CONTROL OF AN UNMANNED AERIAL VEHICLE RELATIVE TO A WALL

By
Ryan Sass

MECHATRONIC CAPSTONE PROJECT REPORT

Submitted to
the Department of Mechanical Engineering

of

SANTA CLARA UNIVERSITY

in Partial Fulfillment of the Requirements
for the degree of
Master of Science in Mechanical Engineering

Santa Clara, California

2016

Control of an Unnamed Aerial Vehicle Relative to a Wall

Ryan Sass

Department of Mechanical Engineering
Santa Clara University

2016

ABSTRACT

Automated control of UAV aircraft reduces pilot workload and increases capability for payload operation. However, most commercially available UAV aircraft do not possess automated control relative to other objects, just absolute position control of the aircraft's location. This ability to maintain relative control would further increase payload usefulness and reduce pilot workload even further. This paper covers the development and design of an off-the-shelf UAV to operate autonomously in relation to a stationary wall. This project demonstrates that simultaneous yaw and pitch control loops can be executed on a microcontroller, installed on the drone aircraft, to perform wall-standoff control. For more robust automated control, improvements beyond this project will be necessary to achieve finer control motion and more sophisticated control. However, this first step in demonstrating automated aircraft control relative to an object can still provide practical uses for drone payload operation.

Figure 1 - Capstone Project Drone in Flight
Image courtesy of Ryan Cooper, Santa Clara University

Acknowledgements

The development of this project would not have been achieved without the support of the following people and entities:

- Dr. Christopher Kitts, Santa Clara University – Project advisor, project guidance and overall project planning.
- Mr. Ryan Cooper, Santa Clara University – Adafruit Micro SD Card Breakout Board Software Development and Implementation, authoring of sections 3.6.7 and 3.7.
- Jacob Osoke, Andrew Drape, Brayton McKnight, Franz Plum, Santa Clara University – Lidar Sensor Mount Development and Prototyping.
- Anne Mahacek, Santa Clara University – Maker Lab access, and tool borrowing.
- Killian Poore, Santa Clara University – General guidance and reference for RC peripherals.
- Aero Micro - RC Part Store.
- 3D Robotics – X-8 and Pixhawk Troubleshooting and Support.
- Current UAS – Motor Replacements and X-8 Support
- RC Timer – Fast and quality Motor Replacements!

Table of Contents

1. Introduction

1.1. Unmanned Aerial Vehicles and Drones

Unmanned aerial vehicles or UAVs are becoming more and more prolific. Ranging from mere inches in size and carrying no payload, to tens of feet across and the ability to carry weapons and heavy sensors, the usefulness of these computer-piloted aircraft continues to expand into more and more applications. The type of UAV varies from aircraft resembling conventional looking airplanes and helicopters, to more unique shapes, such as quad-, hex-, and octocopters.

Figure 2 - RQ-4 Global Hawk UAV
Image courtesy of Author Stacey Knott, Wikipedia, and The United States Air Force

One rapidly expanding UAV market involves quad/hex/octocopter vehicles. These aircraft, commonly referred to as 'drones', usually vary in size from a few inches to approximately 2-3 feet across. They are commonly identified by their four, six, or eight motors arranged in a somewhat circular pattern. They are usually electrically powered, although they can be powered by other means such as gas motors.

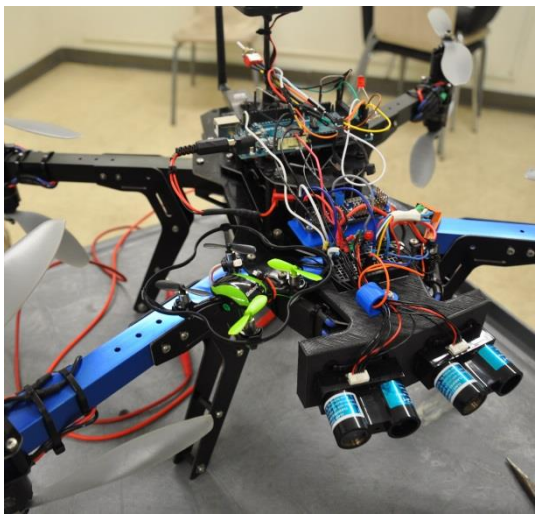

Figure 3 – UDI Nano Quadcopter sitting on the capstone project drone
Image courtesy of Ryan Sass, Santa Clara University

These drones have a few unique characteristics that make them very simple for someone to use. The airframe is simple to build (or prebuilt), there are no control surfaces aside from the spinning propellers to build, mount, or maintain, and when combined with a simple flight computer mounted to the airframe, these drones are very stable and simple to fly. While many drones or remote controlled aircraft are flown using a hand-held remote controller, many controllers allow automated navigation via command uplink or preloaded waypoints.

1.2. Flight Computers

Copter-style vehicles are difficult to pilot without the assistance of a flight computer. This is because small disturbances in aircraft movement lead to large changes in aircraft velocity, making the vehicle difficult to control. Drones equipped with onboard flight computers will sense the orientation and acceleration of the aircraft in all three dimensions, as well as the control the input from a user's remote control. From this basic data, the computer will calculate the

proper orientation the aircraft should assume, and, having specified the configuration of the type of aircraft (usually H, X, or circular hex or octo), it will adjust each of the 4, 6, or 8 motors to achieve aircraft control under the four degrees of independent control: roll, pitch, yaw, and thrust. These adjustments occur in response to the user's input as well as to keep the aircraft stable and compensate for things such as slight differences in motor performance at a given power setting. This makes tasks such as a copter aircraft hovering incredibly simple.

As the sophistication of these flight computers increases, the functionality of these drones does as well. The sensors that these computers use extend from just accelerometers to measure the aircraft's orientation. More sophisticated computers use compasses for heading, barometers for altitude, GPS for precise location, and telemetry for sending parameters to a ground station. These modern flight computers can perform many autopilot features, including holding heading/position/altitude, flying at a specified speed/heading/altitude, automatically taking off and landing, and loitering or circling over a certain location. They can also combine these functions to fly complete missions, flying to a sequence of pre-specified GPS coordinates. They can activate payloads automatically as well, and some have failsafe modes for responding to conditions such as low battery, the loss of a communication link, or flying outside a pre-defined area. All these automated tasks can be transmitted to the aircraft via telemetry link before or during flight. Because of this sophistication, these flight computers can maintain absolute control over the stability and position of an aircraft.

Figure 4 - APM 2.6 Flight Computer
Image courtesy of 3DR

1.3. Drone Uses and Applications

This absolute control over an aircraft allows drones to support a wide range of automated tasks including aerial surveillance, aerial surveying, filmmaking, journalism, law enforcement, search and rescue, scientific research, disaster relief, archaeology, cargo transport, and agriculture. In almost all cases, the drone itself operates as the platform that a payload will operate from to achieve given tasks.

Figure 5 - md4-1000 drone testing for DHL package delivery
Image courtesy of Author Frankhöffner and Wikipedia

Common types of payloads include cargo, cameras, weapons, radar, sensors, and transmitting devices. These payloads can be as simple as a user-activated camera, or as complicated as computer-assisted, visual tracking devices. The payload can be used for commercial applications, such as shipping of

goods, research applications such as geological surveying, or military applications, such as firing of weapons.

Operation of the payload and the aircraft sometimes requires either automation or multiple users. Consider, for instance, a drone equipped with a camera equipped to pan and tilt with user control being used for surveillance. In order to properly surveil an object of interest, not only must control of the aircraft be maintained in order to provide the proper location for the camera payload, but the camera itself must be operated in order to properly record the object itself. If the aircraft moves, aircraft and camera control must be performed simultaneously in order to maintain surveillance.

This is where flight computer automation can prove practical and reduce operator workload. For instance, the operator flying the surveillance of a drone can set the aircraft to maintain or circle it's a location described by GPS coordinates while the operator focuses on operating the payload.

While these flight computers can enable users to autonomously accomplish dirty, dull, dangerous, or remote tasks, there are some limitations to what these drone computers currently can accomplish.

1.4. Current Shortcomings of Today's Drones

One major shortcoming of drones is their lack of automated control relative to other objects. This relative control is paramount for object recognition and avoidance, as well as a maintaining flight automation without accurate GPS connection, such as in the event of bad reception or flying indoors. Most drones are not equipped with visual or proximity sensors to detect objects that could pose risk of damage to either the aircraft or the imposing object.

Most drones are not equipped with proximity sensors or flight computers configured to recognize objects that pose risk to drones. At the moment, the most prudent way for drones to avoid objects while performing automated flying missions is to set flight parameters so that the drone either flies above the hazard or flies with enough space between it and the hazard to avoid a collision.

Figure 6 - Flight parameters to avoid hazards, such as this building
Image courtesy of Ryan Sass, Santa Clara University

While automated flight to avoid flying near objects is good practice to avoid collisions, it does limit what drones can accomplish. Things such as building inspections, indoor flight, close

proximity remote photography, and 3D object mapping are limited to manual drone operation because current available drones are not equipped with object recognition or collision avoidance capability. Despite this, manual flying of a drone in close proximity to objects is difficult and a considerable risk. Things such as disturbances, user disorientation, confusion, or distraction can still result in collision due to user error.

Automating the aircraft to perform relative control – such as keeping a certain distance from an object, would reduce pilot workload and provide safer drone operation while the user controls the payload or drone position.

Figure 7 - A drone performing a building inspection
Image courtesy of Building Enclosure Consulting LLC

An example where this relative control would prove useful would be using building inspections. If the intended need is to photograph the side of a tall building at a close enough range for inspection purposes, manually lifting a person to each location to take a picture could prove difficult, unsafe, and impractical. Having a drone hold a steady distance away from the side of the building, while a user operates the drone's position vertically as well as laterally in order to position the payload properly would be much safer and faster.

1.5. Capstone Project Description and Purpose

The purpose of this project is to exhibit control of yaw orientation and distance of a drone relative to a wall. To do this, the following tasks were performed: design of a sensor package, development of dual control loops in an Arduino microcontroller, integration of the sensor package and microcontroller with the existing drone and its autopilot, and experimentation and testing of the drone to prove project capability. The result of the project is a drone that operates the pitch and yaw axes of flight control autonomously to demonstrate basic "plane lock" with a vertical wall. This "plane lock" can be characterized as the drone still exhibiting user control in the roll (left / right) and thrust (up / down) axes but

Figure 8 - The complete automated drone system used for this capstone project
Image courtesy of Ryan Sass, Santa Clara University

holds orientation and distance along an invisible vertical plane parallel to the wall at a set distance away. This allows the drone to fly autonomously and follow the wall's features, easing operation of manual flight.

Observing the drone fly when automation is first activated midflight, the drone will align itself (yawing or turning left and right) and proceed to maintain a pre-programmed normal distance to a wall (back and forth) with an accuracy of about two feet. The drone will still require user control of the roll (left and right) and thrust (up and down) axes, as well as user control for modes of flight such as liftoff, pre-automated flight positioning, and landing.

1.6. Project Performance Objectives

Because this project is an exercise in demonstration of control loop implementation on a drone aircraft, discernable, measurable, and reasonable project objectives were hard to define because it was not clear at the start of the project if this type of control actuation could even be implemented. A few performance objectives were set that could at least give the project direction. These performance objectives included:

- Demonstrate Aircraft Control with the use of yaw and pitch control loops.
- Demonstrate Yaw Control within ± 20° from normal.
- Demonstrate Pitch Control within ± 3 ft. of a given set point.

Due to the unknown nature of the vehicle's performance under automated control and ease or ability of operation, no further project definitions, such as response time or settling time, were defined. FAA regulations and existing school policies required that project testing had to be performed indoors in a very limited workspace. This prevented more aggressive control objectives for this project. Nevertheless, basic control was achieved and refinements are being planned as future work.

2. System Overview

2.1. System Component Description

The project is comprised of three separate sub-assemblies that have been merged together onto one operational aircraft. The drone itself is a standalone system. With an RC transmitter and telemetry antenna connected to a ground station, the drone is capable of manual flight, as well as fully autonomous flight, provided GPS connection has been established and the drone's parameters and proper flight mode have been set. This aircraft has been specifically designed to fly payloads, and the Pixhawk flight computer that controls it has been developed by a team of engineers outside the scope of this project. Both are consumer products that have been modified for this project in minor ways to meet project goals.

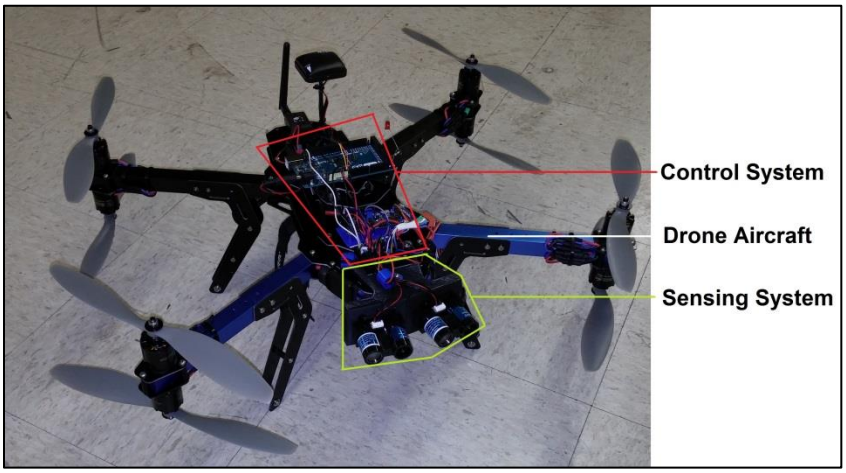

Figure 9 - Project aircraft with subsystems in outline
Image courtesy of Ryan Sass, Santa Clara University

The other two systems that comprise a majority of development for this project are the sensors and control system. This system is almost completely isolated from the Pixhawk and drone in terms of operation. These systems include an Arduino, PWM board, Lidar sensors, SD Card reader and 9V power supply, all of which are mounted on the top of the aircraft. This system provides the distance sensing and calculation, control loop execution, and control signal actuation to the Pixhawk and drone assembly to achieve automated flight in the "plane lock" mode. The Pixhawk is not designed to operate in this fashion when purchased directly off the shelf.

It is important to understand, too, how this actuation occurs. From the point of view of the Pixhawk, the operation of the drone under manual or automated Arduino control is indistinguishable, and the Pixhawk does not change any of its operational flight behavior during the shift between automated or manual control. The only change that occurs in flight is the origination of input signals for yaw and pitch: either coming from the RC receiver by way of the RC transmitter from a user, or from the PWM board by way of the Arduino board from the control loop iteration.

In this way, the Pixhawk and drone provide a stable flight test platform in which to demonstrate the control capability of the Arduino. When all neutral control signals are sent to the Pixhawk,

the Pixhawk continues to maintain stability and level orientation of the aircraft. When any input signals move from neutral, the Pixhawk works to control the stability of the aircraft in the new flight attitude, without any regard to the Arduino control loops or set points. Therefore, the control exhibited by the Arduino would be no different than if a user was attempting to control the drone manually.

2.2. Drone System Overview

2.2.1. Introduction to X-8 Drone

The drone aircraft is a 3D Robotics X-8, an octocopter type platform. The X-8 platform is an aircraft with 4 main motor booms, oriented in an X shaped pattern. This means, in comparison with a conventional aircraft, the 'nose' of this aircraft is situated approximately 45° between two of the motor booms. The '8' in the X-8 description is to indicate that this drone has 8 motors, 2 motors mounted on each motor boom. It has dimensions of approximately 22" x 28" x 12" (560mm x 710mm x 305mm).

Figure 10 - 3DR X-8 Drone
Image courtesy of 3DR

2.2.2. Propellers, Motors and Airframe

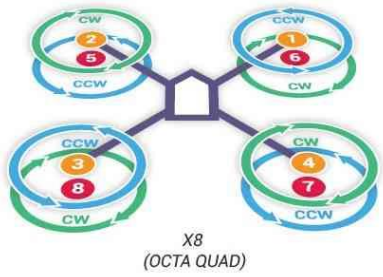

Figure 12 - Motor spin direction for the X-8
Image courtesy of 3DR

The propellers are approximately 10" in diameter. There are four clockwise spinning propellers and four counter-clockwise spinning propellers. They are mounted to each of the motors in a pattern where each motor boom has counter-rotating propellers. Each motor's propeller spin direction is opposite of the closest three motors. For reference, the top right motor spins CCW[1]. Establishing motor spin direction is important for proper control of the aircraft, as roll, pitch, and yaw, and aircraft stability are achieved by varying the speed of the proper motors in order to achieve each type of movement.

The airframe of this aircraft is comprised of carbon fiber for the fuselage and aluminum square channel for the motor booms. It has 'feet' type landing skids, which are comprised of carbon fiber panels with spacers inserted to give substantial width to the feet.

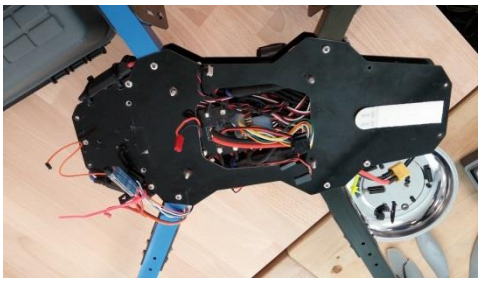

Figure 11 - Interior of aircraft where speed controllers and compnents are located
Image courtesy of Ryan Sass, Santa Clara University

The main fuselage is comprised of two pieces of carbon fiber sandwiching the ends of the four aluminum motor booms. This gives the fuselage an upper, lower, and

interior location for mounting components. There is also a smaller upper housing on top of the upper part of the fuselage for extra mounting locations.

