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

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

***Abstract*— [TODO: write the abstract]**

I. Introduction

[COMMENT: Add more motivation here? like search and rescue, need for dense maps?] Mapping and navigating an environment where GPS (global positioning system) signals are degraded or entirely unavailable such as an earthquake damaged building is not a trivial task. Often these GPS-denied environments are inaccessible to ground robots due to difficult terrain which lends itself better to the use use of UAVs (unmanned aerial vehicles) to carry out some or all of the mapping. When navigating indoor environments with a UAV, collision with any obstacles can be catastrophic so measures must be taken to avoid any collisions. We use a combination of a high-level reactive path planner and a low-level obstacle avoidance filter to avoid obstacles.

Most dense mapping approaches rely on high quality GPS data to patch together data to generate a map [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 must be used.

Because of the limited flight time of UAVs, the mapping process can be streamlined by using multiple UAVs collaborating by dividing up the environment between them then combining all of the maps into a single one. One recent approach to this problem by Micheal et al. was to use a ground robot and a UAV to collaboratively map an earthquake damaged building where the UAV acted as an extension to the ground robot, flying into the hard-to-reach areas 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 [4]. In our approach to this, 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.

The remainder of the paper is organized as follows: Section II describes the framework used to navigate and map the environment, and background on what previous work has made this research possible. Section III details the planning and control schemes used to successfully navigate the unknown area. 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

[COMMENT: move some of this up into the introduction?] The goal of this paper is to show how to successfully navigate and map a GPS-denied environment using multiple UAVs collaborating with each other. For map building to be successful, a flight path that produces high quality loop closures and good coverage of its environment is required. The framework presented in this section assumes that a high quality path will be supplied either by the user or by a high-level coverage path planner. The focus of this paper is first, to show that properly estimating UAV states allows for successful GPS-denied navigation, and secondly, to demonstrate how to streamline the mapping process by merging multiple maps into a single map. A high-level network diagram is shown in Fig. 1 which outlines the framework used in this paper to successfully generate a single merged map from multiple UAVs in a GPS denied environment. Each section of the diagram will be described in detail throughout the paper.

B. Sensors

[COMMENT: talk more about sensors here?] Since we are operating in a GPS-denied environment, we are not able to rely on GPS measurements to give us global information about where the UAVs are located. The sensors used by the UAV to estimate its state are a 3D color and depth (RGB-D) camera, a planar light detection and ranging (LiDAR) laser scanner, a single-beam LiDAR range finder and an inertial measurement unit (IMU) on the onboard flight controller. Using only these sensors and the flight computer, we must be able to accurately estimate the states of the UAV enough control its attitude and position. We talk about how these sensors are used to accurately estimate the UAVs state in the next section.

C. Estimation

Estimation is the most critical element in enabling autonomous flight. Without good position and attitude estimation, autonomous navigation algorithms cease to function. We use a graph-SLAM approach based from that developed by Thrun et al. [5] to navigate and generate the maps. When using a graph-SLAM approach, loop closures can cause problems with the estimation if it not done correctly. Every time the UAV sees the same objects from a similar location as before, a new loop closure is detected and 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

