Robust autonomous micro aerial vehicle (MAV) navigation with onboard, environment-agnostic, multi-sensor SLAM

Parthiv N. Krishna

This project involved the development of autonomous Micro Aerial Vehicles (MAVs) that use a multi-sensorial localization system to navigate and deliver payloads. A quadrotor MAV was designed, built, programmed, and tested. The MAV was designed with a maximum takeoff weight (MTOW) of approximately 4 kg to allow for greater lifting ability. A primary focus of this project was developing a platform that can use a variety of sensors to function in almost any environment. Two primary programmable processors were used: an ARM Cortex M4 for motor control and an Nvidia Jetson TX2 for high-level processing. These two processors are supported by other smaller co-processors for sensor data translation. Sensing methods such as GPS, Optical Flow, LIDAR, and Camera-Based 3D Mapping were combined to navigate indoors and outdoors, and to transition between long-range outdoor flight and targeted indoor flight. Currently, the MAV successfully navigates indoors and outdoors in lighted and low-light conditions. The MAV is guided to target positions from a remote laptop, and it intelligently plans paths to reach the desired locations. Potential applications of this project include search and rescue, surveillance, and package delivery.

Keywords: *robotics, drones, sensor fusion, path planning*

Introduction

Autonomous MAVs for Mapping

Micro Aerial Vehicles (MAVs), commonly known as drones, are small aerial robotic systems that range in weight from less than 50 g to 5 kg, and are used in both civilian and military fields (Qiu & Duan, 2017). In addition to their central processor and any actuators, MAVs generally have numerous sensors on board to track flight parameters such as orientation, position, and altitude (Michael et al., 2013). Using these sensors, an MAV can fly autonomously by following a path through predetermined waypoints, or by intelligently planning routes after sensing the environment.

In order to plan routes autonomously and intelligently in areas where navigational aids (like blueprints or GPS) may not be available, an MAV would have to employ Simultaneous Localization and Mapping (SLAM). Thrun and Leonard (2008) propose SLAM as a method for an autonomous system to determine its location in an unknown environment. By sensing its environment, an estimation for its location can be calculated, consisting of numerous parameters such as position (consisting of x, y, and z) and orientation (consisting of pitch, roll, amd yaw) (Bailey & Durrant-Whyte, 2006, Part I). However, it is important to note that the estimated location is just that: an estimate. The accuracy of the estimate is proportional to the amount of data collected, so the quality of the estimate increases with more measurements (Bailey & Durrant-Whyte, 2006, Part II).

Mapping the interior of buildings autonomously has many potential applications. Michael et al. (2013) designed a system consisting of a terrestrial and an aerial vehicle that could map buildings that had been damaged by earthquakes. Rudol and Doherty (2008) suggested MAVs as a method of real-time monitoring of disaster scenarios such as earthquakes and tsunamis. Using photogrammetry and distance measurements from LIDAR sensors, Rudol and Doherty (2008) were able to create a fairly large (~95kg) MAV that could find civilians through thermal imaging. However, due to its large size, it was forced to make aerial passes instead of flying inside a building. This made it better suited to tsunamis or hurricanes, where civilians may be stranded outdoors as opposed to inside a damaged or destroyed building. Guo et al. (2015) also

developed a cooperative interior mapping method using data from mobile devices. The use of mobile devices is limiting for search and rescue, as it requires the building to be safe for the person carrying the device to navigate.

Limitations of Previous Work

Past work in this project used primarily IMU-based odometry and localization, which was inherently limited for many reasons. While a downward-facing ultrasonic rangefinder was used to get accurate measurements of MAV altitude, the location of the MAV in relation to the room was calculated solely based on integration and linear quadratic filtering of the IMU measurements (Krishna, 2018). This resulted in a high amount of drift, as there was no 'loop-closing' with additional sensors (Krishna, 2018). The reliance on a single IMU was also a problem because IMU failure would always lead to a crash.

Mapping and navigation was also limited. The MAV could only navigate and generate maps in well-lighted environments due to the use on a single visual-spectrum camera. On top of this, all localization, and thus all maps generated, were only relative to the takeoff point of the MAV. Absolute (global) positioning of the MAV was impossible due to the lack of a global coordinate system reference (Krishna, 2018). Autonomous navigation was limited to "exploration," where the MAV flew towards areas where there was less map data. This made it unable to fly in a controllable manner, for example, to specific rooms of interest.

Extension of Previous Work

Past methods of MAV localization generally relied on either outdoor navigation systems (such as GPS) or indoor navigation systems (such as LIDAR scanning systems). The main drawback of this is that MAVs are then limited to either indoor or outdoor navigation (Sun et al., 2018). MAVs configured for outdoor use are ineffective indoors, as they cannot detect and avoid obstacles like walls or furniture. On the other hand, MAVs configured for indoor use cannot successfully navigate in open outdoor environments, because there are few features for scanning LIDAR sensors to detect, and the indoor-specialized MAVs are generally too small to handle windy conditions (Pestana et al., 2013).

Krishna (2018) used a 450mm quadcopter base platform to lift a relatively light sensor payload. In order to lift the increased sensor payload required for this project, the new MAV base platform was larger and had a higher Maximum Takeoff Weight (MTOW). Potential options included adding more motors to create a hexacopter or octocopter. However, a larger quadcopter was ultimately selected to minimize the changes needed to the flight controller logic. The more powerful platform is better-suited to outdoor flight, as the increased thrust capabilities enable it to resist windy conditions more successfully. The use of GPS navigation on this base platform provides a good starting point with outdoor localization and navigation (but no obstacle avoidance) abilities.

Maintaining a consistent hover position is difficult with IMU-based localization alone. While Krishna (2018) used it relatively successfully in indoor environments, Sun et al. (2018) determined that additional sensors are required for positional holding in outdoor environments due to the presence of wind. Lucas & Kanade (1981) described an optical flow algorithm that effectively matches features between successive images to track the distance traveled between the two of them. Lee & Song (2004) proposed optical flow software as a method of localization of terrestrial (ground-based) robots. By pointing the camera at the ground and tracking the change in position of points under the terrestrial robot, the position of a terrestrial robot can be accurately estimated (Lee & Song, 2004). Optical flow software can similarly improve hovering stability and provide a method of estimating the position of an MAV. Honegger et al. (2013) developed a self-contained optical flow board that contained a camera, ultrasonic sensor, and processor; a similar solution was used in this project as an optical flow-based position estimator.

Krishna (2018) argued that LIDAR systems are inferior to camera-based solutions for three-dimensional mapping when the maps are intended to be interpreted by humans. LIDAR units do have higher weight and mechanical complexity than other systems (Krishna, 2018). However, they do provide a few benefits when compared to camera systems for navigation. LIDAR enables rapid obstacle detection with minimal processing overhead, as the distance to the nearest object at an arbitrary angle can be found very quickly (Zhang & Singh, 2015). With additional processing, successive laser scans from the LIDAR can be

matched to generate two-dimensional maps and location estimates (Kohlbrecher et al., 2013). Hess et al. (2016) developed an algorithm to quickly match LIDAR scans and generate maps.

According to Zhang & Singh (2015), visual-inertial odometry (VIO) and localization vastly outperforms IMU-only odometry and localization, at the expense of much greater processing power requirements. Qin et al. (2018) developed a robust method of fusing IMU data with a single camera system for localization. 3D mapping with an MAV system is a common solution for outdoor environments (Colomina & Molina, 2014). Gonçalves & Henriques (2015) used an MAV-based 3D mapping system for topographical monitoring of coastal areas. MAVs flying above buildings can also be used to create 3D models (Li-Chee-Ming & Armenakis, 2014). Krishna (2018) proposed a method of generating full-color 3D maps of the interior of buildings using a camera. For this project, both VIO and 3D mapping were run from the same camera to provide dual functionality.

Each of the sensors functions well in different environments and serves slightly different purposes. GPS works in outdoor environments, and can be used as a global reference frame. Optical flow can be used in any lighted environment, indoor or outdoor, and was used for hover stability and localization. LIDAR can be used in any environment provided that ambient lighting is not excessively bright; it was used for mapping and obstacle avoidance. Finally, 3D mapping and VIO were used for robust mapping and localization in lighted environments. Combining the various features of these sensors allowed for mitigation of their limitations and overall robust multi-environmental mapping and localization.

This project extended the work from the previous year with the intention of greatly improving robustness in localization, mapping, and navigation. The primary improvements came in the fusion of various sensors, including: multiple IMUs, a GPS receiver, a downward-facing optical flow camera, a 360° LIDAR sensor, and a forward-facing global shutter camera.. The main goal was to localize in a variety of environments and successfully perform indoor-outdoor transitions, where the MAV can seamlessly enter an indoor environment (i.e., a building) from an outdoor one, via an entryway like a door or window. This would greatly improve performance in search and rescue scenarios, as an MAV could start outside a building and enter it autonomously, eliminating the need for a human to enter the building at all. It could also be applied in other scenarios, such as general surveillance, warehouse management, architectural surveying, package delivery, and more.

Krishna (2018) used a mapping-oriented method of navigation, i.e., the navigation of the MAV was determined solely with the intention of improving map density. For applications like payload delivery and search and rescue, this behavior is not desirable. Thus, these new developments allowed for the selection of target locations to which the MAV would autonomously navigate.

Project Purpose

The purpose of this project was to develop a Micro Aerial Vehicle (MAV) platform that uses a conglomeration of various sensors to map its environment, localize itself within its environment, and navigate through its environment. Sun et al. (2018) successfully navigated a MAV to target waypoints in an outdoor environment using various sensors to procedurally improve the position and orientation estimates. Krishna (2018) navigated in an indoor environment using IMU-only localization. By extending the IMU-only localization with additional sensors, this project allowed for seamless transitions between various environments (i.e., low-light indoor, well-lighted indoor, and outdoor) while maintaining accurate estimates of location and creating dense three-dimensional maps.

Objectives

1. Development of MAV platform with GPS.
2. Integration of Optical Flow hardware and software.
3. Addition of LIDAR-based SLAM.
4. Implementation of 3D mapping and VIO.
5. Fusion of the sensor methods above.
6. Optimization of navigation to use waypoint guidance.
7. Testing of the system indoors and outdoors.
8. Detection of current environment and automatic selection of appropriate sensors.
9. Addition of payload manipulation hardware.
10. Creation of a user-friendly ground control station.