The lower part of the fuselage holds the battery as well as an indicator buzzer. The interior houses the Electronic Speed Controllers (a module which regulates power going to a motor and modulates speed based on a control signal), power distribution blocks and power module for the motors and 5V peripherals, and the PPM sum receiver (the component that combines the 8 PWM input channels). On the upper part of the fuselage, the flight computer, a Pixhawk PX4, the safety switch, telemetry antenna, and GPS & compass module are mounted. In the space of the landing skid feet, the RC receiver is mounted for manual user control via RC transmitter.

Figure 13 - Top of the X-8 showing motor booms and landing skids
Image courtesy of 3DR

2.2.3. Pixhawk Operation

The Pixhawk flight computer manages all flight operations. In normal operation, the drone can be flown manually by a user sending signals from an RC transmitter to the RC receiver. The receiver is connected to a PPM sum receiver and combines all 8 control signals and sends them directly to the Pixhawk. From there, the Pixhawk will handle control of the aircraft motors to achieve the drone flight commanded from the user. The Pixhawk can also be programmed to fly autonomously by uploading parameters via telemetry or USB and establishing a GPS connection[2].

This means the drone has many modes of operations and can be flown completely manually, in partial autonomous modes (such as Altitude hold mode where throttle level is controlled), or completely autonomous modes (such as flying a route with waypoints, or loitering around a user selected point at a specified altitude.

Figure 14 - Pixhawk PX4 flight computer
Image courtesy of 3DR

All these modes of flight operation assume the craft is being flown outdoors, as the Pixhawk does not have a way of recognizing any potential obstacles if flying low enough to pose a risk of collision. For the rest of this report, it will be assumed the drone is operated only in the complete manual Pixhawk mode when switching between user control and automated Arduino flight. The project was designed around the Pixhawk operating only in this STABALIZE mode, and in LAND mode for landing purposes.

2.2.4. Pixhawk Flight Modes

The two flight modes the Pixhawk uses for this project were LAND and STABALIZE mode.
LAND mode – where the thrust axis is set to safely lower the aircraft to the ground – is used as a failsafe mode in the event of an unsuitable flight condition during normal operation.
STABALIZE mode – allowing a user to fly the vehicle manually, but will self-level the roll and

pitch axes, and hold the yaw heading when flight controls are neutral. In either of these Pixhawk modes, no relative control is calculated by the Pixhawk, the Pixhawk only executes the commands issued by the user or the automation controller.

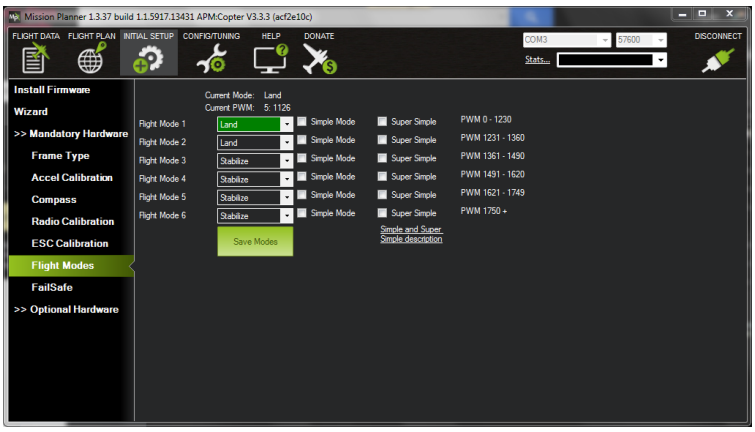

Figure 15 - Pixhawk flight modes
Image courtesy of Ryan Sass, Santa Clara University

2.2.5. Control Signals To and From the Pixhawk

For the Pixhawk to achieve proper flight orientation and control, the Pixhawk conventionally receives flight control signals from a user via RC transmitter - in this case an 8-channel transmitter. Four of these channels are dedicated for the main axes of flight: pitch, roll, yaw, and thrust. The other four channels are auxiliary channels, two of them being utilized in this project – one for specifying Pixhawk flight modes and the other for enabling automation of the pitch and yaw axes.

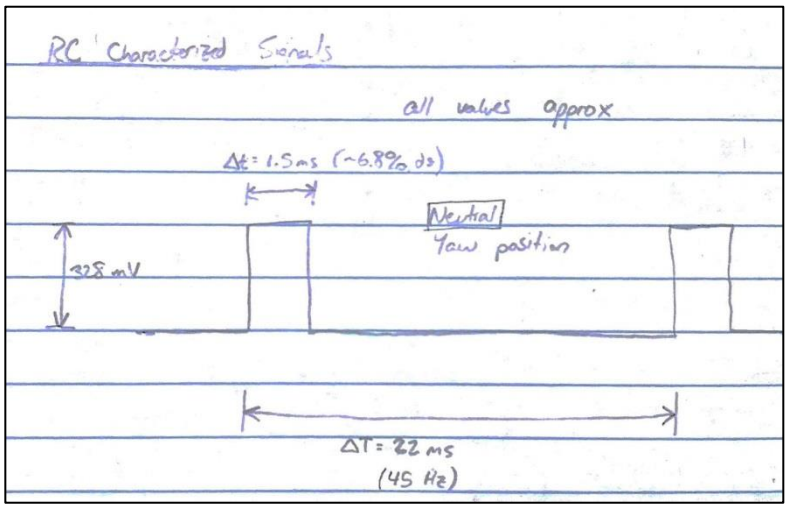

Figure 16 - RC Control Signals
Image courtesy of Ryan Sass, Santa Clara University

These 8 channels from the user are sent from the RC transmitter to the RC receiver on the aircraft. The signals, from transmitter to receiver are in the form of PWM wave – Pulse Width Modulation, where signal changes varying on the amount of the cycle being "high" versus "low". The characterized signal is ~380mV, with a 45 Hz rate, and about a 7% duty cycle at 50% axis rate. The signal will vary from about 5% to 9% at the extreme ends of an axis (i.e. 0% and 100%).

From the RC receiver, the 8 signals, or channels, are combined through a PPM sum receiver. This PPM (Pulse Position Modulation) sum receiver combines each of the 8 signals into a single signal by spacing each pulse a given distance apart and summing their signals into a single 8 pulse signal. This signal will carry all 8 channels worth of information during one transmission cycle. From the PPM sum receiver, the signal is sent to the Pixhawk where it is read.

Once the Pixhawk reads the incoming signals from the PPM sum receiver, it interprets the commands it receives and calculates how the aircraft must move in response to the

corresponding control signals. Once the proper movement is determined, the Pixhawk calculates how each of the 8 the motors must be adjusted to achieve the proper movement, and control signals to each of the ESC's are updated. These PWM signals for each of the 8 motors are sent out to each of the respective motor's ESCs, which adjust each motor speed accordingly.

2.2.6. Basic Power

The Pixhawk receives power via the power module, a voltage regulator that takes the drone's 14.7V battery power and regulates part of it to 5V, which supplies the Pixhawk. This power module supplies the Pixhawk all the power it needs to power all other drone components, including the GPS/compass module, telemetry antenna, RC receiver, buzzer, and safety switch. The motors receiver power via the ESC's which receiver power from the 14.7V battery through the power distribution blocks. This modulates power going to each one ESC in order to protect overloading an individual motor.

2.3. Sensing System Overview

2.3.1. Introduction

The sensing system is the main component that was developed to sense relative distance and orientation to a vertical wall. The sensing system includes two Lidar sensors, a specially designed mount affixed to the front of the aircraft, and an Arduino to support operation of the sensors.

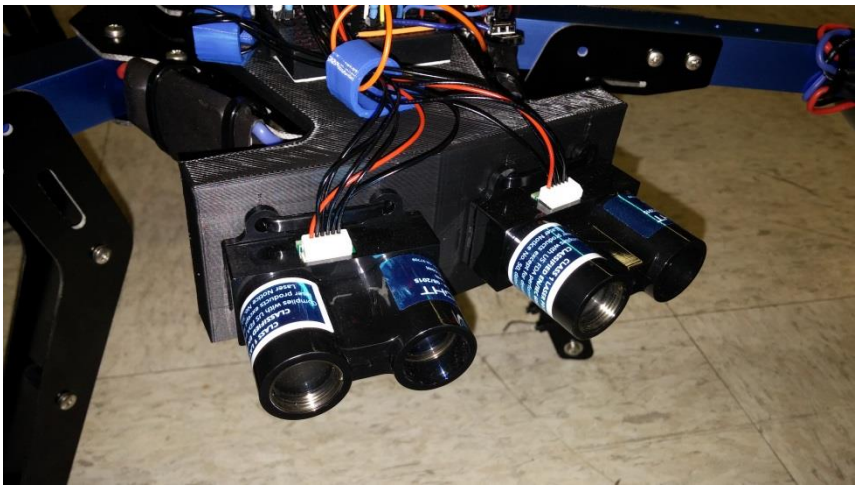

Figure 17 - Sensing system close-up showing mount and sensors
Image courtesy of Ryan Sass, Santa Clara University

2.3.2. Lidar Sensors

Two LIDAR-Live v2 "Blue Label" distance sensors were used to sense the yaw angle and distance the aircraft is from a wall. Each sensor emits a laser light pulse to determine the distance it is from an object, and outputs a calibrated signal corresponding to the distance the sensor reads.

The Lidar sensors operate best over a distance range approximately 250 cm to 1250 cm from an object. The Lidar sensors can perform measurement sensing at speeds up to 500 readings a second[3]. It is powered by 5V from and communicates via I2C communication protocol. Both power and communication for the two sensors were handled directly by the Arduino board.

Consideration must be taken with these sensors to also provide proper error rejection. If the sensors fall out of range or to not get an accurate reading, the values provided from the Lidar sensors could be wildly inaccurate.

2.3.3. Mount

These mount was specifically designed to hold the two Lidar sensors to the aircraft and in the proper position. This mount holds the two sensors in a differential-style configuration. The sensors are each tilted 7.5° from a direct normal position when facing the wall. This angling of the sensors increases the sensor package's sensitivity to slight changes in yaw angle.

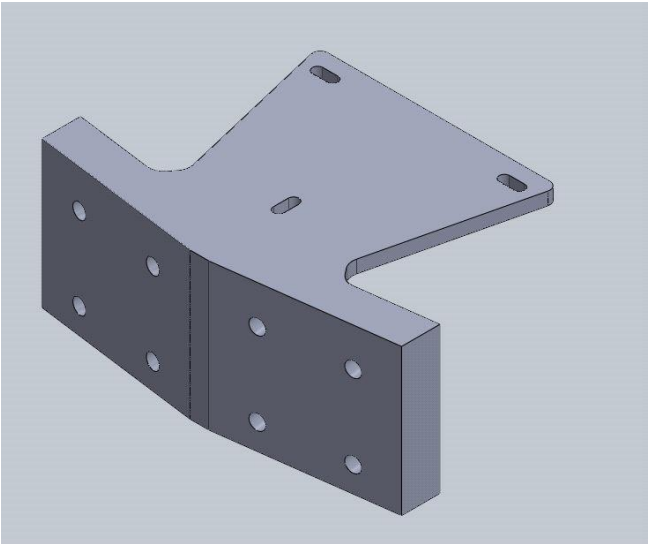

Figure 18 - Lidar sensor mount 3D model
Image courtesy of Ryan Sass, Santa Clara University

With differential style sensing, careful consideration should be used when sensing at long distances, as the further away the sensor is from an object, the wider the wall must be for the sensors to read accurately. This differential sensing is important, though, because it allows the calculation of yaw angle to the wall. When performing relative distance control, it is necessary to stay as 'squared up' to the wall as possible to provide the most accurate distance readings, which in turn improves distance control accuracy.

The Lidar sensor package was mounted on the very front of the aircraft, pointing forward. This positioned the sensors such that their laser projections occurred at a level between the upper and lower spinning propeller blades. This allows the Lidar sensors to take readings with minimal obstruction or interference from the propeller blades.

2.4. Control System Overview

2.4.1. Introduction

The control system consists of the two control loops that operate simultaneously to automate the yaw and pitch control of the aircraft during flight. These two control loops operate using PID-style control. The yaw control system uses only proportional control while the pitch control uses proportional and derivative control.

Originally, the control loops were to work separately – control yaw until the craft is squared to a wall and then control pitch. This proved to be problematic and not very functional during flight, so the two control loops actually work in parallel.

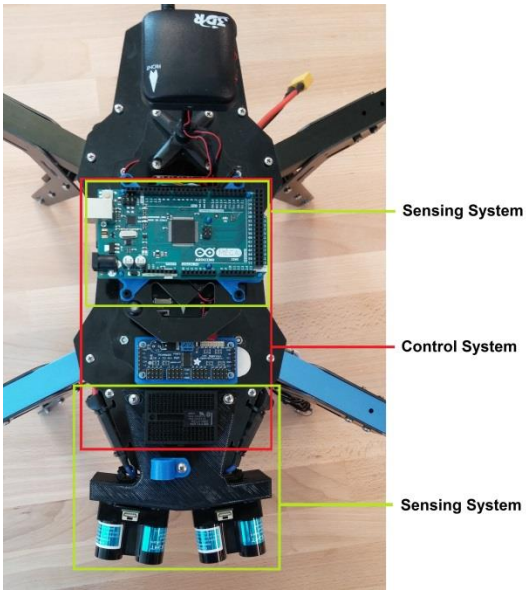

Figure 19 - Control and Sensing System Layout
Image courtesy of Ryan Sass, Santa Clara University

The basic operational flow involves continual control for the yaw command. If the yaw angle from the wall is within a certain range of acceptability, the pitch control loop will also operate.

Both control loops are loaded and executed on an Arduino board, the same as the Arduino board used for the sensing system as well. There are breakout boards for the control system to handle recording data as well as proper output for system integration. The operation of the control loops is continuous, and a user set switch from the RC transmitter will activate when the control loop signals are sent to the Pixhawk. This allows for full manual control until the user is ready to operate in autonomous mode.

2.4.2. Yaw Proportional Control

The yaw control is a proportional control loop. This means that the further the drone is turned away from the wall, the larger the control signal will be to turn it back to its proper orientation. As it reduces its yaw error, the control signal to turn the aircraft more reduces until it reaches a deadband, and the yaw command will return to neutral.

When controlling the yaw of the drone, the Pixhawk does not exhibit drift of yaw. This is characteristic of the STABALIZE flight mode, however strong enough control signals will produce minor yaw drift after returning abruptly to neutral. Because of this, derivative control is not necessary to achieve acceptable control.

2.4.3. Pitch Proportional/Derivative Control

The pitch control is performed with a proportional and derivative control loop. This means that the further away the drone is from a given distance from the wall, the harder it will push to return to the set point. However, the derivative part of the control loop will oppose the push of the proportional control, especially if the drone moves very quickly. This derivative control provides correction from large control overshoot of the set point as well as control loop stability.

When controlling the distance the drone is from the wall, the Pixhawk can drift a decent amount when under neutral pitch control, even when the drone is ‘trimmed’ properly. This means that if the drone is given a decent ‘push’ forward and then return to neutral, the Pixhawk does not bring the drone back to an immediate stop, but just returns the aircraft back to a level orientation. Without drift correction, the drone will coast, much like how a speeding boat will continue to coast after throttling down. Due to this aircraft drift, the derivative control is very important. The drift that occurred from proportional control alone was enough to force the aircraft to oscillate in an unstable control mode.

2.5. Integration Overview

The integration of these two systems allows the aircraft to operate as a standalone package, with all components (except the RC transmitter) on board the aircraft. The sensing package is mounted to the front of the aircraft, and its signals and power being provided and received by an Arduino microcontroller mounted on board the aircraft.

This control loop system is mounted to the top of the drone. The control signals are continually generated by the Arduino, and through the use of a remotely activated switch, are toggled to be received by the Pixhawk for flight automation.

This method of integration allows the drone to switch between operation in manual and automated flight modes seamlessly. This occurs because both manual and automated control signals operate continuously throughout the flight but only one set of signals reach the Pixhawk. Therefore, as long as the Pixhawk remains in its STABALIZE mode the drone can switch back and forth between automated mode and manual control mode without any interruption to the Pixhawk.

Roll and thrust will remain under control of the user, and during flight operation, minor adjustments will be required to maintain proper lateral and altitude spacing. This is where demonstration of the functionality of relative control automation is useful for things such as surveying or inspection of objects such as tall building, walls, or areas that are hard for humans to normally reach.

3. Detailed Design Description

3.1. Introduction

The following chapter will discuss in detail what was developed for the project over the course of the year. The project is broken into the following different areas: sensing system, control system, system integration, electrical wiring and layout, software design, data collection, and Pixhawk configuration. This project stated from scratch, and as a result, there were many different aspects of the project to develop in order to successfully demonstrate operability.

3.2. Sensing System Design

3.2.1. Lidar Sensing System

The design of the sensor system is a specially designed mount with two Lidar sensors attached to it. The system attaches to the front of the aircraft, and holds each Lidar sensor at a slight angle (7.5°) from facing directly forward. This angled difference makes the sensors, acting in unison, more sensitive to angular orientation.

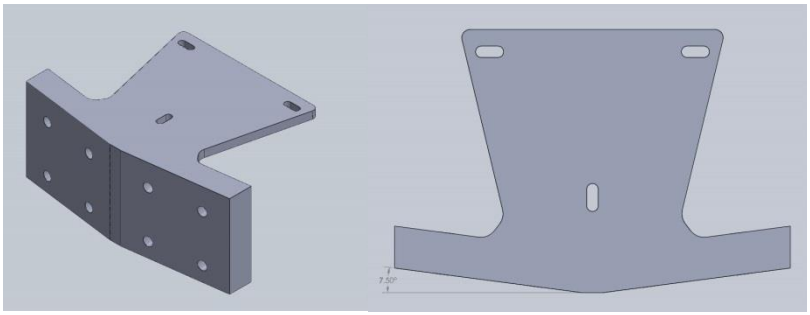

Figure 20 – Isometric and top view of the sensor system mount
Image courtesy of Ryan Sass, Santa Clara University

The 7.5° mounting angle was chosen after the first design iteration trying a 10° mounting angle. This larger angle required a much wider area of wall in order to provide accurate measurements, especially when the sensor system is at further distances. The mounting angle was reduced in order to reduce the area of wall required, but also maintain some sensor sensitivity.