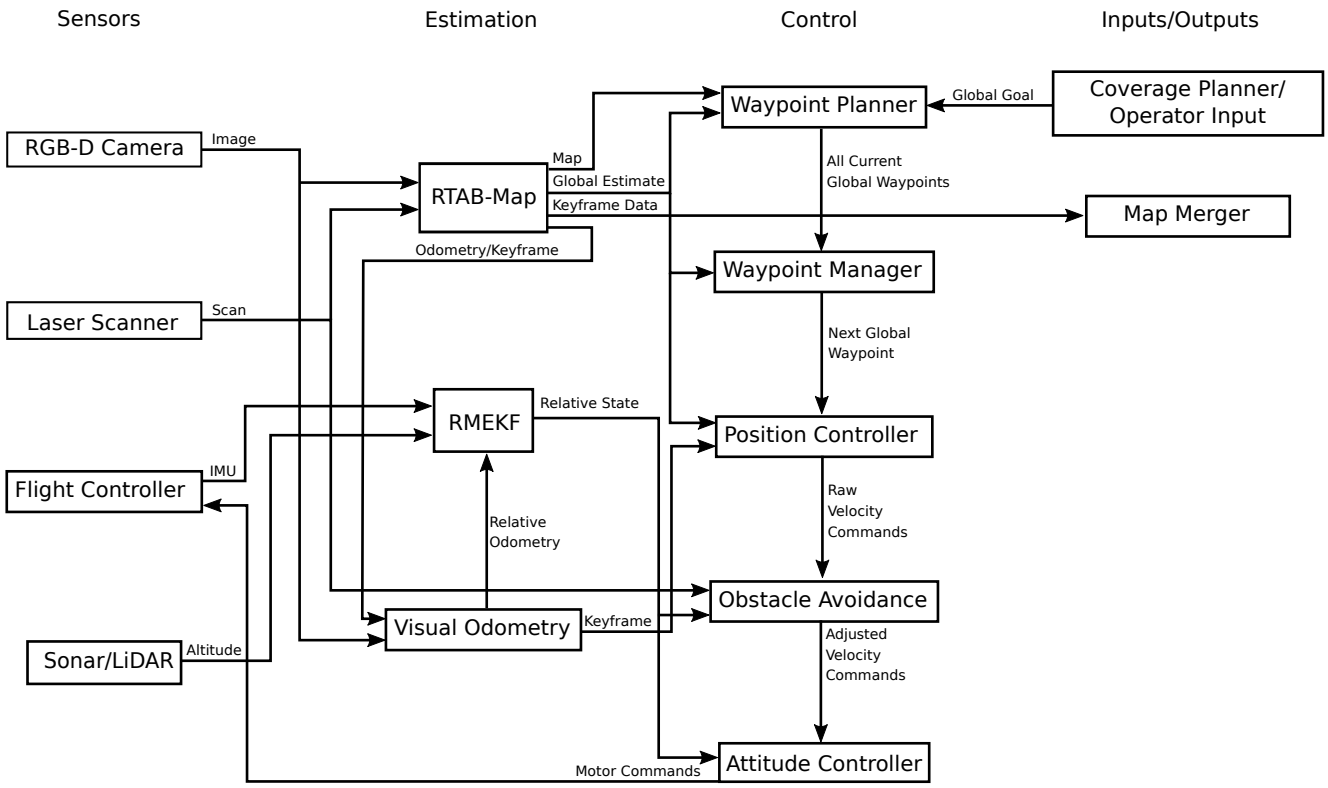

Fig. 1. The network diagram of the relative navigation framework proposed in this paper

estimator can diverge because the new position lies outside of the covariance bounds. 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.

The current global state estimates are stored in a transformation tree as shown in Fig. 2. The $world$ frame is the inertial NED (north east down) frame of the world with a static origin. The UAV's starting location with respect to the world is set as a static transform between the $world$ and map frames with a rotation into the inertial NWU (north west up) orientation. The $base\_link$ transform represents the current estimated position of the the UAV in the NED orientation and $camera\_link$ represents the position in the NWU orientation, the $camera\_base\_link$ transform 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 $odom$ 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. This way the transform between the $odom$ frame and the robot frame $camera\_link$ and $base\_link$ stays continuous when loop closures are detected even though the position estimate is not.

The $keyframe$, $keyframe\_world$ and $keyframe\_camera$ frames are used to track the relative visual odometry for the relative estimation. More specifically, the relative visual odometry is stored in the tree as the transform between the $keyframe$ and $camera\_link$ frames.

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: RTAB-Map (real-time appearence-based mapping), developed by Labbe et al. [6][7][8], is a powerful open source software library that uses graph-SLAM with appearance-based loop closures to generate high-quality, dense 3D maps using only a RGB-D camera and a laser scanner. When coupled with its depth enhanced visual odometry algorithm called RGBD-Odometry [8], it is able to accurately estimate its position with respect to the map that it is building. RTAB-Map is primarily designed for use with slow moving ground robots which do not need high-rate state estimation to work. We found the estimation rates of RTAB-Map on our hardware to be in the 10-30 Hz range which is not sufficient on its own to use for autonomous navigation with a UAV. We used 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 multi-session mapping, it does not allow for mul-