Engineering Constraints

This project was constrained primarily by the total cost. This led to careful selection of parts and spares throughout the development to minimize cost. A generous limit of $1500 was set.

The ability to effectively navigate an indoor environment is itself a constraint, as it limited the maximum size of the MAV. To preserve maneuverability indoors, the maximum size of the MAV was constrained to a 65 cm (26 in) cube. This was selected to provide clearance to pass through a standard 36 inch wide doorway. With an MAV of this size, the maximum takeoff weight generally doesn't exceed 4 kg (9 lb). A conservative limit of 2.5 kg was set to allow for the MAV to carry a payload.

A bare minimum battery life constraint was set at 5 minutes to provide enough time to cover long distances outdoors and navigate larger buildings. To accomplish this as quickly as possible, the MAV must be able to reach an outdoor speed of 9 m/s (approximately 20 mph), and an indoor speed of 0.5 m/s (approximately 1 mph). Coupled with the minimum 5 minute airtime, this would correspond to flying outdoors to a building 500 meters (~⅓ mi) away, entering it and mapping out a 1000 m^2 (~10,000 sq. ft.) area, then returning to the takeoff point.

System Overview

The MAV takes the form of a quadcopter drone with two main programmable processors which are supported by a variety of sensors and transceiver devices. These components all work together to provide the desired functionality.

Overall, the MAV conforms to all of the constraints. The total cost of the components on the MAV was under $1050; a Bill of Materials can be found in *Appendix 1*.

The MAV also meets the dimensional and weight constraints. The MAV in ready-to-fly configuration has a size of 57 cm (L) x 57 cm (W) x 32 cm (H), approximately 22.5" x 22.5" x 12.5". The overall weight without cargo is 2.05 kg, or 4.5 lbs, allowing for a total of nearly 2 kg of theoretical payload capacity. However, the payload is limited to 1 kg to maximize battery life, agility, and airspeed.

The battery life was tested to be around 10 minutes under typical use cases. A more detailed explanation of the methods by which battery life was determined can be found in the *Testing* section. The MAV is capable of flying 12.5 m/s (28 mph) outdoors; its indoor speed varies between 0.5 m/s and 2 m/s (1-4 mph) depending on the density of obstacles in its immediate environment.

For a visual overview of the major components for the MAV, see *Fig. 1*.

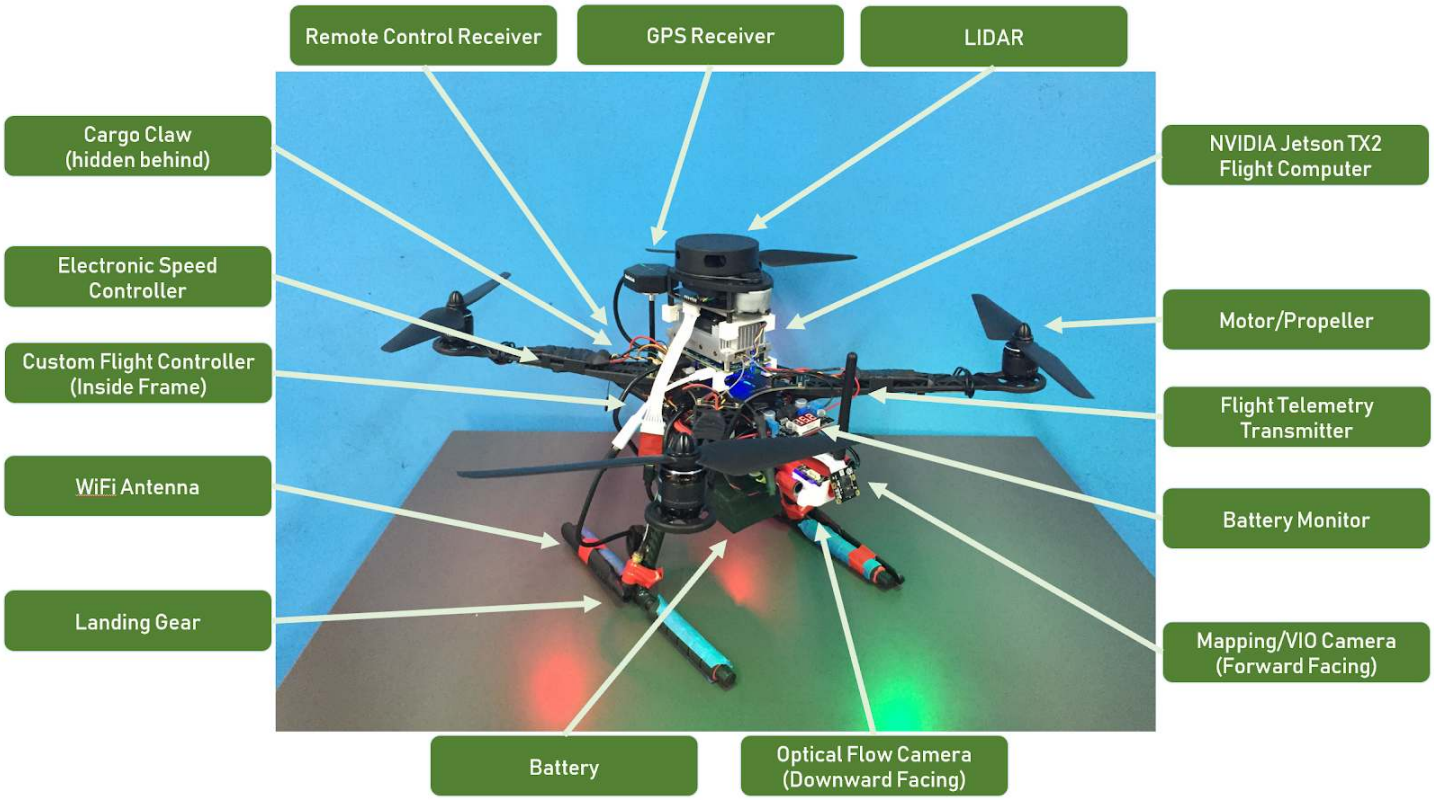


Figure 1: The MAV with major components labeled.

Hardware Overview

The final MAV consists of various hardware subsystems, each of which is described in detail in the following section. These include:

- Drone Platform
  - Structure
  - Power Delivery
  - Thrust Generation
  - Cargo Claw
- Flight Controller
  - Processing
  - Sensing
- Flight Computer
  - Processing
  - GPS
  - Optical Flow
  - LIDAR
  - VIO and 3D Mapping
- Wireless Transceivers
  - WiFi
  - Remote Control Receiver
  - 915 MHz

Drone Platform

The structure of the MAV is an off-the-shelf S500 frame that was modified for this project. The main "arms" are made of Glass Fiber Reinforced Nylon while the main central "hub" is two printed circuit boards that allow for integrated power distribution. Additional mounting holes were drilled to facilitate the attachment of other components. The landing gear was changed to carbon fiber tubes, reducing weight while increasing robustness.

Power is sourced from the onboard 4S Lithium Polymer (LiPo) battery, which has a capacity of 4,000 mAh (milliamp-hours). The four cells in series (4S) allow for the output voltage to be 16.8V when fully charged and a nominal output voltage of 14.8V. The battery output leads are connected directly to the Battery Monitor, then to the rest of the MAV.

The Battery Monitor is a small custom-designed printed circuit board (PCB) which contains circuitry to measure and track various battery characteristics. A 3.3V regulator, a 5V regulator, and a variable voltage regulator set to 10.5V are all onboard; these provide smooth voltages to the more sensitive components onboard the MAV. A resistor divider scales the 0-16.8V battery voltage by dividing it by 4. This creates a 0-4.2V voltage, which is read by an Atmel ATMEGA328 microcontroller via the analog input pins. The ATMEGA328 also measures the total MAV current draw by using a CS-100A hall-effect current sensor, rated for a maximum of 100A. The battery voltage is shown on a three digit eight-segment display, connected via three 74HC595 shift registers. Battery voltage and current are sent to the Flight Controller over I2C.

The 3.3V regulator powers the Flight Controller, the GPS module, and the 915 MHz transmitter. The 5V regulator is used to power the remote control receiver, the Cargo Claw motor, the LIDAR module, and the onboard cameras. The variable voltage regulator, set to 10.5V, is used to power the Flight Computer.

The raw battery voltage out of the Battery Monitor is provided directly to the four electronic speed controllers (ESCs). These ESCs connect to the four brushless (BLDC) motors and switch the DC voltage out of the battery to pairs of poles on the BLDC motors to allow them to spin. The BLDC motors are of the 2312 form factor and each have their own propeller, with a 9.5" diameter and a 4.5" pitch. Two of the motors, the front left and the back right, spin clockwise, while the other two spin counterclockwise. This allows for balancing of angular momentum as well as control of the MAV yaw rate. Each motor/propeller combination can produce a maximum of 1.08 kg (2.38 lbs) of thrust, generating a total of 4.32 kg (9.52 lbs) of thrust.

The MAV has the capacity to carry around 1kg (~2 lbs) of payload. While there is more thrust available, this capacity is limited to maintain maneuverability and battery life. To manipulate payloads, a general purpose cargo claw was designed and 3D printed out of ABS plastic (see *Fig. 2)*. If the MAV would repeatedly be carrying the same cargo, a more specialized claw/manipulator could be designed and mounted to handle it more effectively.

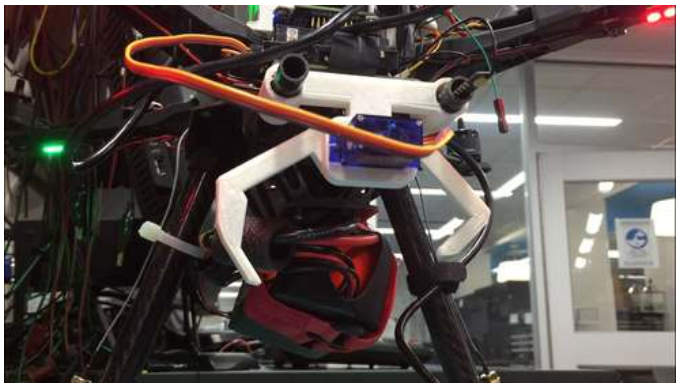


Figure 2: The cargo claw in the open position.

