2019-10-16

Collaborative UAV Planning, Mapping, and Exploration in GPS-Denied Environments

Jacob Moroni Olson
Brigham Young University

Collaborative UAV Planning, Mapping, and Exploration in GPS-Denied Environments

Jacob Moroni Olson

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Timothy W. McLain, Chair
Randal W. Beard
Marc D. Killpack

Department of Mechanical Engineering

Brigham Young University

ABSTRACT

Collaborative UAV Planning, Mapping, and Exploration
in GPS-Denied Environments

Jacob Moroni Olson
Department of Mechanical Engineering, BYU
Master of Science

The use of multirotor UAVs to map GPS-degraded environments is useful for many purposes ranging from routine structural inspections to post-disaster exploration to search for survivors and evaluate structural integrity. Multirotor UAVs are able to reach many areas that humans and other robots cannot safely access. Because of their relatively short operational flight time compared to other robotic applications, using multiple UAVs to collaboratively map these environments can streamline the mapping process significantly. This research focuses on four primary areas regarding autonomous mapping and navigation with multiple UAVs in complex unknown or partially unknown GPS-denied environments: The first area is the high-level coverage path planning necessary to successfully map these environments with multiple agents. The second area is the lower-level reactive path planning that enables autonomous navigation through complex, unknown environments. Third, is the estimation framework that enables autonomous flight without the use of GPS or other global position sensors. Lastly, it focuses on the mapping framework to build a single dense 3D map of these environments with multiple agents flying simultaneously.

Keywords: planning, mapping, collaborative mapping, UAV, GPS-denied, genetic algorithms, relative navigation

ACKNOWLEDGMENTS

I would like to acknowledge the help and advice of my advisor, Dr. McLain, through the process of completing the thesis. I would also like to acknowledge the National Institute of Standards and Technology (NIST) for funding the research that made this possible. I would like to thank James Jackson and Dan Koch for bringing me up to speed with the relative navigation framework and for always being willing to give advice for my research. Most of all, I would like to thank my wife, Sara Olson, for supporting me while I pursued this research. I could not have done it without her.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1. INTRODUCTION

1.1 Problem Statement

During a disaster such as an earthquake or fire, buildings are often left structurally unsound. Sending in a human team to inspect the building can unnecessarily put human lives at risk. This project seeks to minimize the problem by sending in a swarm of intelligent unmanned aerial vehicles (UAVs) deployed from a larger, tethered UAV shown in Figure 1.1. The swarm is able to map a building, scan for damaged structures, and identify the source of a fire to determine the level of damage and whether it is safe to send in first responders.

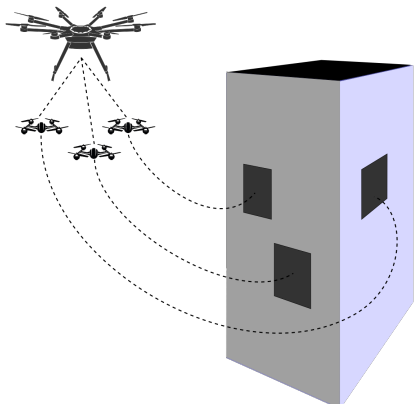


Figure 1.1: Survey UAVs deploy from Sentinel UAV to map a GPS-Denied area

Recent advances in GPS-denied navigation [1] make it possible for UAVs to safely and accurately localize themselves in GPS-degraded environments without colliding with obstacles, making indoor exploration and mapping possible.

Small multirotor UAVs are better equipped for indoor flight than fixed-wing UAVs because of their ability to hover in place and take off and land vertically. One inherent issue of multirotor UAVs, especially small ones, is that they have a limited flight time due to their smaller payload

capacities. Because of their limited flight time, the optimization of search routes can greatly improve their success in exploring and mapping an area. They must also be able to collaborate in their efforts to map and scan a building to quickly deliver results to a ground station. There are many mapping algorithms that work well with a single UAV, but the concept of collaborative, simultaneous mapping with UAVs is a not as developed. Leveraging multiple UAVs simultaneously mapping an area and meshing the data into a single map will greatly decrease the time it takes to survey and map an environment. This will make it more feasible for UAVs to quickly survey a building in an emergency and get the information to first responders without risking human lives.

The objective of this proposed research is to develop a method that leverages multiple small multirotor UAVs by optimally planning their search routes, and efficiently meshing data from the on-board sensors of the UAVs into a single map that is sufficiently organized that a human can read it and extract appropriate information for action.

1.2 Background

There are many components to the mapping problem for UAVs that are integral to its solution including the sensors used to collect the data for maps, the algorithms and data structures used to generate and display maps, and the planning used to navigate through maps.

1.2.1 Sensors

The sensors mounted on the UAVs greatly affect how they are able to map their environments. Monocular cameras are the most lightweight, but are the hardest to use in mapping a three-dimensional environment. Without other sensors, they are unable to create a map with an absolute scale. Another technology with improved potential for 3D perception is the stereo camera, which is made up of two cameras side by side with a known offset. The known offset between the cameras makes it possible to extract depth information from their images. Stereo cameras are able to create a map with an absolute scale, but are still limited in their ability to reliably extract depth information from the environment. In the last several years, 3D color and depth (RGB-D) cameras have emerged. Rather than just using a color stereo pair like standard stereo cameras, RGB-D cameras have a single color camera, a stereo pair of infrared (IR) cameras, and an IR

projector. The IR projector projects a pattern onto the environment that the IR cameras use to get more accurate depth readings than a standard stereo camera. The depth information is paired with the color camera to produce a high-resolution color image with an associated depth for each pixel in the image. These cameras have already had a significant impact on robotics and autonomy applications [2]. Until recently, however, these cameras have had a large form factor compared to monocular and stereo cameras, did not work outdoors, and their depth range was severely limited, making it hard for them to be used on small UAVs. Intel, one of the industry leaders in RGB-D camera technology, recently released a line of small lightweight RGB-D cameras [3] that work both indoors and outdoors with much better range. The small form factor and longer range make these cameras more appropriate for use on a UAV.

Another type of sensor that should be noted is a scanning LiDAR sensor. Rather than return an image of the environment, these sensors are able to return a 360-degree representation of the environment with an order of magnitude greater range than RGB-D cameras. There are two types of scanning LiDAR sensors, 3D sensors that use multiple beams to generate a three-dimensional representation of their environment [4], and 2D sensors, which use a single beam to generate a planar representation of their environment [5]. These sensors are powerful, especially the 3D sensors, but due to the size and weight of the 3D LiDAR sensors, it is infeasible to carry one on the size of UAV that will be used in this thesis. There are smaller and lighter 2D scanning LiDAR sensors that are more feasible and one will be used for obstacle avoidance and to improve long range measurements in this research.

1.2.2 Data Structures and Algorithms

There are different approaches to the way data is stored and used in generating 3D maps. Most approaches use a form of simultaneous localization and mapping (SLAM). There are several different types of SLAM for various data structures and purposes [6]. Occupancy grid mapping is a common technique used when generating dense maps of the environment. These maps can be represented in different ways: Voxels, which are 3D pixels used for reconstructing and representing an environment in 3D space, are generated using 3D occupancy grid mapping. Depending on their size, these are either data heavy or low resolution making them feasible for limited areas only. Octomaps, a newer representation, are like voxels but are able to scale with detail. This allows

for more resolution in detailed areas, but because of the lower resolution in large occupied or unoccupied areas they are less data heavy than a standard voxel grid and can represent much larger areas [7]. Pointclouds are a dense representation of 3D data. They can provide the most detail and resolution, but do not represent vacant and occupied space as well as octomaps. By default, LiDAR sensors and RGB-D cameras return pointclouds, but often these are converted to voxels or octomaps to make the map less data heavy. Figure 1.2 below shows a comparison between a pointcloud representation and an octomap representation of a scene with an image of the scene as a reference.

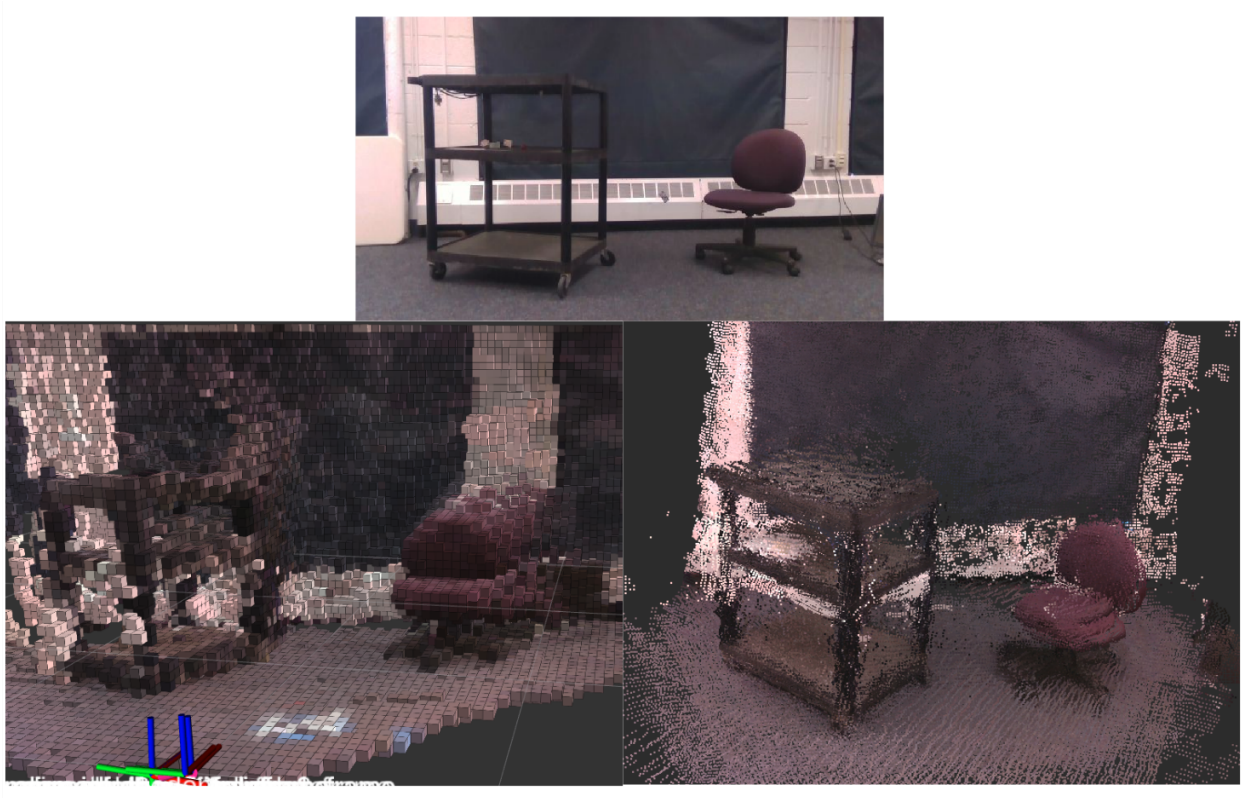


Figure 1.2: Comparison between an image(top), an octomap(left), and a pointcloud(right)

Over the last few years, 3D mapping technology has greatly improved. Labbe et al. [8–10] have developed a capable open-source software package called RTAB-Map designed for creating 3D maps from RGB-D cameras. This package simplifies the problem of creating reliable 3D maps and is able to generate both point clouds and octomaps. It is also able to mesh multiple maps

together into a single map using loop closures. It currently lacks the framework to build maps with multiple UAVs simultaneously.

Because of the limited flight duration of multirotor UAVs, when mapping areas that are larger than a single UAV can map in a flight, effectively merging maps from multiple agents into a single map is essential. Mangelson et al. [11] recently derived a way to robustly merge multiple maps from a factor-graph SLAM approach. They also discuss how this merging can happen real-time and improvements their approach makes over other methods of merging multiple maps.

In 2012 Micheal et al. [12] tackled some of the collaborative mapping problem by using ground and aerial robots to collaboratively map an earthquake-damaged building. They used a LiDAR scanner and an RGB-D camera and mapped the building using a voxel grid. They were able to successfully merge the maps into a single well-structured map. Their results lacked simultaneous mapping and a sufficient map resolution to be used effectively by search and rescue teams.

More recently in 2013, Zou et al. [13] performed collaborative simultaneous SLAM with multiple handheld cameras to generate a single map and localize all agents on the map. In 2017, Schmuck et al. [14] implemented monocular SLAM using multiple UAVs with downward facing cameras flying simultaneously to create a single shared map. Both of these instances successfully showed that it is possible for multiple agents to simultaneously generate a shared map. This research will be a good stepping stone, but it is different than the goal of this research project in that the map produced by the agents was sparse and would not be useful to first responders assessing post-disaster damage.

1.2.3 Planning

In 2007 Bryson et al. [15] demonstrated the ability to plan multiple flight paths and navigate an area using multiple UAVs flying concurrently using an EKF-SLAM (extended Kalman filter SLAM) algorithm. This research was successful in having multiple agents find the same landmarks and create a single landmark map used to localize all UAVs. This research was done outdoors, not in a GPS-denied environment, and it was a landmark-based EKF-SLAM. The resulting map is only the landmark locations, generating only a sparse representation of the environment. This project aims to generate a dense, detailed representation of the environment that will be useful to first responders.

The path planning for this project will have the focus of generating flight paths that produce a sufficient map of the environment for humans to act upon the information in the map. In 2015 Martin et al. [16] tackled a similar problem by using genetic algorithms to optimize flight paths that were able to effectively map uneven terrain and generate much more accurate maps than standard grid search flight paths. A similar approach to this will be used to plan efficient flight paths that are able to provide detailed, accurate maps of the indoor environment.

From this brief survey of the current research, it can be seen that although various and significant contributions have been made in the field of collaborative planning and mapping, there is still great opportunity for continued research. This research helps move some of the current research from the single UAV platforms to intelligent swarm applications, and helps make the maps generated by these intelligent swarms more usable by humans.

1.3 Research Contributions

In completing the proposed research, the following contributions have been made:

- A genetic algorithm coverage planner was developed to plan efficient paths through an indoor GPS-denied environment that have sufficient coverage and loop closures to generate useful maps.
- A low-level reactive path planner was designed to work in conjunction with an obstacle avoidance algorithm to enable safe navigation through complex environments.
- The functionality of the relative navigation framework was expanded to enable flight through new and unknown environments.
- The functionality of the RTAB-Map software package was expanded to allow for multiple agents to simultaneously map an environment and combine their maps into a single combined map in near real-time.

1.4 Thesis Outline

The remainder of the thesis is organized as follows: Chapter 2 details the software and hardware architecture used in this research to obtain results. Chapter 3 describes the relative navigation

and mapping framework used to autonomously map and navigate a GPS-denied environment with multiple UAVs simultaneously. Chapter 4 explores the multi-agent coverage path planner developed to plan appropriate paths for mapping a GPS-denied environment. Finally, Chapter 5 reviews the conclusions and recommendations after completing the research.

CHAPTER 2. SYSTEM ARCHITECTURE

This project was completed by first developing the software to work in a capable, high fidelity simulation environment, then extending this capability into hardware.

2.1 Software

Several open-source software packages were used to complete the research. A brief explanation of each package is included below

2.1.1 ROS

The software used in this research is built to work with Robot Operating System (ROS) as the communication framework [17]. It is a highly capable publish-subscribe inter-process communication library used widely across robotics disciplines designed to simplify the software interface between robotic platforms. The use of this framework made it possible to have multiple UAVs and base-station computers seamlessly communicate the necessary information to accomplish this research.