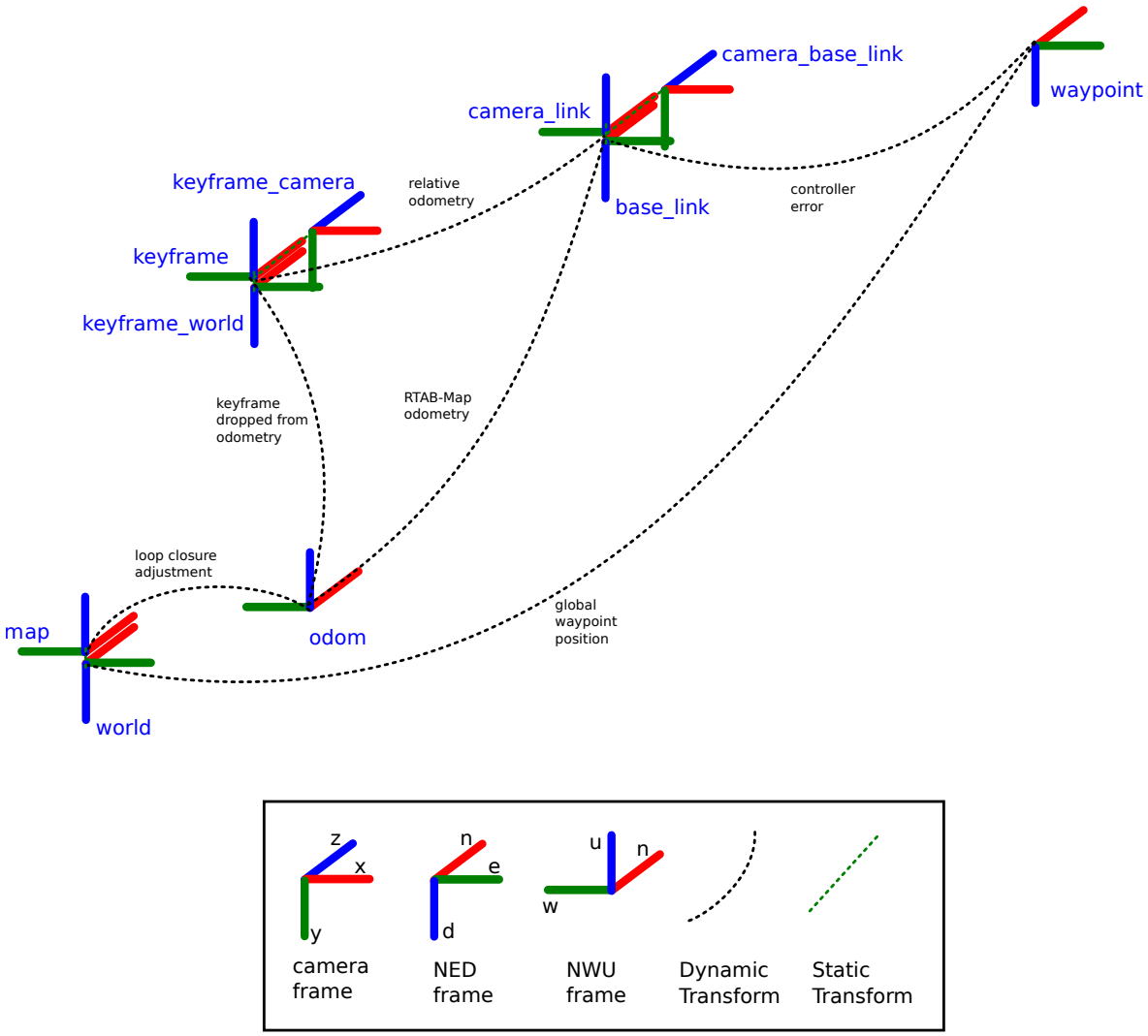

Fig. 2. The transformation tree of the reference frames used in estimation and control.

tiple 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 as previously used in the transformation tree and their respective transforms.

We use the current position estimate of RTAB-Map for the position controller and waypoint manager. Because of the inevitable inaccuracies and the lower estimate rates of RTAB-Map, we do not use these estimates to do attitude control on the UAVs. Rather, we use a relative navigation framework to estimate the attitude and relative state.

2) Visual Odometry: The odometry and keyframe information generated by RTAB-Map is used to produce a relative odometry message that is sent the relative estimator. RTAB-Map produces a global visual odometry which 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. As previously mentioned, this relative odometry information is stored as the transform between the *keyframe* and *camera_link* frames. Because of the constant resetting of the keyframe transform, the odometry used by the relative estimator is less susceptible to drift over time.

3) RMEKF: The Relative Multiplicative Extended Kalman Filter (RMEKF) which is the heart of the relative navigation framework established by Wheeler et al. [9][10] and Koch et al. [11] was shown to successfully estimate the UAV's relative state sufficient to autonomously navigate in GPS-denied environments that had been previously 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 RMEKF uses the IMU measurement from the flight controller, the relative visual odometry as explained previously, and the attitude measurement from the LiDAR single beam range finder as the measurement updates. Then using the multirotor dynamics model, the RMEKF is able to accurately estimate 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 which would cause stability issues in the velocity and attitude controllers onboard the UAV.

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 inputs to the cascading control scheme, as shown in Fig. 1, are the current obstacle map, estimated position, and goal position. After calculating the control efforts needed, it outputs the motor commands to the flight controller.

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

2) Waypoint Manager: After receiving the current set of waypoints, the waypoint manager selects the appropriate current 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 and when the error crosses 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 the UAV rather than the state space, sudden shifts in the UAVs position estimate caused by loop closures have minimal effect on the controller and it is able to continue controlling the error to zero. The output of the position controller is a velocity command for the UAV.

4) Obstacle Avoidance: Before passing the velocity control 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 along with the input velocity command. It uses a cushioned extended-periphery avoidance (CEPA) technique developed by Jackson et al. [12] to alter the velocity command when necessary to push the UAV away from obstacles while continuing as close to the direction of the incoming velocity command as possible. 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 takes the velocity input commands and outputs the motor commands to the flight controller.

E. Inputs/Outputs

The input for each agent in the system is the desired goal location 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 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

The reactive planner we use is a form of rapidly-exploring random trees (RRT) planner like that originally developed by Lavalle et al. [13] with a path smoothing approach similar to that proposed by Beard and McLain in 2012 [14]. RRT path planning is effective for planning in real time with dynamic obstacles because it randomly searches the full, non-discretized environment for feasible paths rather than searching the environment exhaustively which can require heavy discretization. Since the planner uses a dense 2D grid map of all currently known obstacles, some adjustments were made for it to efficiently plan and re-plan when new obstacles were 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 its 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 uses RRT to plan 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. To avoid obstacles in the path, it is necessary for the planner to be able to dynamically re-plan paths as new obstacles are added to the map and loop closures adjustments are made.