To sense the distance from the wall, the two sensor distance readings are averaged. To sense the relative angle from the wall, a sophisticated calculation is performed.

In order to find the angle the Lidar sensor is with the normal of the wall, it is assumed that the sensors were mounted 15° from one another, and that the distance between sensors was sufficiently smaller than the distance from the wall, and was assumed to be zero. This allows triangulation to occur using law of cosines formula:

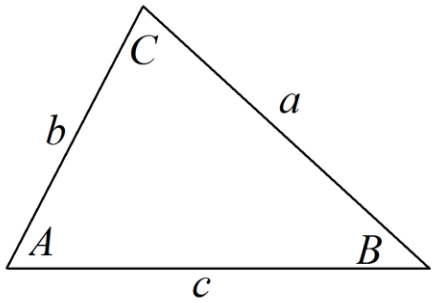

Figure 21 - Law of Cosines
Image courtesy of Ryan Sass, Santa Clara University

$$c^2 = a^2 + b^2 - 2ab\cos C$$

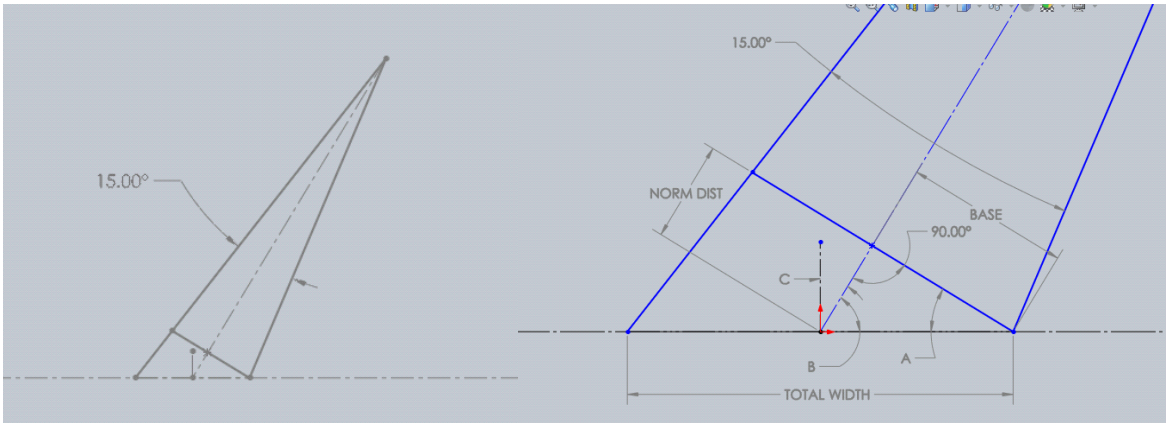

Figure 22 - Approximate geometry of the Lidar sensors with yaw error
Image courtesy of Ryan Sass, Santa Clara University

With respect to Figure 22, to find the normal angle, C, requires finding angle A, as well as the NORM DIST and TOTAL WIDTH. Then the law of cosines and Pythagorean's theorem can be applied to find the value of A, which is complimentary to C.

Knowing the angle the sensors are relative to each other, along with their sensor distance readings, it is possible to calculate the TOTAL WIDTH using the law of cosines. Calculating the NORM DIST requires taking the smaller sensor distance value subtracted from the average sensor distance value. This gives value to the hypotenuse and opposite sides of a right triangle formed between the shorter-distance sensor and the average distance. This is enough information to use Pythagorean's theorem to find the angle A. Taking the inverse sine of the NORM DIST divided by half the TOTAL WIDTH and converting to degrees gives the yaw angle the aircraft is away from normal to a wall.

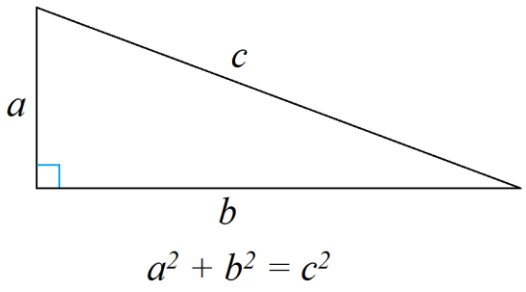

Figure 23 - Pythagorean Theorem
Image courtesy of Ryan Sass, Santa Clara University

This formula can be summarized and simplified in the following formula:

$$yaw\ angle = \sin^{-1}\frac{average\ sensor\ value - smaller\ sensor\ value}{\frac{1}{2}\sqrt{(LSV)^2 + (RSV)^2 - 2(LSV)(RSV)\cos 15°}}$$

Where:

$$LSV = Left\ Sensor\ Distance\ Value$$
$$RSV = Right\ Sensor\ Distance\ Value$$

This formula helped determine accurate angle measurements, even when at extreme angles. While at small angles, a linear approximation can be assumed, at larger angles greater than 30° of yaw, a large discrepancy can be seen between a linear approximation and the calculated

approximation. The calibration chart, shown below, depicts this difference and shows how altering the yaw angle can produce a linear approximation at lower values, but its accuracy diminishes at higher angles.

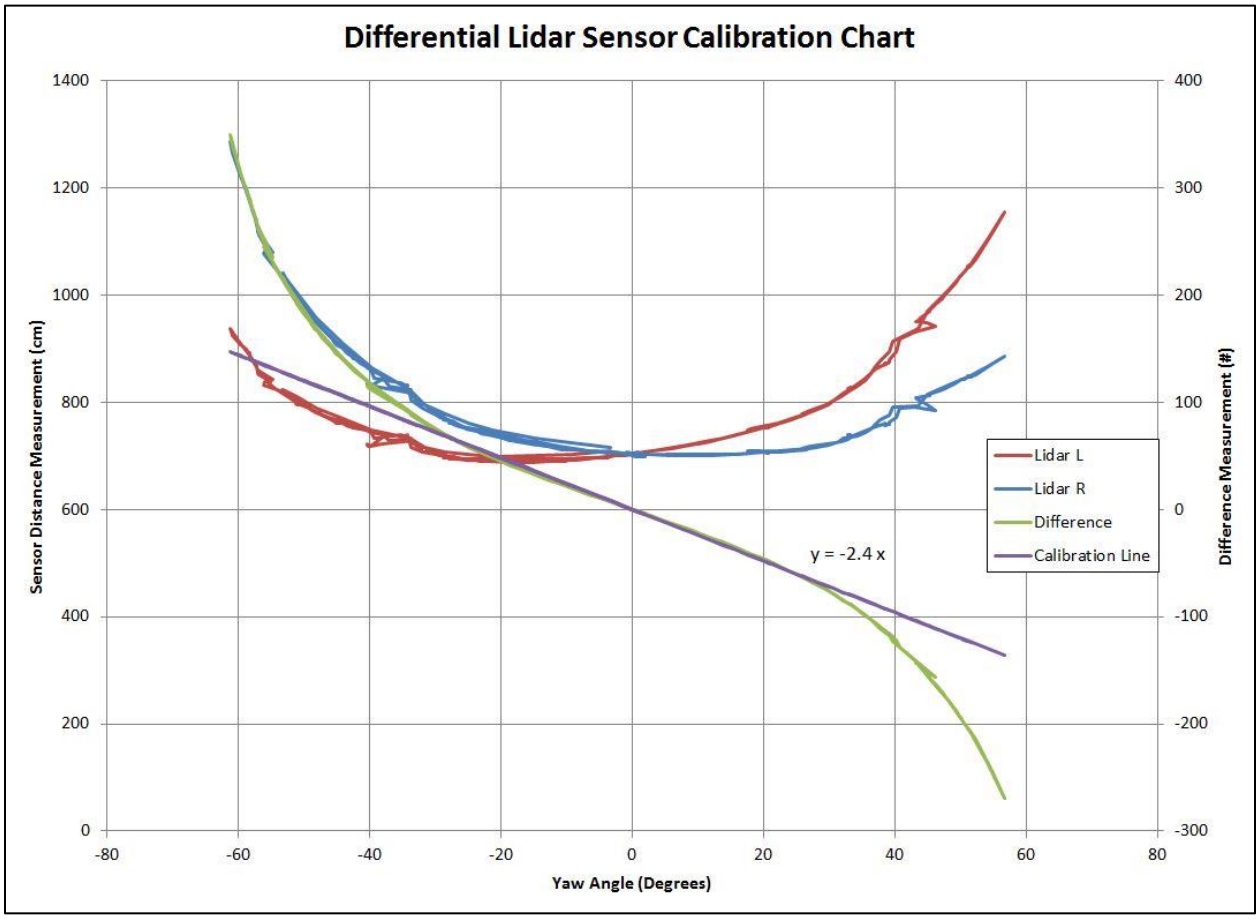

Figure 24 - Differential Lidar Sensor Calibration Chart
Image courtesy of Ryan Sass, Santa Clara University

Therefore, while a simple linear approximation between the sensors can be expressed by the following formula:

$$simple\ yaw\ angle = -2.4 * (RSV - LSV)$$

Where:

$$LSV = Left\ Sensor\ Distance\ Value$$
$$RSV = Right\ Sensor\ Distance\ Value$$

It is clear that yaw angles larger than 30° and especially larger than 40° exhibit large sources of error. Therefore the equation for yaw angle was chosen for greater accuracy.

3.2.2. Arduino Microcontroller

Figure 25 - Arduino Mega 2560 r3
Image courtesy of Ryan Sass, Santa Clara University

The Arduino control board chosen for this project is the Arduino Mega 2560 r3. This board, while large in comparison to the rest of the Arduino product line, is still small enough to fit on top of the drone, and has enough computing power and speed to maintain reasonable control of an aircraft – a fast moving vehicle.

The Arduino handles all the I2C communication and power with the Lidar units. It calculates distance and yaw angle readings from the Lidar sensors, records this information using the SD card reader board, and uses the Lidar data information in the control system.

3.3. Control System Design and Map

3.3.1. Overall Control Loop Design

The overall design of the dual control loop system is the heart of the automated control. For each iteration of the main control loop, yaw control is performed and it is determined if the pitch control loop should be performed or if the pitch control output signal should be set to be neutral. This determination is based on the state of the sensing system, the calculated yaw angle, and the relative distance from the wall. If the drone is too close or far away from the wall (i.e. out of sensor range or too close for safe operation), too askew from the wall to determine safe distance control, or there's a sensor reading error, it will prohibit the drone from controlling pitch and restrict the drone to only yaw control. This avoids unexpected pitch commands when in unusual flight circumstances.

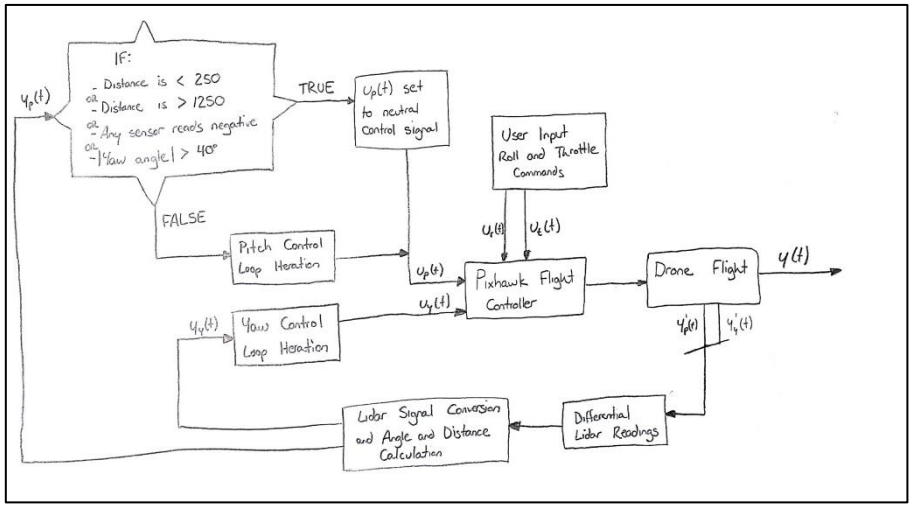

Figure 26 - Overall control loop flow diagram
Image courtesy of Ryan Sass, Santa Clara University

Assuming none of the error conditions are met, the overall control loop control executes one iteration of yaw angle control and then executes one iteration of pitch angle control. This process is continually repeated as long as the Arduino is powered. The output control signals are

generated from the PWM board. The control loops continually operate, even when under manual control. The PWM control signals, however, are not sent to the Pixhawk when in manual control.

These two loops are specifically designed to work in unison, as working with gated style control proved to be difficult to achieve smooth pitch control and was unreliable.

3.3.2. Yaw Control Loop

The yaw control loop is modeled after a PID control loop; however it only utilizes proportional control. The yaw angle is defined by the angle measured between the centerline of the aircraft and a normal line to the wall. Therefore, when the aircraft is positioned squarely nose first towards the wall, it is considered to have 0° of yaw angle. Any yaw angle that deviates from this has either positive or negative degrees of yaw error.

This defines the set point of the yaw control loop. The control loop is hard coded to maintain 0° of yaw angle, with a deadband of ±3°. The control loop follows proportional control – the higher the absolute value of degrees turned away from the set point the aircraft is, the larger the control signal generated to return the aircraft to the set point. The control loop is also coded to limit the amount of control signal, thereby reducing excessive control input at large angles of error.

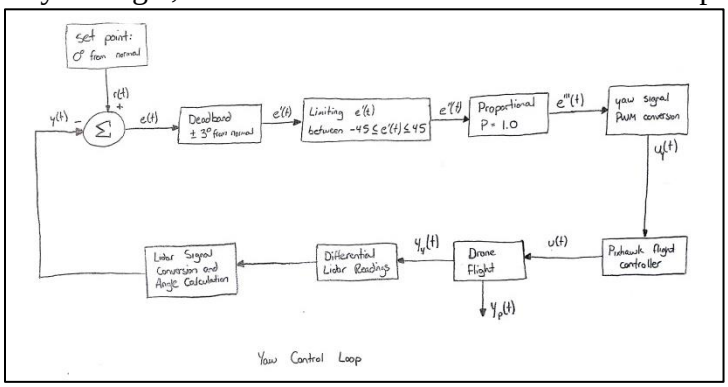

Figure 27 - Yaw control loop flow diagram
Image courtesy of Ryan Sass, Santa Clara University

While the yaw control loop exhibits some overshoot with proportional control, there is very little drift produced by the control input, and the overshoot is rapidly corrected. During flight tests, yaw error settled inside the deadband within 3 seconds. A level of overshoot was purposely tuned fort the controller, as a more critically damped system was difficult to observe during video recording of test flights of the aircraft. Over-embellishing the control response helps demonstrate the control authority of the yaw control loop.

3.3.3. Pitch Control Loop

The pitch control loop is modeled after a PID control loop; however it omits the Integral part of the control. The distance from the aircraft to wall is controlled by the pitch command, assuming the aircraft is oriented normal to the wall. The aircraft must have an absolute yaw angle value of less than 40° in order for the pitch control loop to be active. Actuating outside these yaw parameters runs the risk of putting the drone in an unsafe position to fly, in the event that there is something between the wall and the normal distance to the aircraft.

The aircraft is hard coded to maintain a distance of ~22 ft. or 670 cm from the wall. The proportional control behaves just the same as the yaw proportional control does – pushing harder towards the set point the larger the distance the aircraft is away from the set point. There is a very small deadband for the distance, and there is also an upper limit of how much control signal can be provided. However, due to the fact that there is substantial allowance by the Pixhawk for drifting once a neutral signal is given to the Pixhawk, derivative control is also necessary for the drone to avoid drastically overshooting the set point or behave in an unstable control loop setting.

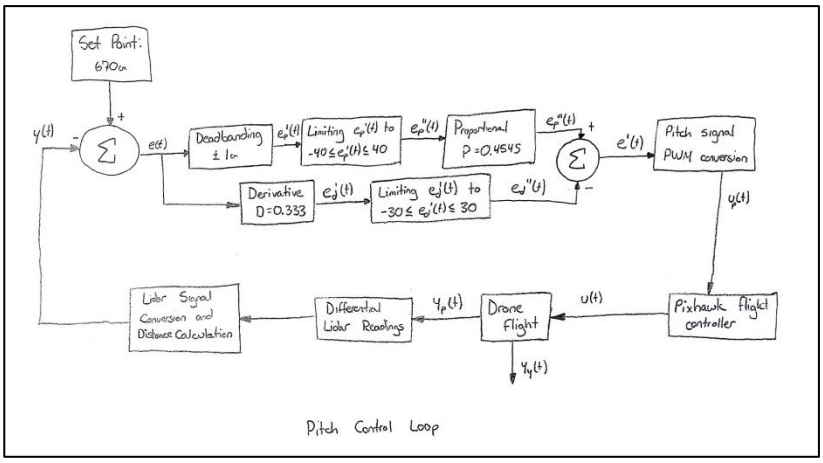

Figure 28 - Pitch control loop flow diagram
Image courtesy of Ryan Sass, Santa Clara University

The derivative control portion of the control loop is necessary to correct for the drift caused by large control inputs that can occur from the Arduino. This derivative control, which keeps memory of the previous iteration's amount of distance error, opposes fast movement towards the set point, and contributes to its settling time. When the aircraft overshoots the set point, the derivative control promotes more movement away from the set point; however the proportional control is tuning to overpower the derivative control input in order to avoid it being an issue for control.