2.1.2 Relative Navigation Framework

The relative navigation framework developed by Leishman et al. [1, 18–22] makes it possible for a UAV to robustly navigate indoors and in other GPS-denied environments where global information is not available. This is accomplished using a keyframe approach and only estimating the UAV's state relative to the current keyframe. The usage and functionality of this framework will be explained with more detail in Chapter 3.

2.1.3 RTAB-Map

The open-source visual odometry and 3D mapping software developed by Labbé et al. [8–10], real-time appearance-based mapping (RTAB-Map), is a capable and configurable package optimized for usage with RGB-D cameras. Using only the input from an RGB-D camera, it is able to generate high-quality dense 3D maps of the environment. RTAB-Map also has a depth-enhanced visual odometry algorithm that we use, along with the relative navigation framework to enable autonomous flight. The functionality of RTAB-Map was extended as part of this research to allow for simultaneous multi-agent mapping in real time which will be explained in further detail in Chapter 3.

2.2 Simulation Environment

The simulation environment was built to reflect, as closely as possible, the real physics and interface of the real world.

2.2.1 Gazebo

Gazebo is an open-source simulation environment with high-fidelity physics designed to approximately replicate reality [23]. This environment allowed me to run several tests as I was developing the software and allowed me to prototype much faster without risking crashing expensive equipment. One complexity of simulating a functional environment is designing detail of the environment correctly. If there is not enough detail and texture in the environment, the robotic vision techniques, such as visual odometry and feature-matching loop closures, do not work. But if there is too much repetitive detail in the environment, false loop closures are detected too often and the mapping algorithm fails to create a coherent map. Fig. 2.1 shows an example of a detailed environment that worked for simulation testing.

2.2.2 ROS Flight

ROS Flight is an open source autopilot software built for use with ROS [24]. It takes advantage of the functionality and utility of ROS and is able to run both on hardware and in a

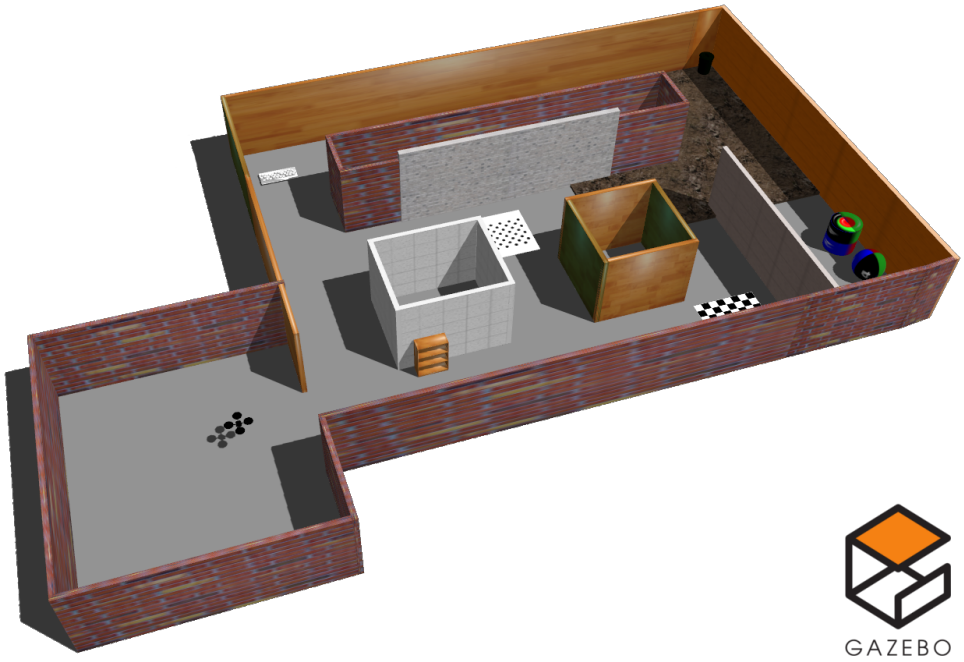


Figure 2.1: An example of a Gazebo environment used in simulation.

software-in-the-loop (SIL) mode. This helps bridge the gap between simulation and the real world because the same firmware is running in both.

2.2.3 Distributed Computing

Because of the heavy computational load of simulating multiple UAV's simultaneously, a distributed computing approach is used in simulation. A base-station computer with a dedicated graphics processing unit (GPU) is used to simulate the environment and the agent's sensors. The sensor information is sent via high-speed Ethernet connection to the onboard flight computers used on the hardware to simulate real sensor readings. The onboard computers run the software stack to navigate and map the environment. Since the computers used in the distributed computing setup are the same ones flown on the airframes, the distributed computing setup performs two functions: It lightens the computational load of the base-station computer enough to successfully simulate multiple UAVs simultaneously flying in the environment, and also allows for the simulation to emulate the computational load experienced when running the hardware. Fig. 2.2 shows the network configuration of this setup.

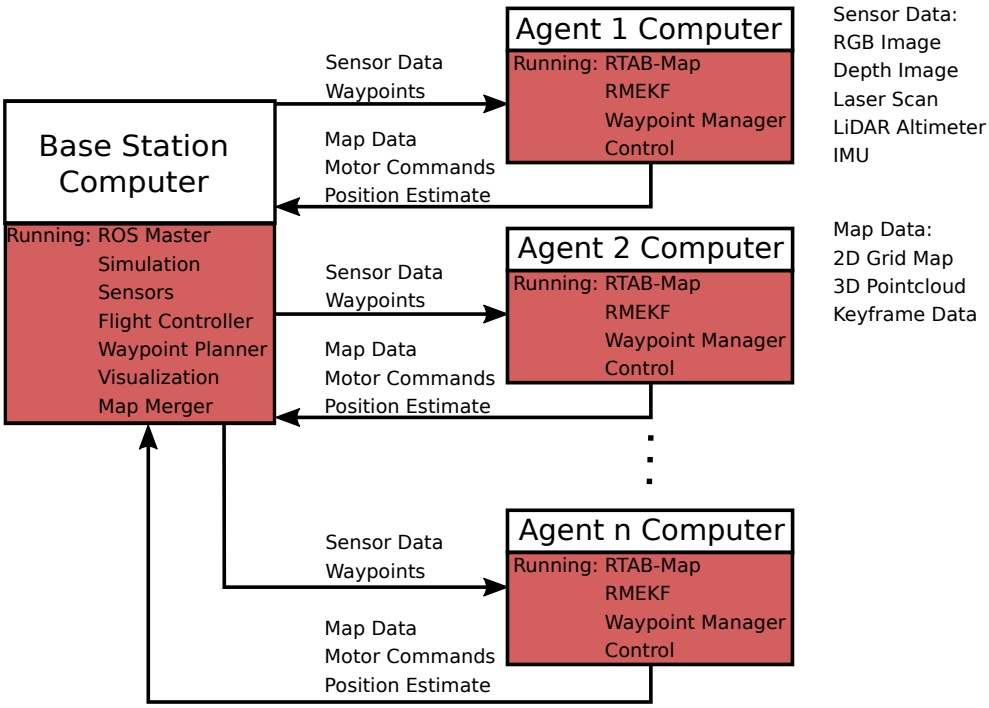


Figure 2.2: The network diagram of the distributed computing setup used for simulation.

2.3 Experimental Hardware

The hardware used for this research was selected to balance both the size and functionality of the UAV. The airframe is limited to be small enough to safely fly through an average-sized door, but large enough to efficiently carry the necessary payload to explore and map a GPS-denied environment. The network diagram for the hardware setup outlined in Fig. 2.3 shows the similarity between hardware and simulation. The computation load of the onboard computers remains essentially the same between simulation and hardware. It differs only in that the sensor data feeds directly into the onboard computer rather than being generated by the base station.

2.3.1 Airframe

The airframe, as shown in Fig. 2.4, chosen for this research is the Tarot LJI 500-X4 Quadrotor. The maximum width of the UAV including the propeller guards is 32 inches. This airframe was selected because it is easily customizable and configurable to carry the necessary payload and allow for safe indoor flight. If this concept were to be made into a product, streamlining the pay-

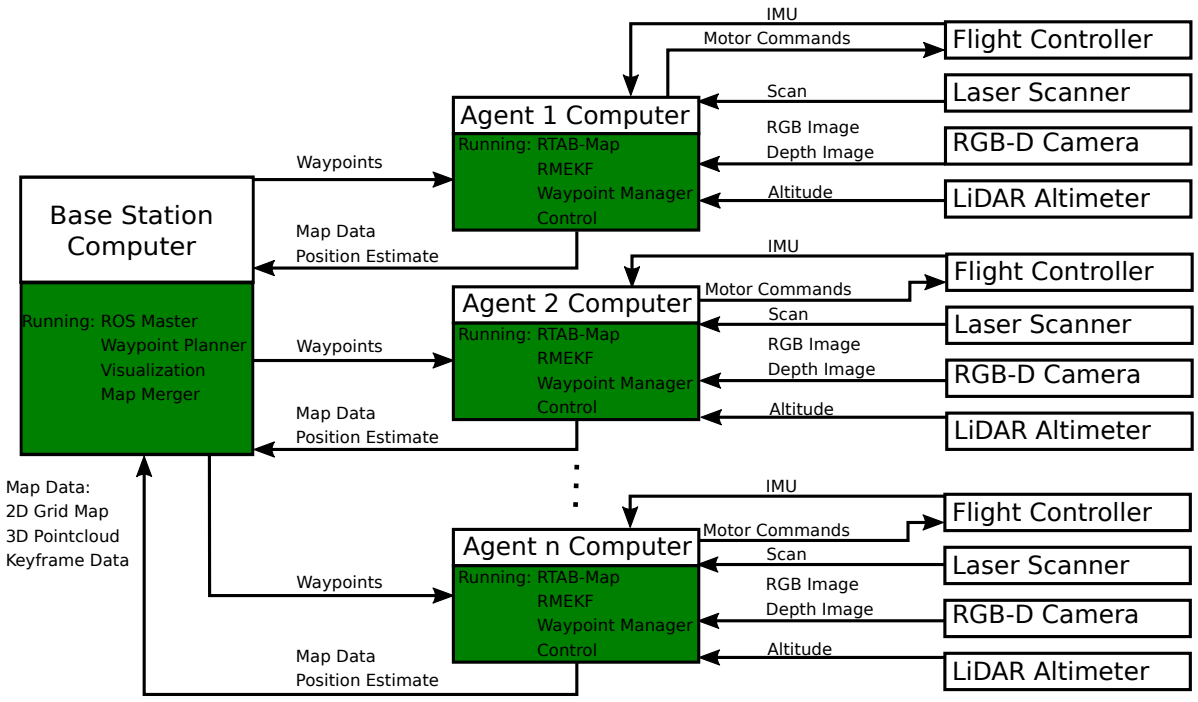


Figure 2.3: The network diagram of the hardware setup used for this research.

Figure 2.4: Customized Tarot LJI 500-X4 Quadrotor airframe built with sensors attached.

load and optimizing the motor and propeller combination would help to significantly reduce the necessary airframe size.

2.3.2 Sensors

The sensors used for the hardware were selected for their capability, weight, and price. Although some sensors perform better in some ways than the sensors used, the size, weight, and

Figure 2.5: Image of Intel RealSense d435 RGB-D camera.

price of the other sensors made them infeasible to carry on a UAV platform that fit the necessary criteria for this research.

Camera

The camera that was selected for this research is the Intel RealSense D435 RGB-D camera [3] shown in Fig. 2.5. As explained in Chapter 1.2.1, the D435 is a highly capable RGB-D camera that has a range of 10+ meters. It is also significantly smaller and lighter than other RGB-D cameras available on the market.

Planar LiDAR Scanner

We use a planar LiDAR scanner to ensure real-time obstacle avoidance would be possible and assist in the mapping process. The sensor we selected is the RPLiDAR A2 planar laser scanner [5] shown in Fig. 2.6. It has a range of 18 meters which is sufficient for indoor use, and a relatively low cost and small form factor which makes it ideal for use with a UAV.

Single-Beam LiDAR Altimeter

We use a single-beam LiDAR altimeter to help with the state estimation to enable an altitude hold control loop. The sensor we used is a TF-Mini [25] shown in Fig. 2.7. It has a small form factor, a low price point, and a range of 12 meters, which is sufficient to measure the height-above-ground of our flights.

Figure 2.6: Image of RPLiDAR A2 planar laser scanner.

Figure 2.7: Image of TF-Mini single-beam LiDAR altimeter.

Flight Controller

We use the F4 Advanced Flight Controller [26] shown in Fig. 2.8 as our flight controller. It uses a STM32F405 processor and has an integrated IMU that we use with the state estimation. The flight controller is running the ROS Flight firmware to handle attitude control and communication with the onboard computer.

2.3.3 Communication

To establish consistent communication between the UAVs and the base station, we use a high-speed long-range point-to-point WiFi system. This system enables much faster data transfer speeds at a much longer range than a standard WiFi connection. We use the Ubiquiti Rocket and Bullet AC system [27, 28] show in Fig. 2.9 to establish this connection. In open-air these are

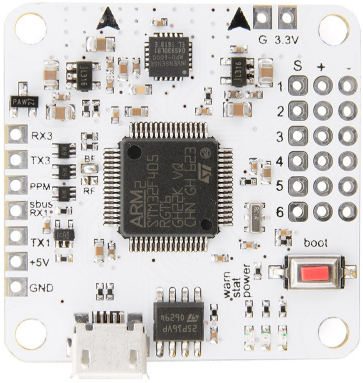


Figure 2.8: F4 Advanced flight controller used for attitude control.

Figure 2.9: Ubiquiti Rocket AC (left) and Ubiquiti Bullet AC (right) used for communication.

capable of speeds of up to 300 Mbps and up to one mile range. This allows for sufficient data transfer for our flights.

Figure 2.10: Gigabyte Brix i7-8550 onboard computer used to process mapping, estimation, and navigation algorithms.

2.3.4 Computation

The onboard computer for the UAVs needed to be both light weight and powerful. We selected the Gigabyte Brix i7-8550 [29] shown in Fig. 2.10 because it has a small form factor with a very powerful processor. The 8th generation Intel i7 processor is able to handle all of the necessary computation without saturating the CPU. This allows for the mapping and navigation algorithms to run at a sufficient frequency to achieve good visual odometry and map generation. The Brix also allows for up to 32 GB of RAM. This enables storing enough data in memory to quickly re-optimize large maps in real-time.

CHAPTER 3. RELATIVE NAVIGATION FRAMEWORK FOR MULTI-AGENT MAPPING

3.1 Introduction

[1] 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 [16, 30]. 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 [12]. More recently,

[1] The following chapter is composed from paper "Multi-Agent Mapping and Navigation of Unknown GPS-Denied Environments Using a Relative Navigation Framework" to be submitted to ICUAS 2020

Mangelson et al. detailed a method to effectively merge maps collected from multiple robots acting separately after the mapping process is complete [11]. 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 chapter is organized as follows: Section 3.2 presents the framework we use to navigate and map an environment along with background information about previous work that we build upon. Section 3.3 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 3.4. Results showing and evaluating the generated maps are presented in Section 3.5. Finally, conclusions are presented in Section 3.6.

3.2 Technical Approach

3.2.1 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 chapter 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. 3.1 that outlines the framework we use to successfully generate a single merged map from multiple UAVs in a GPS denied environment.