B. Reactive Path Planning

Fig. 3 shows the process of the dynamic path planning 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 it begins. A path is planned to the current goal which avoids the obstacles that are initially detected. As the UAV flies, the obstacle map is continuously updated with new obstacles. the path is constantly being checked for collisions with any obstacles. If collisions are found, a new path is planned to the goal from the current location which avoids the newly discovered obstacles. This process continues until the agent is able to successfully reach 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.

Since this planner is used in conjunction with the CEPA obstacle avoidance node explained in II-D.4, the UAV is able to effectively avoid obstacles in emergency situations at a higher rate than the planner runs. The obstacle avoidance also is able to correct a velocity command that would have caused a collision. The UAV is also able to navigate through complex maps that would not be possible with only obstacle avoidance. If a possible path to the goal exists, the RRT planner will find a way to reach it. More recently, new improvements to RRT have been explored such as RRT* developed by Karaman et al. [15] which is able to guarantee asymptotically optimal paths. Since this planner is used as a form of exploration of unknown environments, we decided to not extend the planner to use RRT* to allow it to be more random in the flight path to encourage more exploration of the map while flying paths.

1) Efficient Collision Detection: Most implementations of RRT have few large obstacles that are planned around but the obstacles we use are from a 2D grid map generated by RTAB-Map. So rather than having just a few large obstacles, there are many small obstacles. So using a standard obstacle detection check with each propagation of the RRT would be extremely inefficient. To maximize the efficiency during planning, only obstacles within range of the candidate node and its connection to the tree are checked. An example of how this works is illustrated in Fig. 4.

As the RRT tree propagates, before each candidate node is added to the tree, an obstacle collision check is done on the obstacles within range of the new node. The obstacles are determined to be in range if they are either between both x and y locations of the line formed connecting the candidate node to the tree or within one buffer radius of those points. The green bounding box in Fig. 4 shows which obstacles would be included in this obstacle check. To check the if the obstacles would collide, we check to see if their perpendicular distance to the line is less than the buffer radius. With the following equations.

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

with

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

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

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

(x_1, y_1) and (x_2, y_2) are the endpoints of the candidate node line and (x_{obs}, y_{obs}) is the location of the obstacle being checked. 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 proposed in this paper is able to generate a combined map from multiple agents on a base station computer in near real time while the UAVs are still simultaneously mapping the 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, the data from the keyframe consisting of the color and depth information from the keyframe, and the features extracted from the color image are stored in a database referred to as RGB-D Cache which is hosted on the base station computer. A 3D color and depth (XYZRGB) pointcloud is generated using the color and depth images from the RGB-D camera onboard 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 similarly to how individual maps are generated for each UAV [6][7][8]. The first step is to search for loop closures in the image features 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 remove some noise and filter out the ceiling to make the map more understandable to the operator.

V. Results and Discussion

A. Simulation

We were able to successfully autonomously navigate and map a simulation environment in ROS Gazebo with multiple agents and combine 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 to mimic the flight controller and did not use any ground truth information available in simulation. Fig. 6 shows the simulation setup used to map the environment.

Fig. 7 shows the results from mapping the simulated environment with two agents and combining the maps. You can see that neither agent saw everything in the combined map, but they were able to successfully merge together into a map that had all of the features from each individual map. It is also important to note that this map is generated and combined as the agents are flying in real time.

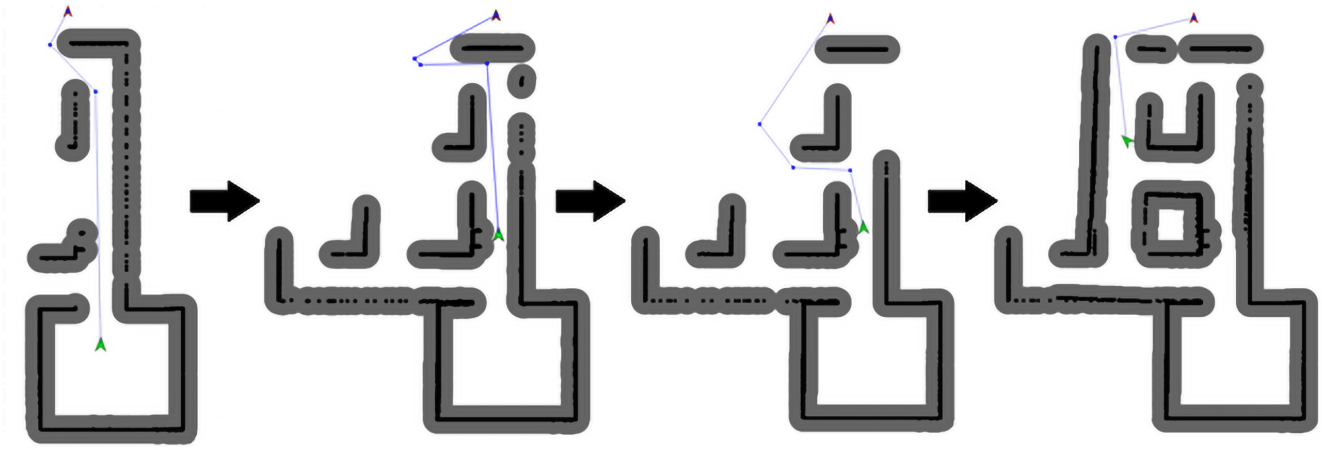

Fig. 3. An example of how the reactive path planner works as the UAV flies the planned path. The current estimated position is marked by the green arrow, the current goal position is marked by the red arrow, and the current path is marked with blue lines. Detected obstacles and safety buffers are represented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

