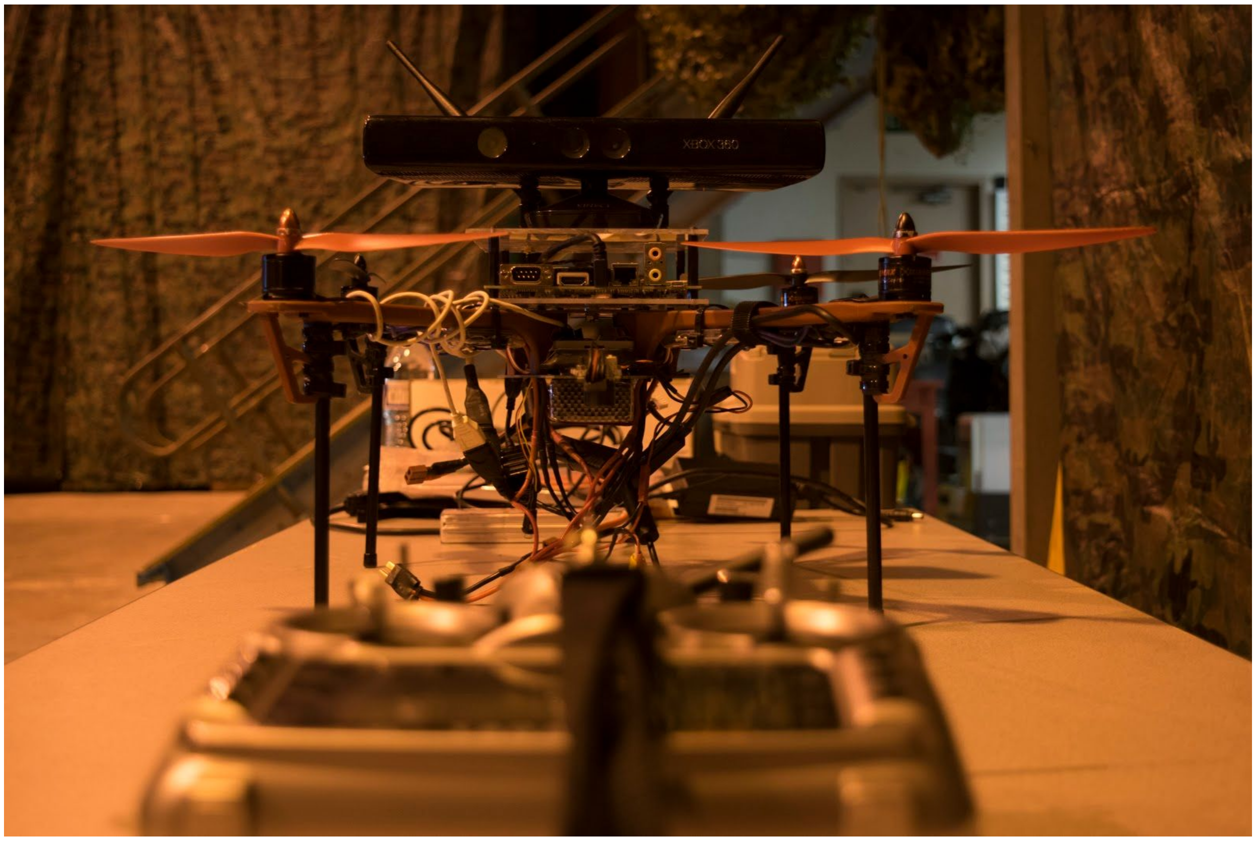

Henry Gaudet, Blaid Burgess, Michael Park, and Yasy Mandi

SEARCH AND IDENTIFICATION DRONE

FINAL REPORT

Table of Contents **2**

1. Abstract

In this paper we discuss a drone that can semi-autonomously navigate its way through an indoor environment while simultaneously mapping the area and sending the data to a laptop ground station over a wireless network connection. We discuss the key details of the drone's hardware as well as all of the necessary software. We present the challenges we faced trying to build full functionality into the drone with only ten weeks of working time, the problems we overcame as well as the problems we did not overcome, and the schedule we followed to achieve our goal.