3.2.2 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 [3]), a planar light detection and ranging (LiDAR) laser scanner, a single-beam LiDAR range finder, and

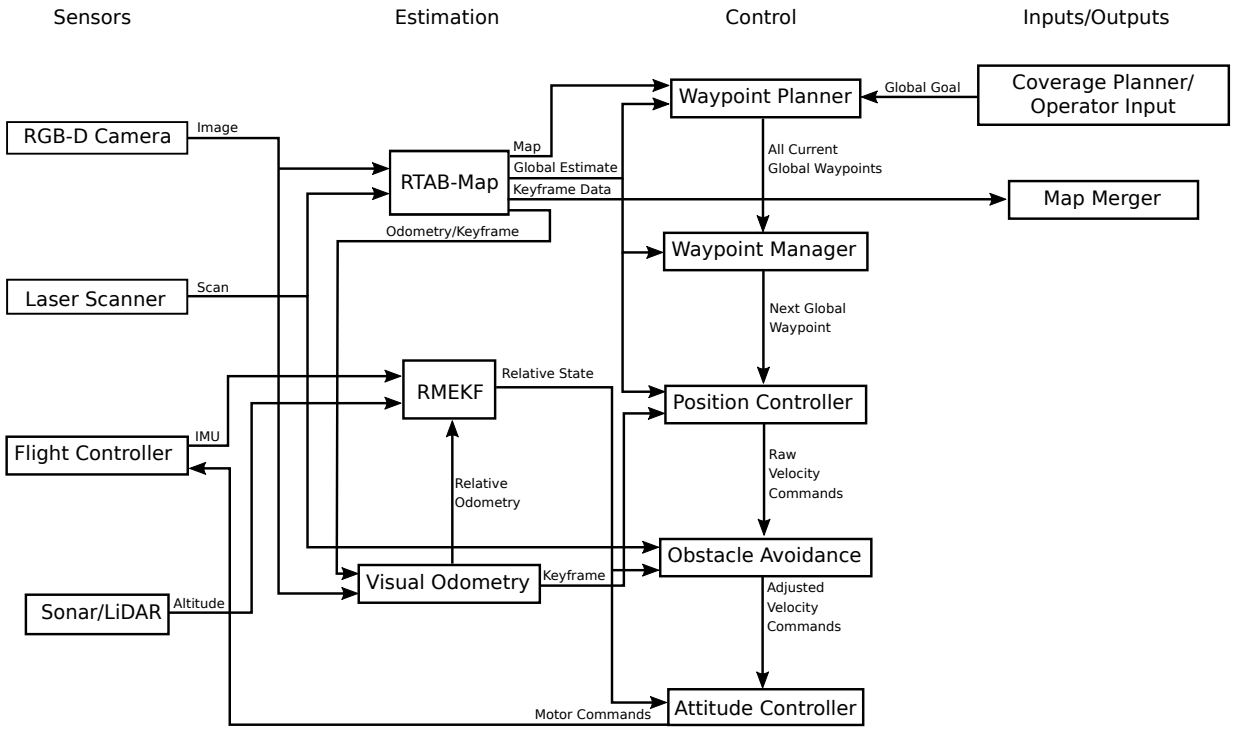


Figure 3.1: The network diagram of the relative navigation framework proposed in this chapter.

an inertial measurement unit (IMU) 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.

3.2.3 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. [31] 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

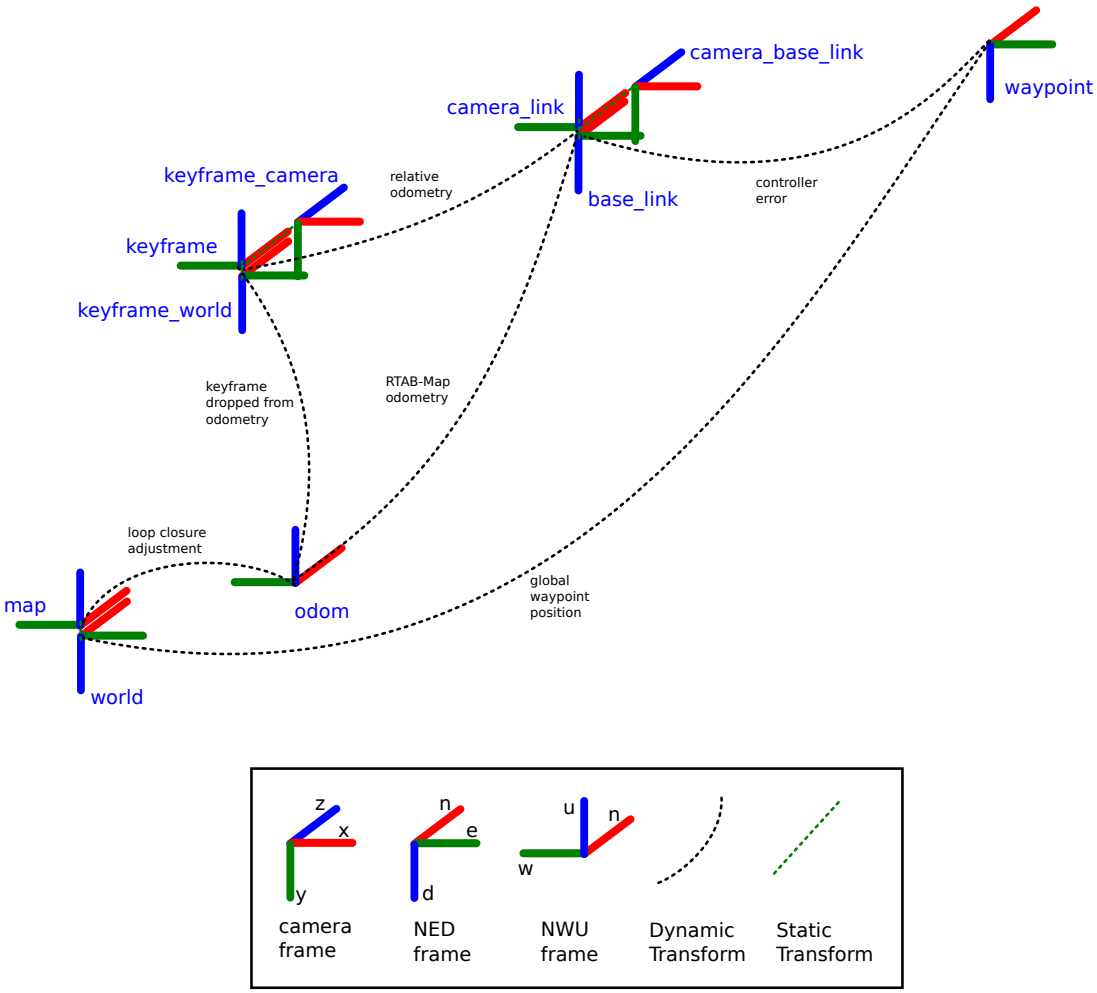


Figure 3.2: The transformation tree of the reference frames used in estimation and control.

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. 3.2. 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.

RTAB-Map

The real-time appearance-based mapping (RTAB-Map) package, developed by Labbe et al. [8–10], 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-Map [10], 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 chapter 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 3.4.

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.

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 3.1 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 [19].

RMEKF

The relative navigation framework [1, 21, 22] 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 chapter 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.

3.2.4 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 3.1.

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 3.3.

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.

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.

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 [32] 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.

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).

3.2.5 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 3.4.

3.3 Planning

When designing the reactive planner, we explored using both node-based optimal algorithms and sampling based algorithms [33]. Node-based (or heuristic search) algorithms like A* work well to find optimal paths around obstacles [34]. 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) [35] 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* [36], 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 [37] 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.

3.3.1 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.

3.3.2 Reactive Path Planning

Fig. 3.3 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.

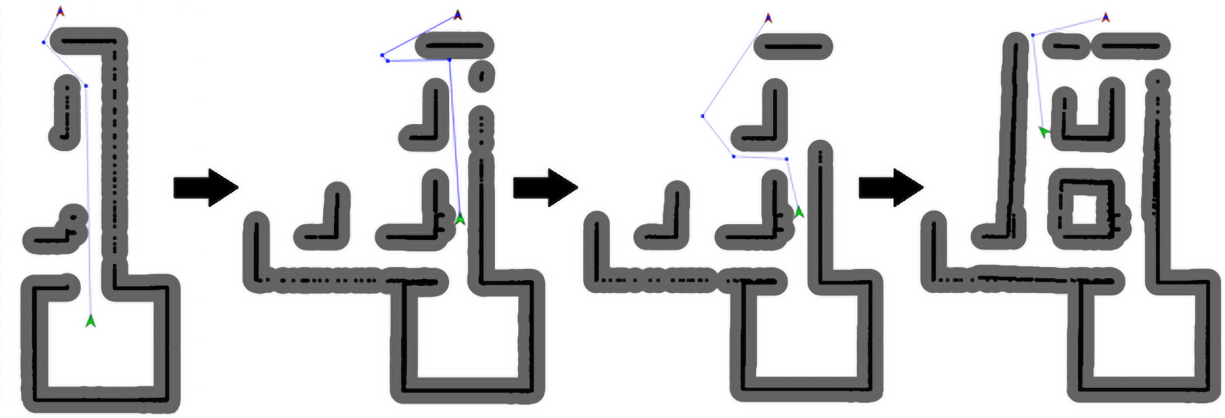


Figure 3.3: An example of how the reactive path planner works as the UAV flies the planned path. The current estimated position is marked by the green arrow, the current goal position is marked by the red arrow, and the current path is marked with blue lines. Detected obstacles and safety buffers are represented with black and grey respectively.

Since this planner is used in conjunction with the CEPA obstacle avoidance node explained in 3.2.4, 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.

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. 3.4.

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.

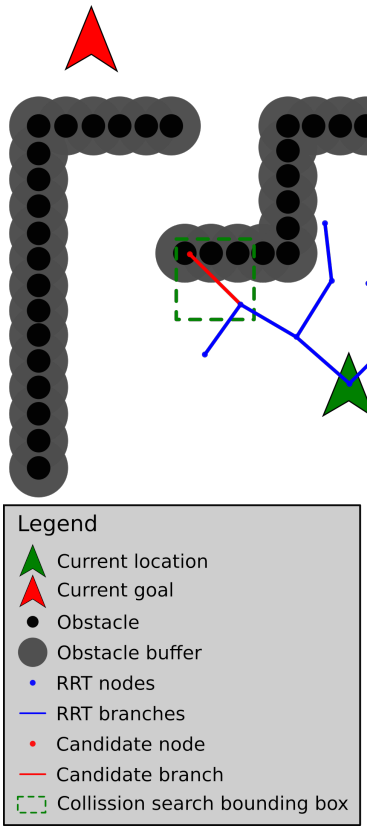


Figure 3.4: Example of one step of collision check with the RRT planner.