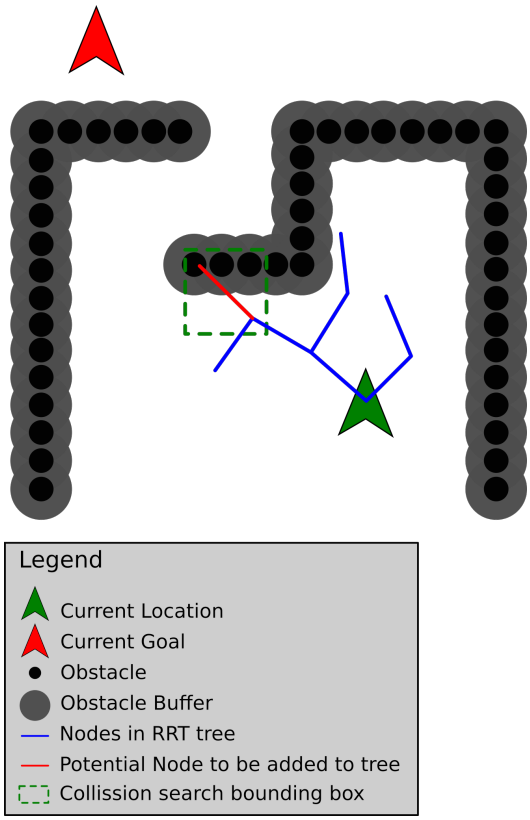

Fig. 4. Example of one step of collision check with the RRT planner.

Flying in simulation helped prove out the reactive planner and obstacle avoidance. It was also helpful to debug and sort out the plumbing of the relative navigation framework and control schemes. The area where the simulation falls short, is with the computer vision applications, Gazebo is good at mimicking realistic physics and dynamics from the real world, but the environments are significantly less detailed than the real world. This made designing a simulation world more difficult. When there was too little detail added to the world, the visual odometry algorithms would often fail or perform poorly. If too much repetitive detail was used, RTAB-Map would find too many false loop closures and the mapping would fail. The simulated world developed and used to obtain results as shown in Figs. 6 and 7 was able to minimize these issues, but still failed to produce results on par with a real world test. After proving the setup in high-fidelity simulation, we moved to hardware to get results especially with the computer vision aspects of the research.

B. Hardware

We were able to successfully combine maps generated from multiple UAVs in near real time in manual flight. Fig 8 shows an example of a map built in a cluttered environment with large amounts of glass. Although noisy, it is evident that the map merging works in hardware and generates a usable map of the environment. This map was generated from from pushing two agents on carts rather than flying them for safety concerns. Fig 9 shows an example of a map built from a more structured hallway environment with little glass. The map generated here is more structured than the cluttered environment. We were unable to tune the RMEKF sufficiently to fly autonomously in an uncontrolled environment for these results, but we were able to prove that with a sufficiently tuned estimator, the planning, control, and obstacle avoidance will work using the simulated world.

VI. Conclusions

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, may 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0926580514000193

[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, dec 2015. [Online]. Available: http://www.mdpi.com/2072-4292/8/1/26