Flight Controller

The Flight Controller is an ARM Cortex M4 microcontroller which has the primary goals of reading data from various IMUs and controlling the speeds of the four motors. This allows for precise control over the translation and rotation rates of the MAV in all three dimensions. The challenge of controlling this system is that it is inherently underactuated; i.e., it has six degrees of freedom (X movement, Y movement, Z movement, pitch, roll, and yaw) while only having four control inputs (the speeds of the four motors). As a result, the degrees of freedom are fundamentally linked; it is mechanically impossible to actuate one degree of freedom while keeping all others the same. The flight controller balances all of the data from sensors and a mathematical model of the MAV to control it efficiently and reliably.

Three IMUs provide gyroscope and accelerometer data to the Flight Controller. These are an InvenSense MPU6000, an STMicro L3GD20H, and a Bosch BNO055. The MPU6000 and L3GD20H are connected to the ARM Cortex M4 over SPI, while the BNO055 is connected over I2C. These IMUs were selected because they range from high update rate and low precision (MPU6000) to low update rate and high precision (BNO055). On top of this, three IMUs allow for redundancy, so a single IMU failure would not cause the MAV to crash.

The Flight Controller connects to the ESCs using Pulse Width Modulation (PWM) signals (see *Fig. 3)*. The ESCs read this signal, which is updated 250 times per second, to determine the speed at which to spin the motors. A 1ms pulse width corresponds to 0% (idle) throttle, while a 2ms pulse width corresponds to a 100% (max) throttle. All pulse widths between 1ms and 2ms correspond linearly to different throttle percentages.

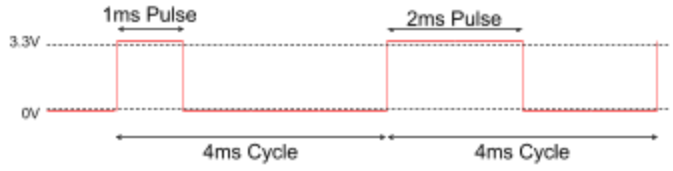


Figure 3: PWM Signal Example

The various components are connected on a PCB. Additional passive electronics, such as voltage regulators and clock crystals, can also be found on the Flight Controller PCB.

Flight Computer

The Flight Computer is an NVIDIA Jetson TX2, with the stock carrier board replaced by the CTI Orbitty carrier board. The Orbitty carrier is significantly smaller and lighter (See *Fig. 4*), which makes it more appropriate to use on the MAV. The Orbitty has fewer ports and hardware connectivity options onboard; however, there were a sufficient number for use on this MAV. The Flight Computer is connected to the Flight Controller using a 926100 baud (926.1 kilobits per second) UART connection.

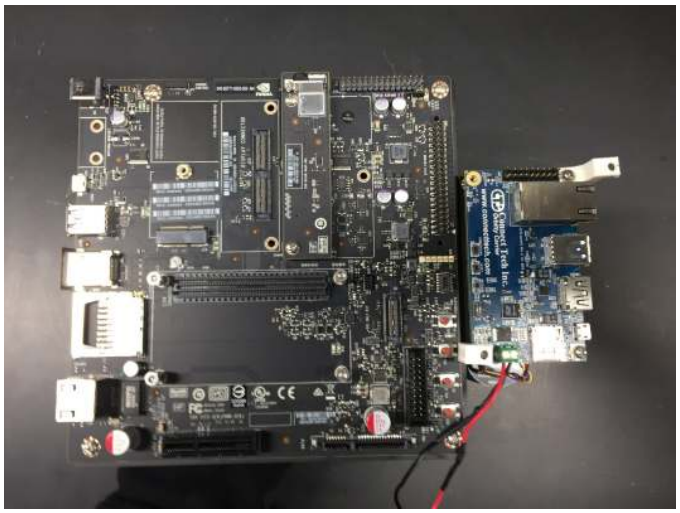


Figure 4: Size comparison of the stock Jetson carrier (left, black) and the CTI Orbitty carrier (right, blue).

Various sensors are connected to the Flight Computer. Each individual sensor functions best in specific situation since they each have their own strengths and weaknesses. They are all fused to enable multi-sensor SLAM that works indoors and outdoors in various lighting conditions.

GPS data is collected using a u.blox NEO-M8N receiver module. It can receive data from four different Global Navigation Satellite Service (GNSS) systems. These are GPS (USA), Galileo (EU), GLONASS (Russia), and BeiDou (China). By combining data from all of these, the accuracy and precision of the M8N receiver module is increased. The M8N also provides compass data to have an absolute measurement of the MAV heading. The M8N is powered off of the 3.3V regulator from the Battery Monitor. It is connected to the Flight Computer over a UART line running at 115200 baud (115.2 kilobits per second). The GPS location updates at 5 Hz, and when it is connected to enough satellites it can output altitude data as well.

The Optical Flow measurements and calculation is handled by an downward-facing ArduCam MT9V034 camera module connected to an ARM

Cortex M4F. A LV-EZ1 ultrasonic rangefinder is also used for altitude measurement. The entire Optical Flow assembly (camera, coprocessor, ultrasonic sensor) is mounted to the MAV using a 3D printed case (See *Fig. 5*). The camera is a global shutter camera. This means that the entire frame is exposed simultaneously, which is crucial for timing and accurate distance tracking. Standard rolling shutter cameras expose different areas of the frame at slightly different times, which would not have worked for this timing-sensitive application. The MT9V034 runs at a very low resolution (752x480 downscaled to 188x120) and in grayscale. The ARM Cortex M4F (running at 168 MHz) processes the camera data, tracks the movement of features in the images, and updates the position estimate 250 times per second. The positional estimate information is read by the Flight Computer using an I2C connection.

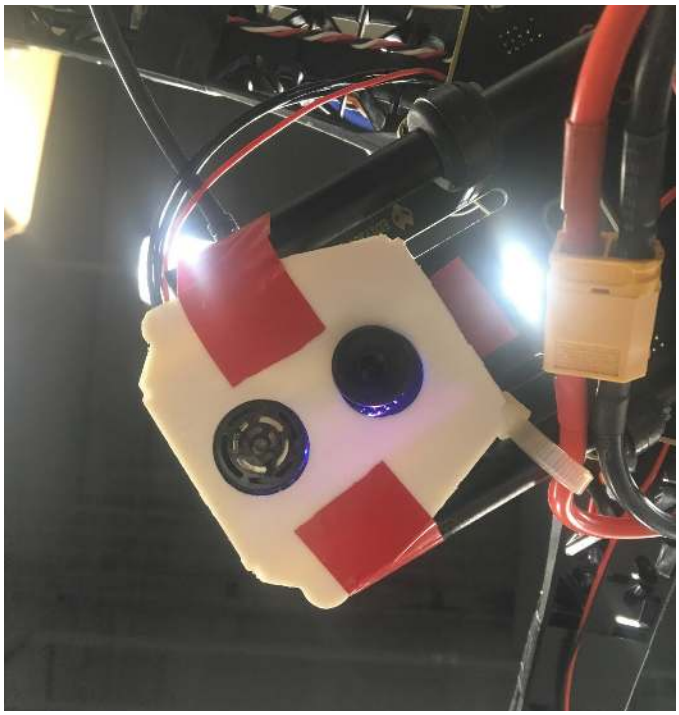


Figure 5: The Optical Flow assembly, consisting of the ultrasonic rangefinder (left) and camera (right).

A RoboPeak RPLIDAR A1M8 is attached to the MAV using a 3D printed mount. The LIDAR module is elevated above all other components to avoid static occlusions, which would reduce the amount of useful data provided (See *Fig. 6*). It rotates 360°, allowing it to measure distances along a two-dimensional plane. The LIDAR module can collect distance measurements at a rate of 2000 Hz, and it is configured to spin at 2Hz. This means it collects 1000 measurements per revolution. It can detect objects up to 12 m (39 ft) away with 0.2 cm (0.078 in) resolution on the distance measurements. The RPLIDAR outputs data using a UART signal; this UART is connected to an FTDI Serial-USB converter board. This allows it to be plugged in to a USB port on the Flight Computer, which enables bidirectional data transfer and supplies power to the LIDAR module. The 2D map is updated at 2Hz but data is only sent back to the laptop at 0.5 Hz.

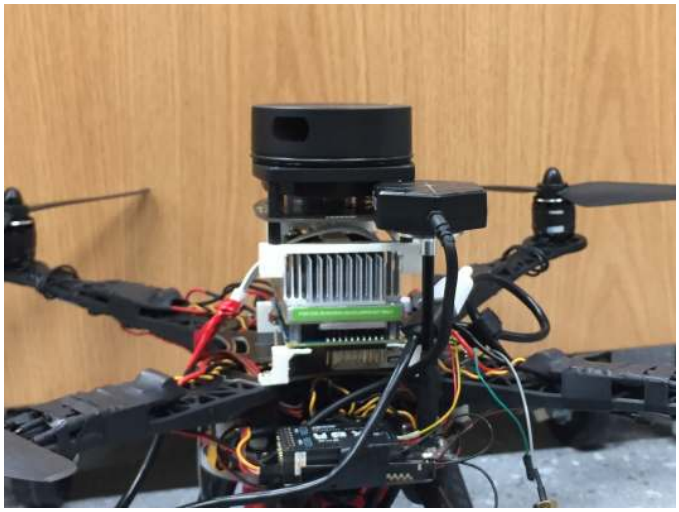


Figure 6: The LIDAR module is elevated above all other components on the MAV.

Visual-Inertial Odometry (VIO) and 3D mapping are handled by an ArduCam AR0134 full-color global shutter camera. This is connected via a USB shield to the Flight Computer. Image data is sent at 54 frames per second at a resolution of 1280x960. On the Flight Computer, this image is left in full-color mode for 3D map generation, and also converted to grayscale for VIO calculation. The 3D mapping algorithm creates dense full-color maps using images from the camera. The VIO algorithm fuses frame differences from the image data with synchronized inertial data sent from the Flight Controller over a UART connection. This allows it to robustly estimate position at a rate of 54Hz. 3D maps are generated in real-time at 1 Hz.

Wireless Transceivers

Three wireless transceivers transmit data to the ground and receive data from ground-based transmitters; each serves a unique purpose.

A 2.4 GHz WiFi transceiver is found onboard the NVIDIA Jetson TX2; it is used to send back high-bandwidth data. This includes 2D and 3D maps that are sent to the remote laptop, as well as target indoor waypoints that are received from the laptop. Functionally, the Jetson acts as a WiFi router that the laptop connects to in order to share data.