2. Introduction

Our project was to build an indoor quadcopter with mapping capabilities. The intended use of this drone was for military or police missions that required surveillance of dangerous areas. Our drone replaces the need for people to physically enter a building or other enclosed space without knowledge of what is inside and reduces risk to bodily harm or death. We achieve this goal by implementing a simultaneous localization and mapping (SLAM) algorithm to provide the robot with an estimate of its xyz location and xyzw orientation within a local reference frame and use that estimate to move along a set of waypoints. In this way, the robot is able to create an internal visual representation of its environment and semi-autonomously navigate its way through.

Our team was split up into two teams to work on these two problems:

- Blaid and Yasy worked on the SLAM algorithm and mapping functionalities
- Henry and Michael worked on making sure the drone had flight capabilities and could autonomously follow a path

Implementing the SLAM algorithm required the use of a highly integrated software that allowed every part of the drone, both hardware and software, to be able to seamlessly communicate with one another without sacrificing individual function or feature. A lot of the research was thus focused on finding these matching parts. The software the team ended up going with is as follows:

- ROS: Robot Operating System
- OpenKinect Library: allowed communication between the Kinect sensor and the companion computer running Ubuntu 114.04
- Kinect Fusion: gave us access to the information provided by the sensors on the Kinect
- Point Cloud Library: an all-inclusive library of numerous point cloud systems with built in functions for easy use
- RTAB Map: a SLAM package included in ROS that creates a depth map from point cloud input
- Ardupilot: a navigation library with functions that allow for autonomous flight

Building the drone and achieving autonomous flight required integration of all the hardware components and facilitating effective communication among the devices. This required extensive research about each of the hardware components we used and clever engineering to work around obstacles with device driver compatibility. We used the following hardware in our build:

- DJI Flamewheel F450 quadcopter frame
- Floureon 11.1V 3S Lipo Battery 5500mAh 35C
- Intel 3160.HMWG.R Dual Band Wireless AC 2.4 & 5.8 Ghz B/G/N/AC
- 4x Crazepony Emax Mt2213 935kv Brushless Motors
- 4x 20A ESCs
- Pixhawk 2.4.8 Flight Controller w/ APM 2.4.8
- NVIDIA Jetson TK1 w/ L4T 21.5
- XBox 360 Kinect Sensor
- Maxbotix MB1242 Ultrasonic Range Finder

3. Technical Material

This project was split up into two teams; the quadcopter hardware team made sure the drone read in coordinates and flew the correct way, and the software team implemented the necessary libraries and algorithms required for it to operate SLAM. Testing was done incrementally and results were intuitive. As we implemented various features (mapping an area, having the drone follow coordinates, taking in measurement readings) we could just see what the drone output and checked for correctness.

The hardware team focused on these main things:

- Creating an interface between the Jetson and the Pixhawk
- Implement autopilot software on the drone
- Implement path planning algorithm for the drone to read in coordinates and autonomously follow it

The software team focused on these main things:

- Finding a platform that could communicate with the operating system on the drone
- Importing the proper libraries that had SLAM capabilities and making sure they worked with our equipment
- Implementing the SLAM algorithm
- Creating a point cloud depth map
- Allowing the drone to localize itself and read in a set of coordinates to follow

Hardware Configuration

Henry and Michael assembled the drone and found which libraries we would need to use to make our project work. The components of the drone were four high power motors, four electronic speed controllers to facilitate variable motor speed, and one radio frequency receiver for wireless communication. For the embedded components of our robot, we used an NVIDIA Jetson TK1 as the GPU and companion computer to the Pixhawk the flight controller, and a

Microsoft Kinect sensor for its depth imaging capabilities. Using these components we achieved stable piloted flight.