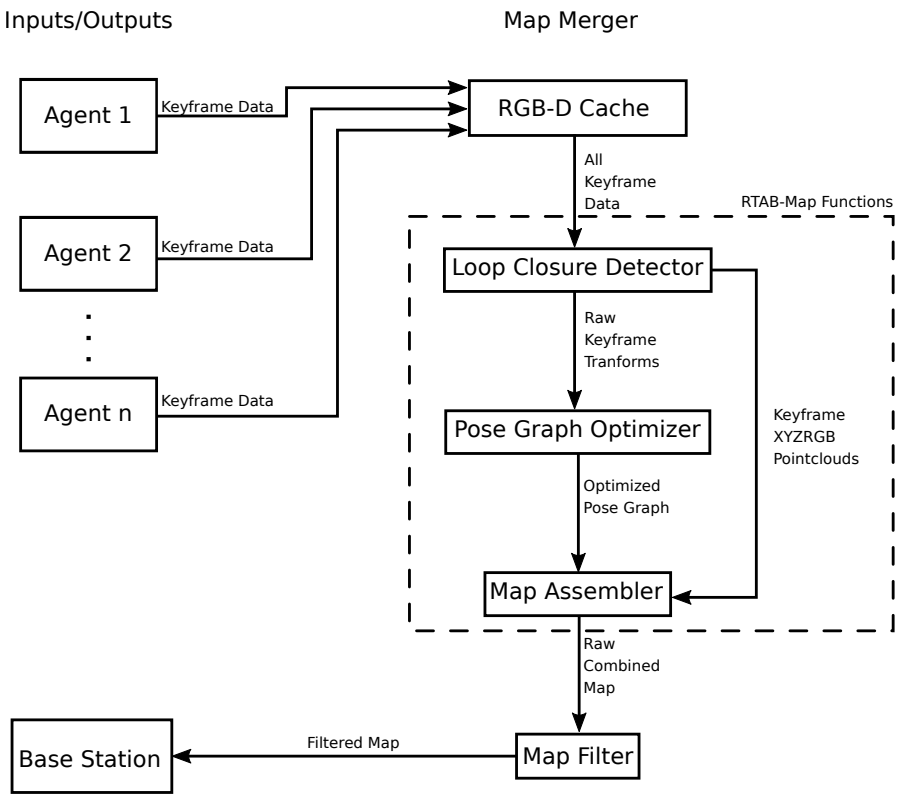

Fig. 5. The network diagram for the multi-agent map merging node proposed in this section.

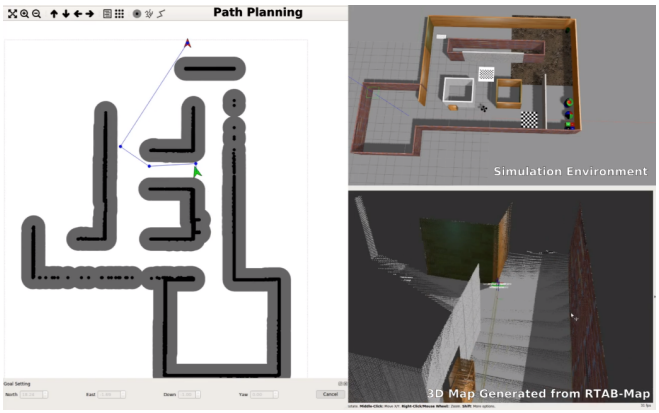

Fig. 6. Setup used for the simulation results. The reactive planner is shown on the left, the simulation world is shown on the top right, and the current map is shown on the bottom right

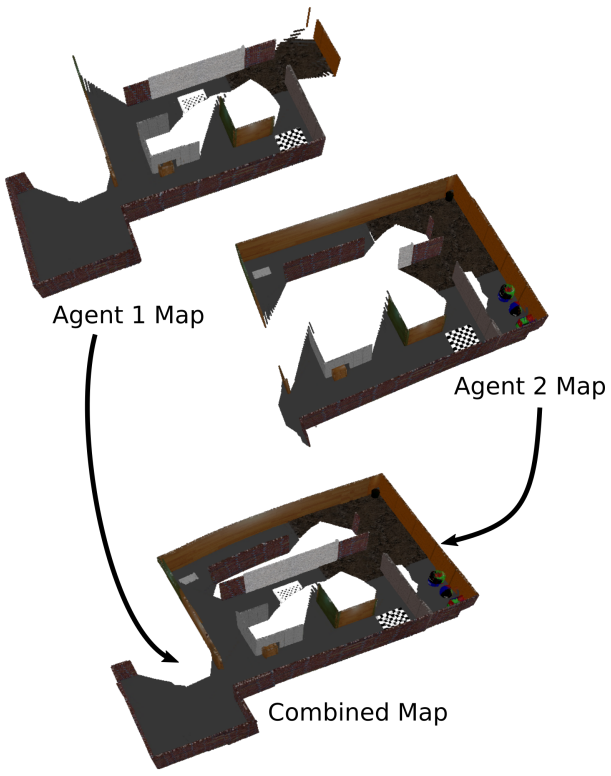

Fig. 7. Map generated from combined maps in the simulated environment.

[3] N. Michael, S. Shen, K. Mohta, Y. Mulgaonkar, V. Kumar, K. Nagatani, Y. Okada, S. Kiribayashi, K. Otake, K. Yoshida, K. Ohno, E. Takeuchi, and S. Tadokoro, "Collaborative mapping of an earthquake-damaged building via ground and aerial robots," *Journal of Field Robotics*, vol. 29, no. 5, pp. 832–841, sep 2012. [Online]. Available: http://doi.wiley.com/10.1002/rob.21436

[4] J. G. Mangelson, D. Dominic, R. M. Eustice, and R. Vasudevan, "Pairwise Consistent Measurement Set Maximization for Robust Multi-robot Map Merging," *ICRA 2018*, 2018. [Online]. Available: http://robots.engin.umich.edu/ joshuagm/pubs/jmangelson-2018a.pdf