The Remote Control Receiver is an FRSky X8R; it is used to receive manual flight commands from the FRSky Taranis X9D+ remote controller. These commands are transmitted using the ACCST wireless frequency-hopping protocol; the X8R encodes the positions of the various axes and buttons in to the S.BUS protocol. The S.BUS signal is connected directly to the Flight Controller, where it is decoded to the raw axis and button positions.

The 915 MHz transmitter sends unidirectional data to a 915 MHz receiver connected to a USB port on the laptop. The lower frequency of this public band allows for longer range data transmission than WiFi, at the expense of lower bandwidth. As a result, the data transmitted is limited to mission-critical data, specifically system health information, current GPS coordinates, and battery monitoring information. It is connected to the Flight Controller over a 19200 baud (19.2 kilobits per second) UART connection.

Software Overview

The software stack on the MAV (See *Fig. 7*) can be divided into the following logical sections:

- Flight Controller
    - Sensor Data Collection
    - Filtering and Estimators
    - Motor Control
    - Telemetry
- Flight Computer
    - Robot Operating System (ROS) Core
    - Global Positioning
    - LIDAR
    - Optical Flow
    - 3D Mapping
    - Sensor Fusion
    - Path Planning
    - Flight Controller Communication
    - Data Transmission
- Battery Monitor
- Optical Flow
- Remote Laptop
    - Maps and Waypoints
    - Ground Control Station
    - Flight Modes

Flight Controller

The ARM Cortex M4 Flight Controller runs the open-source NuttX Real-Time Operating System (RTOS). On top of this base system, there are various C++ programs that each handle one aspect of the sensing, estimation, and motor control logic. Each of these tasks runs in parallel and constantly exchanges data with the other tasks.

Sensor data is read from the three IMUs at the update rate of the sensors -- 1000 Hz for the MPU6000, 200 Hz for the L3GD20H, and 20 Hz for the BNO055. Each of these outputs angular velocities in all three dimensions and linear accelerations in all three dimensions. The data from the MPU6000 is run through a low-pass filter to reject high frequency vibrations (i.e., from motors). After this, the data from all three is fed into an orientation and position estimator, which is fundamentally an Extended Kalman Filter (EKF).

1. MPU6000 (fast) angular velocities are integrated once to estimate attitude.
2. MPU6000 linear accelerations are corrected from the MAV's relative axes (x, y, z) to Earth absolute axes (north, east, down). This is because the IMU measures linear accelerations relative to the MAV attitude at the time they were measured, but the global position estimate needs to be relative to a static (Earth) reference frame.
3. Corrected MPU6000 linear accelerations are integrated to estimate linear velocities.
4. Linear velocities are integrated to estimate position.
5. Over multiple iterations, the integrated values (original estimate) from just the MPU6000 tend to drift and become very unreliable. To account for this, gyro and accelerometer noise estimates, pulled from a multivariate normal distribution, are used to create a State Covariance Matrix (SCM). The SCM predicts the error in the system.
6. When a measurement from a more accurate sensor (i.e. L3GD20H or BNO055) is available, the difference between the original estimate and the new sensor value is calculated.
7. The SCM and the new measurement are combined to correct the original estimate.
8. The predicted error is scaled down since a more accurate measurement was made.

This estimation algorithm provides sufficiently accurate measurements at high frequency by combining the high update rate of the MPU6000 with the high accuracy of the L3GD20H and the BNO055.

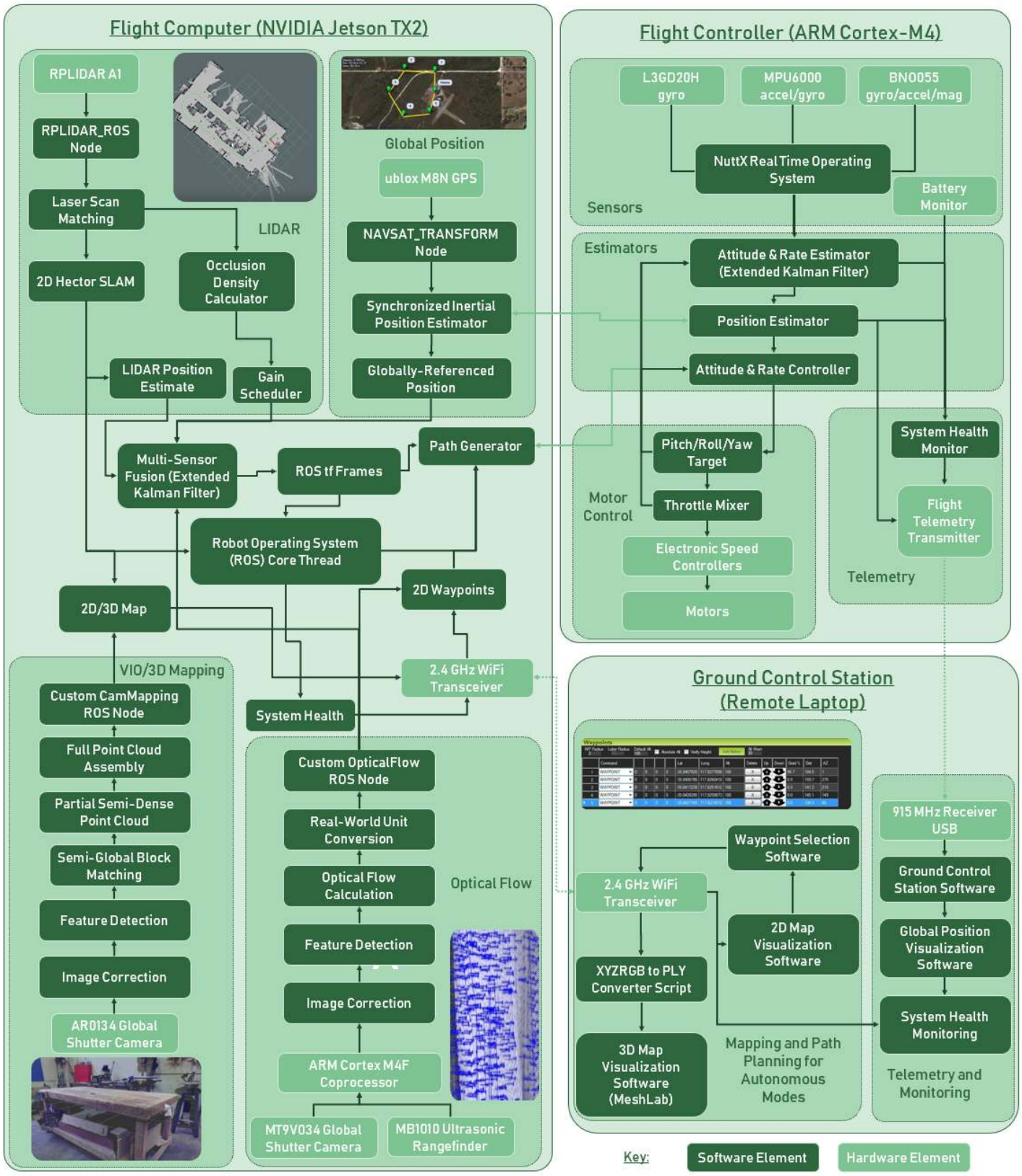


Figure 7: A logical block diagram of the software on the Flight Computer, Flight Controller, and Remote Laptop. This diagram also includes some of the hardware elements and shows their connections and interactions with the software stack.

The next stage is the attitude and rate controller. This directly controls the MAV angular position (attitude) and indirectly controls the MAV linear velocities (rates).

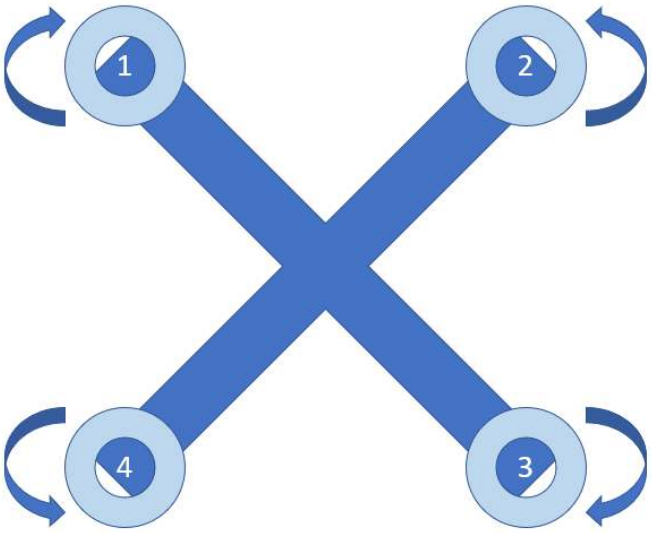


Figure 8: The four motors of and their rotation directions (the top represents the front). Motors 1 and 3 rotate clockwise (CW) while motors 2 and 4 rotate counterclockwise (CCW).

MAV altitude, pitch, roll, and yaw can be directly controlled. By modulating the speeds of the four motors on the MAV (see *Fig. 8*), these four degrees of freedom are actuated.

1. The desired MAV altitude and attitude (pitch/roll/yaw) is received as a desired setpoint.
2. The current altitude/pitch/roll/yaw values are measured as process variable.
3. These errors are tracked and used to calculate the values for the throttle, pitch, roll, and yaw outputs. This is done using four PID controllers (see *Eq. 1*), one for each degree of freedom being controlled.

$$o(t) = K_p e(t) + K_i \int_0^t e(\tau)\, d\tau + K_d \frac{de(t)}{dt}$$

Equation 1: The equation for a PID controller. The output at a given point in time o(t) is the sum of three components. The first component is Proportional to the error (approaches the setpoint), the second is an Integral that represents the history of error values (reduces steady-state error), and the third is a Derivative that represents the rate of change of the error (rejects disturbances). Each of these components has an associated constant K which is "tuned" to achieve the desired response.

4. The throttle, pitch, roll, and yaw outputs are linearly combined to generate desired speeds for the four motors (see *Eq. 2*).

$$\begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} T \\ P \\ R \\ Y \end{bmatrix}$$

Equation 2: The four motor speeds (V) are linear combinations of the Throttle, Pitch, Roll, and Yaw values that are calculated by the PID controllers.