3.3.4. Mechanical Actuation

Mechanical actuation of the drone is performed by the Pixhawk which receives control signals from the Arduino and PWM board or the user. To achieve motion, the Pixhawk will control altering groups of motors in order to achieve the motion desired. For the three primary control axis – roll, pitch, and yaw - actuation requires motors change their speed in two groups of four motors, and the corresponding adjustments adjust so the net sum of the thrust produced remains constant. It is important to note that as the moment produced by thrust changes the orientation of the aircraft, this will change direction of thrust with respect to gravity, and gives the appearance that net thrust changed. However during pure single axis control changes this is not actually the case.

The fourth axis of control, thrust, is the exception to motor actuation. When the throttle axis is actuated all 8 motors work in unison. Increasing speed of all the motors increases thrust and lift, but no changes in moment of any axis occur. Decreasing speed of all the motors decreases thrust and lift, but again, no changes in moment of any axis.

For the actuation of any control axes, motor adjustment is executed and controlled by the Pixhawk, which receives user commands and determines what motor actuation to perform to achieve proper flight axis movement.

To roll, the Pixhawk adjusts the group of four motors on the left and right side of the aircraft. This reduction in thrust on one side and increase on the other will create a moment and roll the aircraft. To pitch, the Pixhawk adjusts the group of four motors in the front of the aircraft and the back of the aircraft. This effect is the same as the roll actuation, just performed along the pitch axis instead of the roll axis. To yaw, the motors actuated are based on their spin direction. The counter clockwise and clockwise spinning motors adjust speed in opposition. The end effect is no net change in thrust, but a change in net torque produced by the motors. This creates the yaw movement. To achieve complete flight actuation, the adjustments the Pixhawk makes to the motors is a superposition summation of all four axes of flight applied to each motor.

Unlike a conventional helicopter, thrust is generated by the speed at which the propellers spin, not the angle at which the propeller blades meet the air. This means that in the event of one or multiple motor failures, the Pixhawk will attempt to regain flight stability; however this compensation may be extremely difficult to control. Also, unlike a conventional helicopter, in the event of battery failure, autorotation is not possible and the drone will glide like a brick.

3.3.5. Arduino and PWM Servo Board

The Arduino and the PWM servo board are the primary components that handle the control loops and output. The Arduino board handles execution of the control loops, and is the same component shared with the sensing system described in Section 2.3.

The PWM board, described later in Section 3.4.4, handles generating the output signals as determined by the Arduino, in order to achieve signal the Pixhawk to achieve mechanical actuation. In communicating with the PWM servo board via I2C, the Arduino acted as the master controller, while the other peripherals were assigned as slaves.

3.3.6. SD Card Reader Board

The Adafruit Micro SD card breakout board (254), also referred to as the SD card reader board, is a small board designed to handle the data recording of the control loops for performance analysis. The SD card reader board is attached to the Arduino through digital pins 50-53, and transfers data via SPI Protocol. This protocol is a much faster data transfer method than serial output, and therefore reduces the impact to control loop

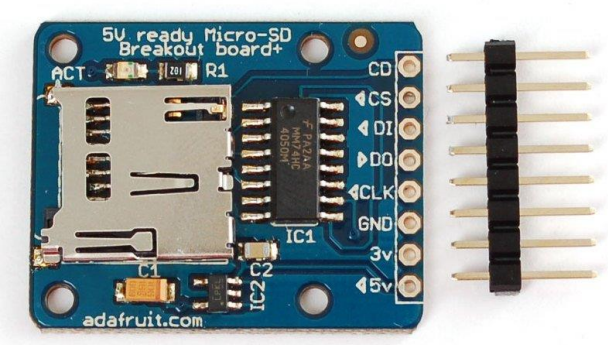

Figure 29 - Adafruit Micro SD card
Image courtesy of Adafruit

execution speed by the Arduino control loops perform to report the data.

Data recording and transfer is a challenge for this project. Because the drone is airborne, umbilical style data transfer isn't practical and transfer via wireless communication wasn't explored due to time constraints. Also, handing serial data logging can reduce the Arduino's speed to execute control loops as well, and could affect how well the control loops perform. Luckily, the change in performance using SPI protocol and no data recording is not noticeable from a visual perspective. This breakout board allows data recording of the Lidar sensor input and the PWM signal output values in a real-time setting with minimal changes to the operation of the software or flight performance.

3.4. System Integration

System integration required sending the output signals generated by the control loop system to the Pixhawk in the same size and frequency as normal RC signals. It also required the ability to toggle between automated and manual control for flight safety. This was achieved by a RC activated switch and relay. Other aspects of integration cover the mounts installed on the aircraft and proper Pixhawk configuration.

3.4.1. Pitch and Yaw Signal Handling

In order to signal the Pixhawk to perform motion, whether in automated or manual control mode, all control signals go through a PPM sum receiver. This component accepts eight PWM input signals – including pitch and yaw, combines all eight into a single signal, and sends them directly to the Pixhawk where actuation occurs.

In order to integrate two separate modes of flight, manual and automated, handling to two sets of control signals for pitch and yaw was required. To handle switching between manual control and automated control, a RC activated switch and DPDT relay were utilized to handle directing the sets of control signals into the Pixhawk inputs.

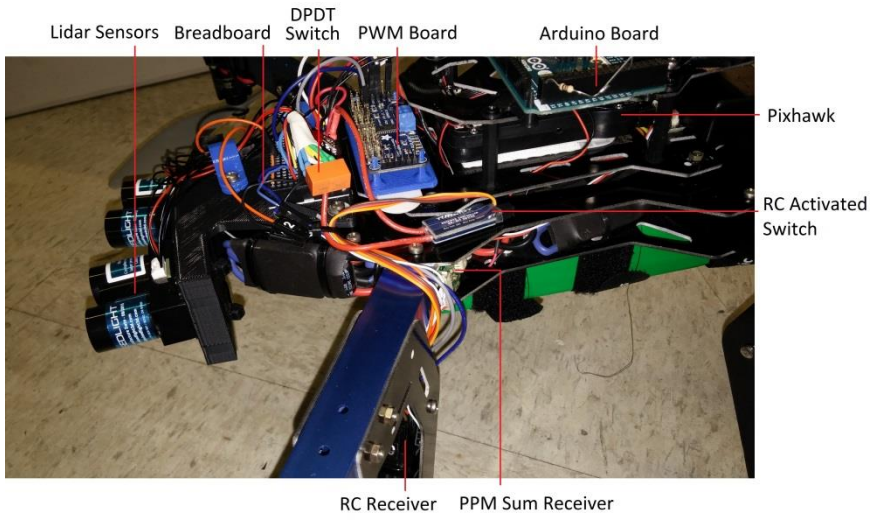

Figure 30 - Physical Layout of the pitch and yaw signal handling components
Image courtesy of Ryan Sass, Santa Clara University

The DPDT relay's switch contacts connected the PPM Sum receiver's yaw and pitch inputs to either set of control signals. The manual yaw and pitch signals (FROM the RC receiver) are connected

to the NC switch contacts of the DPDT relay, and the automated yaw and pitch signals (FROM the PWM board) are connected to the NO relay contacts. Manual control was purposefully chosen to transmit through the NC contacts. In the event of Arduino failure or relay failure, the relay would most likely switch off and the NC contacts would return to their normal position. This would maintain manual control input during component failure.

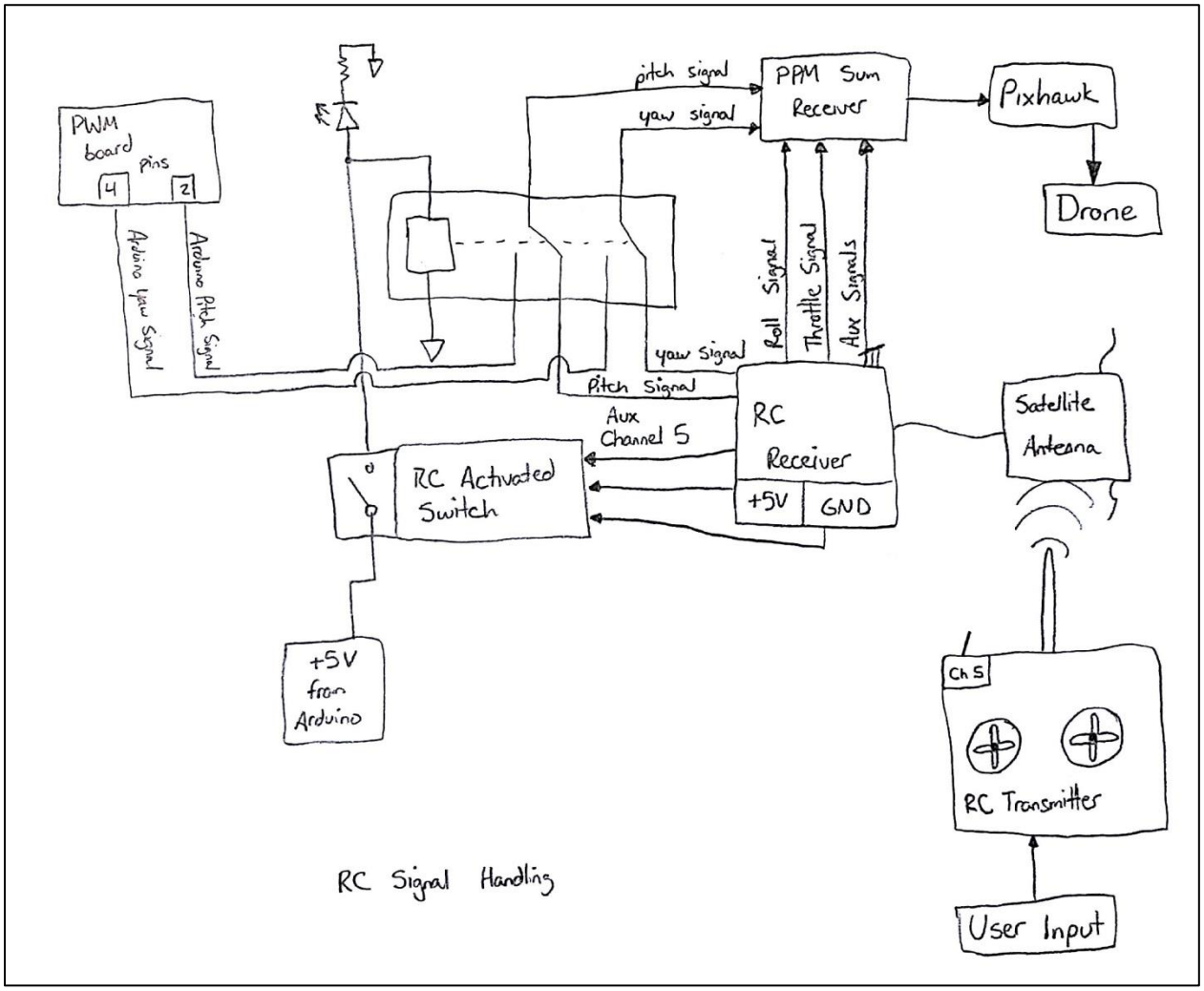

Figure 31 - Signal flow chart for manual and automated control
Image courtesy of Ryan Sass, Santa Clara University

3.4.2. RC Activated Switch

The Turnigy Receiver Controlled Switch, referred to as the RC activated switch, is used to provide power to the DPDT relay. The DPDT relay handles switching between the automated and manual yaw and pitch control inputs. When the DPDT relay receives power at the coil, it connects the NO contacts to the poles. When no power is supplied to the relay, the DPDT switch connects the NC contacts to the poles. The RC activated switch switches power to the relay coil, and in effect controls the switching between manual and automated control.

This switch operates by reading auxiliary channel 1 from the RC receiver and will open or close its internal SPST switch if the RC duty cycle of the signal it receives is high or low.

This switch allows reliable switching between automated and manual control of the two axes of fight control. To make this switch operate with an RC transmitter, an auxiliary channel on the transmitter that is operated by a SPDT switch must be chosen to activate the RC activated switch

Figure 32 - Turnigy Switch
Image courtesy of Ryan Sass, Santa Clara University

3.4.3. DPDT Relay

The DPDT relay and RC activated switch are the key safety features to allow development of Arduino automated flight. Because automated flight of the aircraft would be incredibly risky to test for the entire duration of a test flight, the DPDT switch allowed the state of automation to be toggled.

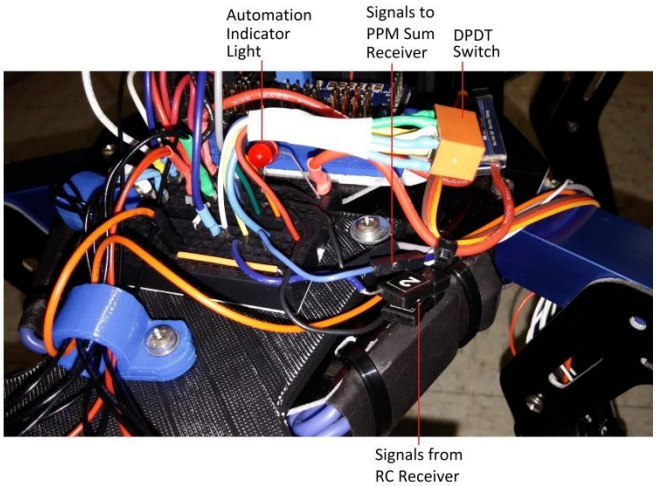

Figure 33 - Close-up of DPDT Relay
Image courtesy of Ryan Sass, Santa Clara University

The relay was a concern in the electronic design because of the physical nature of the switching contacts and susceptibility to vibrations, especially when mounted to a drone with 8 spinning motors. However, during flight testing and operation, this did not prove to be problematic. More extensive testing should be considered to find the most robust relays or discreet device available to handle the switching of the signals.

3.4.4. PWM Servo Board

The Adafruit 16-Channel 12-bit PWM/Servo Driver –I2C interface – (PCA9685) is designed to generate PWM signals specified from an I2C command[5]. Its primary design is to handle controlling power and signal processing for small RC servos to operate. However, in this project, the power handling is disregarded and the PWM output signals are strictly used in order to interface with the Pixhawk's PPM sum receiver to send proper control signals.

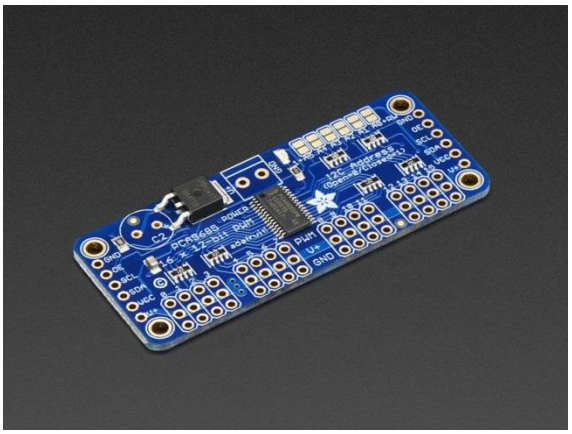

Figure 34 - Adafruit Servo Board
Image courtesy of Adafruit

A PWM servo board was chosen for two reasons. First, the board is programmable to adjust the PWM signal frequency to the proper value. Second, the board provides a decent resolution of duty cycle for the PWM signal, since the duty cycle of a flight control axis (from 0% to 100% range) ranges from 5% – 9% PWM duty cycle. These considerations were aspects the Arduino board could not perform without loss of resolution or performance.

The PWM servo board output signals required characterization and tuned in order to match its output signal to that of an RC signal. This was performed using a digital oscilloscope as well as changing the parameters in the software of the driving Arduino. Once these values were tuned and set, no confirmation checks were performed, as there is nothing built in to do so. The only feedback that the signals produced by the board are correct is when control authority by the Arduino is exhibited. This board is powered by 5V from the Arduino and communicates via I2C communication protocol.

3.4.5. Signal Triggering

While it was a concern that the signals from the RC receiver and the PWM board going to the PPM sum receiver might have to have a common trigger in order to successfully be combined, this did not prove to be necessary. It appears the PPM Sum Receiver can handle the switch successfully regardless of when each signal starts its duty cycle.

The only point to note is the PPM sum receiver's blue signal light, which during normal operation has a slow-steady blink. This will change to rapid blinking for a few seconds when the signals sent to the PPM sum receiver toggle between the automatic and manual set of signals. This is a clear indication that the PPM sum receiver is dealing with the change in signal start times, however signal triggering did not prove necessary and the PPM sum receiver handled it flawlessly.

3.4.6. Mounting

Mounting to the drone was a very important to put all the added hardware onto the drone. Components had to be to access for service or troubleshooting, be securely kept out of the way of the propeller blades, maintain a low enough profile to avoid snagging on things or coming loose while flying, and positioned in areas to avoid destruction in the event of a crash.

The mount for the Lidar sensors was created using a 3D printer, and created in ABS plastic. It included slotted holes for mounting to the aircraft to facilitate easy installation as well as allowing more tolerance for the 3D printer. The Lidar sensors are attached to the mount via four through holes and zip ties.

The other mounts created for this project included the PWM board mount and the Arduino mount. Both of these mounts were created with a 3D printer in are of ABS plastic. The PWM board is affixed to the aircraft via adhesive backed plastic Velcro. The Arduino board is affixed to the aircraft via mounting screws. The screws were already in use to hold the top carbon fiber

plate to the aircraft. To install the peripherals to the mount, a press-fit style installation was used. This style was chosen because it provided secure installation during flight and easy removal during service, maintenance, or removal. While inverted flight was not a consideration in any of the flight modes of this aircraft or project, the mounts proved robust enough even during periods of aircraft inversion, such as accelerometer or compass calibration.