[5] 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,

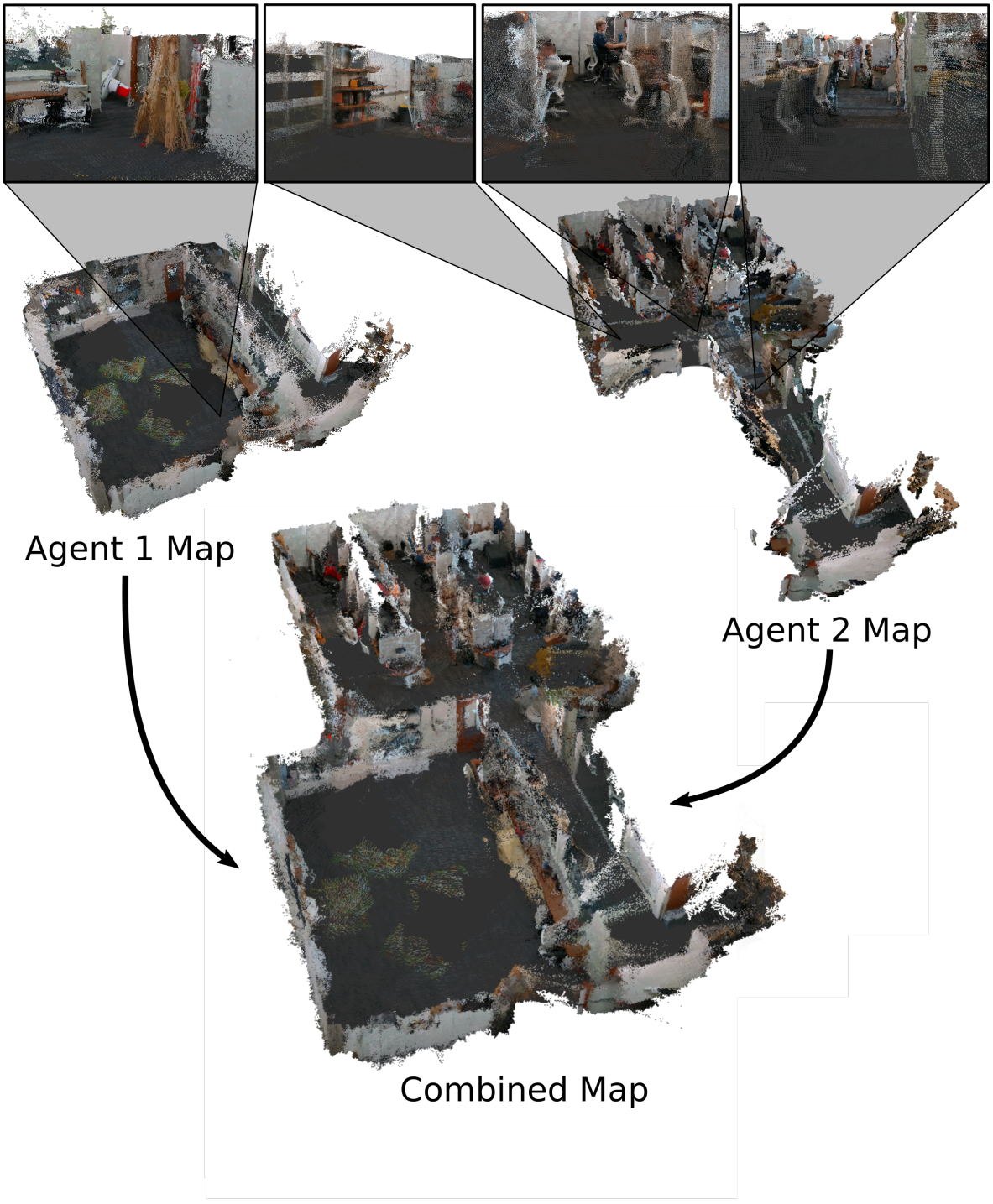

Fig. 8. Example of hardware results of merging maps from two agents into a single map in a cluttered environment. Above the individual agent maps are examples of the detail in the pointclouds when zooming in.