5. The four motor speeds (each ranges from 0 to 1) are linearly converted into PWM pulse lengths (ranging from 1ms to 2ms).
6. These PWM pulse lengths are output to four pins on the ARM Cortex M4 that are connected to the ESC inputs.

MAV horizontal velocity cannot be directly controlled. This is a result of the of the fundamental underactuated nature of the quadcopter system. Due to the inability to directly control the MAV linear velocities, they must be indirectly controlled by instead controlling the angular position of the MAV. Controlling the angular position of the MAV has the side-effect of also affecting the horizontal MAV velocity. This side-effect is intentionally leveraged: the MAV angular position is directly controlled, allowing for indirect control over horizontal velocities.

1. Desired x and y velocities are received and become the PID setpoints.
2. The current x and y velocities are measured and become the PID process variables.
3. The error in the velocities are calculated by subtracting the process variable from the setpoint.
4. The errors are fed into two PID controllers (see *Eq. 1*), with the output being a pitch angle target (for Y) or a roll angle (for X) target. For the MAV to move forward, it must pitch forward. For the MAV to move to the right, it must roll right. To achieve a movement in some arbitrary direction, the pitch and roll commands are combined.
5. The output target pitch/roll are then fed into the attitude controller.

Functionally, this control method where one set of PID controllers feeds setpoints into another set of PID controllers is known as cascaded PID control. This method works well here, since some degrees of freedom are not directly controllable.

The Flight Controller sends attitude and position

estimates to the Flight Controller at 250Hz (the rate at which they are calculated). It receives the current GPS location from the Flight Computer at 5Hz. In addition, it reads the battery voltage, current, and approximate total mAh drawn from the Battery Monitor. The current altitude, attitude, horizontal velocities, GPS location, and battery data is sent to the ground via the 915 MHz transmitter at 5Hz.

Flight Computer

The NVIDIA Jetson TX2 Flight Computer runs the Ubuntu 16.04 Linux distribution with the Robot Operating System (ROS) Kinetic framework. ROS consists of a Core process, which handles various Topics that can be read from and written to by Nodes. The architecture of the Flight Computer software is largely a conglomeration of open-source (named in ALL_CAPS) and custom Python (named in CamelCase) ROS nodes, each with a very specific purpose.

GPS data is read using a modified version of the open-source NAVSAT_TRANSFORM node in ROS. Modifications were made to allow it to directly communicate with the u.blox NEO-M8N module, since by default it works with a Garmin 18x. The raw data from the GPS module is processed to package it in a format that is ready to transmit on the /gps ROS topic. Timestamped data is posted to the /gps topic at 5Hz, the rate at which it is received from the NEO-M8N. Each packet includes the latitude, longitude, compass heading, altitude (if measureable), and number of connected satellites.

LIDAR data is first read using the open source RPLIDAR_ROS node provided by RoboPeak, the manufacturer of the RPLIDAR A1M8 module. This node connects to the RPLIDAR over a USB port, reads the raw data, and interprets it. The data is then sent to the /laserscan ROS topic, where it can be interpreted. The data sent includes the points detected by the LIDAR in polar coordinate form (angle, distance). The /laserscan topic is read by a LidarMapper node which was modified from the open source HECTOR_MAPPING node. The LidarMapper node matches sequential scans, overlays them, and generates two-dimensional maps. From these maps, it is also able to generate a rudimentary MAV location estimate.

1. An entire scan (1000 points, 1 revolution) is read from the /laserscan topic and stored. This is the Reference Scan, and it is represented as a series of 2-element vectors (x and y coordinates of each point).
2. When a new scan is available from /laserscan, this Current Scan is compared to the Reference Scan.
3. An optimization algorithm finds the ideal linear transformation (rotation and translation) to overlay the Current Scan over the Reference Scan. In other words, the Transformation Matrix multiplied by the Current Scan ideally equals the Reference Scan. In practice, the Transformation Matrix is found that makes the two sides of the equation as equal as possible.
4. The inverse of the Transformation Matrix is taken to be the estimated MAV translation and rotation between the successive laser scans. This is posted to the /laserpos topic.
5. An Iterative Occupancy Map (see *Fig. 9*) is generated, using each transformed laser scan as additional data to be added to the map. If a certain point is repeatedly shown to be occupied (i.e. there is a wall or object there), there is higher confidence that it is truly occupied. Once this confidence crosses a configurable threshold, it is added to the map.
6. The Current Scan becomes the Reference Scan. The process repeats with a new scan.

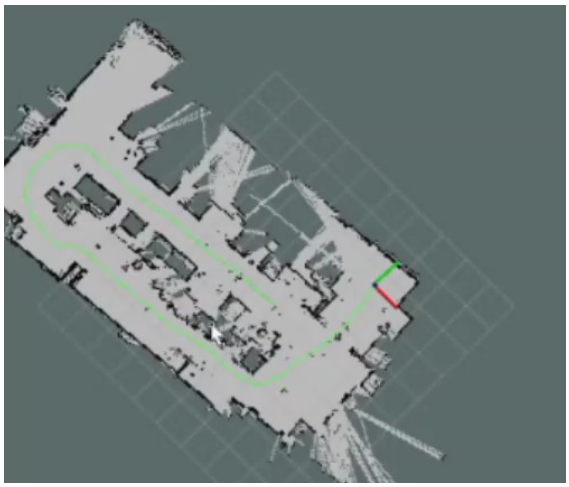


Figure 9: Sample 2D map from LIDAR data. The green line represents the estimated MAV trajectory.

Optical Flow is processed entirely by the ARM Cortex M4F coprocessor, so no image processing or real-world measurements occur on the Flight Computer. As a result, the OpticalFlow ROS node simply interprets the data coming from the coprocessor over I2C. After data is read, it is

converted to packets that are published to the /opticalflow ROS topic. These packets include the x, y, and heading positions and velocities calculated by the optical flow coprocessor.

The 3D Mapping is handled by a CamMapping ROS node; this node is a direct port of the 3D mapping algorithm described by Krishna (2018). This algorithm was modified to work with the RPLIDAR as the rangefinder, and also to also add visual-inertial odometry (VIO) capabilities.

1. Images are captured via the webcam as the MAV moves, with timestamps and associated position/orientation estimates from the flight controller (see *Fig. 10* top).
2. Images are then processed in pairs using a Semi-Global Block Matching (SGBM) algorithm to generate depth maps, treating them as pairs from a stereo camera. Features (corners and edges) are located within each image.
3. Detected features are matched between the two images.
4. The distances (in pixels) that each feature moved between the two images are measured. Pixels that move further between the two images are considered to be closer to the MAV, while groups of pixels that move less are further away (parallax effect). This allows for relative distances to be measured, i.e. the yellow group of pixels is 2x further away from the camera than the blue group of pixels (see *Fig. 10* middle).
5. Synchronized data from the LIDAR provides a measurement to the closest object in the image. Based on the relative depth map, this distance is extrapolated into over 100,000 real-world distances for every pair of images, creating a partial full-color 3D point cloud (see *Fig. 10* bottom).

Each partial point cloud is captured at a certain position and orientation, estimated by the IMU fusion on the Flight Controller. Thus, each one can be transformed using a matrix that takes into account the position and orientation of the MAV at the time of capture. By applying this linear transformation, which is unique to each cloud, they can be placed relative to a common coordinate system. This allows a large full-color point cloud to be constructed as a three-dimensional map of the entire environment.

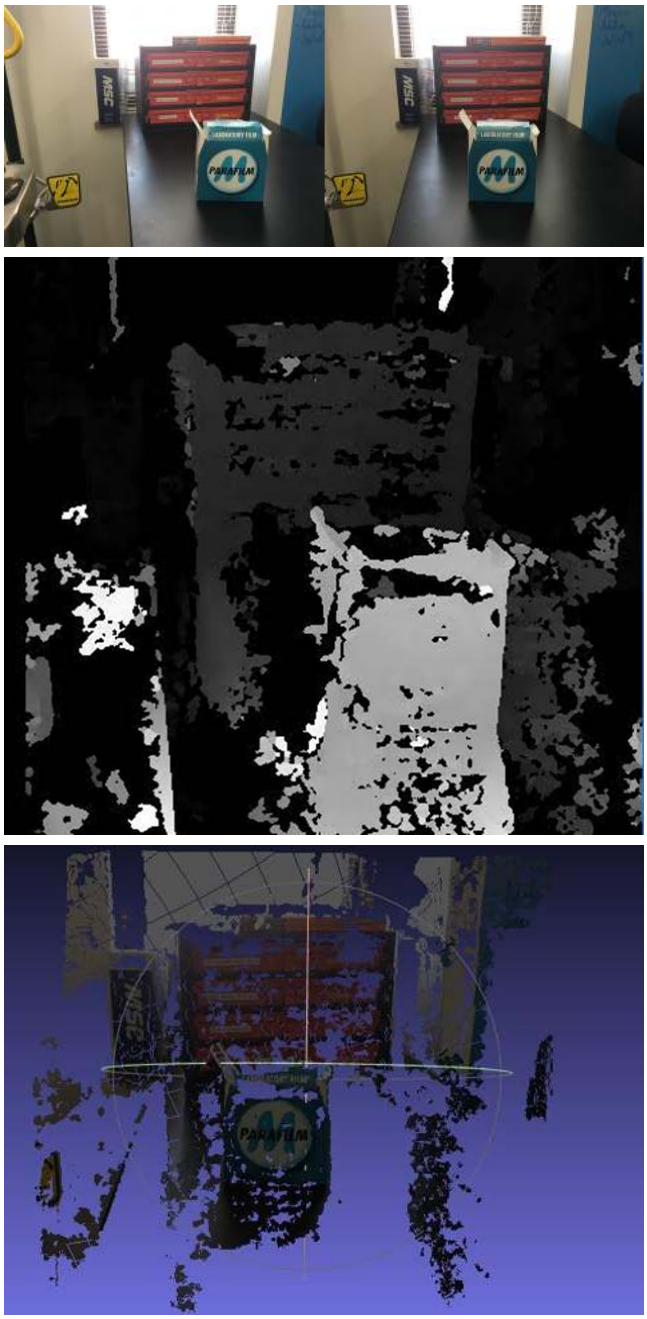


Figure 10: 3D mapping example, showing the process from images (top) to depth map (middle) to point cloud (bottom).

Each time a point is determined, it is published to the /map3d topic in a packet that includes the x,y,z coordinates along with the RGB color code.