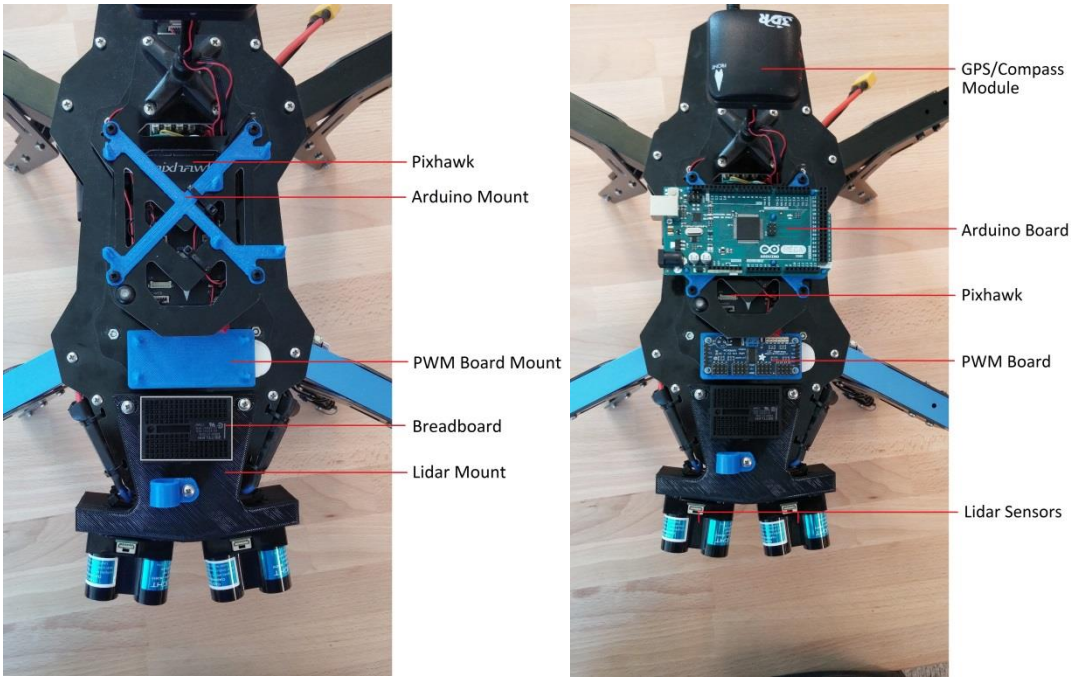

Figure 35 - Control and Sensing system mounts with their corresponding hardware in place
Image courtesy of Ryan Sass, Santa Clara University

The Arduino board is mounted just above the Pixhawk, and directly behind the PWM board. When installed, the Pixhawk does not usually require regular access. Mounting the Arduino on top of it reduces the ease of access to the Pixhawk. This proved to be a suitable solution though, because the Arduino required more frequent access in order to troubleshoot as well as plug wires into. This proved an adequate location that maintained low profile while allowing very easy access.

The breadboard, used for handling power and signal distribution, is mounted directly behind the differential sensors on the top of the aircraft. It is mounted in a position where it is fairly close to all components including the PPM sum receiver to allow for short wiring requirements between individual components.

All of these added components and mounts did add weight to the aircraft; however this did not affect the overall flight characteristics of the drone. The Pixhawk is designed to compensate for any change in weight balance, and while the added weight shift is believed to be minimal for this size of drone, the Pixhawk managed to compensate without issue. The drone also did not exhibit

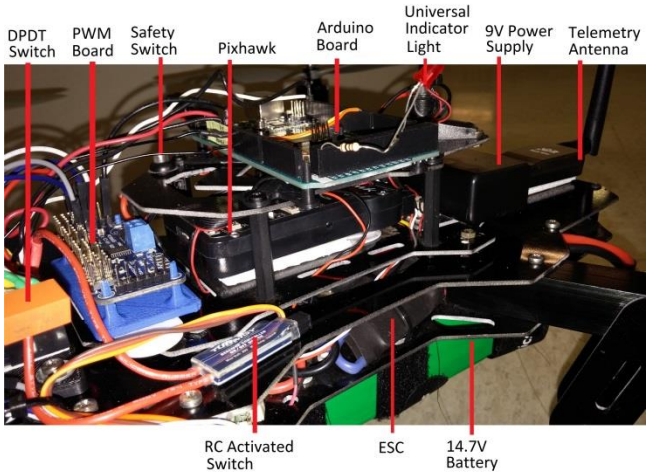

Figure 36 - Components mounted to the aircraft
Image courtesy of Ryan Sass, Santa Clara University

any signs of reduction in movement agility. The motors still provided a substantial amount of lift, translating to strong control authority of the drone.

3.4.7. Pixhawk Flight Mode

For this system to operate, the Pixhawk must be set to STABALIZE flight mode during the duration of flight. This includes when operating in either manual or automated control modes. Only in emergency situations was the LAND mode ever used.

3.5. Electrical System Wiring and Layout

3.5.1. Power and Grounding

There are two main sources of power for this drone. The primary source of power comes from the large 14.8V Lithium Polymer battery. This battery supplies power to the 8 motors and the power module, which generates 5V for the Pixhawk, the RC receiver, GPS and compass, telemetry antenna, PPM sum receiver, and RC receiver. The other power supply is the 9V battery that powers the Arduino. The Arduino's regulated 5V output powered the two Lidar sensors, the PWM board, and the SD card board.

While the 14/5V Pixhawk and aircraft electrical systems are almost completely isolated from the Arduino's 9/5V supply, there is still signal sharing between the PWM board and the PPM sum receiver. This means that the two systems must be grounded together in order to have proper signal voltage levels.

To achieve proper grounding, the negative contacts on the RC receiver, which receive power and ground directly from the PPM sum receiver, are connected to the ground on the bread board, which is connected to the PWM board, as well as the Arduino, Lidars and SD card reader board. While all the components are essentially grounded to one another, there might be a difference between ground of components on the far end of the ground loop circuit. Most importantly however, the ground between the PWM board and the RC receiver, the two components that send similar data signals from different sources of 5V power are grounded almost directly together. This was specifically chosen to provide the shortest grounding path between the two components for clarity with each signal being generated. It proved to be successful for signaling purposes as the PPM sum receiver recognizes both signals without any notable issues.

While power directly from the drone's battery, ranging from 13.5V – 16.5V, was considered as a possibility for powering the Arduino (it can accept up to 20V), it did not prove to be regulated

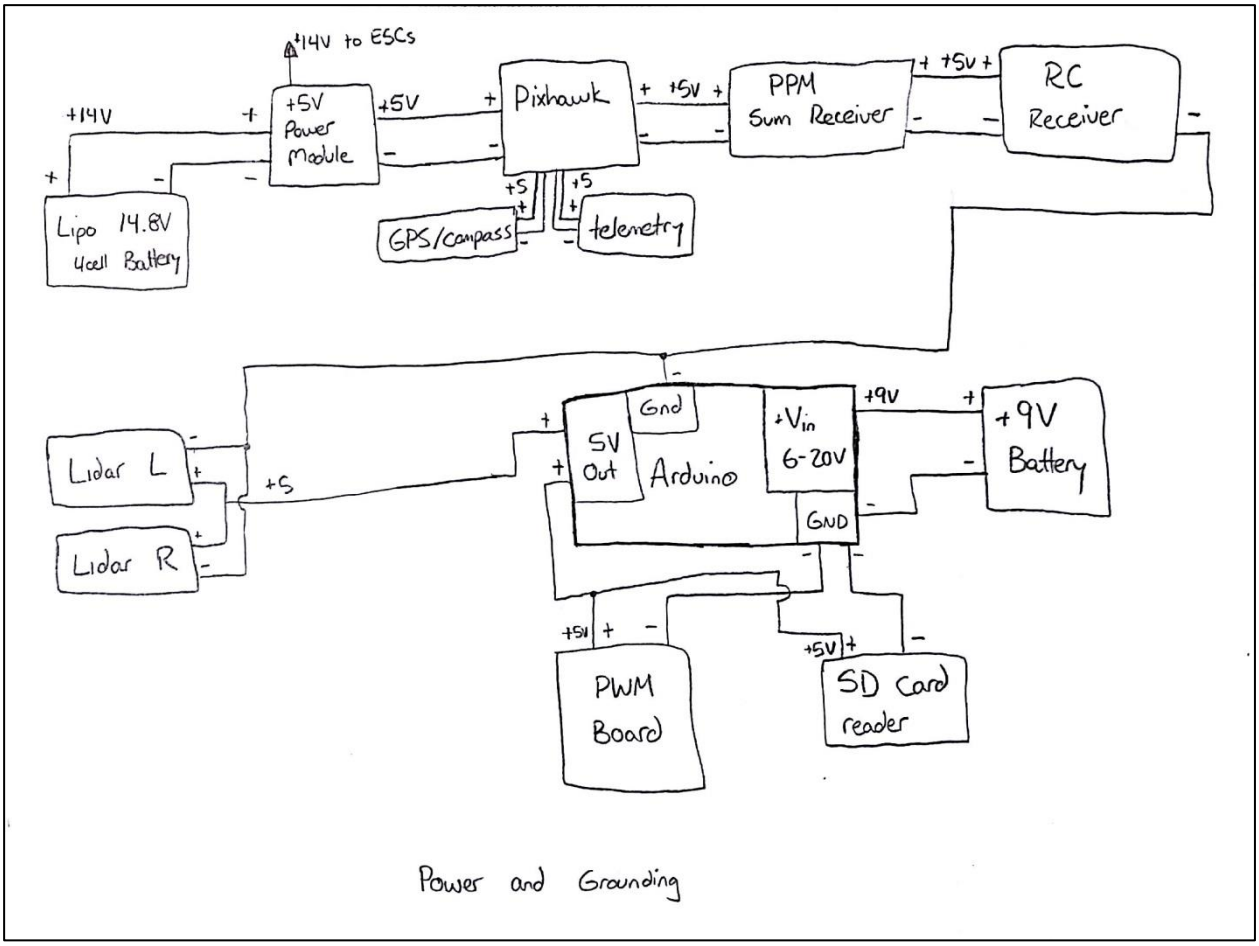

Figure 38 - Power and ground wiring schematic
Image courtesy of Ryan Sass, Santa Clara University

properly or provide stable enough results with the Arduino's 5V regulated output. Therefore, while future modifications should include an improved power supply for the Arduino control loop assembly, 9V power is used for this project and worked satisfactorily.

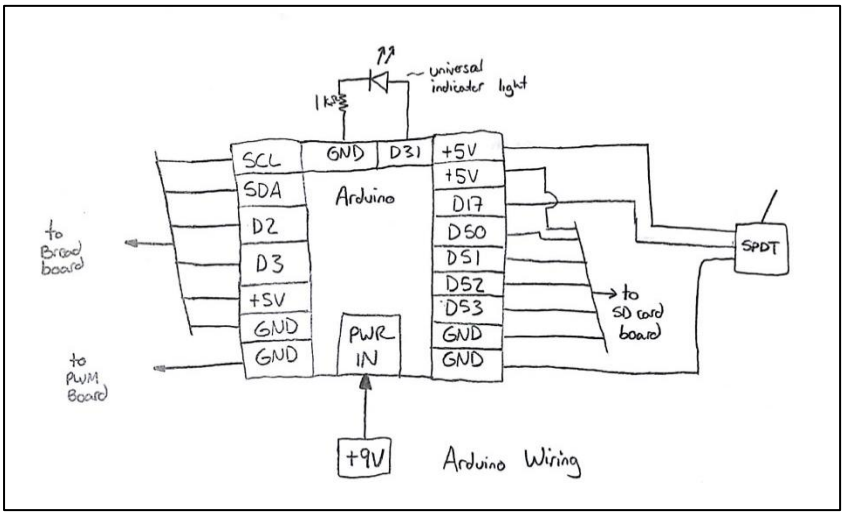

Figure 37 - Arduino board wiring
Image courtesy of Ryan Sass, Santa Clara University

3.5.2. Arduino Wiring

The Arduino was the main component that powered most of the other controller components, as well as sending communication signals to each of

these components. With the exception of the SD card, most of the wiring for the Arduino is connected to a breadboard so that power and communication signals are easily distributed to other components. The Arduino communicates with the PWM Board and the Lidar sensors via I2C, which requires Power, Ground, SCL, and SDA connections, all of which can be connected together, since individual communication is done via addressing for I2C. The SPDT switch is the signaling switch to indicate when to record and stop recording data to the SD card board.

Also, a universal indicator light, powered through pin 31, is attached to the board directly. This will be described in more detail in Section 3.5.6, but was used for in-flight trouble shooting, and ultimately indicates the status of pitch control in the final version of the software.

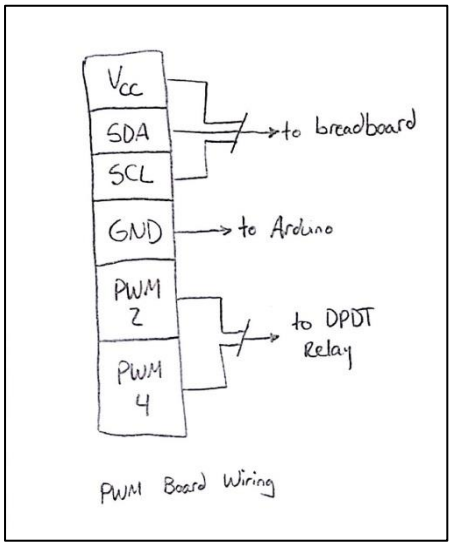

Figure 39 - PWM Board Wiring
Image courtesy of Ryan Sass, Santa Clara University

3.5.3. PWM Board Wiring

The PWM board requires simple connections to the Arduino, as it only needs power, ground, and SCL/SDA connections for I2C. With the exception of ground (for the sake of shorter wiring), all the wires from the PWM board run directly to the breadboard, since all connections to the PWM board are shared with other components.

Wiring of the PWM output pins are also connected to the breadboard, where they are routed through the DPDT relay.

3.5.4. Lidar Wiring

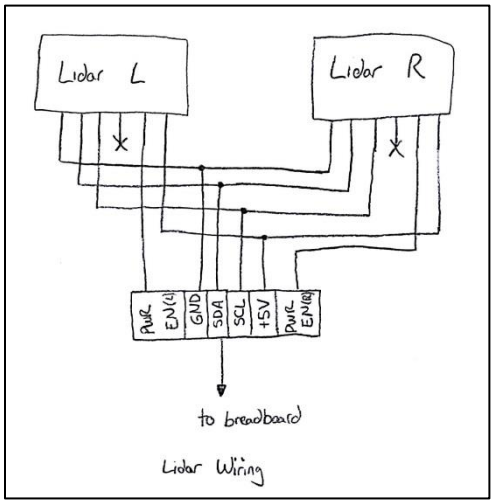

Figure 40 - Lidar Wiring
Image courtesy of Ryan Sass, Santa Clara University

The Lidar is wired in a similar fashion to the PWM board. All the connections, including the PWR EN connections, go directly to the bread board. The 5V, GND, SCL, and SDA connections are related to I2C communication.

Since each sensor had 5 connections (PWR, GND, SCL, SDA, PWR EN), all but the PWR EN connections are shorted together and treated as 1 sensor input. This proved better space efficiency for wiring on the breadboard. The PWR EN wiring had to be treated independently because each signal is used for changing the addresses of the Lidar sensors for I2C communication, and are specifically used when multiple sensors are connected using I2C communication.

3.5.5. Breadboard Wiring

The breadboard is the junction between many components. Connections of power or signal included the Arduino, Lidar Sensors, PWM Board, RC Activated switch, DPDT relay, PPM Sum receiver, and RC receiver. The connections of +5V, GND, SCL and SDA all arrive at this board to be connected to other components.

The grounding of the RC receiver to the Arduino/ PWM Board ground is also wired here.

This is also the location of the Arduino automation control light, which provides visual feedback to the user that the Arduino has taken control of the yaw and pitch axes of flight.