Algorithm 1 Obstacle Range Filter

Require: *candidate_branch*, *obstacle_list*, r_{buf}

```
for all obstacles in obstacle_list do
    if x_obs + r_buf ≤ min(x_1, x_2) then
        continue
    else if x_obs − r_buf ≥ max(x_1, x_2) then
        continue
    else if y_obs + r_buf ≤ min(y_1, y_2) then
        continue
    else if y_obs − r_buf ≥ max(y_1, y_2) then
        continue
    else
        filtered_obstacles ← 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. 3.4 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

$$d = \frac{|\Delta y\, x_{obs} - \Delta x\, y_{obs} + \Delta s|}{\sqrt{\Delta y^2 + \Delta x^2}}, \tag{3.1}$$

where

$$\Delta x = x_2 - x_1 \tag{3.2}$$

$$\Delta y = y_2 - y_1 \tag{3.3}$$

$$\Delta s = x_2\, y_1 - x_1\, y_2. \tag{3.4}$$

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.

3.4 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 environment. Fig. 3.5 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.

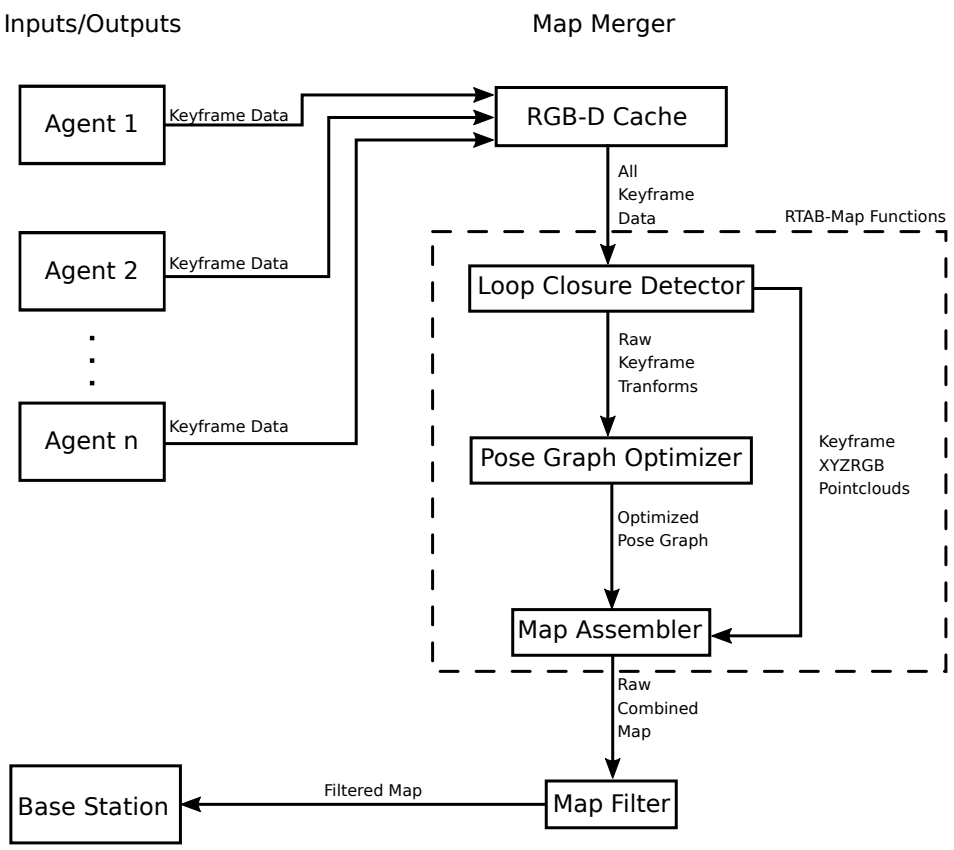


Figure 3.5: The network diagram for the multi-agent map merging node proposed in this section.

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 [8–10]. 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.

3.5 Results and Discussion

3.5.1 Simulation

We successfully navigated and mapped a simulation environment autonomously in ROS Gazebo [23] 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 [24], 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. 3.6 shows the simulation setup used to map the environment.

Fig. 3.7 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. 3.6 and 3.7 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,

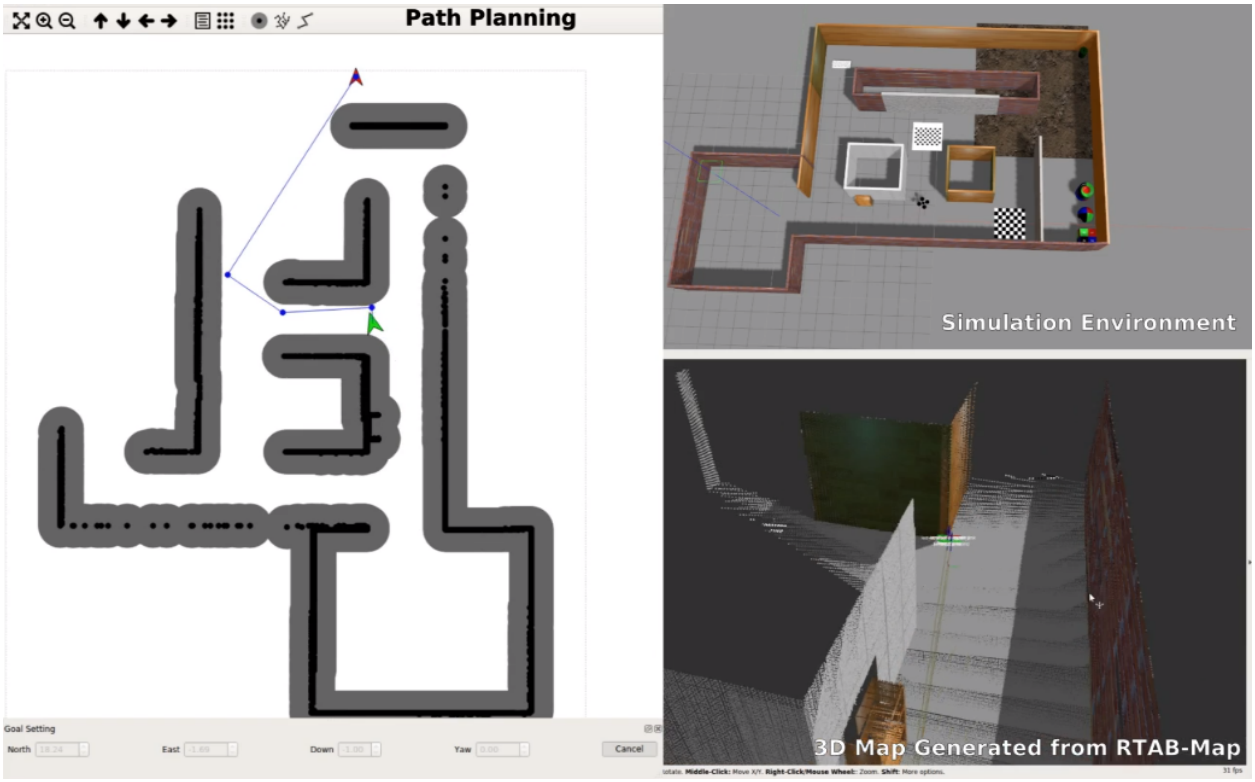


Figure 3.6: Setup used for the simulation results. The reactive planner is shown on the left, the simulation world is shown on the top right, and the current map is shown on the bottom right.

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

3.5.2 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 two vehicles and were able to expand it up to four vehicles. We were able to successfully combine maps generated from multiple UAVs in near real time with manual flight. Fig. 3.8 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

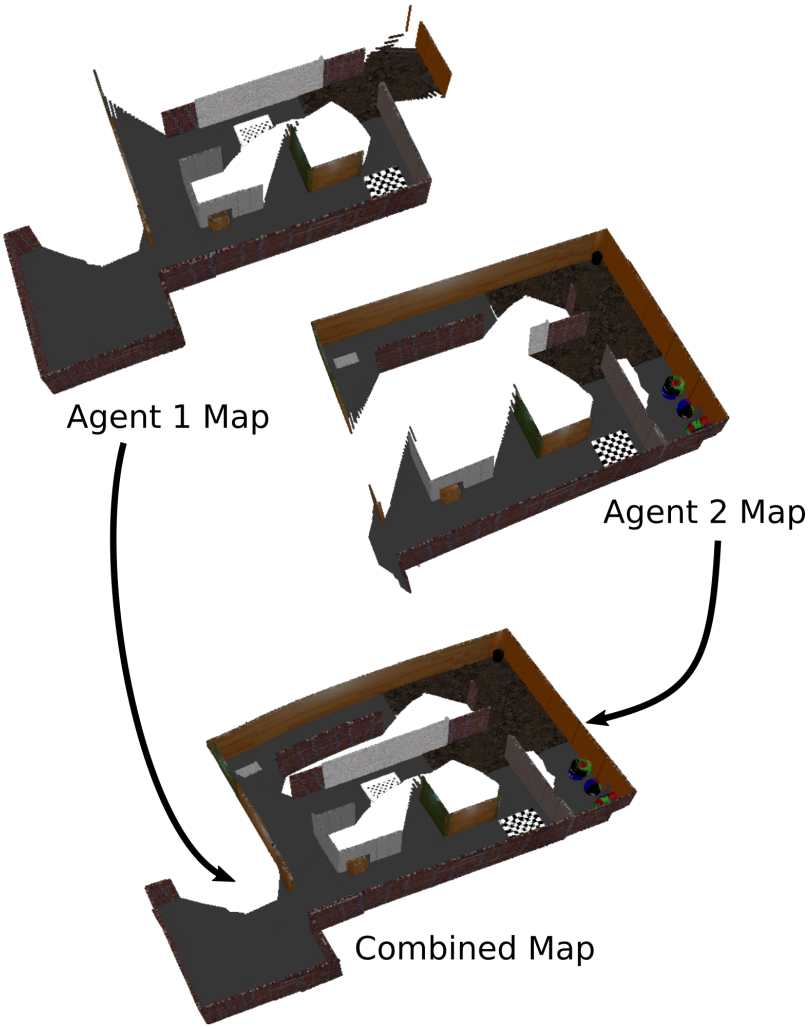


Figure 3.7: Map generated from combined maps in the simulated environment.

Fig. 3.9. By the end of the map merging process with this data, the combined map generation 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. 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.

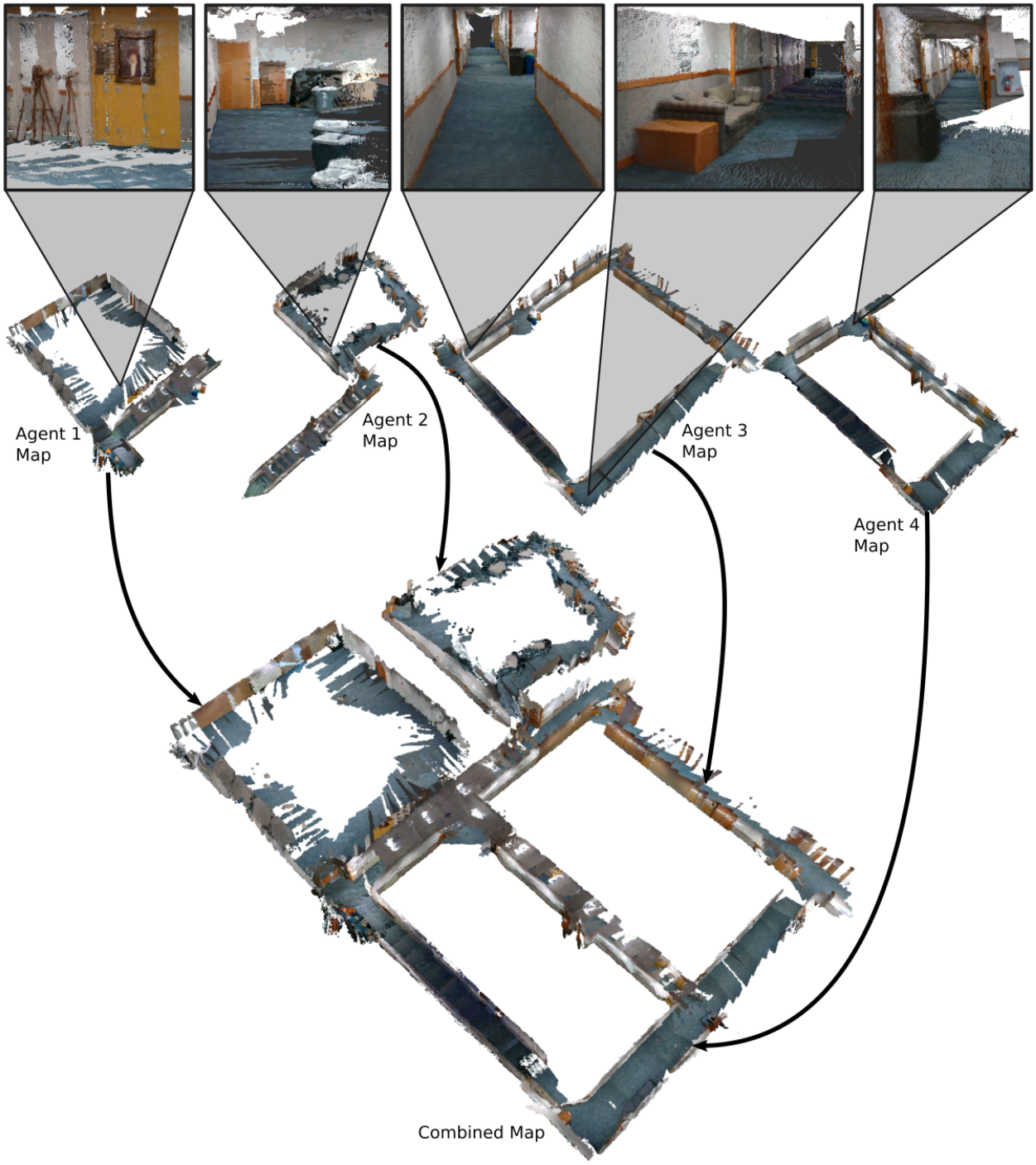


Figure 3.8: Example of hardware results of merging maps from four agents into a single map in an indoor environment. Above the individual agent maps are examples of the detail in the pointclouds when zoomed in.

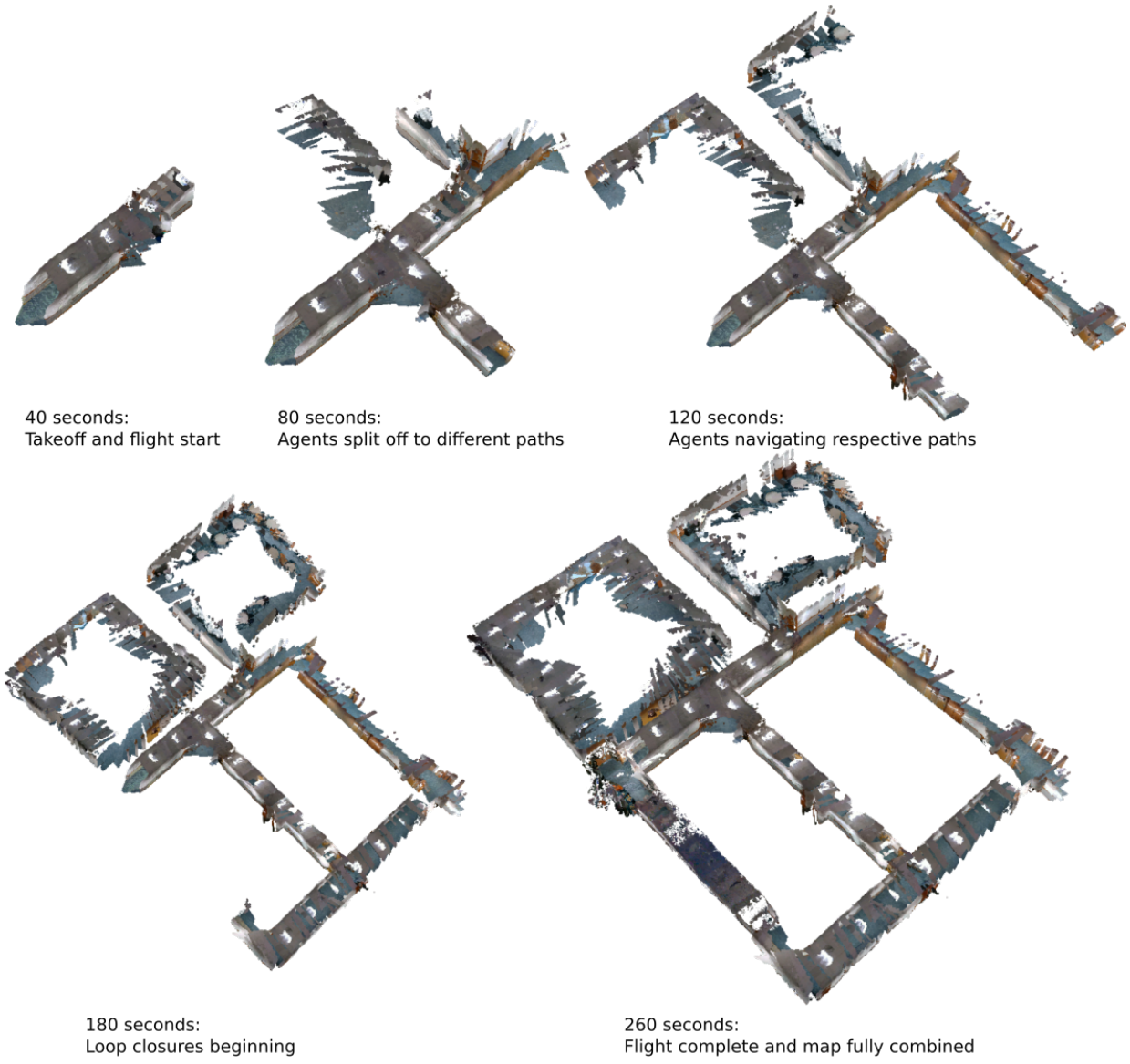


Figure 3.9: Example of the merged map being generated in near real-time. The combined map is updated every 30-40 seconds as agents are flying.

3.6 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.

3.7 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.

CHAPTER 4. COVERAGE PLANNING

4.1 Introduction

[1] Autonomously generating an internal 3D map of a building requires intelligent control of the mapping vehicle. This ranges from manually planning a flight path with simple obstacle avoidance to full-stack exploration and path-planning algorithms.

Generating a flight path is a nontrivial task, especially when multiple objectives are considered. For example, the algorithm may want to maximize observed coverage of the space while minimizing flight time. This becomes tedious with each new space and more difficult as the complexity of the space increases with more rooms and inter-connecting hallways. When planning paths to achieve good coverage of the area rather than obstacle avoidance or traversability, it becomes much more complex. To plan effective flight paths in more complex flight spaces, using path-planning algorithms becomes a necessity.

When generating paths for the purpose of building a 3D map of the environment in a GPS-denied area such as indoors, enforcing loop closures becomes critical. Loop closures both between paths of different agents and between different segments of the same path are critical to successfully generate a usable map. Loop closures between different segments of the same path significantly reduce drift that occurs in visual odometry algorithms necessary for navigation and estimation with UAVs. Loop closures between paths of different agents ensure that the individual maps of each agent will combine into a single map.

There are several different approaches to coverage path planning that have been explored over recent years. These approaches range from 2D coverage where the robot must pass over all points in its known environment, common for uses such as cleaning robots or mine searching [38]

[1]The following chapter is composed from paper "Optimal Multi-Agent Coverage and Flight Time with Genetic Path Planning" to be submitted to ICUAS 2020. Brady Anderson and Craig Bidstrup assisted in developing the algorithm to generate paths for a single agent with no loop closures. I then extended the algorithm to incorporate loop closures and plan paths for multiple agents simultaneously.

to using coverage planning to generate 3D maps of outdoor terrain using viewpoint information from the robot's camera [39]. These approaches differ from the coverage planning desired for generating 3D maps of indoor environments with multiple UAVs in that they do not enforce loop closures in the generated paths.

The complexity of the design space renders many simple optimization routines improper for this problem. Further, by the structure of the problem, derivatives are not available for each design variable. Thus, a derivative-free approach is required. We chose to use a genetic algorithm because they are well suited to this type of problem. As outlined by Gen et al. [40], genetic algorithms are a powerful tool for solving complex, multi-objective optimization problems such as this. Genetic algorithms have been used in coverage path planning before, Yakoubi et al. details an approach to 2D coverage planning of a cleaning robot using a multi-objective genetic algorithm [41] and Hameed discussed using them for 3D terrain [42]. Similarly to [38], and [39], these approaches do not enforce loop closures and the objective is to visit all areas on the ground. Since the goal of this optimization is to plan a path that when flown will generate a high-fidelity 3D map of an indoor area with a forward-facing camera, viewing all the walls and obstacles is more important than covering all floor space. The genetic algorithm proposed in this chapter takes these differences into consideration. We found our genetic algorithm was able to effectively plan single agent and multi-agent paths in an arbitrary environment and satisfy the constraints of the problem.

The remainder of the chapter is organized as follows: Section 4.2 describes the method for simplifying the design space to create a reasonable problem for the genetic algorithm to solve. Section 4.3 describes the approach and architecture used to generate optimal paths for a single UAV, then the method is extended to multi-agent path planning. Results showing and evaluating the generated paths are presented in Section 4.4. Finally, conclusions are presented in Section 4.5.

4.2 Optimization Setup

Modeling a problem for optimization requires a balance between fidelity and tractability. Model accuracy is especially important because any inaccuracies in the model can lead to solutions that are not truly optimal in the physical system. However, if the design space is too large, or ill formed, optimization algorithms can fail to converge. This is especially problematic when derivatives are not available, or there are multiple local optima.

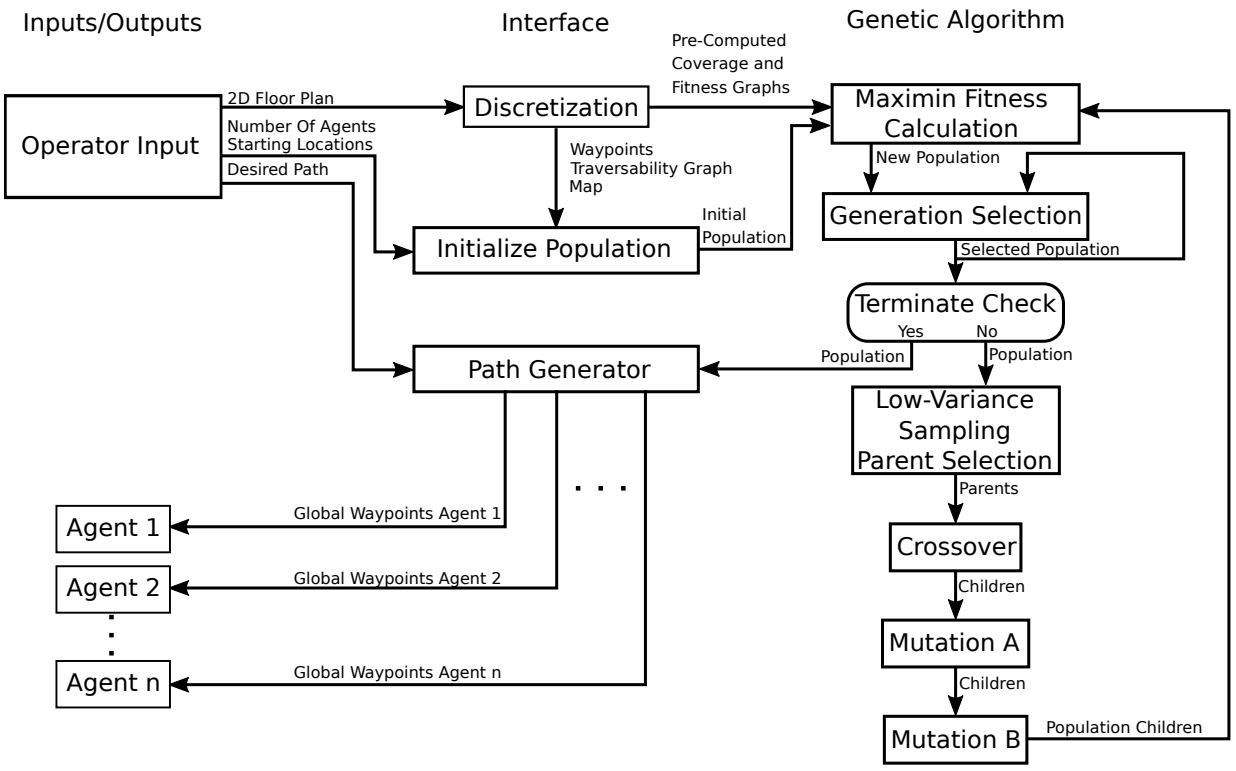


Figure 4.1: The network diagram for the proposed coverage planner showing the inputs and outputs of each component.

4.2.1 Problem Statement

The goal of our optimization is to generate paths through a simple floor plan with the intent of building a 3D map using a UAV and a forward facing RGB-D (color and depth) camera. We start with a simple 2D floor plan of the area of interest and a known scale of the map. Then, with as little human interaction as possible, generate a flight path for the UAV to map the building, maximizing coverage while minimizing flight time so that the path is feasible with real hardware. We do this first with a single agent, then extend it to work for several agents collaborating to build a single map. A network diagram with the layout of the proposed optimization is show in Fig. 4.1. Each component of the network will be explained in detail in later sections.

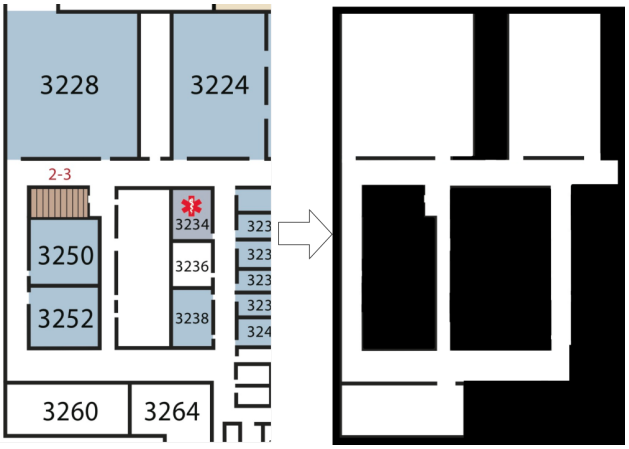


Figure 4.2: The initial 2D floor map of the area of interest and the segmented floor map reflecting explorable and occluded spaces.

4.2.2 Discretization

The open-ended nature of path planning can make optimization difficult. To simplify the design space, we discretize it by shrinking the map and constraining the path to only fly through predetermined waypoints before applying a genetic algorithm.

Map Generation

The first step to discretizing the design space is to simplify the 2D floor plan. The only part of discretization that must be done manually is segmenting explorable areas and occluded (inaccessible or uninteresting) areas on the 2D map. Fig. 4.2 illustrates how this segmentation works. The white space represents explorable areas and black space represents occluded areas.