Sensor Fusion is handled by the SensorFuser ROS node. It takes in data from the various position estimators -- Flight Controller IMUs, GPS, Optical Flow, and LIDAR SLAM -- to create more robust and accurate MAV position estimates. This reads data from /imu, /gps, /opticalflow, and /lidarpos and feeds them into an Extended Kalman Filter. This Extended Kalman Filter reacts differently to different environment types. For example, in outdoor environments, GPS measurements are more reliable. The current environment is selected

automatically, using a configurable threshold of LIDAR object density, number of GPS satellites connected, and average camera brightness level. This Gain Scheduler weights different sensors differently depending on the environment, and modifies the Extended Kalman Filter accordingly. These environments include Outdoors, Sparse Indoors, Medium Indoors, and Dense Indoors. Each of these four environments can be modified by Bright, Normal, and Low lighting, for a total of twelve modes (see *Table 1*). These modes also affect the speed of the MAV.

Table 1: The various environment and lighting modes of the Localization (Loc.) and Avoidance (Av.) system. Different conditions necessitate the use of different sensors for each purpose.

Lighting	***Environment Type***			
	Outdoors	*Sparse Indoors*	*Medium Indoors*	*Dense Indoors*
Bright Lighting	**Loc.** IMU, GPS, OF **Av.** Camera **Speed** up to 12.5 m/s	**Loc.** IMU, OF, LIDAR **Av.** LIDAR, Camera **Speed** 2 m/s	**Loc.** IMU, OF, LIDAR **Av.** LIDAR, Camera **Speed** 1.5 m/s	**Loc.** IMU, OF, LIDAR **Av.** LIDAR, Camera **Speed** 0.5 m/s
Normal Lighting	**Loc.** IMU, GPS, OF **Av.** LIDAR **Speed** up to 12.5 m/s	**Loc.** IMU, OF, LIDAR **Av.** LIDAR, Camera **Speed** 2 m/s	**Loc.** IMU, OF, LIDAR **Av.** LIDAR, Camera **Speed** 1.5 m/s	**Loc.** IMU, OF, LIDAR **Av.** LIDAR, Camera **Speed** 0.5 m/s
Low Lighting	**Loc.** IMU, GPS **Av.** LIDAR **Speed** up to 12.5 m/s	**Loc.** IMU, LIDAR **Av.** LIDAR **Speed** 1.5 m/s	**Loc.** IMU, LIDAR **Av.** LIDAR **Speed** 1 m/s	**Loc.** IMU, LIDAR **Av.** LIDAR **Speed** 0.5 m/s

Path Planning is handled by the PathPlanner ROS node. This node subscribes to the /waypoints and /tf topic to gather information about the path it should generate. It then publishes paths to the /path topic. The PathFollower ROS node then generates the necessary MAV velocities to follow the paths at each moment in time. The following algorithm generates paths:

1. A straight-line path is drawn from the current /tf position to the desired waypoint (See *Fig. 11*).

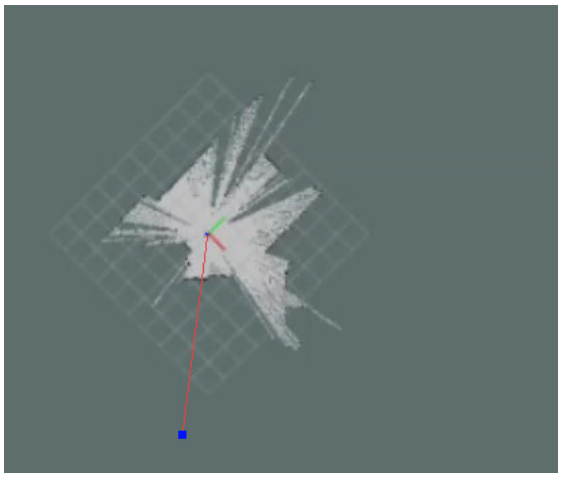


Figure 11: An initial straight-line path (red) to the target waypoint (dark blue) is generated.

2. If indoors and an obstacle is in the way, create a waypoint to the left or the right of the obstacle (See *Fig. 12*). The direction is dependent on which side is more open.

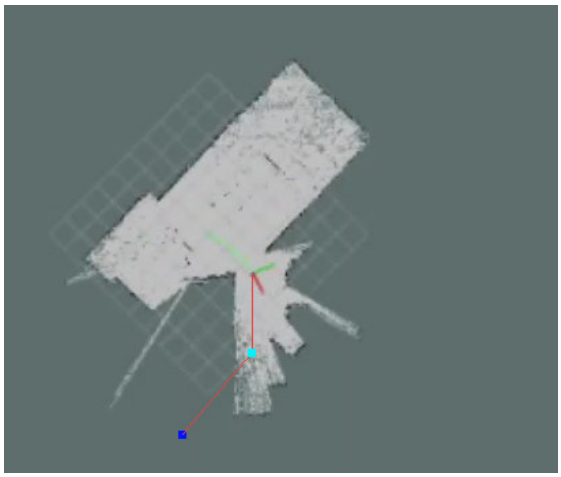


Figure 12: An intermediate waypoint (light blue) is automatically generated to avoid the walls.

3. If approaching a bottleneck, add a waypoint directly in the center of the bottleneck. For example, a narrow doorway (see *Fig. 13*).
4. Regenerate the path with the new waypoints and output the path to the /path topic.
5. A PathFollower ROS node subscribes to /path. Depending on the displacement vector needed to follow that path, the PathFollower generates MAV velocity targets in the x, y, and heading axes. These are then sent to the /flightcommands topic, which is ultimately sent to the Flight Controller so that it can control the motor speeds.

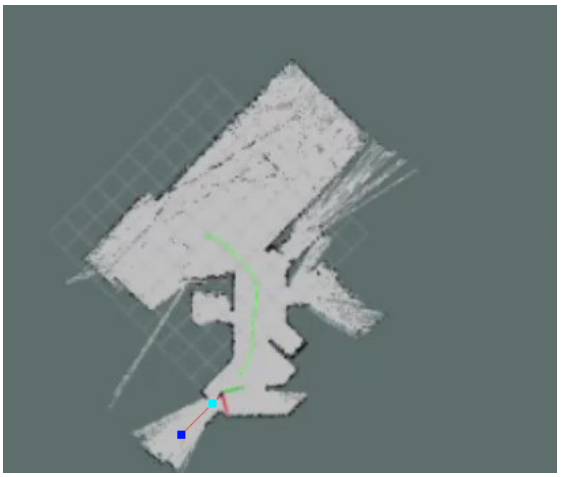


Figure 13: An intermediate waypoint (light blue) is generated in the center of a bottleneck (doorway). This allows the MAV to avoid crashing into either side of the bottleneck.

6. This entire process is repeated at 5Hz, constantly updating the paths as new information is collected. This also allows for corrections if the path is not being followed ideally (i.e., if there is some sort of drift or disturbance to the MAV flight).

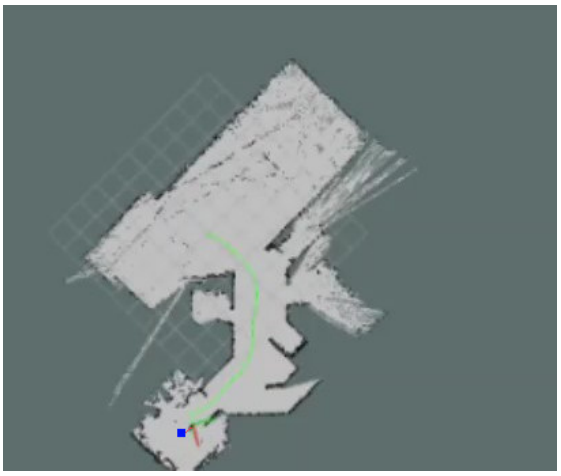


Figure 14: The MAV approaches the target waypoint. The entire path it flew is visible (green line).

A FlightControllerComms ROS node handles communication with the Flight Controller. The primary goal of this ROS node is to read data from /flightcommands and send them to the Flight Controller so that it can control the MAV flight characteristics to follow paths. The FlightControllerComms node also subscribes to the /gps topic and sends GPS location information to the Flight Controller via UART. It also reads IMU data from the Flight Controller, posting them to the /imy topic for use in multi-sensor fusion and visual-inertial odometry.

The Flight Computer sends data directly to the ground using its onboard 2.4GHz WiFi transceiver. The Remote Laptop connects to a wireless network that is created on the MAV, allowing it to transmit and receive information. The Flight Computer reads waypoints from the the /waypoints topic; each waypoint includes a waypoint type (GPS or indoor), then a waypoint formatted according to the type. It also reads from the the /mode topic for selection of flight mode. Both /waypoints and /mode are published from the Remote Laptop. The Flight Computer sends various topics to the Remote Laptop, including /tf for location data, /map2d and /map3d for map data, and /telemetry for system health information.

Battery Monitor

There are two primary coprocessors on the MAV, the Battery Monitor and the Optical Flow processor.

The Battery Monitor (Atmel ATMEGA328 microcontroller) runs a relatively straightforward C++ program that reads the voltage on two analog pins. One of these pins is connected to the 3MΩ/1MΩ voltage divider, so multiplying the measured voltage by 4 provides the battery voltage. The other pin is connected to the output of the CS100-A hall effect current sensor, which outputs 20mV/A. Therefore, the measured voltage is multiplied by 50 to calculate the total current draw of the MAV system. These values are measured at 25Hz.

To track the total amount of battery drain, the Battery Monitor performs a summation of the current draw multiplied by the amount of time for each measurement (see *Eq. 3*). By comparing the mAh drawn with the total battery capacity of 4,000 mAh, an the remaining battery percentage can be estimated.

$$mAh(t) = \sum_{t} I(t)\,\Delta t$$

*Equation 3: The total mAh at a given time t is calculated by first multiplying each current measurement by 40ms (Δt) and then adding each of these measurements. This is equivalent to the Riemann approximation of an integral. The A*ms units are converted to mAh by dividing by 3600.*

The Battery Monitor sends the battery voltage, current, and approximate total mAh drawn are sent to the Flight Controller whenever it is polled over the I2C bus.

Optical Flow

The Optical Flow coprocessor runs a C++ program that tracks various features across multiple frames, computes the optical flow between these frames, and converts the pixel motions into real-world velocities and distances. Pixels are binned at 4x4 (creating a final resolution of 188x120) to increase sensitivity, reduce noise, and increase frame rate.