Navigation

Once we achieved piloted flight, we focused on trying to make the drone follow a set of waypoints, but this became a significant obstacle. The Ardupilot software API allows the programmer to provide a list of waypoints to the flight controller and the controller will then maneuver the robot through those set of coordinates, so long as the waypoints consist of GPS coordinates. For this project we wanted the robot to navigate indoors so we couldn't use GPS coordinates. The software exposes several flight modes, but allows the waypoint-following behavior only in "guided" mode, which can only be entered if the device detects an attached GPS module or optical flow sensor. We had neither of these pieces of hardware so we instead did some research into the source code and discovered that the mechanism by which the drone follows the GPS waypoints is by using an extended kalman filter (EKF). This meant that the actual code responsible for the path following behavior was not dependent on coordinates in GPS form, but could in fact be local coordinates within a consistent frame of reference. Our solution was to spoof the GPS coordinates with coordinates we received from our SLAM algorithm's 3D pose estimation, using the x- coordinate for latitude and y-coordinate for longitude, and maintaining a local north-east-down (NED) reference frame with the robot's starting position as (0,0,0). In this way, we were able to get the drone to follow a short list of simple waypoints without an attached GPS or optical flow module.

Ultimately the drone was not able to follow the waypoints in a robust fashion, i.e. it would successfully follow waypoints using our technique about 30% of the time. We believe this low success rate is due to errors in the pose estimation received from the SLAM algorithm compounded with over corrections in the controller's attitude adjustments leading to an unstable platform an ultimately a crash. We further believe that installing an optical flow sensor to get a second estimate of the robot's 2D x-y position would allow for a better and more stable localization estimate and would decouple the flight controller's localization estimate from the estimate computed by the SLAM algorithm.

SLAM - Mapping and Localization

Blaid and Yasy worked on finding the correct interface and operating system that was compatible with the drone. Since we used an NVIDIA Jetson, which does not have as much software support available, the task was difficult. After weeks of research and failures, we found and used ROS, the Robot Operating System. ROS provided us with an all-inclusive system that allowed everything to properly communicate with each other, made compiling and debugging very simple, and gave us access to all of the necessary libraries.

Once the operating system was set up we imported OpenKinect Library, Kinect Fusion, Point Cloud Library, and RTAB Map. The OpenKinect Library enabled us to to connect the Kinect to Windows and allow communication between them. Kinect Fusion let us use the Kinect sensors to allow for 3D scanning, which we later used in making our point clouds. The Point Cloud Library was the biggest piece of our project as it made it very easy to create various point clouds, with various uses and functionalities. Finally, the RTAB Map gave us an easy interface to upload our point clouds and create the depth map.

We wrote a script that automatically ran all of the necessary files and commands and made it very easy to launch the program. Once everything was launched and the RTAB Map was running, the program automatically localized the drone and gave it a good sense of where it was within the map, marking it with various localized, blue vectors. This let the drone figure out where it could go. From there, we had access to a lot of useful information, such as pose estimates, that could be used to make the drone autonomous.

4. Milestones

Week 1 - Week 3

Setup

- Coordinate and finalize schedules and meeting times with.
- Finalize the specification details of the project.
- Research and find what materials and which software we will need for our project.
- Assemble hardware.

Week 4

Interfacing with PCL and Kinect sensor - Blaid and Yasy

- Interface with the kinect to read in sensor data, use the appropriate APIs to access the data and create a point cloud map, write the map to a file, open it for visualization.
- Problem: we kept running into compile and runtime errors trying to compile software and dependencies from source for our linux environment. We had to try many different packages carrying point cloud libraries and slowly got rid of our compile errors one by one. Ultimately we were successful using ROS as a package manager instead of compiling separate software from source.

Interface Jetson and Pixhawk - Michael & Henry

- Create an interface between the Jetson and the Pixhawk so that 3D coordinates created through the SLAM algorithm can be passed to the flight controller, enabling the robot to move about.

Week 5

Working with point clouds and managing memory - Blaid and Yasy

- Stitch together the various point clouds generated by the Kinect using computer vision registration techniques to create a complete map.
- Problem: We were able to create and save different point clouds taken from different angles of the same space, and were able to successfully save them. However, we were unable to open up the files to view them and begin stitching them together. This caused us to have to look for a different way to implement what we were doing and we lost a week's worth of time.

Indoor autopilot - Michael and Henry

- The Ardupilot software wants absolute 3D coordinates (i.e. GPS), so a solution needs to be developed: either a fix using existing autopilot functionality or an entirely new indoor autopilot system using attitude control functionality exposed by the Ardupilot API.

- Test Flight #1: Failed Mission

Week 6 - Milestone Mark

Create depth map - Blaid and Yasy

- Continue working on point cloud registration
- Deliverable: A dynamically created 3D virtual depth map using Kinect sensor.