Once this map is segmented, it is scaled to represent the minimum resolution desired in the optimization. This parameter is tunable to balance run time and fidelity. Once the map is properly scaled, we begin generating waypoints.

Waypoints

To further discretize and simplify the design space of the optimization, we place waypoints on the map and constrain the flight path to fly only through these waypoints. First, a grid is overlaid onto the map with a resolution of one half the width of the narrowest accessible hallway. This

ensures that there are waypoints in every hallway and room to make the space fully traversable. Next, all waypoints overlapping occluded space are removed.

After removing occluded waypoints, the remaining waypoints are further pruned and adjusted to better cover the space. To remove waypoints that would cause the UAV to collide with obstacles, we generate a safety buffer around all occluded space greater than half the width of the UAV. Any waypoint that falls within this safety buffer is moved perpendicularly away from the wall so that it is outside the safety buffer. The waypoints are then pruned to remove redundancy. The distance between adjacent waypoints is checked and any that are closer than the initial grid resolution are replaced with a single waypoint at the centroid of the cluster. This prevents overcrowding of waypoints in hallways and corners to make it simpler to generate flight paths in these areas. We create another waypoint grid that also removes waypoints that are not near walls to simplify the design space even more. Fig. 4.3 shows an example of this waypoint pruning process. We chose to do this simplified discretization approach rather than computing Voronoi graphs of the map as done by Sipahioglu et al. [43] which tend to collapse rooms down into few waypoints. We wanted to allow more thorough exploration in open areas such as large rooms.

4.2.3 Objective Functions

We split the optimization into two competing objectives: The first maximizes coverage of the given map, and the second minimizes flight time. The maximin fitness function, which will be described in Section 4.3.1, is used to determine the relative weighting between the two objectives.

Maximize Coverage

The first objective is to maximize coverage of the environment. To simplify the problem, we assume the UAV only flies forward. We use the camera's minimum and maximum viewable depths (d_{min} and d_{max}) and horizontal field of view (θ_{fov}) to construct the camera viewable area. We control the direction of the camera at each waypoint to face directly towards the next waypoint in the path. Then at each camera pose, we subtract all explorable area occluded by obstacles from the viewable area by dividing the viewable area into m sections and setting the distance of each section $d_{view,i}$ to

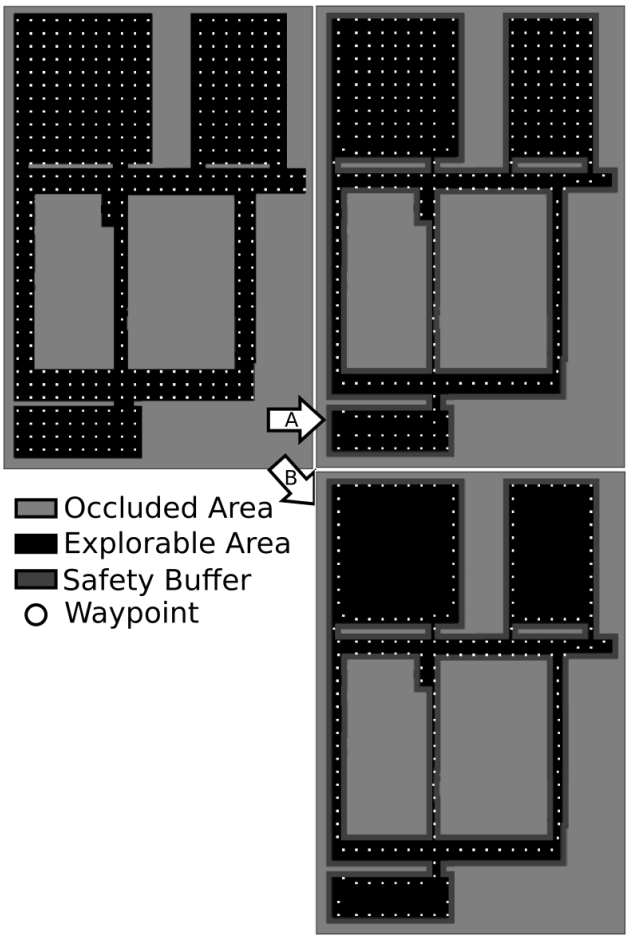


Figure 4.3: The initial waypoint grid with only occluded waypoints removed (upper-left), the modified waypoint grid after applying the safety buffer and anti-crowding (upper-right), and the modified waypoint grid after applying safety buffer, anti-crowding, and pruning waypoints not near walls (lower-right).

$$d_{view,i} = \max(\min(d_{max}, \min(d_{obs,i})), d_{min}) \text{ with } i = 0...m, \tag{4.1}$$

where $d_{obs,i}$ is an array composed of the distances to all obstacles in section i. As m increases, the coverage calculation becomes more accurate, but also more more computationally costly. An example of how the coverage is evaluated is shown in Fig. 4.4.

Next, we set the modified viewable area as explored at a waypoint. After repeating this process for every waypoint in the path, we calculate the percent of the explorable space in the map that is seen in that path. Mapping and visual odometry algorithms work better when the color and depth information from the camera is feature rich. Because of this, flying near walls is more

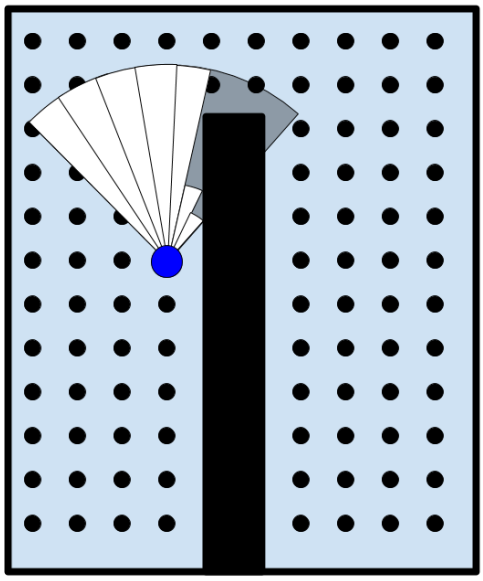


Figure 4.4: An example of how the coverage would be evaluated at a single waypoint with m=7. The blue dot is the agent at a waypoint, the white area is what would be counted as viewed and the grey area represents the maximum viewable area.

beneficial than flying in open space like the middle of large rooms. To capture this, we compute a second coverage value, which is the percent of safety buffer (computed when placing waypoints) seen by the path. We then perform an alpha blend (shown in equation 4.2) on the two coverage types to get a single value that is used as the first objective value

$$\mathscr{C} = \alpha_{cov}\mathscr{C}_{open} + (1 - \alpha_{cov})\,\mathscr{C}_{walls}, \tag{4.2}$$

where $\mathscr{C}$ is the value used for the coverage constraint, $\mathscr{C}_{open}$ is the coverage of the open explorable space, $\mathscr{C}_{walls}$ is the coverage of the safety buffer, and α_{cov} is a tunable parameter to weight the importance of walls versus empty space in the coverage calculation. See Fig. 4.5 for an example path and corresponding coverage from this step.

Minimize Flight Time

The second objective function minimizes flight time. To achieve this, we assume a constant-velocity flight and compute the total distance flown in each path. To avoid excessive meandering and incentivize flying straight lines, we also added a turning cost proportional to the change in

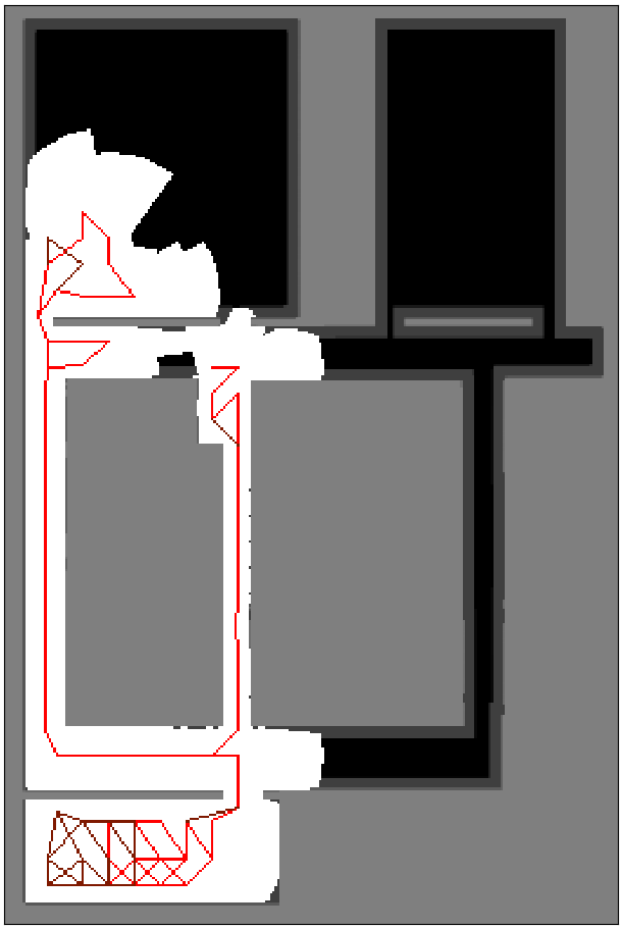


Figure 4.5: An example path and its corresponding coverage. The red line is the path flown, light grey represents obstacles, white represents explored area, black represents unexplored open space, and dark grey represents unexplored walls

heading angle between waypoints. We apply the cost as an exponent of the change in heading angle by $\frac{\pi}{2}$ to heavily penalize turning more than 90 degrees at any time as given by

$$flight\ time = \sum_{i=0}^{l} \left(d_{w,i} + \rho \left| \frac{\Delta\psi_i}{\pi/2} \right|^{\rho} \right), \tag{4.3}$$

where l is the total number of waypoints in a given path, $d_{w,i}$ is the distance between the current waypoint and the next waypoint, ρ is a tunable parameter to weight the cost of turning, and $\Delta\psi_i$ is the difference in heading angle between the current and next waypoint. This favors paths with fewer waypoints and less turning.

4.2.4 Traversable Graph

To save computation when generating or modifying paths, we pre-compute the traversability graph. The UAV is allowed to move to any waypoint defined in the traversability graph for its current position. For any given waypoint, we want to constrain movement only to adjacent waypoints that are not obscured by obstacles. To achieve this, we split up the space around each waypoint into octants. Then, we find the nearest waypoint in each octant and add an edge between it and the current waypoint. To prevent flying into obstacles, we prune the graph to remove any traversals that collide with obstacles.

Using this traversability graph, we generate feasible paths for the UAV. Given a desired path length and starting location, a random waypoint is selected from the traversability graph. The probability of choosing the next waypoint favors forward motion, with probability decreasing as turning angle increases to reduce meandering. The probability drops off according to a logarithmic scale with a tunable weight to control how much the path will favor forward movement. To avoid backtracking, we also encode a short-term memory so that the UAV will not return to a recent waypoint unless there is no other choice.

Because of the computational cost of computing the modified viewable area at each waypoint, we use the information in the traversable graph to precompute the modified viewable area for every possible waypoint traversal. We also do this for the distances and angles between traversable waypoints used in computing flight time. By pre-computing this information and storing it, the optimization is able to run significantly faster.

4.3 Approach

4.3.1 Single Agent Architecture

Performing coverage planning for a single agent is simpler than multi-agent planning. We begin by detailing the architecture for a single agent as follows. The methodology of genetic algorithms is to structure the optimization problem in a way that mimics the genetics and evolution process observable in nature. The variables of the optimization are stored in a chromosome data structure similarly to how genes are stored in nature. To enforce survival of the fittest, fitness pressure is applied through mate selection. Then to move from generation to generation, reproduction

mechanisms such as gene crossover and mutation are implemented on the new organisms introduced to the population from mating. Fig. 4.1 shows the flow of the genetic algorithm used in this research.

Chromosome

The chromosome defining each member organism in a generation has a minimum, initial, and maximum number of genes. Each gene represents one waypoint in a path and its value encodes a location in the grid of possible waypoints, as seen in Fig. 4.3. The first generation of parents is spawned by generating a random path within the 2D map. The first step to create this path is to seed the same initial waypoint to each organism in the generation's population, representing the idea that the doorway to enter the building is always in the same place. After the initial waypoint is seeded, the traversability graph is used to populate the rest of the path. All organisms in the first generation start with the same initial number of waypoints, however, the chromosome length is allowed to vary in crossover and mutation between the minimum and maximum number of genes. Once the chromosome is defined, both the coverage and flight-time objective values are computed for that organism. To start the first generation with only feasible organisms, any chromosomes that do not satisfy the starting constraints are discarded and a new organism is generated in its place. This process is repeated until the desired number of organisms has been added to the first generation.