The fundamental principle of optical flow is that the features of an image at time *t* will still be in the image at time *t+1*. However, these objects will be in different places in the new frame. In practice, the brightnesses (or intensities) of pixels are tracked by grouping pixels into features that are tracked. These include corners, edges, and flat regions (large regions with constant brightness). This principle is known as the Brightness Constancy Constraint (see *Eq. 4.1*).

$$\{4.1\} \quad I(x+\Delta x,\ y+\Delta y,\ t+\Delta t) = I(x,y,t)$$

$$\{4.2\} \quad I(x+\Delta x,\ y+\Delta y,\ t+\Delta t) = I(x,y,t) + \frac{\delta I}{\delta x}\Delta x + \frac{\delta I}{\delta y}\Delta y + \frac{\delta I}{\delta t}\Delta t$$

$$\{4.3\} \quad \frac{\delta I}{\delta x}\Delta x + \frac{\delta I}{\delta y}\Delta y + \frac{\delta I}{\delta t}\Delta t = 0$$

$$\{4.4\} \quad \frac{\delta I}{\delta x}\frac{\Delta x}{\Delta t} + \frac{\delta I}{\delta y}\frac{\Delta y}{\Delta t} + \frac{\delta I}{\delta t}\frac{\Delta t}{\Delta t} = 0$$

$$\{4.5\} \quad \frac{\delta I}{\delta x}V_x + \frac{\delta I}{\delta y}V_y + \frac{\delta I}{\delta t} = 0$$

$$\{4.6\} \quad \nabla I \cdot \vec{v} + \frac{\delta I}{\delta t} = 0$$

Equation 4: Derivation of optical flow equation.

In *Eq. 4.1*, the Brightness Constancy Constraint is given, i.e. the intensity of the pixel at (x,y,t) is equal to the intensity of the pixel at (x+Δx, y+Δy, t+Δt). *Eq. 4.2* follows separately under the assumption that the movement between frames is small, so the change in I can be modeled by the first-order derivatives of I with respect to x,y,t multiplied by the change in x,y,t. Subtracting *Eq. 4.1* from *Eq. 4.2* yields *Eq. 4.3*. Dividing by Δt yields *Eq. 4.4*. Replacing Δx/Δt and Δy/Δt with the x and y components of velocity, and cancelling Δt/Δt, yields *Eq 4.5*. Ultimately this creates *Eq. 4.6*, which is the final representation of the optical flow equation. The dot product of the intensity gradient and the velocity vector between the two frames plus the change in intensity between the two frames is 0.

Besides *Eq. 4.6*, an additional equation must be provided to fully constrain this system. By assuming that the displacement between two frames is approximately constant in a small area around a given pixel, an additional constraint can be constructed. In other words, the optical flow algorithm seeks to minimize the square of the magnitude of the gradient in small windows around each pixel of interest (See *Eq. 5*). This keeps predicted motion as uniform as possible.

$$\sum_{x,y \in \Omega} W^2(x,y)\left[\nabla I(x,y,t) \cdot \vec{v} + I_t(x,y,t)\right]^2$$

Equation 5: The quantity to be minimized, an additional constraint to the optical flow problem.

Since this creates a system with more equations than variables, the goal is not to solve it perfectly, but rather to create a solution that optimizes this quantity (i.e., makes it as close to 0 as possible). By tracking a relatively small number of features, generally 25 to 30, accurate estimates of the pixel velocity between successive frames can be made.

This pixel velocity is converted to real-world velocities using trigonometry. Each pixel in the 2D image is a projection of the 3D world, therefore each pixel represents a particular x and y angle (see *Fig. 15*). Since the horizontal field-of-view (FOV) is known to be 33°, and the vertical FOV is known to be 21°, each pixel in the 188x120 image represents approximately 0.18° in the x and y directions.

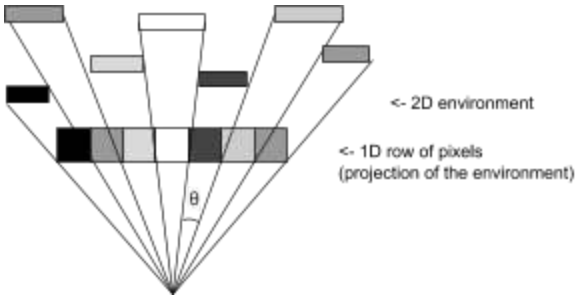


Figure 15: Each pixel represents a particular angular slice (θ). This figure only shows one a 1D line of pixels as a projection of a 2D world, in reality this would be extended to a 2D rectangle of pixels as a projection of a 3D world.

A given pixel velocity v=<V_x, V_y> is multiplied by 0.18° to get a movement between two consecutive frames as angles (θ_x, θ_y). For example, a movement of 100 pixels horizontally and 50 pixels vertically would correspond to an angular change of <18°, 9°>. Then, this angle is converted to a real-world ground displacement d=(d_x, d_y) (see *Fig. 16*). The MAV altitude, h, is measured using the ultrasonic rangefinder.

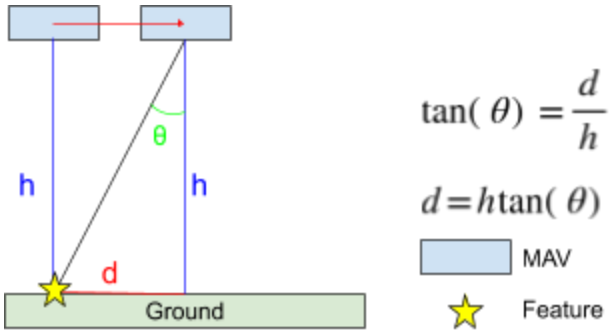


Figure 16: Conversion from image angles to real-world displacement. The MAV moves a distance d at a height h, causing the feature to move in the image by an apparent angle θ. This process is repeated for x and y directions.

This real-world ground displacement is derived with respect to time to calculate real-world groundspeeds in the x and y directions. The data is sent to the Flight Computer 250 times per second over I2C.

Remote Laptop

The remote laptop runs multiple separate programs that each achieve a specific task related to the remote control and monitoring of the MAV.

Two-dimensional maps are visualized using RViz, which is included with ROS (See *Figs. 9, 11-14,* and *18* for examples). These maps are visualized next to live camera feeds from the AR0134 forward-facing camera. RViz allows for 2D navigation targets (waypoints) to be selected; these waypoints are sent back to the MAV for interpretation and path-planning.

A Python script converts the three-dimensional map data stream from ROS in to an ASCII PLY (polygon file format) which allows for the data to be saved. The PLY file can be read by various 3D mesh visualization softwares, such as the open-source MeshLab software (see *Fig. 17*).



Figure 17: 3D point cloud map generated by the MAV, visualized in MeshLab.

The open source Ground Control Station software QGroundControl is used to monitor the MAV's altitude, attitude, horizontal velocities, GPS position, and battery characteristics. QGroundControl also allows for the selection of GPS waypoints for long-range outdoor navigation.

The Remote Laptop also allows for the selection of various flight modes that are available for use on the MAV.

- Takeoff: climb to cruise altitude. Outdoors, this is 50ft while indoors it is 4ft.
- Land: descend until the ultrasonic sensor measures an altitude of <3in (i.e. MAV is on the ground).
- Hover: maintain heading and spatial location, counteract any deviations from the originally set location and heading.
- Manual: Use remote controller input for horizontal movement, climb/descend, and yaw inputs. When the sticks are released and centered, maintain the current position, altitude, and heading.
- Assisted Manual: Manual mode, but use LIDAR and cameras to detect obstacles. Reject controller inputs if they would result in approaching an obstacle at a configurable distance (default 1ft).
- GPS Guidance: Receive target GPS coordinates from the Remote Laptop. Climb to 50ft altitude and follow straight-line path to the desired location.
- Indoor Exploration: Fly autonomously towards areas that have comparatively lower map density to collect more data. Avoid potential obstacles.
- Indoor Navigation: Receive target waypoints from the Remote Laptop. Generate paths to navigate to the desired waypoints, and fly autonomously to the target waypoints while avoiding obstacles.

Testing

Throughout the development of the MAV, small tests of individual sensors and software components were continually run. In general, tests of each individual sensor were completed before any attempts at sensor fusion. These tests helped verify the functionality and accuracy of the sensors and associated software.

Once the software was close to complete, various flight tests were performed in indoor and outdoor environments.

Indoor tests of the full software stack were

completed. These tests included verification of the Indoor Exploration and Indoor Navigation flight modes, as well as evaluation of the accuracy of the generated maps. Tests were completed first in a garage, then on two levels of a house. All lighting conditions were present within each test, for example, bright lighting in a sunlit room, normal lighting in an artificially-lit hallway, and low light in a room with the lights off. The MAV was observed to handle all of these environments, intelligently switching to the different lighting modes when necessary, and slowing down when in darker environments.

The 2D maps were compared to existing blueprints of the environment that was mapped (see *Fig. 18*). Distance measurements and room areas were verified to be accurate to within 2% of their blueprint value.

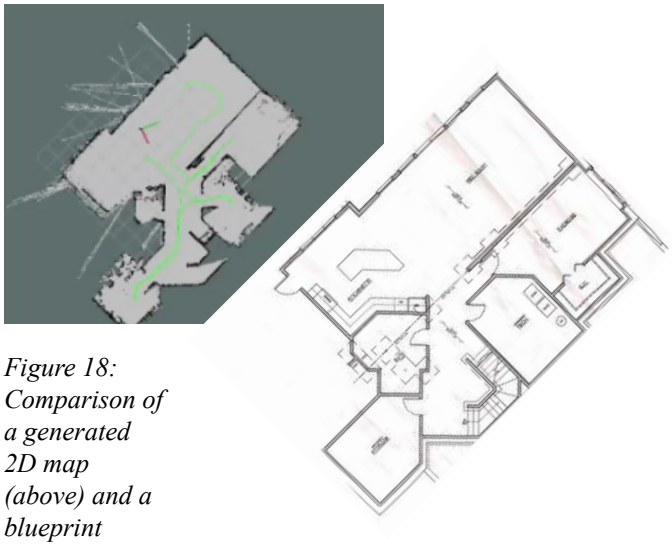


Figure 18: Comparison of a generated 2D map (above) and a blueprint (right) of the same area.