Static path following - Michael and Henry

- Complete autopilot software.
- Deliverable: Autonomously navigate a fixed set of 3D points.
- Test Flight #2: Failed Mission
- Revision: We ran into issues when trying to send the pixhawk waypoints in an indoor environment. We figured a work around by sending spoof GPS coordinates, but this delayed our static path following milestone.

Week 7

Drone localization - Yasy

- Use the depth map, gyro, and altitude readings to help the drone create a pose estimation within the environment.

~~Dynamic map loading - Blaid~~

- Make efficient use of memory by storing parts of the map that are unnecessary to disk to be loaded when needed again.
- Revision: Due to the previously mentioned setbacks we faced with software compatibilities, we ran out of time to implement this module.

~~Dynamic path planning - Michael~~

- Develop a path planning and exploration algorithm (i.e. maze-solving) so that the drone can autonomously explore its environment.
- Revision: Due to the previously mentioned setbacks we faced with test flight failures, we ran out of time to implement this module.

Week 8 - Milestone Mark

Drone localization - Blaid and Yasy

- Deliverable: Drone will be able to add an accurate pose estimation to the depth map.
- Revision: We came to realize that the localization data we received from the Kinect was not enough to effectively keep the drone stable. Thus we decided to acquire an optical flow sensor to aid with this, however, we ran out of time to integrate the component.

~~Dynamic path planning - Michael and Henry~~

- Deliverable: Drone will be able to create a planned path and add it to the depth map.
- Test Flight #3: Mission Success
- Revision: Since we didn’t have time to create a path planning algorithm, we were unable to finish this module.

Week 9

Finish any unfinished features

- Begin integrating everything and testing.
- Depending on time constraints, implement optional features.

Week 10 - Milestone

Complete all Presentation material.

- Create a presentation video.

Deliverable: The finished Drone capable of performing SLAM in an indoor environment.

- Have the drone all put together and working.
- Make sure all aspects are integrated and functional .
- Work on fixing any last minute bugs or feature modifications.

5. Conclusion

We were successful in building a drone that could map an indoor area and send the information over a wireless connection to a laptop ground station. We were unable to achieve fully autonomous flight but were able to achieve semi-autonomous flight albeit with a low success rate. We have a strategy to correct this problem and believe it will result in a higher success rate while following a list of waypoints consisting of local coordinates.

Future Work

The future of the project will likely see the Kinect replaced with a stereo vision system. While the depth-mapping technology is attractive for this application, what we found was that our sensor consumed a large amount of power reducing our flight time significantly. We have researched alternatives such as the Intel Realsense line of products, but all of these sensors are limited by their short-range sensing capabilities. In order to be a precise and effective mapping and aerial navigation platform, the robot would need longer range visual detection capabilities than are available with the current generation of depth-sensing cameras (at least at a reasonable price). We therefore conclude that a stereo vision system gives us an advantage in choosing the best imaging sensor with enough resolution to implement our software effectively at ranges up to 12 meters.

As discussed above, the robot will likely see the addition of an optical flow sensor installed on its underside facing downward. The sensor continuously calculates a velocity estimate based on the speed it moves over the ground. This extra information allows for a second 2D position estimate that can be used with the estimate from the SLAM algorithm and input into a kalman filter to get a more accurate 2D estimate, diminishing the need for as severe over corrections as we experienced using only the SLAM pose estimate.

Finally the robot will likely see an upgrade from an 11.1V circuit to a 14.8V circuit with several voltage regulators. Using our 11.1V circuit led to some unexpected behavior when the battery was about halfway discharged. We believe this is due to fluctuations in elecrical demand from the kinect sensor and the GPU. To correct these fluctuations, we believe that upgrading the battery to a 4-cell Lipo from a 3-cell will allow for a constant input voltage to the kinect sensor (or whichever replacement sensor is installed) and to the GPU. Additionally, the upgrade would likely see an increase in flight time, as the higher voltage will turn the motors equally fast with less input energy.

6. References

Any references that you cited in the document

- RTAB-Map: Homepage
- MavLink: Homepage
- ROS: Homepage