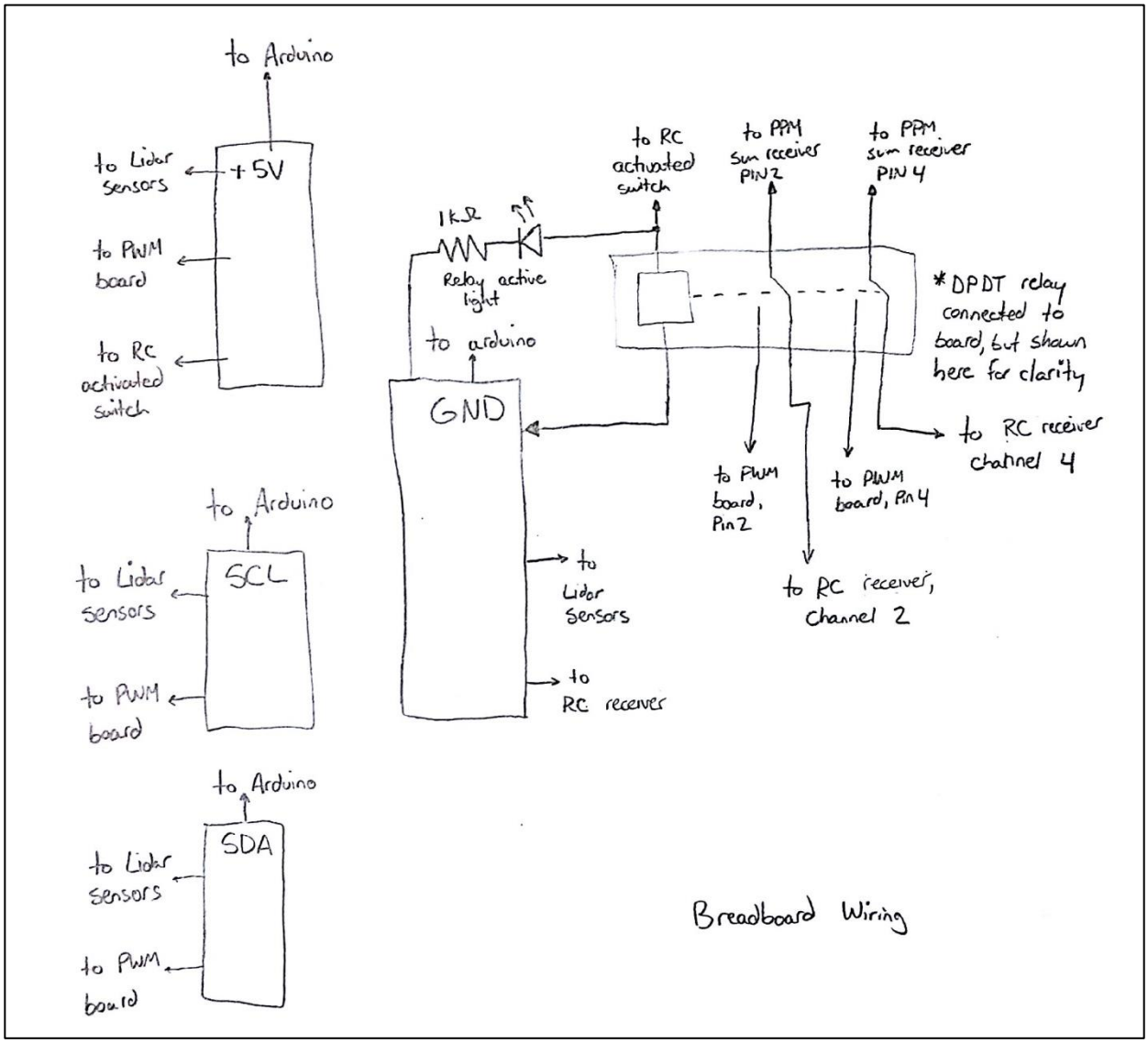

Figure 41 - Breadboard Wiring
Image courtesy of Ryan Sass, Santa Clara University

3.5.6. LED Indicator Lights

There are two LED indicator lights on the aircraft. Each is designed in a similar manner but each are meant to indicate two separate pieces of information. Both are red LED's, and are connected to 5V in series with a 1 kΩ resistor. This gave the user visual binary feedback to confirm states of the aircraft during flight.

One indicator light is the automation indication light. This light illuminates when the DPDT relay is active. The light is an indicator that the DPDT relay's NO switch contacts are closed, and the Arduino is in control of the pitch and yaw axes. This allows visual feedback of when the aircraft is operating in automation and when it is not. The power supplied to this light comes from the Arduino and travels through the RC activated switch SPST contact. This design was chosen in the event of Arduino failure or RC switch failure; the light will always provide accurate feedback of the state of the relay. If the light is active, the relay is too. This light is mounted to the breadboard directly. The wiring for this light can be seen in Figure 41.

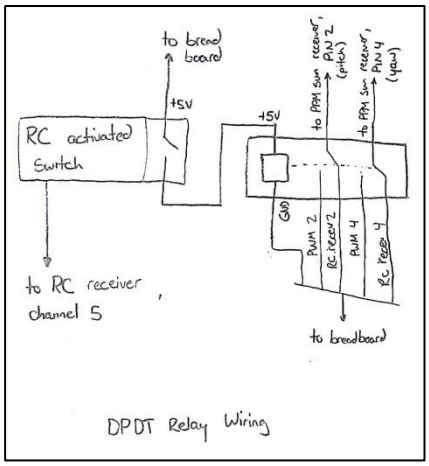

Figure 42 - DPDT relay and RC activated switch wiring
Image courtesy of Ryan Sass, Santa Clara University

The second indicator light is the universal indicator light, mounted directly to the Arduino. This light, connected from digital pin 31 to ground, will illuminate when certain flight conditions are met. This provided feedback of the state of the control system under certain flight conditions in the dynamic test environment. This light is considered universal as it is user programmed to indicate any single parameter the user can choose. Over the course of its use, it reported state of things including excessive yaw drift, sensor read error, set point or deadband locations, and pitch control active/inactive. The light can only be used to indicate one status, and its state has to be known ahead of time when the Arduino was programmed to be understood or useful. The wiring for this lights can be seen in and Figure 37.

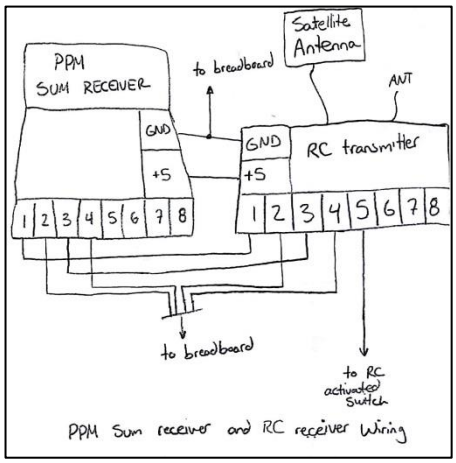

Figure 43 - PPM sum receiver and RC receiver wiring
Image courtesy of Ryan Sass, Santa Clara University

3.5.7. DPDT Relay and RC Activated Switch

It can be seen how the RC activated switch and the DPDT relay are connected. All the wiring for these devices is connected through the bread board, with the exception of the AUX 1 channel from the RC receiver. This connection provides the RC activated switch the RC PWM signal and is connected directly to the RC receiver.

3.5.8. PPM Sum Receiver and RC Receiver Wiring

The PPM sum receiver is the device that takes 8 PWM signals, commonly generated from the RC receiver, shifts and combines them into one signal that the Pixhawk then reads. The PPM sum receiver receives power and ground from the Pixhawk directly, and supplies the RC receiver with power and ground. The PWM sum receiver has 8 input channels to accept incoming signals. The Pixhawk expects that the assignment of signals for the first four channels is as follows:

1. Roll
2. Pitch
3. Throttle
4. Yaw

The rest of the 4 auxiliary channels can be assigned to any number of switches or dial on the RC transmitter, depending on the configuration of the Pixhawk and the aircraft. One important channel to note is Channel 5. This channel of the PPM sum receiver is the indicator channel to the Pixhawk about what flight mode the Pixhawk should be operating in. Based on the configuration of the Pixhawk, up to 7 different user assigned Pixhawk modes can be selected depending on the RC duty cycle of the signal going into Channel 5.

For this project, this channel 5 signal was assigned to a 3 position switch that the RC receiver output on its AUX 3 channel. Originally the project intended to utilize three different flight modes: Land, Stabilize, and Alt Hold. Ultimately the Alt Hold mode was removed from the selection and Stabilized mode was assigned in its place.

Most of the signals from the RC receiver are wired directly to the PPM sum receiver with the exception of ELE, RUD, and AUX 1. ELE and RUD (short for Elevator – the control signal for pitch – and Rudder – control signal for yaw) are wired to the breadboard along with Channels 2 and 4 from the PPM sum receiver. AUX 1 channel on the RC receiver is wired directly to the RC activated Switch.

3.5.9. SD Card Board Wiring

The SD Card Board was wired directly to the Arduino. It included digital pins 50 – 53 for communication, as well as +5V and GND. Also, a SPDT switch was attached to the Arduino connecting either +5V or GND to digital pin 17 in order to signal the SD card to write data or to stop writing data and complete the creation of the file. This switch is important to toggle; after flight data was recorded on the SD card, the file had to be finalized in order to be a functional, usable file.

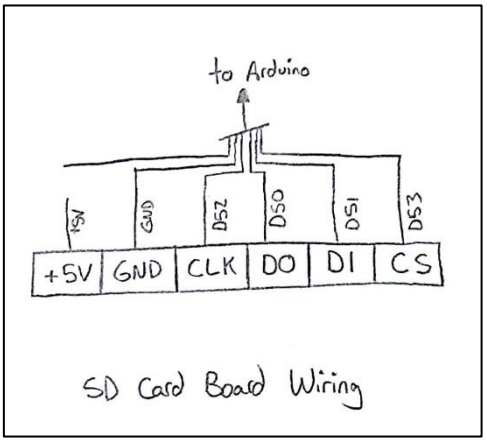

Figure 44 - SD Card Board Wiring
Image courtesy of Ryan Sass, Santa Clara University

3.6. Software Design and Flow

3.6.1. Introduction

The software developed for this project was written for the Arduino, however small parts of software configuration are required for the Pixhawk as well.

The bulk of the code was developed to be compartmentalized for easier understanding, as well as easy alteration as the code developed during flight testing.

While a the majority of the code was created and developed as an individual effort, the software to handle the data writing to the SD card reader board integrated into the final software was a joint development effort, with software creation to handle SD card interface handled by Ryan Cooper from Santa Clara University.

Evolution of the software was numbered in versions. The primary number of the software indicated the following development notes:

1. v0.x – Experimental and Early Stage Software. Intended for development of each different part of the software, and not intended for complete automated control.
2. v1.x – First control iteration of combined automated pitch and yaw control. Pitch and Yaw control worked in a gating manner – i.e. the yaw control loop would align the drone to the wall then stop executing and the pitch control loop would then distance the craft from the wall. Any deviation from the yaw error deadband during pitch control would terminate the pitch control loop and execute the yaw control loop again until back into the yaw error deadband.
3. v2.x – Second control iteration of automated pitch and yaw flight. Pitch and yaw control worked simultaneously.
4. +v2.5 –Control loop flight software including SD card writing software added
5. v2.6 – Final version of the code used for this project.

In the following sections, the individual aspects of the software are described more in detail.

3.6.2. Arduino Software

The Arduino software was written to handle the dual control loops execution, the sensor input handling, the output signaling, data recording, and processing the control algorithms.

The Arduino software is very simple in its implementation. Upon powering the Arduino, a one-time ‘setup’ command list is executed. Then, an eternal looping ‘loop’ command list is executed again and again. The loop command list will continue until either a reset is initiated or power off occurs.

The software for the control loop was simple to process. It includes pinging, reading, and processing the input from the Lidar sensors. Calculations including averaging the input signals

and calculating the yaw angle offset occur. It will also calculate if the pitch control loop should be executed for the loop iteration or set the pitch control signal to neutral. It then initiates an iteration of the yaw control loop to determine the yaw output value. Then, depending on if the pitch control loop can execute, it performs an iteration of the pitch control loop. If the pitch control loop cannot execute, the pitch control output value is set to neutral. Finally, the software pings, and sets the output PWM values for the yaw and pitch control signals that were calculated by the control loops. This loop is executed in a matter of milliseconds and is then repeated again and again.

3.6.3. Main Code Flow

The overall flow of the software was developed to be incredibly basic. In the Arduino code, there is a "setup" routine that is performed once at the start of every power up or reset cycle, and a "loop" routine that is executed indefinitely until a reset or power off occurs.

The setup routine of the code involves loading proper libraries, initializing variables and arrays, configuring certain Arduino I/O pins and configuring the addresses for the Lidar and PWM board, establishing PWM output frequency, and initializing the SD card for file writing.

The loop routine of the code is where the control loops are executed, the Lidar are used to take measurements, and distance and angle values are calculated. Throughout the looped code there are small lines of execution for the SD card software to write data during execution. The basic flow is as follows:

1. SD Card Data write and switch check
2. Lidar Sensor Read and Signal Processing
3. Yaw Control Loop Iteration
4. Pitch Control Loop Iteration OR Set to Pitch Neutral (depending on the state of the yaw, as determined in the Lidar Signal processing)
5. Set PWM board output Values

Once an iteration of this loop is executed, the Arduino will start again from the top of the list and run through these functions again and again, until a power off or reset. The control signals are continually sent out, regardless of if the user has switched to automated control or not. In this way, there is no need to sync the Arduino with the RC activated switch or delay for initialization when automated control is desired. The signals are just routed to the DPDT switch and channeled to the PPM sum receiver when the user switches to automated control.

3.6.4. Lidar Sensor Read and Signal Processing

The Lidar sensor read function will take five sensor reading values from each sensor, with a 2 millisecond delay between each reading. It will average each respective sensor's value based on the five readings. The number value each Lidar sensor returns after a measurement is the amount of centimeters, in straight line distance, to the object the light reflected off of.

After obtaining average sensor values, the two signals go through processing in order to determine distance, angle, and yaw state. First, the distance is determined, simply by taking the average the two Lidar sensor signals. While this is error in this method of distance calculation, as the drone is actually slightly closer than the given average reading values (due to the angular shift of the sensor package), the difference between the actual distance and the calculated distance the aircraft is from the wall is small enough to be considered negligible.

Next, the angle of the yaw is calculated. This calculation is described in greater detail in Section 3.2.1. With the angle and distance determined, error checking is performed on the two sensors, checking for conditions where pitch control should be disabled for aircraft safety and stability in unusual flight situations. These conditions that are checked are as follows:

1. Distance to the wall is less than 250 cm (Too close)
2. Distance to the wall is greater than 1250 cm (Too far away)
3. Any sensor returning a negative value (Usually not reading off the wall, and not getting proper measurements from the sensor)
4. Calculated yaw angle is greater than 40° to the left or right. (Too mis-aligned from a wall, pitch control would not be a safe mode of operation because of the lack of clear space to move laterally).

If any of these conditions are met, pitch control is disabled so that a user may take control of the aircraft if during these conditions the drone is in an unsafe orientation or flight condition (i.e. drifting rapidly towards or away from an obstacle).

3.6.5. Control Loops

The control loops perform one iteration during each execution of the Arduino's "loop" routine. The yaw control loop operates indefinitely, and there are no conditions under which the yaw control will stop operating. The pitch control loop, however, does have conditions, described in Section 3.6.4, during which it will and will not operate for flight safety considerations. This allows the pitch control to not fly the aircraft in an unsafe manner during bad sensor readings.

The yaw control loop compares the current yaw angle to a hard coded set point, in this case, 0° yaw angle. Knowing yaw direction with respect to the wall, based on which sensor reading is larger than the other, the control loop will executes a deadbanded, limited, proportional control iteration to correct this yaw error.

The pitch control loop compares the current distance of the aircraft from a given set point, in this case, a hard coded 670 cm distance from the wall. Knowing the aircraft's current distance as well as its approximate distance from the last iteration, it executes a deadbanded, limited proportional and limited derivative control loop iteration to correct this error. Unlike the proportional control portion of this loop that gives the drone and increase in signal to return to the set point as the drone moves further away, the derivative term actually opposes the proportional control input,

thereby controlling the speed by which the drone will approach the set point and reduces set point overshoot.

More details of these control loops are discussed in Section 3.2.

3.6.6. PWM Output

The PWM output function is a very simple routine. Having the current pitch and yaw control adjustments calculated by the control loops' execution, the PWM output function adds these adjustments to the neutral output value to give a proper control signal output value. For the pitch command, a hard coded added 'trim' value is included to this addition which helps compensate for the natural forward or backward drift of the aircraft. This value is determined during flight tests of the aircraft flying with neutral commands.

This function follows the following calculations to determine output values:

$$PWM\ Output_{yaw} = PWM\ neutral\ value + yaw\ control\ adjustment$$

$$PWM\ Output_{pitch} = PWM\ neutral\ value + pitch\ control\ adjustment + trim\ value$$

Once these values are set, the function then calls the PWM board to update the PWM output values, and the signals then can travel to the PPM sum receiver for a change in flight control.

3.6.7. SD Card Read/Write

The Arduino sends data to an SD card breakout board using the Serial Peripheral Interface (SPI) protocol. The breakout board writes this data onto an installed SD card. To keep the SD file operations simple, the file names generated on the SD card are limited to a few characters, in this case assigning each successive filename a number. Filenames would always be an increment in numbering from the previous created file. Once the file is created, the Arduino can continuously write data to the file on the SD card. The most important file operation, though, is closing the file to ensure generated data will be properly saved prior to the Arduino powering off. A SPDT switch was used to indicate when the Arduino should initiate commands to close the file in order to retain all the data.

3.6.8. Pixhawk configuration

The Pixhawk must also be configured to operate with the Arduino, otherwise the input controls will not function as expected. Using the Pixhawk software, called Mission Planner, to set up complete calibration for the drone – including accelerometers, compass, GPS, ESC's, radio, and battery monitor - a few other things need to be configured.

First, the stability control rates for pitch and roll can be tuned on the Pixhawk for a snappier or softer feel as the drone flies. These values were tuned to be more on the soft side, as the primary

mode of flight for this drone was in a small space indoors and fast, rapid movements while under automated control was undesirable.

Second, the flight modes need to be configured to one of the RC transmitter switches in order to communicate to the Pixhawk which mode to operate in. Also, the flight modes needed to be assigned so that the Pixhawk will operate in a non-GPS required mode (such as Stabilize). This allows the drone to take off and fly. If in a GPS required flight mode, the drone will refuse to fly until a GPS connection is established.

Finally, the battery failsafe should be configured so that the drone doesn't run through the entirety of the batteries prior to landing as well as protecting the integrity of the batteries. Due to the nature of a drone, no autorotation can occur, so if the motors all stop due to battery failure, the drone will fall directly to the ground.

3.7. Data Collection and Processing

The data logged to the SD card during test flights included the following five values:

1. Elapsed time in milliseconds
2. Readings from both the left and right Lidar sensors
3. Yaw and pitch control output values the Arduino calculated

These values were continually logged, updating approximately every 200ms. This update time reflects the total time to not only log the data, but process the control loops, read the Lidar data, and update the PWM board as well.

Once data was recorded and finalized after a test flight, it was transferred off of the SD card to a computer. From there, the delimited file could be reviewed using a program such as Microsoft Excel.

3.8. Pixhawk Flight Modes, Failsafes, and Parameter Settings

3.8.1. Pixhawk Firmware and Interface

To send parameters and update values for the Pixhawk, Mission Controller, a software interface specifically developed to interface with the Pixhawk, is used to set any values or parameters requiring adjusting (such as flight modes, failsafe modes, or stability control parameters). These values are updated by connecting the computer to the Pixhawk via telemetry antenna or USB.