Crossover

Given a set of two parents, crossover creates two children. Crossover occurs with a defined probability. If crossover does not occur, the children are clones of the parents. Otherwise, a search is done along the length of both chromosomes to determine where, if at all, both paths cross through the same waypoint. Next, one of the common waypoints is randomly chosen, and the tails of the chromosomes are swapped. See Fig. 4.6 for an example. If at any point a chromosome becomes longer than the maximum allowable length, it is truncated to the maximum length. Similarly, if the chromosome becomes shorter than the minimum allowable length, the chromosome is lengthened with a random feasible path up to the minimum chromosome length. To reduce the frequency of

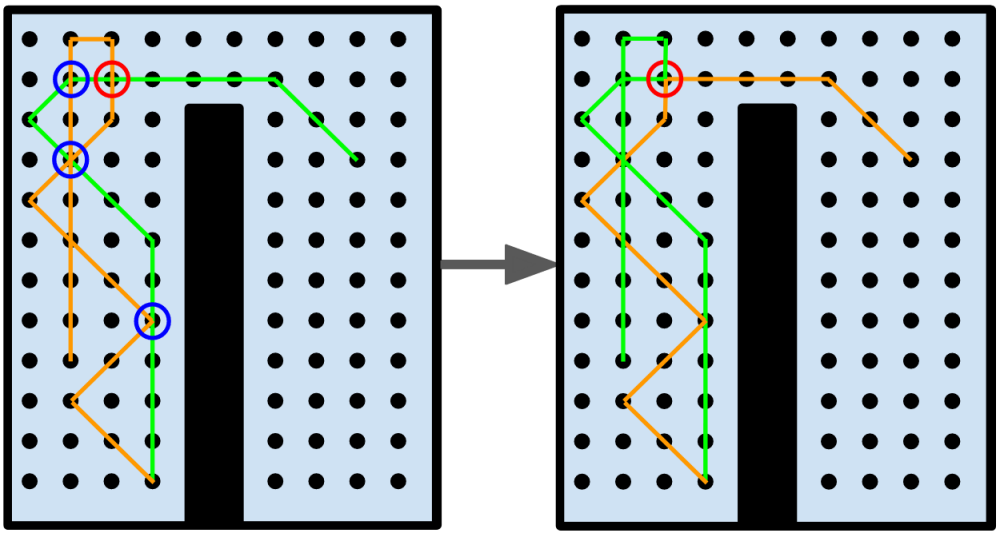


Figure 4.6: An example of the crossover mechanism. The orange and green paths represent the paths of the selected parents. First, common waypoints are found (marked in blue and red). Next, one of the common waypoints is chosen at random (red). Finally, the tails of the two paths are swapped, as shown on the right.

truncation and regeneration of paths, a parameter is set to only select points for crossover if they are within a set range of each other.

Mutation

After a new chromosome is generated, either by cloning or crossover, it is subjected to two different types of mutation ($\mathscr{A}$ and $\mathscr{B}$), each with a defined probability of occurring. The first type of mutation ($\mathscr{A}$) randomly selects both a random gene in the chromosome and a random length between the current and maximum chromosome length. Using the traversable graph, a new path is randomly generated from that gene onward, up to the randomly chosen length. See Fig. 4.7 for an example of this style of mutation.

The second mutation ($\mathscr{B}$) is intended to straighten and shorten the paths. A random segment (random in position, static in length) of the chromosome is selected. The Hamming distance from the first waypoint in the segment to all other waypoints in the segment, starting with the last point, is computed. If a point is found to be within two waypoints of the first point in the segment, all waypoints in the segment between those two are replaced with a shortcut, as illustrated in Fig. 4.8. This is only completed according to a defined sub-probability of mutation, which is

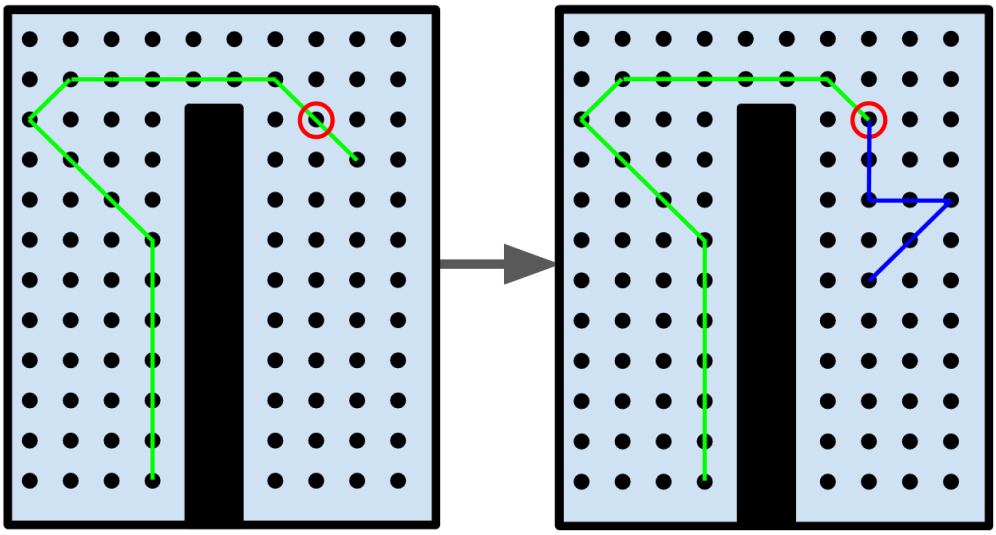


Figure 4.7: An example of the Mutation $\mathscr{A}$ mechanism. A random waypoint in the path is chosen (marked in red) and the tail is replaced with a random path of a random length.

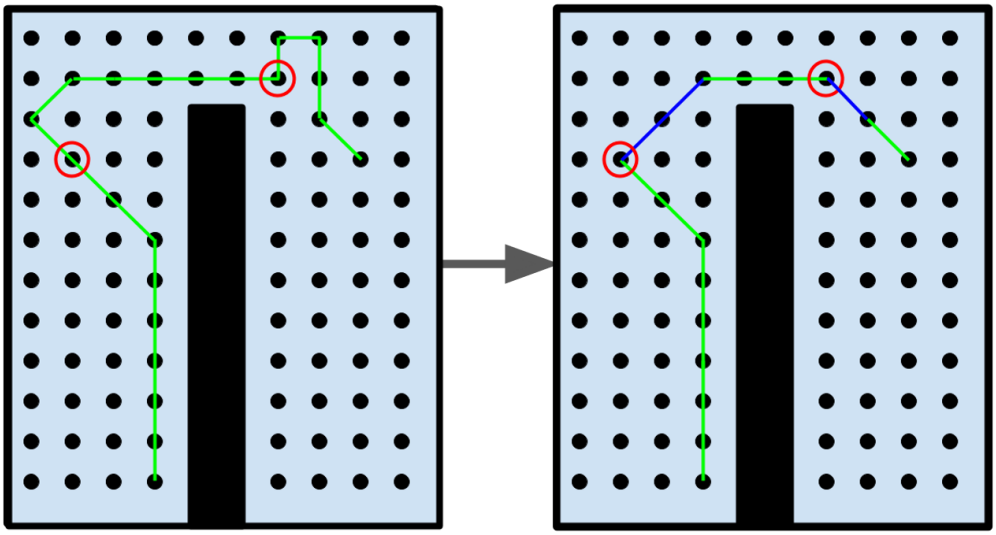


Figure 4.8: An example of the Mutation $\mathscr{B}$ mechanism. Multiple random points are chosen (marked in red). From each point, the path is searched up to a certain length to find places where shortcuts or alternate paths can be taken (shown as blue segments).

typically much higher than the overall mutation probability. The segment is left as is if no points are within a Hamming distance of two. This type of mutation is applied to the chromosome a specified number of times with the intent of finding multiple useful shortcuts in the path.

Constraints

Most of the constraints of the system are modeled into the discretization, traversability graph, and upper and lower bounds of the chromosome length as described in Section 4.2. However, constraining the minimum coverage and loop closures could not be applied directly to the model.

The coverage constraint was applied by adding to the cost to make it worse than the worst feasible design. Since we are using two objectives, we implemented this constraint by adding a large cost to the flight-time objective to guarantee that all organisms violating the constraint will be dominated by other designs (according to the maximin fitness).

Rather than start with a high constraint that would eliminate many of the original designs, we initialized the constraint with a low value. Over several generations we increase the constraint slowly to push the designs into the desired region of objective space. This also inadvertently increases the average flight time, but as the system continues to evolve, more efficient flight paths are found even with the new, strict coverage constraint and the average flight time decreases again.

We defined loop closures as anywhere that the path passes through the same waypoints in the same direction more than once as shown in Fig. 4.9. For a path to satisfy the loop closure constraint, it has to have a set number of loop closures. To avoid useless loop closures, a loop closure is only counted towards the constraint if it is separated by at least a defined number of waypoints. Similarly to the coverage constraint, we enforce this constraint by adding a large cost to the coverage, ensuring that all paths that violate the constraint are dominated by other designs.

As described in [44], constraints are applied to the fitness value. Because we have two objectives, and our non-modeled constraints directly relate to only one of the objectives, we implement these constraints on the objective values prior to computing the fitness value for the organism.

Maximin and Elitism

After all of a generation's children have been created, the fitness of the parents and children is computed by using the maximin fitness function as described by Balling [45]. In brief, the maximin scheme encourages a spread of designs across the Pareto front between competing objective values, with a target of minimizing the objectives. It does this by calculating the distance

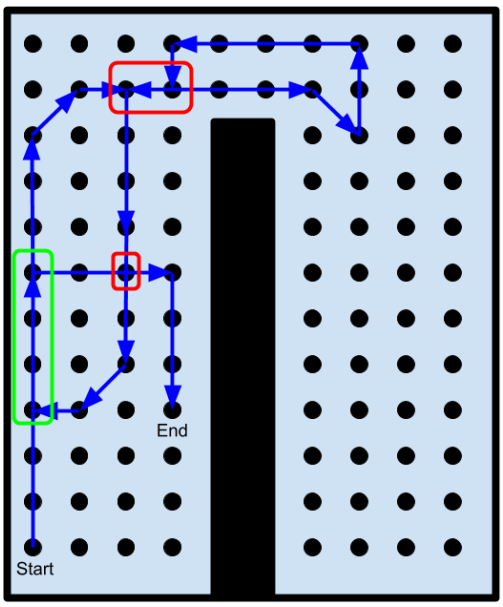


Figure 4.9: An example of single agent loop closure. A valid loop closure is marked in green where the path passes through the same waypoints in the same direction. Examples of times when the path crosses over its self but not loop closures are marked in red.

between each of the objective values of all the organisms, and finds the minimum difference of both objectives to each other. It then computes the maximum of all of these minima as the fitness value. Following the notation in [44], if f_k^i is the computed k^{th} objective value for organism i, the maximin fitness for organism i is computed by,

$$\mathit{fitness} = \max_{i \neq j} \left(\min_k \left(f_k^i - f_k^j \right) \right). \tag{4.4}$$

Once all parents and children have fitness values computed with respect to each other, the best N, comprising of the most negative fitness values, is selected as the next generation, where N is the size of a generation.

Low-Variance Sampling

We use a Roulette-style sampling method to pick parents in a way that gives organisms with better designs (determined by the maximin fitness) a higher probability of being selected as a parent for the crossover process. We employ a low-variance sampling technique as described in [46]. This method normalizes the population's fitnesses and treats them as a set of probabilities.

A naive approach randomly samples N times from the pool of parents according to their respective probabilities. To reduce the possibility of a parent being sampled disproportionately to its probability, we draw a single uniform random number r between 0 and $\frac{1}{N}$. The probabilities are then stacked up and compared to r. Whichever organism corresponds to the bin that r falls into gets sampled and $\frac{1}{N}$ is added to r and again compared to the stack of probabilities. This process is repeated N times. We also apply fitness pressure with the low-variance sampling with a tunable parameter γ with a value between 0 and 1. If $\gamma = 0$, no fitness pressure is applied, and the sample is random. If $\gamma = 1$, it will only select the most fit organism for the next generation.

4.3.2 Multi-Agent Architecture

When modifying the algorithm to generate multi-agent paths, the crossover and mutation mechanisms remained largely unchanged. Each agent was treated separately and crossover and mutation were applied independently on each path. The maximin fitness function, elitism, and low-variance sampling all remained unchanged; however the chromosome and constraints did need to be modified to accommodate the multi-agent architecture. The following section details these changes and additions to the architecture to generate multi-agent paths.

Chromosome

The chromosome was expanded to be of size $n \times l_{max}$ where n is the number of agents and l_{max} is the maximum path length where each row represents the path of its respective agent. The genes each row have the same representation as in the single agent case. To accommodate for paths that are shorter than the maximum length, any path shorter than l_{max} is padded with -1 at the end to populate any unused genes in the chromosome. We also allow for the agents to either start at the same waypoint as each other (if there were only one entry to the map), or different waypoints (if there are different entrances into the map).

Constraints