Outdoor flight stability tests were performed in wind conditions including calm ~0 mph, light breeze ~7 mph, and moderate breeze ~15 mph (see *Fig. 19*). The stability of the MAV was tested by commanding it to takeoff to 5m altitude, hold position for 5 minutes, then land. The distance between the takeoff location and the landing location was measured to characterize the rate of drift. This process was repeated 3 times during each wind condition. In the conditions tested, the MAV was very stable in the position hold; drifting less than 5 cm (2 in) per minute.

GPS navigation was tested by randomly generating GPS waypoints within a 10m x 10m space, then navigating to those locations. Qualitatively, this functionality appeared to work as designed. However, a method for quantitatively measuring the accuracy and precision of the GPS navigation was not determined. During calm wind conditions the MAV was flown at a top speed of 12.5 m/s (28 mph); though it could probably exceed this speed, this was the highest speed tested and verified.



Figure 19: An outdoor flight test of the MAV.

Discussion

The MAV met all of the constraints and the system as a whole functioned quite well, exceeding expectations in speed and accuracy. The ability to effectively navigate a wide variety of environments and lighting conditions has numerous potential applications, including search and rescue, general surveillance, warehouse management, architectural surveying, package delivery and more.

One key limitation of the MAV is three-dimensional obstacle avoidance. The LIDAR sensor enables two-dimensional obstacle avoidance, but it misses any objects that are outside of its detection plane. For example, a tall table may be too low to be detected by the LIDAR, but it may still pose a collision risk with the MAV landing gear. To address this, three-dimensional obstacle avoidance is handled by the forward-facing camera; this means that the MAV can only avoid obstacles in three-dimensional when traveling in the forward direction. When traveling in dense environments, the MAV must always fly in the forward direction and rotate to travel in other directions, sacrificing some agility. To enable obstacle avoidance in all directions, more cameras (facing different directions) would be needed.

There are a few developments that would greatly increase the usability of this project in commercial applications or make it more effective

in general.

The first would be to complete the final objective: the creation of a user-friendly ground control station. Currently, the remote laptop runs several independent programs that each achieve a single goal. For development this works fine; however, it isn't an ideal solution for general use. As a result, the development of a single program that combines all of the other ones in an intuitive interface would increase usability, allowing for use of the MAV with minimal training.

Another would be to improve the reliability of the automatic environment detection. Currently, environment switching happens primarily by analyzing the number of GPS satellites and the density of LIDAR data to detect the current environment (indoor or outdoor). By implementing a machine learning classifier, likely a Logistic Regression or Predictive Learning Model, data from all sensors could be interpreted to detect the current environment type more reliably.

A long-term goal is the elimination of LIDAR altogether. With additional processing power still available on the NVIDIA Jetson TX2, it could likely handle more cameras. Replacing the LIDAR with a pure-camera system would reduce weight and moving parts. Additionally, it would allow for three-dimensional obstacle avoidance in all directions, improving the MAV's agility and ability to navigate very dense environments.

References

Bailey, T., & Durrant-Whyte, H. (2006). Simultaneous localization and mapping (SLAM): part I. IEEE Robotics & Automation Magazine,13(2), 99-110. doi:10.1109/mra.2006.1638022

Bailey, T., & Durrant-Whyte, H. (2006). Simultaneous localization and mapping (SLAM): part II. IEEE Robotics & Automation Magazine,13(3), 108-117. doi:10.1109/mra.2006.1678144

Colomina, I., & Molina, P. (2014). Unmanned aerial systems for photogrammetry and remote sensing: A review. ISPRS Journal of Photogrammetry and Remote Sensing, 92, 79-97. doi:10.1016/j.isprsjprs.2014.02.013

Gonçalves, J., & Henriques, R. (2015). UAV photogrammetry for topographic monitoring of coastal areas. ISPRS Journal of Photogrammetry and Remote Sensing, 104, 101-111. doi:10.1016/j.isprsjprs.2015.02.009

Guo, C.X., DuToit, R.C., Sartipi, K., Georgiou, G., Li, R., O'Leary, J., Nerurkar E.D., Hesch, J.A., Roumeliotis, S.I. (2015). Resource-Aware Large-Scale Cooperative 3D Mapping from Multiple Cell Phones. ICRA Late Breaking Results Poster, 26-30,

Hess, W., Kohler, D., Rapp, H., & Andor, D. (2016). Real-time loop closure in 2D LIDAR SLAM. 2016 IEEE International Conference on Robotics and Automation (ICRA). doi:10.1109/icra.2016.7487258

Honegger, D., Meier, L., Tanskanen, P., & Pollefeys, M. (2013). An open source and open hardware embedded metric optical flow CMOS camera for indoor and outdoor applications. 2013 IEEE International Conference on Robotics and Automation. doi:10.1109/icra.2013.6630805

Kohlbrecher, S., Meyer, J., Graber, T., Petersen, K., Klingauf, U., & Stryk, O. V. (2014). Hector Open Source Modules for Autonomous Mapping and Navigation with Rescue Robots. RoboCup 2013: Robot World Cup XVII Lecture Notes in Computer Science, 624-631. doi:10.1007/978-3-662-44468-9_58

Krishna, P. (2018). Development of autonomous unmanned aerial systems for semi-dense point cloud generation in disaster scenarios.

Lee, S., & Song, J. (2004). Robust mobile robot localization using optical flow sensors and encoders. IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA 04. 2004. doi:10.1109/robot.2004.1307287

Li-Chee-Ming, J., & Armenakis, C. (2014). Generation of Dense 3D Point Clouds Using a Small Quadcopter. Geomatica, 68(4), 319-330. doi:10.5623/cig2014-406

Lucas, B., & Kanade, T. (1981). An iterative image registration technique with an application to stereo vision. Proceedings of the International Joint Conference on Artificial Intelligence, 674-679.

Michael, N., Shen, S., Mohta, K., Kumar, V., Nagatani, K., Okada, Y., . . . Tadokoro, S. (2013). Collaborative Mapping of an Earthquake Damaged Building via Ground and Aerial Robots. Springer Tracts in Advanced Robotics Field and Service Robotics, 33-47. doi:10.1007/978-3-642-40686-7_3

Pestana, J., Mellado-Bataller, I., Sanchez-Lopez, J. L., Fu, C., Mondragón, I. F., & Campoy, P. (2013). A General Purpose Configurable Controller for Indoors and Outdoors GPS-Denied Navigation for Multirotor Unmanned Aerial Vehicles. Journal of Intelligent & Robotic Systems, 73(1-4), 387-400. doi:10.1007/s10846-013-9953-0

Qin, T., Li, P., & Shen, S. (2018). VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator. IEEE Transactions on Robotics, 34(4), 1004-1020. doi:10.1109/tro.2018.2853729

Qiu, H., & Duan, H. (2017). Multiple UAV distributed close formation control based on in-flight leadership hierarchies of pigeon flocks. Aerospace Science and Technology. doi:10.1016/j.ast.2017.08.030

Rudol, P., & Doherty, P. (2008). Human Body Detection and Geolocalization for UAV Search and Rescue Missions Using Color and Thermal Imagery. 2008 IEEE Aerospace Conference. doi:10.1109/aero.2008.4526559

Thrun, S., & Leonard, J. J. (2008). Simultaneous Localization and Mapping. Springer Handbook of Robotics,871-889. doi:10.1007/978-3-540-30301-5_38

Zhang, J., & Singh, S. (2015). Visual-lidar odometry and mapping: Low-drift, robust, and fast. 2015 IEEE International Conference on Robotics and Automation (ICRA). doi:10.1109/icra.2015.7139486

Appendix 1: MAV Bill of Materials

Subsystem	Name	P/N	Vendor	Price	Qty	Subtotal
MAV	S500 Quadcopter Frame	B01D4MK5MS	Amazon	$43.99	1	$43.99
	Motors, ESCs, Propellers Set	B00TROIMXW	Amazon	$75.66	1	$75.66
	4S 4000 mAh LiPo Battery	9067000104-0	HobbyKing	$40.42	1	$40.42
	FRSky X8R Radio Receiver	B01HZB6XTI	Amazon	$19.99	1	$19.99
Battery Monitor	ATMEGA328PB-MURCT-ND	ATMEGA328PB	DigiKey	$1.37	1	$1.37
	100A DC Current Sensor	CS-100A	Panucatt	$14.99	1	$14.99
	74HC595PW Shift Register	74HC595PW,118	DigiKey	$0.50	3	$1.50
	3 Digit 8 Segment Display	160-1545-5-ND	DigiKey	$3.23		$3.23
	Battery Monitor Passive Electronics	Many	DigiKey	$20.00	1	$20.00
	Battery Monitor PCB	Custom Part	JLCPCB	$10.00	1	$10.00
Flight Computer	Nvidia Jetson TX2 Development Kit	945-82771-0000-000	Nvidia	$299.00	1	$299.00
	Orbitty Carrier Board for Jetson TX2	PKG900000000691	ConnectTech	$174.00	1	$174.00
Flight Controller	ARM Cortex M4 STM32F301K8U6	497-17411-ND	DigiKey	$2.89	1	$2.89
	L3GD20HTR IMU	497-13931-2-ND	DigiKey	$3.42	1	$3.42
	MPU6000 IMU	497-13931-2-ND	DigiKey	$1.67	1	$1.67
	BNO055 IMU	828-1058-1-ND	DigiKey	$12.07	1	$12.07
	Flight Controller Passive Electronics	Many	DigiKey	$10.00	1	$10.00
	Flight Controller PCB	Custom Part	JLCPCB	$10.00	1	$10.00
GPS	u.blox NEO-M8N GPS Module	387000075-0	HobbyKing	$19.99	1	$19.99
Optical Flow	ARM Cortex M4F MSP432P401RIPZR	296-44715-1-ND	DigiKey	$7.95	1	$7.95
	MT9V034 Camera	387000059-0	OpenMV	$39.00	1	$39.00
	LV-EZ1 Rangefinder	1863-1005-ND	DigiKey	$24.95	1	$24.95
LIDAR	RPLIDAR A1M8 LIDAR Scanner	RB-Rpk-02	RobotShop	$99.00	1	$99.00
Camera	AR0134 Global Shutter Camera	RB-Adu-39	RobotShop	$54.00	1	$54.00
	ArduCam USB Shield	RB-Adu-32	RobotShop	$49.99	1	$49.99
	Total					$1,039.08