The Pixhawk firmware was not directly used in terms of this project, but was used for Pixhawk and drone configuration. The Pixhawk firmware has multiple levels of operation, but the two important ones to worth noting are flight control and flight operation firmware.

The flight control firmware deals with controlling the motors of the aircraft in order to maintain stable control of the aircraft. Multiple parameters are specified to the Pixhawk, including the type

of aircraft, number of motors and motor orientation. With the integrated accelerometers in the Pixhawk, it calculates how the Pixhawk should be oriented to achieve stable level flight, and then calculates – based on its assumed flight frame and shape and motors controlling – what control signals to provide each motor to achieve the proper flight orientation. It then uses control loop feedback to correct the motors it controls in order to achieve actual level flight.

The flight operation software works on top of the flight control software. This software works to actually allow the drone to be controlled, either flying autonomously with preset missions loaded on the Pixhawk or by manual control. In manual control, the Pixhawk will adjust the motors based on the 4 axes of flight control to produce the roll, pitch, yaw, and thrust performance that the user commands.

In the Pixhawk firmware, there is a lot more sophistication, automation, and features that will not be discussed but are useful. However, another important note includes the failsafes the Pixhawk can monitor, including low battery or failed sensors. In the event of a failsafe, the Pixhawk will control the throttle to bring the aircraft to the ground in a safe, controlled descent. It does not, however, control any of the other axes of flight, and they remain under manual user control.

3.8.2. Flight Modes

Flight modes can provide a good failsafe in the event of unsafe flight operation. Originally, the intended operation of this aircraft was to operate in one of three modes, and the signal to switch between them was assigned to a switch on the RC transmitter that has three positions. These modes included:

1. Land –bring the aircraft to the ground in an expedited but controlled descent. Throttle input is automatically set, but roll, pitch, and yaw are still under user control.
2. Stabilize – the aircraft will maintain straight and level flight until manual input is received from the transmitter. All 4 axes of flight are under manual control. Upon returning roll, pitch, or yaw axis controls to a neutral position, the aircraft will return to level flight, although drift may occur.
3. Altitude hold – similar to Stabilize flight mode, however throttle is controlled to hold a given barometric and/or (if enabled) GPS calculated altitude. Roll pitch and yaw control is still enabled, and throttle is automatically adjusted when these flight inputs affect altitude.

While altitude mode was ultimately removed for this project due to the lack of ability to test the robustness and reliability of the flight mode, each mode listed above was a failsafe for the mode that succeeded it. So, if in altitude mode, the user can switch to manual mode and take control of throttle once again. In the event of an unsafe flight mode in manual mode, the user can switch to land mode, and the aircraft will attempt to rapidly but safely descend to the ground. This should reduce the chance of a crash landing if the user would have difficulty in landing the aircraft

manually, such as if in a difficult flight situation or when close to obstacles during an emergency landing.

3.8.3. Important Pixhawk/Mission Planner Parameter Settings

The most important parameter setting is the battery failsafe. It must be configured to the proper monitor and set to monitor voltage and amperage. This will protect the drone from destroying itself or the battery failing mid-flight. For a battery failsafe, the LAND mode is automatically activated, and the reserve battery power was set to the lowest value (1000 MAH), as flying within 3’ to 8’ of the ground required little power to accomplish a fast landing.

Another important set of parameters to check is ensuring the motors of the drone operate in their assigned position. This check is important to perform, with the propellers off, when initially installing a Pixhawk to the aircraft.

Lastly, the roll/pitch sensitivity was tuned to very low values (0.1159) in order to keep the drone’s roll and pitch rates low. Also, the throttle hover setting was set to a mid-level value (500) in order to reduce the sensitivity of the throttle was performed. In this case

4. Testing

4.1. Introduction

Testing for this project was conducted to prove the concept and feasibility of operation. Therefore, the parameters for final flight performance were defined to prove that a microcontroller could achieve autonomous control of a drone. The following performance objectives were set for development and testing:

- Demonstrate Aircraft Control of yaw and pitch axes using control loops.
- Demonstrate Yaw Control within ± 20° from normal.
- Demonstrate Pitch Control within ± 3 ft. of a given set point.

Throughout the course of the school year, testing varied as the project developed. During the fall quarter, testing efforts were directed to prove the overall design of the control loop assembly could successfully recognize a wall and exhibit motor reaction on the then propeller-less drone as the orientation of the wall to the aircraft changed.

Testing efforts during winter quarter were to prove the operation of the control loops on a 2 dimensional vehicle – in this case on a rover bot. This allowed for design verification of the general control loop theory, electrical, and software aspects of the project prior to equipping and testing a drone.

Finally, efforts for testing during spring quarter involved proving successful automated flight of the drone and demonstrating basic control loop response in the yaw and pitch axes.

4.2. Fall Quarter – Demonstration of Project Feasibility and Limited Operation

The first phase of testing for this project involved demonstrating project feasibility. This included proving that hardware choices selected were capable of performing as required. Also, signs of control reaction of the drone motors needed to be demonstrated. The specific list of objectives for project feasibility for the fall quarter included:

1. Demonstrate multiple Lidar Sensors can operate and provide accurate data when connected to an Arduino.
2. Demonstrate output of a PWM signal from the Arduino with proper RC frequency and duty cycle.
3. Demonstrate signal compatibility between the PPM sum receiver/Pixhawk and the Arduino.
4. Demonstrate yaw control loop reaction.
5. Demonstrate Arduino control of the drone.

To accomplish these tasks, extensive research on the Lidar, Arduino, and Pixhawk components were conducted. Characterization of the RC receiver's PWM signals was required in order to determine what the Arduino must produce to match the RC signal and reduce the chance of signal rejection from the PPM sum receiver. Finally, three different batteries of tests were conducted:

Figure 45 - The drone configured for the Arduino and Drone control testing phase, wiring not yet installed
Image courtesy of Ryan Sass, Santa Clara University

- Arduino and Lidar testing – conducted via Serial port feedback on a computer.
- Arduino PWM and PPM Sum testing – conducted via Mission Control software feedback to observe input control signals.
- Arduino and Drone control testing – conducted with the propeller-less drone, a movable wall, Mission Control software feedback, Serial Port feedback, and audible feedback of the motors.

The results of these tests were overall a success. The following notes for the five objectives of fall quarter include:

1. Demonstrate multiple Lidar Sensors can operate and provide accurate data when connected to an Arduino. : Success. Using I2C communication and specific channel address wires it is possible to operate two Lidar sensors and proves very reliable.
2. Demonstrate output of a PWM signal from the Arduino with proper RC frequency and duty cycle. : Limited success. The output of the Arduino board can output PWM signals tuned to RC frequency and duty cycle, however because of the limited resolution of the Arduino, finer resolution would be better for more precise control.
3. Demonstrate signal compatibility between the PPM sum receiver/Pixhawk and the Arduino. : Success. The Arduino can produce signals that are compatible and are recognized by the Pixhawk.
4. Demonstrate yaw control loop reaction. : Success. During physical testing of the Arduino, Lidar, and drone, the drone exhibited sounds of changing motors when the movable wall was rotated between normal to the aircraft and angled. This gives indication that the yaw control loop was reacting to the changes in orientation to the wall.
5. Demonstrate Arduino control of the drone. : Success. The Arduino can exhibit control over a drone.

The success of these tests show the feasibility of the project, as the hardware was proven at varying degrees of functionality. Because the next steps in development were proving control loop operation on a two dimensional vehicle, the first phase of testing helped set up the foundation for what the following quarter's tests would use to function.

4.3. Winter Quarter – Demonstration of 2D Operation

The second phase of testing for this project involved demonstrating simulated operation of the drone to prove successful operation of the hardware. This allowed development of the Arduino control loops in a more forgiving and controlled environment, in this case operating in just the two dimensions of control intended. It also facilitated higher factors of safety in the preliminary stages of the control loop development. This was beneficial as when flight testing occurred, the basics of the control loop were already proven to be functional. The list of objectives for this quarter to demonstrate project operation included:

1. Mount the Arduino control loop hardware, including the Pixhawk, on a BOE robot and demonstrate autonomous vehicle control.
2. Demonstrate incorporation of new PWM board into hardware operation.
3. Demonstrate execution of yaw and pitch control loops operating in unison.
4. Prove control loop repeatability and robustness.

To accomplish these tests, a Board of Education (BOE) robot was used. This vehicle is a three wheeled, two independent wheel drive rover with a flat top to allow for payload mounting. The robot can simulate yaw of the drone by operating the two motors in opposition to one another. When the two drive wheels spin in opposite directions, the vehicle spins just as the drone would yaw. When the two wheels spin in unison, it simulates the pitch movement by moving forward or backwards.

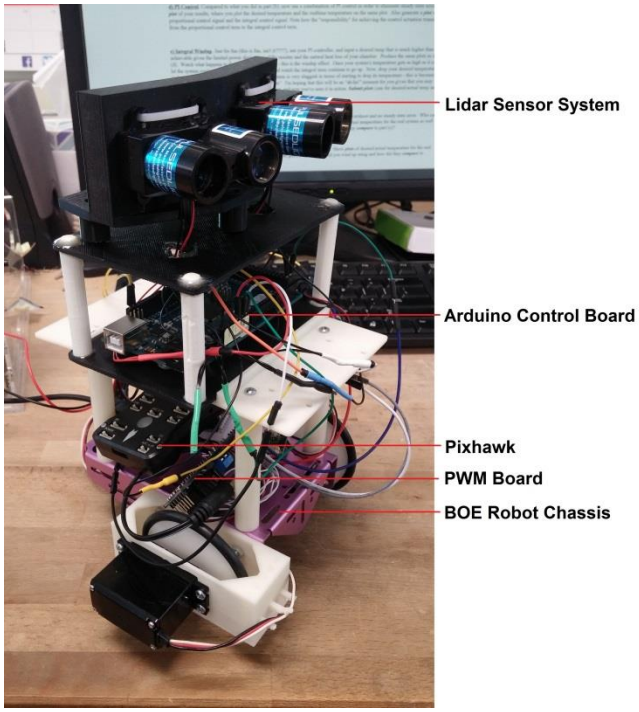

Figure 46 - The testing platform for winter quarter including the BOE robot and capstone project components integrated
Image courtesy of Ryan Sass, Santa Clara University

BOE bot required heavy modification in order to contain the electronics of the project, mount the Lidar Sensors to the robot, mount the motors in the same orientation, and establish a long enough umbilical cord to deliver power. This required special wiring for the umbilical cord, as well as heavy use of 3D printing to develop specially fitted mounts. The mounts created included the Lidar holder, a 'stack' to house the Arduino and Pixhawk, and a special motor mount for one of the motors. These special wheel motor mounts were created in order to operate both motors in the same direction to achieve forward motion. The basic shape and design of the Lidar mounts carried through the project's development.

Control output had to be slightly altered in

the software to achieve proper simulated pitch and yaw commands. Since both commands required operating the motors either in unison or in opposition, this required control loop outputs to be specially altered before sending the PWM output signal so that proper movement was achieved.

The PWM board used for this project was incorporated into this phase of testing in order to provide finer resolution of output signal change. Confirmation of the resolution of the PWM board's operation was conducted with a digital oscilloscope.

The results of these tests were an overwhelming success. The following notes for the four objectives of winter quarter include:

1. Mount the Arduino control loop hardware, including the Pixhawk, on a BOE robot and demonstrate autonomous vehicle control. : Success. All components were mounted, and all components could successfully work in unison to make the BOE robot operate as desired.
2. Demonstrate incorporation of new PWM board into hardware operation. : Success. The new PWM board relieved processing time of the Arduino software as well as provided much greater resolution when changing PWM signals. It improved resolution by a factor of about 15.
3. Demonstrate execution of yaw and pitch control loops operating in unison. : Success. The two control loops, while being gated, both worked very well and control to a set point within a few inches.
4. Prove control loop repeatability and robustness. : Success. The BOE bot exhibited very robust and accurate control that was repeatable and fairly robust. The robot could be turned left or right of the wall, and near or far, in any combination thereof, and the robot would correct its 'yaw' and then proceed to move to the deadbanded set point.

These tests proved that the control loop algorithms and hardware could successfully exhibit control over a vehicle, and that controllability of an aircraft was definitely possible, and that the Arduino control assembly was ready to be installed and tested on a drone aircraft.

4.4. Spring Quarter – Demonstration of Flight Operation

The final phase of testing was demonstration of flight operation of the Arduino control assembly. It involved a lot of objectives as this was the first time the aircraft was to be flown under automated control. The list of objectives for this quarter to demonstrate flight operation included:

1. Mount and equip the aircraft with all components in a secure, but accessible manner for flight operations, service, and data upload.
2. Develop a safe plan of operation for test flight development.
3. Establish data recording of control loop parameters.
4. Establish Arduino Control of the drone.

5. Demonstrate Arduino control loop control of the aircraft.
6. Demonstrate Arduino signal error rejection.

These objectives will be described in more detail in the following sub sections. This was the file phases of testing and development. The testing performed here was described in deep detail in the hopes that future development on this project can benefit from the testing, lessons, and success achieved here.

4.4.1. Objective 1 - Mount and equip the aircraft with all components in a secure, but accessible manner for flight operations, service, and data upload.

One of the most visible pieces of equipment on the drone, and the largest mount was for the Lidar sensors. The Lidar mount went through an initial design iteration with the assistance of a team of undergraduate students. The alignment and general shape of this mount was established in this initial phase, but other design choices were ultimately rejected as they lacked ease of access for service or adequate security.

Initially, the Arduino board was slated to be mounted just beneath the Lidar sensors. However the amount of wires and the style of wiring gave rise to a high risk for propeller strikes, either by falling out of their connection with an upside-down mounted Arduino, or by just general spacing of the wires. This upside-down mounting of the Arduino also proved difficult to remove the Arduino board for service, so the Arduino board mounting location removed from the Lidar mount and placed elsewhere with a separate mount.

The Lidar mount also initially had two fins that straddled the top and bottom part of the nose of the drone; however this proved difficult to mount, so the fit to mount to the top of the aircraft was strengthened, while the bottom piece of the mount was removed from the design. Also, slots for the mount were substituted for holes. This allowed for greater simplicity of installation for the three bolts that hold the mount in place.

The undergraduate team also designed the mount for the PWM board. This design proved very suitable, as it was stable, easy to install, located well, and did not pose any shorting risk on the mount it was located.

The Arduino board mount was chosen to sit on the top of the aircraft behind the Lidar and PWM mounts. The mount was designed in a minimalist style, and had the same push-mount style of board attachment that the PWM mount had. The mount fits screw holes already existing to hold the top plate to the aircraft. The wiring for the control loop package was trimmed and secured so that it would not easily come loose, and did not have enough flex or movement to swing anywhere near the propellers. The breadboard was attached on an open area of the Lidar mount where it was easy to access and helped keep the wiring distances between each peripheral as short as possible.

With the current positions of the Pixhawk and Arduino, it's easy to see and access the Arduino pins, see the Pixhawk state indicator light, as well as attach USB plugs to Arduino and Pixhawk for data upload. Removal of the Arduino or PWM board is as easy as pulling upwards.

Therefore, achieving the task of mounting and equipping the aircraft with all components in a secure, but accessible manner for flight operations, service, and data upload is was successful.

4.4.2. Objective 2 - Develop a safe plan of operation for test flight development.

This objective was developed through trial and error, so throughout the course of testing, three flight incidents occurred. However, a lot of procedural steps can be implemented to avoid these accidents in the future.

The first test performed on the aircraft was an anchored flight test to test motor operation and control signal outputs. This was done by strapping the drone to a stand and placing it inside a secured chamber. Proper control signal response can be confirmed by running the motors and observing the motor response. In this way it was learned that inverting the pitch command on the transmitter was necessary. This is also an important time to confirm the proper connection of each individual motor to the Pixhawk. While it was not known to perform this step, this would have adverted two of the flight incidents during this phase of testing.

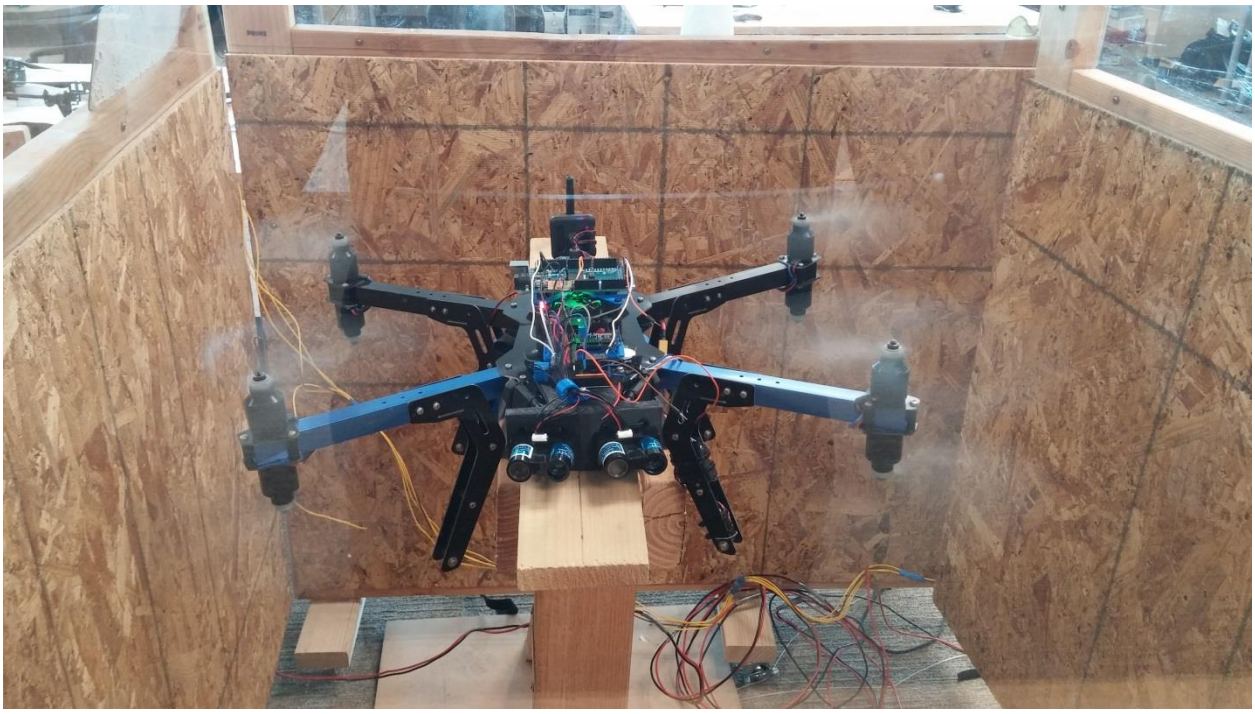

Figure 47 - Automated drone anchored flight tests in flight test chamber
Image courtesy of Ryan Sass, Santa Clara University

The first two accidents occurred due to incorrect connection of the motors to the Pixhawk. This lead to an unstable aircraft prior to it ever leaving the ground and resulted in two turnovers of the aircraft while attempting to lift-off. No amount of user control could have avoided the accident,

only confirming proper connection of the motors to the Pixhawk would have averted these incidents.