The majority of the architecture change between the single-agent and multi-agent case was in modeling the constraints for the multi-agent case. The constraint on coverage was applied to the

combined coverage of the full chromosome, not the individual paths. The loop closure constraint had to be expanded to handle loop closures between agents. For a single 3D map to be generated when dealing with more than one agent, the agents must have loop closures between them. For only two or three agents, this is simple to enforce. For four or more agents, graph theory is necessary to efficiently evaluate the connectivity of the paths.

To keep track of loop closures, we create a symmetric $n \times n$ loop closure matrix C with each element defined by

$$\begin{aligned} c_{ij} &= \text{loop closures between agent } i \text{ and agent } j \\ i, j &= 0...n, \end{aligned} \tag{4.5}$$

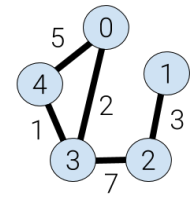
where n is the number of agents. The values along the diagonal of C are the number of loop closures the agent has with its own path. The off-diagonal terms store the total number of loop closures between agent i and agent j. The loop closure constraint from the single agent architecture is applied here using these values to ensure that paths have loop closure with themselves. We also added a new constraint to ensure the agents paths are connected enough to build a single map.

As described by [47], we create the $n \times n$ symmetric adjacency matrix A from the loop closure matrix C with each element defined by

$$\begin{aligned} a_{ij} &= \begin{cases} 1, & \text{if } (c_{ij} \geq 1 \text{ and } i \neq j) \\ 0, & \text{otherwise} \end{cases} \\ i, j &= 0...n \end{aligned} \tag{4.6}$$

which encodes which agents are directly connected to other agents Then we create the $n \times n$ diagonal degree matrix D defined by

$$d_{ii} = \sum_{j=0}^{n} a_{ij}, \;\; i = 0...n$$



$$C = \begin{bmatrix} 7 & 0 & 0 & 2 & 5 \\ 0 & 10 & 3 & 0 & 0 \\ 0 & 3 & 3 & 7 & 0 \\ 2 & 0 & 7 & 2 & 1 \\ 5 & 0 & 0 & 1 & 5 \end{bmatrix}$$

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix} \quad D = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{bmatrix}$$

Figure 4.10: An example connectivity graph with five agents and its respective loop closure matrix C, adjacency matrix A, and degree matrix D. The agent number is labeled on each blue node and the number of loop closures between agents is shown with the black edges between nodes.

so each element along the diagonal represents the number of agents that each agent is directly connected to through loop closure. Next, we define the graph Laplacian matrix L as

$$L = D - A \tag{4.7}$$

which encodes the connectivity of the agents. We can determine whether there are enough loop closures between agents to connect all paths into a single map by looking at the eigenvalues of L. The number of zero eigenvalues represents the number of separate connected components in the graph. So, if all agents are connected to each other, there will be only one zero eigenvalue. If there are not sufficient loop closures to combine into a single map, there will be more than one zero eigenvalue. We use this to set the last constraint used for the multi-agent architecture. There must be exactly one zero eigenvalue of L to satisfy the constraint. An example connectivity graph for a system with five agents is shown in Fig. 4.10

4.4 Results

The designer is required to set many parameters for the genetic algorithm to work as derived in [44]. The parameters we used, along with parameters specific to our implementation, are outlined in Table 4.1.

Table 4.1: Parameters used to obtain optimization results.

Parameter	Value
Generation Size (Organisms)	100
Crossover Probability	0.7
Crossover Time Threshold (Waypoints)	70
Mutation $\mathscr{A}$ Probability	0.3
Mutation $\mathscr{B}$ Probability	0.3
Number of Mutation $\mathscr{B}$ Instances	20
Mutation $\mathscr{B}$ Acceptance Probability	0.8
Mutation $\mathscr{B}$ Search Distance (Waypoints)	5
Objective Scaling Factor: Flight Time	0.0001
Objective Scaling Factor: Coverage	-1.0
Coverage Blending Factor α_{cov}	1.0
Coverage Sections	15
Minimum Chromosome Length (Waypoints)	75
Maximum Chromosome Length (Waypoints)	250
Pixel Scale (meters/pixel)	0.15
Minimum Coverage Constraint Start	0.3
Minimum Coverage Constraint End	0.8
Minimum Coverage Constraint Aging (Generations)	60
Loop Closure Separation Threshold (Waypoints)	30
Minimum Loop Closures	2
Path Memory (Waypoints)	10
Turning Cost Factor ρ	8.0
Log Scale Turning Weight	0.7
Fitness Pressure Factor	0.5

4.4.1 Single Agent

Fitness

Fig. 4.11 shows how the objective values of the organisms evolved from generation to generation. Four different snapshots are shown: the first, randomly generated set of parents, the state of the generation at about a third and two-thirds of the way through the selected number of generations, and last, the final result of the progression of 1000 generations. Fig. 4.12 shows how the average negative coverage and average flight time progresses over the course of the 600 generations. The coverage flattens out temporarily, but then continues to drop after 400 generations. The interesting result displayed is in the flight-time objective value history. After about 30 gen-

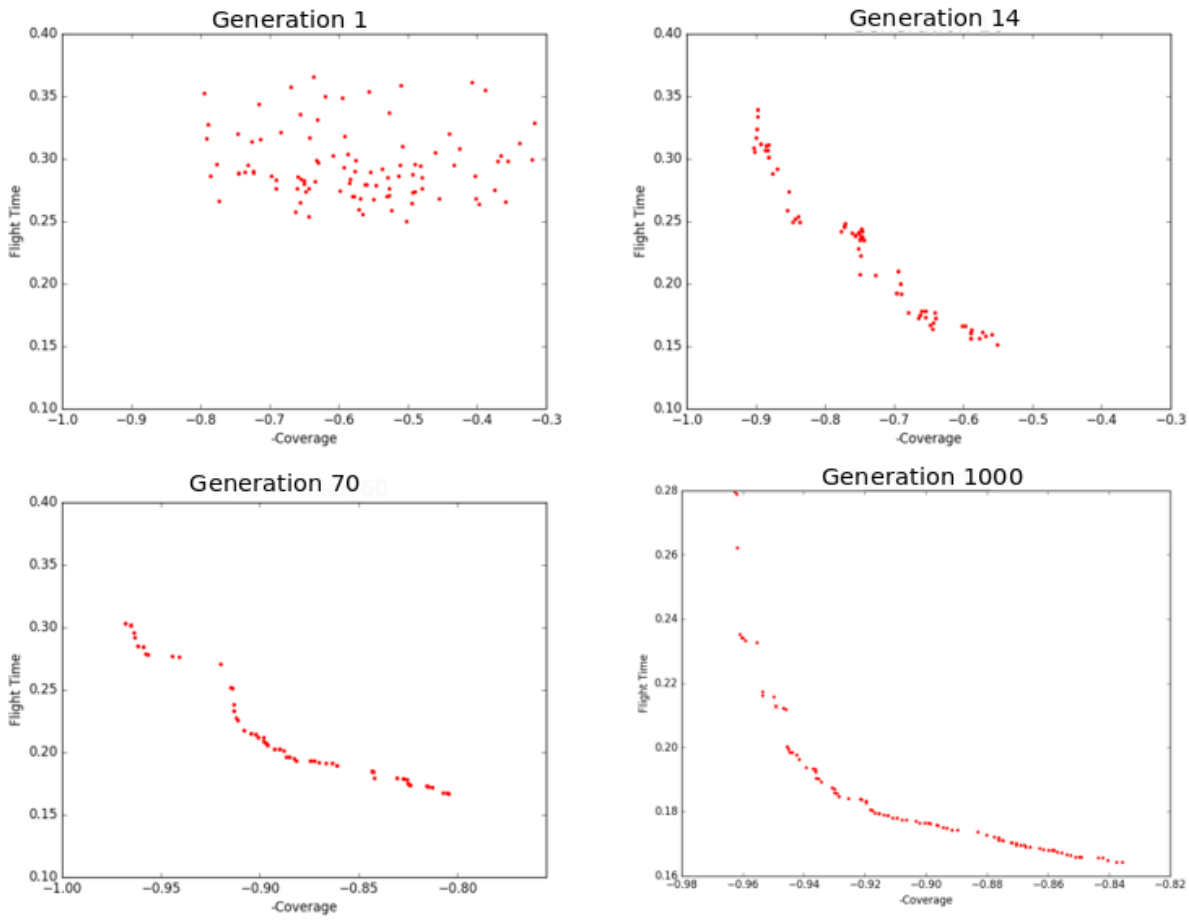


Figure 4.11: History of the Pareto front across several generations. The dynamic constraint placed on coverage prioritizes high coverage but also induces high flight time for the first 1000 generations. Afterward, the algorithm can work with high coverage organisms, optimizing them for lower flight times, while maintaining their high coverage properties.

erations, the flight time starts to rise again. This is due to the dynamic constraint that invalidates all organisms with coverage below the defined threshold at the defined generation number. Once the coverage constraint settles to a constant value, the organisms with good coverage dominate the generation population and the algorithm successfully lowers the flight time again to represent a useful Pareto front that has a good spread of options from which to choose.

Paths Generated

Our algorithm was successful in generating useful paths, as seen in Fig. 4.15. The single agent path covers over 95% of the walls and wastes little time returning to parts of the map that

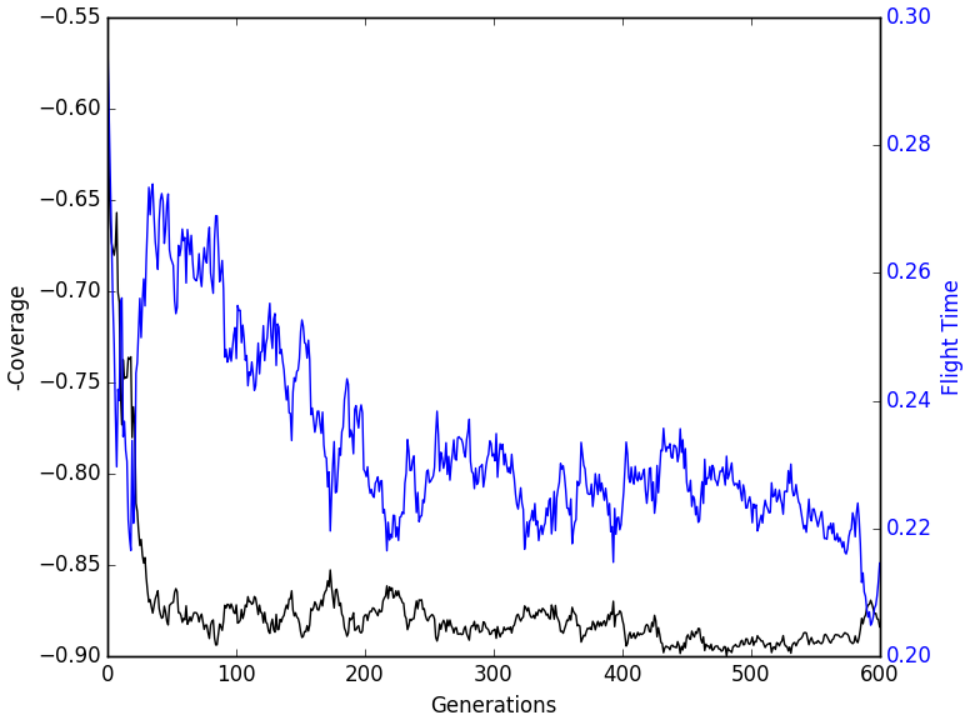


Figure 4.12: A plot of the average objective values over 600 generations. Note how the average flight time initially decreases then sharply increases again, as the rolling constraint on coverage is applied. As the generations are then allowed to evolve, the flight time settles back down below the previous minimum.

have already been visited. The only exception is generating a useful loop closure in the center hallway that would help in generating a 3D map.

4.4.2 Multiple Agents

Fitness

Similar to the single agent case, our algorithm was successful in exploring the design space to generate paths that had both good coverage and low flight times. Fig. 4.13 shows the progression of the average objective values over 500 generations for a case with six agents on a much larger map. The six agents began distributed throughout the map, so the initial average coverage is much higher than the single agent case, so the average flight time does not have the same spikes as with the single agent case.

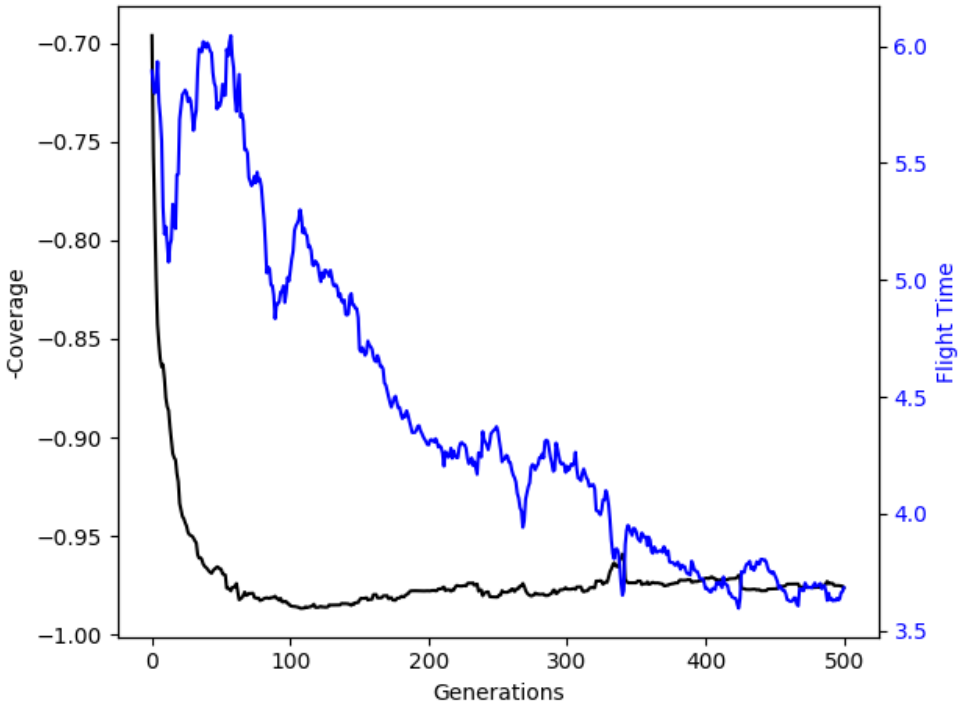


Figure 4.13: A plot of the average objective values over 500 generations for six agents on a large map. Note how the average flight time initially decreases then sharply increases again, as the rolling constraint on coverage is applied. As the generations are then allowed to evolve, the flight time settles back down below the previous minimum.

Paths Generated

Our algorithm was also successful in generating paths for two and three agents on the same map as shown in Fig. 4.15 and for six agents on a larger, more complicated map as shown in Fig 4.14. In the multi-agent case, the paths were not only able to create loop closures with themselves, they were able to connect with each other enough to create a single connected map between all agents.

4.5 Conclusions

We were able to successfully generate paths with sufficient coverage to generate a high-fidelity map of the environment both for the single-agent case and the multi-agent case. These paths also covered the environment efficiently to decrease the flight time necessary to map the

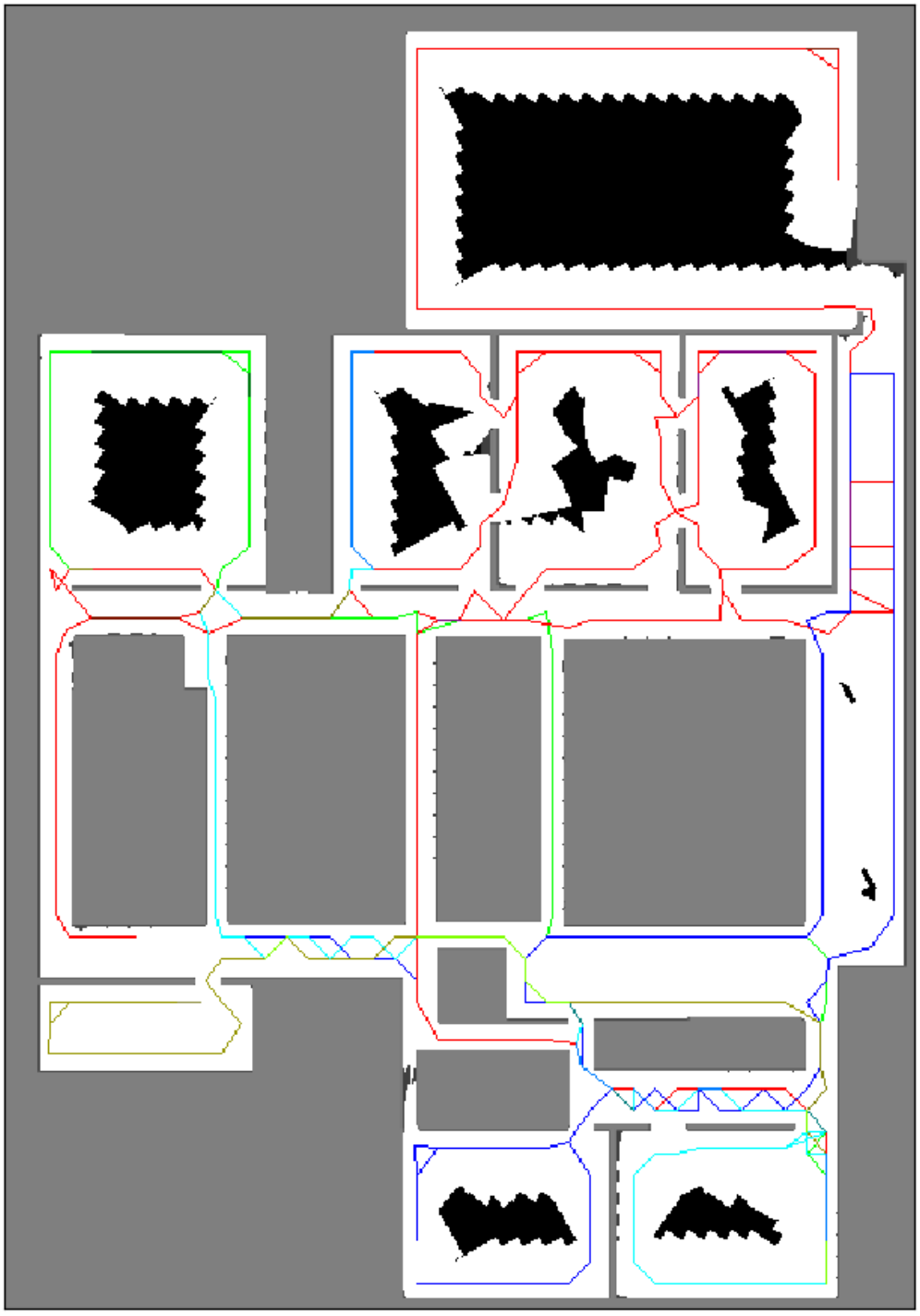


Figure 4.14: An example path generated for six agents on a larger, more complex map. Each agent's path is represented by a different color. Darker colors indicate loop closure. Note the near perfect wall coverage and loop closures between all agents.

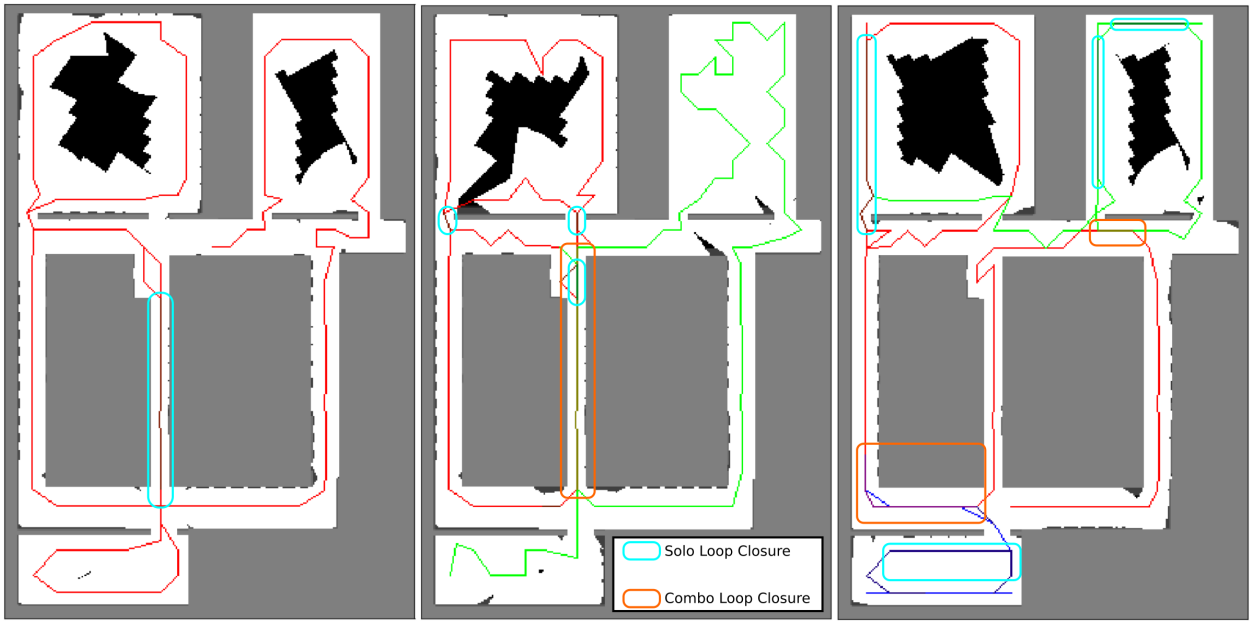


Figure 4.15: Example of final paths generated by the genetic algorithm for a single agent (left), two agents (center), and three agents (right) on the same map. These paths show excellent wall coverage and loop closures both with themselves (shown in teal) and with each other (shown in orange) to generate a single well-structured map.

environment. The results show that with a properly formulated genetic algorithm, efficient area-coverage paths can be generated to assist in building 3D maps in complex environments.

CHAPTER 5. CONCLUSIONS AND RECOMMENDATIONS

5.1 Research Conclusions

In completing this research, a full solution to mapping complex GPS-denied environments with multiple UAVs working collaboratively was explored. The focus of the research was to develop a method to map a partially known environment when a two-dimensional floor plan exists and is available. This research involved the development of new algorithms as well as leveraging and expanding upon multiple existing approaches to the problem.