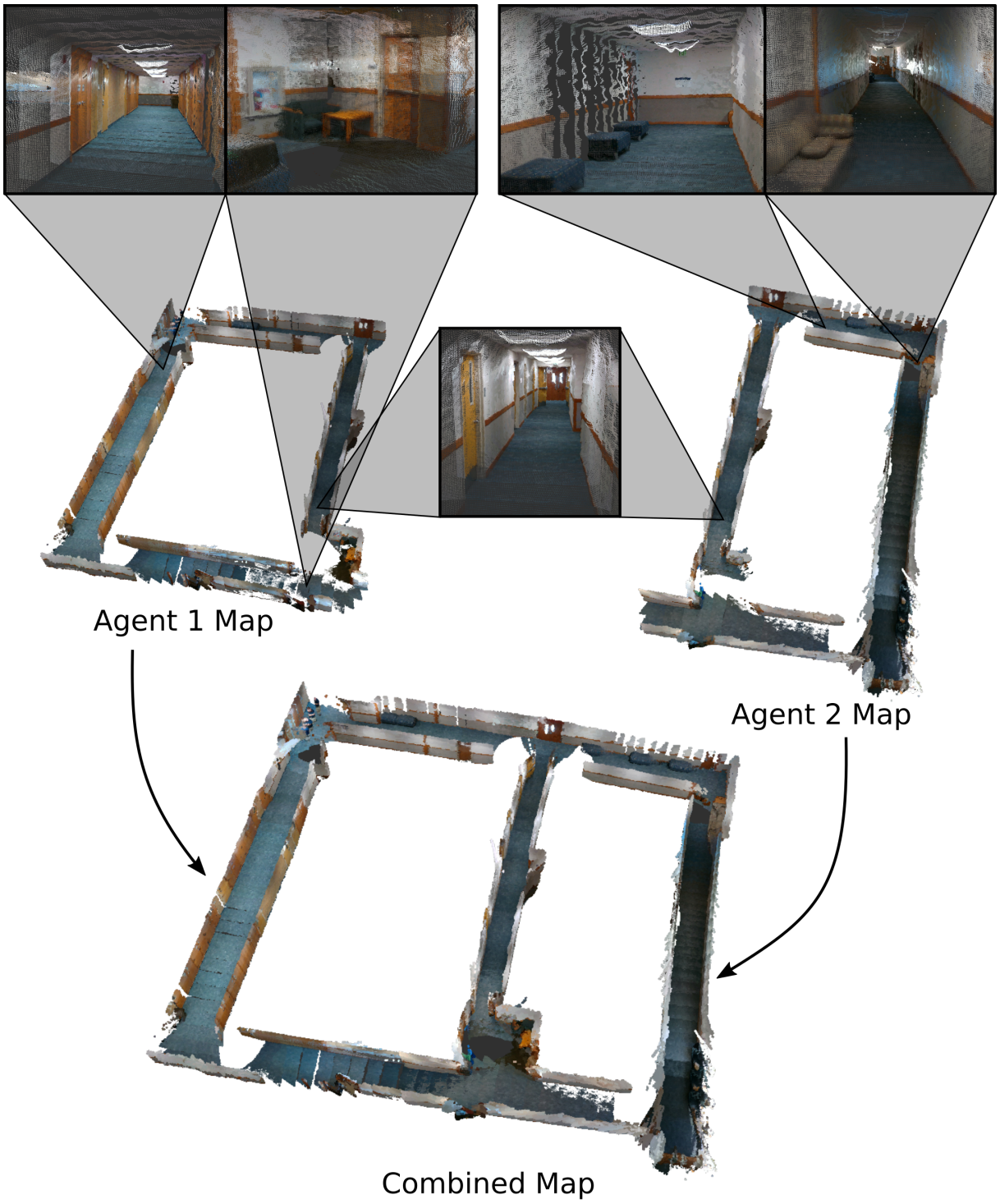

Fig. 9. Example of hardware results of merging maps from two agents into a single map in a simpler hallway environment. Above the individual agent maps are examples of the detail in the pointclouds when zooming in.

no. 5-6, pp. 403–429, may 2006. [Online]. Available: http://journals.sagepub.com/doi/10.1177/0278364906065387

[6] M. Labbe 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, sep 2011, pp. 1271–1276. [Online]. Available: http://ieeexplore.ieee.org/document/6094602/

[7] ——, "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. [Online]. Available: https://introlab.3it.usherbrooke.ca/mediawiki-introlab/images/b/bc/TRO2013.pdf

[8] M. Labbé and F. Michaud, "RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation," *Journal of Field Robotics*, vol. 36, no. 2, pp. 416–446, mar 2019. [Online]. Available: http://doi.wiley.com/10.1002/rob.21831

[9] D. Wheeler, D. Koch, J. Jackson, G. Ellingson, P. Nyholm, T. McLain, and R. Beard, "Relative Navigation of Autonomous GPS-Degraded Micro Air Vehicles," *All Faculty Publications*, aug 2017. [Online]. Available: https://scholarsarchive.byu.edu/facpub/1962

[10] D. O. Wheeler, D. P. Koch, D. O. Wheeler, D. P. Koch, J. S. Jackson, T. W. Mclain, R. W. Beard, 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," vol. 38, no. 4, pp. 30–48, 2018. [Online]. Available: https://scholarsarchive.byu.edu/facpubhttps://scholarsarchive.byu.edu/facpub/1961

[11] D. P. Koch, D. O. Wheeler, D. P. . Koch, D. O. . Wheeler, R. . Beard, T. . Mclain, K. M. Brink, R. Beard, T. Mclain, R. W. Beard, and T. W. Mclain, "Relative Multiplicative Extended Kalman Filter for Observable GPS-Denied Navigation," Tech. Rep., 2017. [Online]. Available: https://scholarsarchive.byu.edu/facpubhttps://scholarsarchive.byu.edu/facpub/1963

[12] 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, jun 2016, pp. 399–405. [Online]. Available: http://ieeexplore.ieee.org/document/7502597/

[13] S. M. Lavalle and S. M. Lavalle, "Rapidly-Exploring Random Trees: A New Tool for Path Planning," 1998. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.35.1853

[14] R. Beard and T. McLain, *Small Unmanned Aircraft Theory and Practice*. Princeton, New Jersey: Princeton University Press, 2012.

[15] 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, jun 2011. [Online]. Available: http://journals.sagepub.com/doi/10.1177/0278364911406761