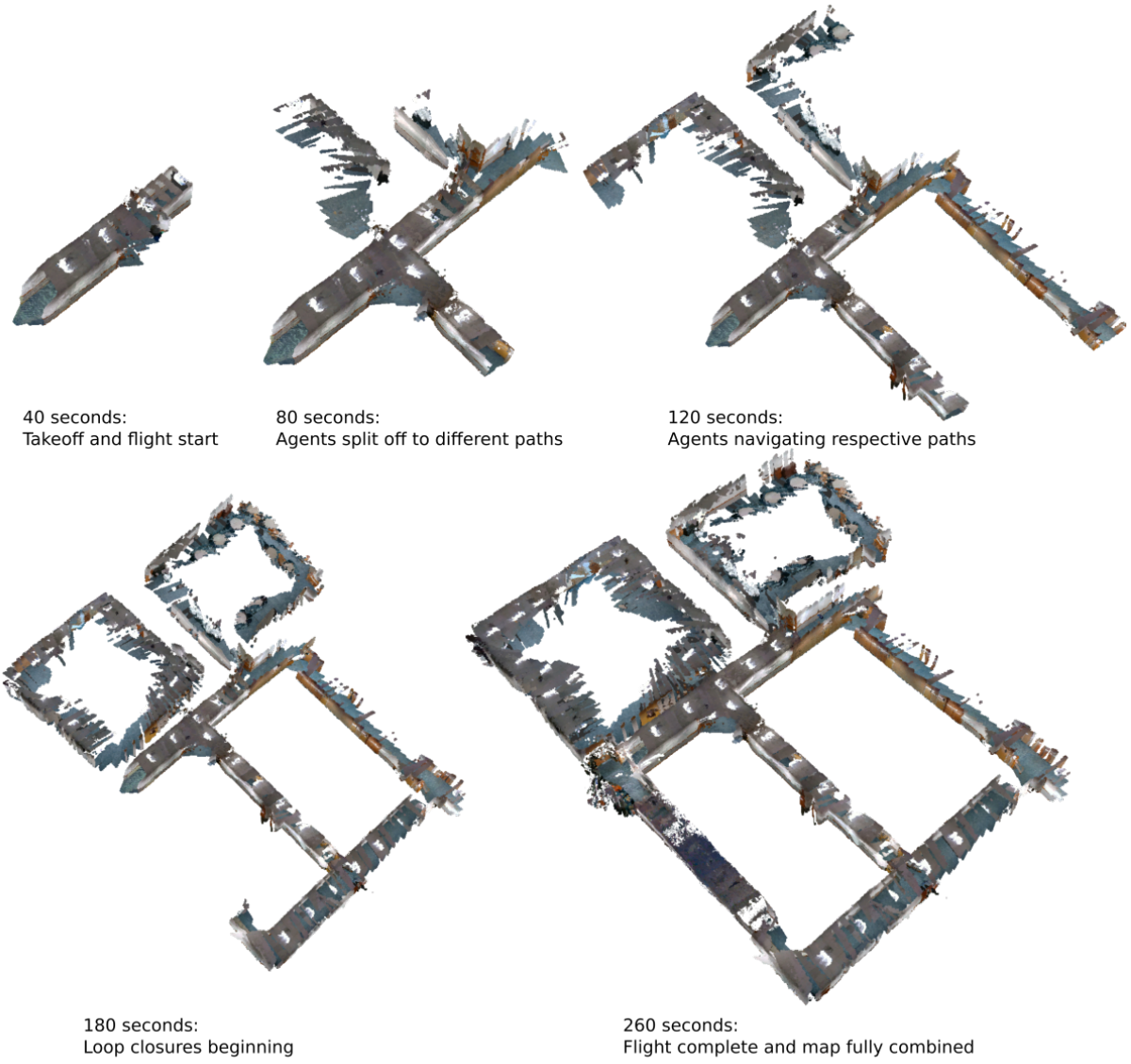

Fig. 9. Example of the merged map being generated in near real-time. The combined map is updated every 30-40 seconds as agents are flying.

[8] ——, "Appearance-based loop closure detection for online large-scale and long-term operation," *IEEE Transactions on Robotics*, vol. 29, no. 3, pp. 734–745, 2013.

[9] ——, "RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation," *Journal of Field Robotics*, vol. 36, no. 2, pp. 416–446, 2019.

[10] R. C. Leishman, T. W. McLain, and R. W. Beard, "Relative navigation approach for vision-based aerial GPS-denied navigation," *Journal of Intelligent & Robotic Systems*, vol. 74, no. 1-2, pp. 97–111, 2014.

[11] D. O. Wheeler, D. P. Koch, J. S. Jackson, G. J. Ellingson, P. W. Nyholm, T. W. McLain, and R. W. Beard, "Relative navigation of autonomous GPS-degraded micro air vehicles," *All Faculty Publications*, 2017.

[12] D. O. Wheeler, D. P. Koch, J. S. Jackson, T. W. McLain, and R. W. Beard, "Relative navigation: A keyframe-based approach for observable GPS-degraded navigation," *IEEE Control Systems Magazine*, vol. 38, no. 4, pp. 30–48, 2018.

[13] D. P. Koch, D. O. Wheeler, R. Beard, T. McLain, and K. M. Brink, "Relative multiplicative extended Kalman filter for observable GPS-denied navigation," 2017.

[14] J. Jackson, D. Wheeler, and T. McLain, "Cushioned extended-periphery avoidance: A reactive obstacle avoidance plugin," in *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2016, pp. 399–405.

[15] L. Yang, J. Qi, D. Song, J. Xiao, J. Han, and Y. Xia, "Survey of robot 3D path planning algorithms," *Journal of Control Science and Engineering*, vol. 2016, p. 5, 2016.

[16] N. J. Nilsson, *The quest for artificial intelligence: A history of ideas and achievements*. Cambridge, MA: Cambridge University Press, 2009.

[17] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Tech. Rep., 1998.

[18] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[19] R. W. Beard and T. W. McLain, *Small unmanned aircraft: Theory and practice*. Princeton university press, 2012.

[20] OSRF, "Gazebo." [Online]. Available: http://gazebosim.org/

[21] J. Jackson, G. Ellingson, and T. McLain, "ROSflight: A lightweight, inexpensive MAV research and development tool," in *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2016, pp. 758–762.