As a part of this solution, a planning algorithm was developed that successfully generates flight paths with sufficient coverage and loop closures to map an environment with multiple agents. A low-level reactive path planner was also developed to work with an obstacle avoidance algorithm to safely navigate complex unknown environments to accommodate for missing information from the provided floor plan. The functionality of the relative navigation framework was improved to allow navigation in previously unexplored areas. Lastly, the functionality of RTAB-Map was extended to allow multiple agents to simultaneously map an environment and combine their maps into a single one.

5.2 Future Research

Although this research was successful in generating a combined map from multiple UAVs, there are areas where more research focus would help to improve the overall solution.

5.2.1 Planning Improvements

The two different levels of planning were able to successfully complete their individual objectives: creating a high-level path to achieve good coverage and loop closures, and creating a low-level path that is able to avoid obstacles in real-time. There is, however, still a gap between

the two planners. After the high-level path planner generates a global path, there is nothing in place to connect the map being generated with the two-dimensional floor plan used to generate the paths. This results in the global estimation drifting far enough from truth at times that it is not possible to fly the global paths generated. To close the gap between the planners, functionality will need to be added to the planning and estimating process to allow the map being built by the UAVs to loop-close with the floor plan used to generate the path. This will give pseudo-global measurements that would be able to correct the drift and inaccuracies of the estimator and enable global path following.

The current implementation of both the high-level and low-level planners generate way-points for the UAV to follow. This results in the planners having less control over the heading angle of the UAV than a trajectory planner would. The loose control of the heading angle results in the map having some unseen areas because the UAV was not facing the direction that was intended. Expanding the functionality of the planners to generate a trajectory for the UAVs to follow would ensure that the camera is facing in the optimal direction at all times and improve the final quality of the map by reducing the number and size of unseen areas in the environment.

5.2.2 Mapping Improvements

The current implementation of simultaneous mapping generates a combined map in near real time, but the combined map is only used by the base station computer. The next step to improve the map merging process will involve sending the combined map information and UAV locations back to the UAVs. This will enable more intelligent planning and ensure that the UAVs can safely avoid each other while flying. Also streamlining and parallelizing the map merging process to allow for full real-time map merging will enable the combined map to be used for reactive path planning and exploration.

5.2.3 Exploration Improvements

Lastly, the current solution to the multi-agent collaborative mapping problem assumes that a two-dimensional floor plan will be provided. This limits the functionality to a small set of use cases. Future research to develop and implement an exploratory planner to autonomously explore

an entirely unknown environment would enable many other use cases for this technology. It would also increase robustness for the current use cases to handle unexpected differences between the given map and the actual environment.

REFERENCES

[1] 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. 1, 9, 25

[2] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, "RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments," *The International Journal of Robotics Research*, vol. 31, no. 5, pp. 647–663, 2012. 3

[3] Intel, "Intel® RealSense depth camera D435 - Intel® RealSense depth cameras." [Online]. Available: https://click.intel.com/intelr-realsensetm-depth-camera-d435.html 3, 14, 20

[4] Velodyne, "Velodyne LiDAR." [Online]. Available: http://velodynelidar.com 3

[5] Slamtec, "RPLIDAR A2 - Slamtec." [Online]. Available: https://www.slamtec.com/en/Lidar/A2 3, 14

[6] S. Saeedi, M. Trentini, M. Seto, and H. Li, "Multiple-robot simultaneous localization and mapping: A review," *Journal of Field Robotics*, vol. 33, no. 1, pp. 3–46, 2016. 3

[7] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous robots*, vol. 34, no. 3, pp. 189–206, 2013. 4

[8] 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. 4, 10, 23, 32

[9] ——, "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. 4, 10, 23, 32

[10] ——, "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. 4, 10, 23, 32

[11] 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, 20

[12] 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. 5, 19

[13] D. Zou and P. Tan, "CoSLAM: Collaborative visual SLAM in dynamic environments," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 2, pp. 354–366, 2012. 5

[14] P. Schmuck and M. Chli, "Multi-UAV collaborative monocular SLAM," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 3863–3870. 5

[15] M. Bryson and S. Sukkarieh, "Co-operative localisation and mapping for multiple UAVs in unknown environments," in *2007 IEEE Aerospace Conference*. IEEE, 2007, pp. 1–12. 5

[16] 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. 6, 19

[17] OSRF, "ROS.org — Powering the world's robots." [Online]. Available: https://www.ros.org/ 9

[18] R. Leishman, J. Macdonald, T. McLain, and R. Beard, "Relative navigation and control of a hexacopter," in *2012 IEEE International Conference on Robotics and Automation*. IEEE, 2012, pp. 4937–4942. 9

[19] 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. 9, 24

[20] R. C. Leishman and T. W. McLain, "Multiplicative extended Kalman filter for relative rotorcraft navigation," *Journal of Aerospace Information Systems*, vol. 12, no. 12, pp. 728–744, 2014. 9

[21] 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. 9, 25

[22] 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. 9, 25

[23] OSRF, "Gazebo." [Online]. Available: http://gazebosim.org/ 10, 33

[24] 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. 10, 33

[25] Benewake, "TFmini." [Online]. Available: http://benewake.com/en/mfeedback.html 14

[26] B. Forooghi, "F4 Advanced Flight Controller - (MPU6000, STM32F405)." [Online]. Available: https://www.getfpv.com/f4-advanced-flight-controller.html 15

[27] Ubiquiti, "Ubiquiti - Bullet AC." [Online]. Available: https://www.ui.com/airmax/bullet-ac/ 15

[28] ——, "Ubiquiti - Rocket® AC." [Online]. Available: https://www.ui.com/airmax/rocket-ac/ 15

[29] Gigabyte, "GB-BRi7-8550 (rev. 1.0) — Mini-PC barebone (BRIX) - GIGABYTE U.S.A." [Online]. Available: https://www.gigabyte.com/us/Mini-PcBarebone/GB-BRi7-8550-rev-10{#}ov 17

[30] 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. 19

[31] 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. 21

[32] 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. 26

[33] 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. 27

[34] N. J. Nilsson, *The quest for artificial intelligence: A history of ideas and achievements*. Cambridge, MA: Cambridge University Press, 2009. 27

[35] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Tech. Rep., 1998. 27

[36] 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. 27

[37] R. W. Beard and T. W. McLain, *Small unmanned aircraft: Theory and practice*. Princeton university press, 2012. 28

[38] H. Choset, "Coverage for robotics–a survey of recent results," *Annals of mathematics and artificial intelligence*, vol. 31, no. 1-4, pp. 113–126, 2001. 39, 40

[39] A. Bircher, M. Kamel, K. Alexis, M. Burri, P. Oettershagen, S. Omari, T. Mantel, and R. Siegwart, "Three-dimensional coverage path planning via viewpoint resampling and tour optimization for aerial robots," *Autonomous Robots*, vol. 40, no. 6, pp. 1059–1078, 2016. 40

[40] M. Gen and R. Cheng, *Genetic Algorithms and Engineering Optimization*. Wiley, 2000. 40

[41] M. A. Yakoubi and M. T. Laskri, "The path planning of cleaner robot for coverage region using genetic algorithms," *Journal of innovation in digital ecosystems*, vol. 3, no. 1, pp. 37–43, 2016. 40

[42] I. A. Hameed, "Intelligent coverage path planning for agricultural robots and autonomous machines on three-dimensional terrain," *Journal of Intelligent & Robotic Systems*, vol. 74, no. 3-4, pp. 965–983, 2014. 40

[43] A. Sipahioglu, G. Kirlik, O. Parlaktuna, and A. Yazici, "Energy constrained multi-robot sensor-based coverage path planning using capacitated arc routing approach," *Robotics and Autonomous Systems*, vol. 58, no. 5, pp. 529–538, 2010. 43

[44] A. R. Parkinson, *Optimization-based design version 3.2*. BYU Academic Publishing, 2019. 51, 52, 55

[45] R. Balling, "The maximin fitness function; multi-objective city and regional planning," in *International Conference on Evolutionary Multi-Criterion Optimization*, Springer. Springer, Berlin, Heidelberg, 2003, pp. 1–15. 51

[46] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. MIT press, 2005. 52

[47] E. W. Weisstein, "Laplacian matrix." [Online]. Available: http://mathworld.wolfram.com/LaplacianMatrix.html 54