Another useful test was the “1-inch flight” tests. A flight “testing pad” was developed to address this issue with the aircraft flipping over during lift-off. It was a large, heavy piece of plastic with four anchors that the drone’s landing legs could be secured to with bailing wire. This allowed the aircraft to fly a few inches off the ground but not flip over. This provided safe troubleshooting of the aircraft when its ability to fly is uncertain.

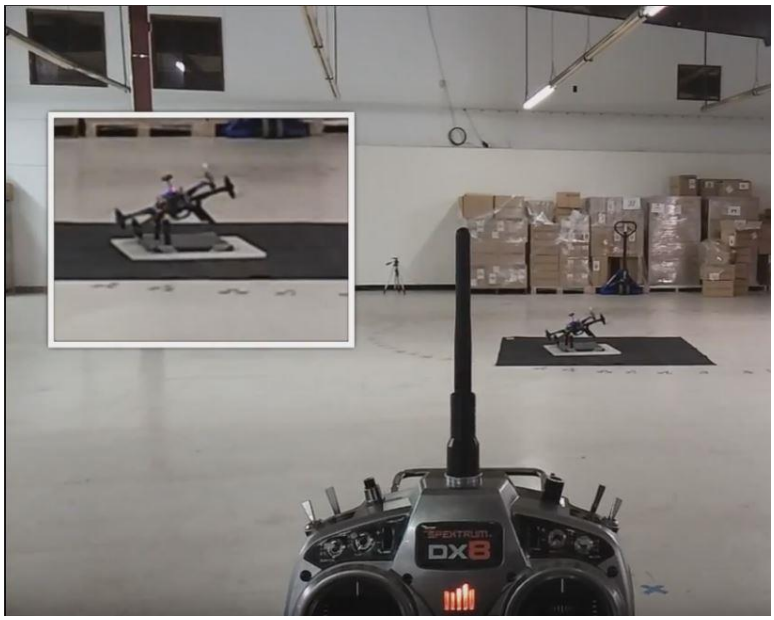

Figure 48 - Drone during "1-inch flight" tests, attempting to tip over. Zoomed inset on the left shows a close-up of drone.
Image courtesy of Ryan Sass, Santa Clara University

Finally, once the drone could fly successfully, tests of the control loops were the next steps to developing the aircraft. These tests were performed in incremental steps to ensure that each portion of the control loops in the software was functioning properly and any issues could be resolved, before the more complicated aspects of the code were tested. This meant performing the following types of flight tests in the following order:

1. Full manual flight test to establish proper manual control of the aircraft and that all drone components operate successfully under actual flight conditions and all control loop components are affixed securely to the vehicle.
2. Manual flight test of DPDT relay operation, with full control in either switch positions of the relay. Red indicator light is attached to the relay to indicate the switch. Manual control signals for pitch and yaw are connected to each corresponding NC and NO contacts on the relay to maintain full control and allow operation of the DPDT relay and the RC activated switch in a flight environment.
3. Manual flight test of DPDT relay operation with simulated automation control. Manual pitch and yaw control signals go only to their respective NC contacts on the relay. No input control signals go to the NO contacts. Establish manual flight control of the aircraft then switch the DPDT relay and confirm loss of pitch and yaw control. Also confirm successful relay operation.
4. Arduino neutral flight test. Establish manual flight, and then switch to Arduino control output signals sending neutral pitch and yaw signals. Confirm neutral PWM signals from Arduino behave as expected.

5. Arduino Yaw Only flight test. Establish manual flight with a slight yaw error and allow Arduino yaw control. Establishes Arduino yaw behavior and control for tuning. Full manual pitch control authority (similar to test 2) is maintained for safety.
6. Arduino Pitch Only flight test. Establish manual flight with a distance from the set point and allow Arduino pitch control. Establishes Arduino pitch behavior and control for tuning without any interruption from the yaw movement. Full manual yaw control authority (similar to test 2) is maintained for safety.
7. Arduino Pitch and Yaw Control. Establish manual flight with distance and/or yaw error and allow for Arduino control. This established the full capabilities of this project.

For many of the type 5 – 7 flight tests, 'dry' flight tests were conducted. These dry flight tests were conducted by placing the drone on a movable cart instead of flying the drone. From there, observation of signal input to the Pixhawk from the Arduino by way of the Mission Controller software, as well as values internal to the Arduino that were specifically coded to output for the dry tests were displayed using serial output on a computer. From here, the movable cart could be moved in relation to the wall and the corresponding displayed output commands. This allowed some preliminary signal characterization and confirmation of controller behavior to an actual flight to reduce the unexpected errors during a live flight.

While the final flight incident occurred during a type 7 test flight, the plan for operation for test flight development was successful enough to avoid any other incidents by flushing out errors far before they became issues during flight. The unfortunate mishap occurred, ironically, because the failsafe of switching to LAND mode was not utilized and an attempt to recover the aircraft under manual control during an unexpected flight circumstance. Had this failsafe been utilized instead of opting for manual control, the aircraft may have either avoided or had a less severe incident.

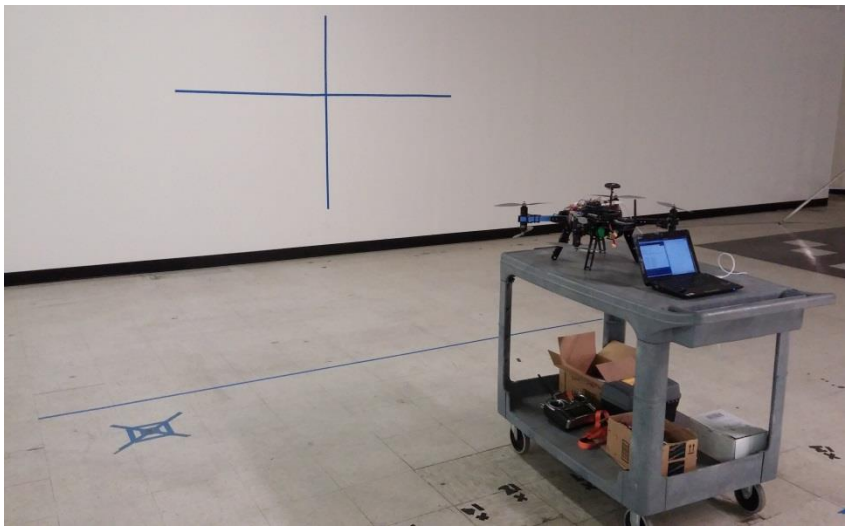

Figure 49 - A 'dry' flight test, with computer connected to observe input and output values in real time
Image courtesy of Ryan Sass, Santa Clara University

These operations, including failsafes such as the land feature for manual control intervention, kept development of the project advancing and a good rate and attributed to the project's success.

4.4.3. Objective 3 - Establish data recording of control loop parameters.

This objective was accomplished later in the development of the project, after the control loops were almost fully developed. This actually proved beneficial, as a comparison of flight control performance with and without data recording could be compared and the difference was determined to be negligible.

The development of this data recording was done in conjunction with an undergraduate student. While the overall code flow was a collaborative effort, the undergraduate student wrote the code that accomplished this task. A lot of dry testing was performed to confirm this data recording, prior to making live flights with data recording. The results of these tests provide invaluable information about the drone's performance that the Pixhawk cannot record. The resolution of this data is in time steps of 200ms, which is be more than substantial at the moment for the speed that this drone flies controlling around its set point. This ability to record data successfully at a fast rate in comparison to the drone itself proved this objective was met successfully.

4.4.4. Objective 4 - Establish Arduino control loop control of the aircraft.

This objective was completed during type 5 – 7 flight tests as outlined in Section 4.4.2. The testing of the Arduino's capabilities for drone control prior to these tests was unknown, so there was an element of "try it, see, and retune". The saving grace to establishing the Arduino control was the ability to switch back to manual mode in the event of unexpected control behavior. This objective was thoroughly demonstrated and successful by the end of this project.

4.4.5. Objective 5 - Demonstrate Arduino control loop control of the aircraft.

This objective, while being performed during an array of type 7 test flights, exhibited different behavior from the similar simulated tests performed during the winter quarter by the BOE robot. Unlike the stationary robot, the drone has more drift in the pitch movement. Also, the aircraft could move the sensor into positions that required much more error rejection than was first thought to be required during winter tests.

During the type 7 flight tests, the original control loop design intent was to perform yaw control until yaw error fell within an acceptable yaw deadband zone, then stop yaw control by issuing a neutral yaw command and execute pitch control until either located within the proper distance deadband zone or the aircraft turns out of the yaw deadband zone. This proved problematic during actual flight tests, though, as the drone would sometimes find itself just outside of the yaw deadband zone while drifting (but not controlling) to or from the set point, and could not exhibit pitch control to correct.

To fix this issue (aside from some control loop tuning), the control loop design was changed to perform both yaw and pitch control simultaneously, unless the drone had large yaw error. This was decided because in that given situation it would be unknown if pitch control at these extreme angles would put the aircraft in an unsafe location to continue pitch control as the drone is

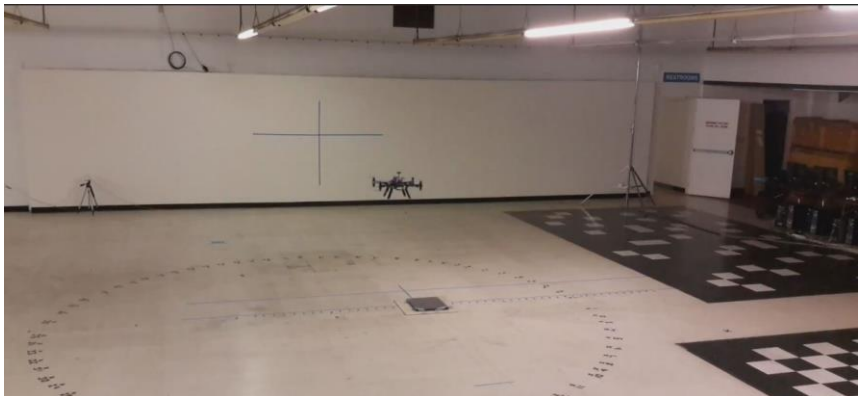

Figure 50 - Drone during flight test
Image courtesy of Ryan Sass, Santa Clara University

moving (i.e. flying the drone diagonally towards the wall and into an unsafe area as it corrects yaw and pitch at the same time). Since the current suite of sensors can't identify the drone's path, it was decided to it was decided that the drone will only attempt to correct yaw at larger error angles until it is once again reduced yaw error to resume pitch control.

The data results of the control loops are located in Section 4.5. The data is a clear depiction of proportional and proportional/derivative control loops in operation. Both control loops reach their set points and maintain their error within the deadband achieving good stability. Therefore, this objective was successfully demonstrated.

4.4.6. Objective 6 - Demonstrate Arduino signal error rejection

This objective was added after the major flight incident that caused the loss of all 8 flight motors and several propellers. Because the aircraft can drift into areas where odd sensor readings can occur, it is important to include this type of error rejection so that the drone does not try and fly out of control. This is part of the reason that caused the crash. It also demonstrated that it was important to have indicator lights to show when the aircraft is within acceptable windows of operation.

Input error rejection was incorporated into the software of the Arduino control loops. This error rejection included:

1. Aircraft distance to the wall is less than 250 cm (Too close)
2. Aircraft distance to the wall is greater than 1250 cm (Too far away)
3. Any sensor returning a negative value (Usually not reading off the wall, and not getting proper measurements from the sensor)
4. Calculated yaw angle is greater than 40° to the left or right. (Too mis-aligned from a wall, pitch control would not be a safe mode of operation because of the lack of clear space to move laterally).

These modes of flight would stop the execution of the pitch control loop and set the pitch output control to a neutral value, allowing either the user to more successfully regain control of the aircraft in manual flight mode, or allow the aircraft to resume yaw control until the yaw error has been reduced to allow for more successful execution of pitch control.

4.5. Flight Position and Control Loop Performance Data

The graph in Figure 51 shows the aircraft distance from the set point in blue and the yaw error from the set point in red. They are all graphed with respect to time and are normalized values.

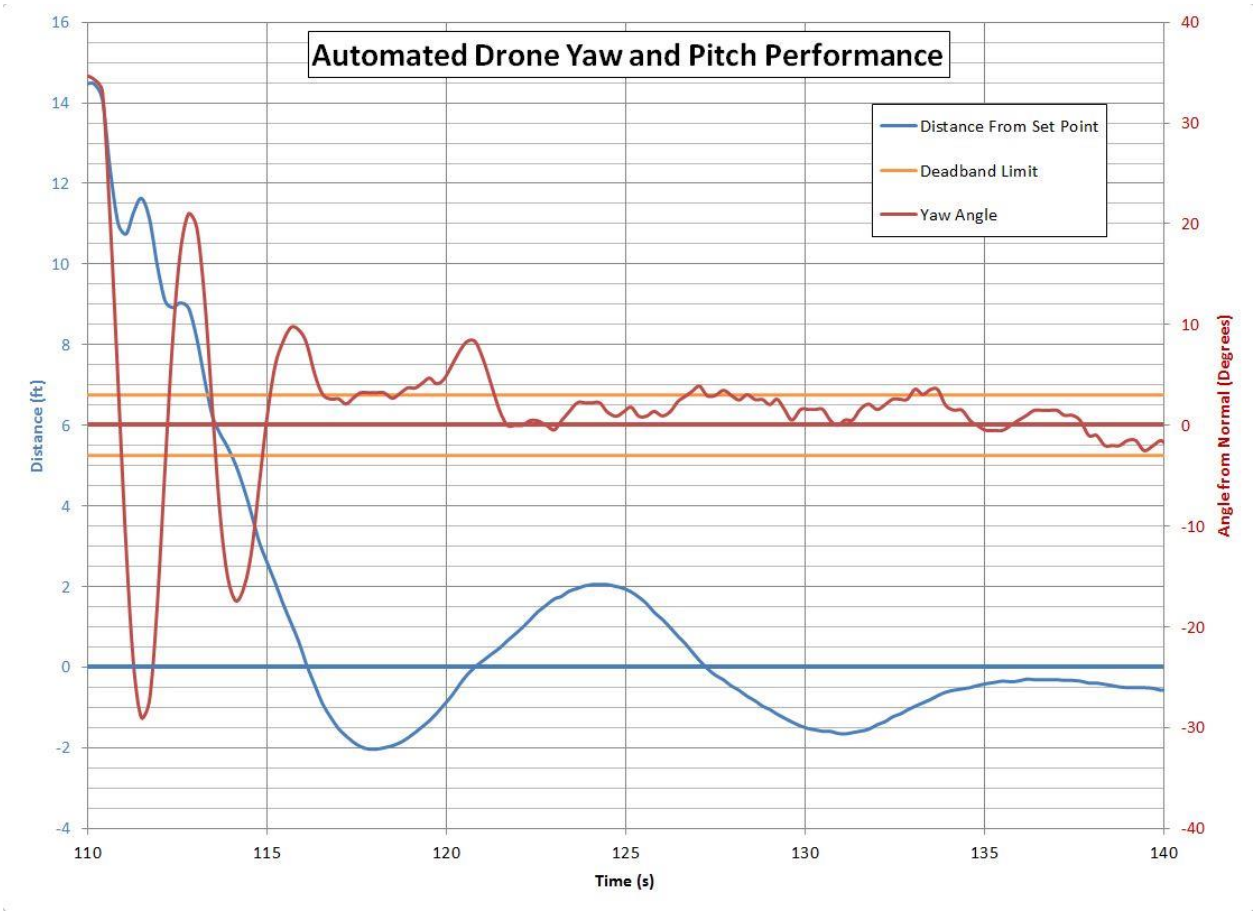

Figure 51 - Performance Graph Showing Aircraft Distance and Yaw Offset from Respective Set Points

The normalized distance values are displayed on the left vertical axis and the normalized yaw angle values are displayed on the right vertical axis. The deadband for the yaw limit is also depicted as the horizontal lines in orange, as this control deadband was large enough to be seen accurately. The set point for pitch is the horizontal line shown in blue, and for yaw the horizontal bar shown in red.

The graph in Figure 52 shows the pitch performance, in blue, with the normalized pitch control output value, in red, overlaid. The left vertical axis displays the values for normalized distance from the set point. The right vertical axis displays the normalized value for control response as added to the neutral output command. The pitch set point is the horizontal bar shown in blue.

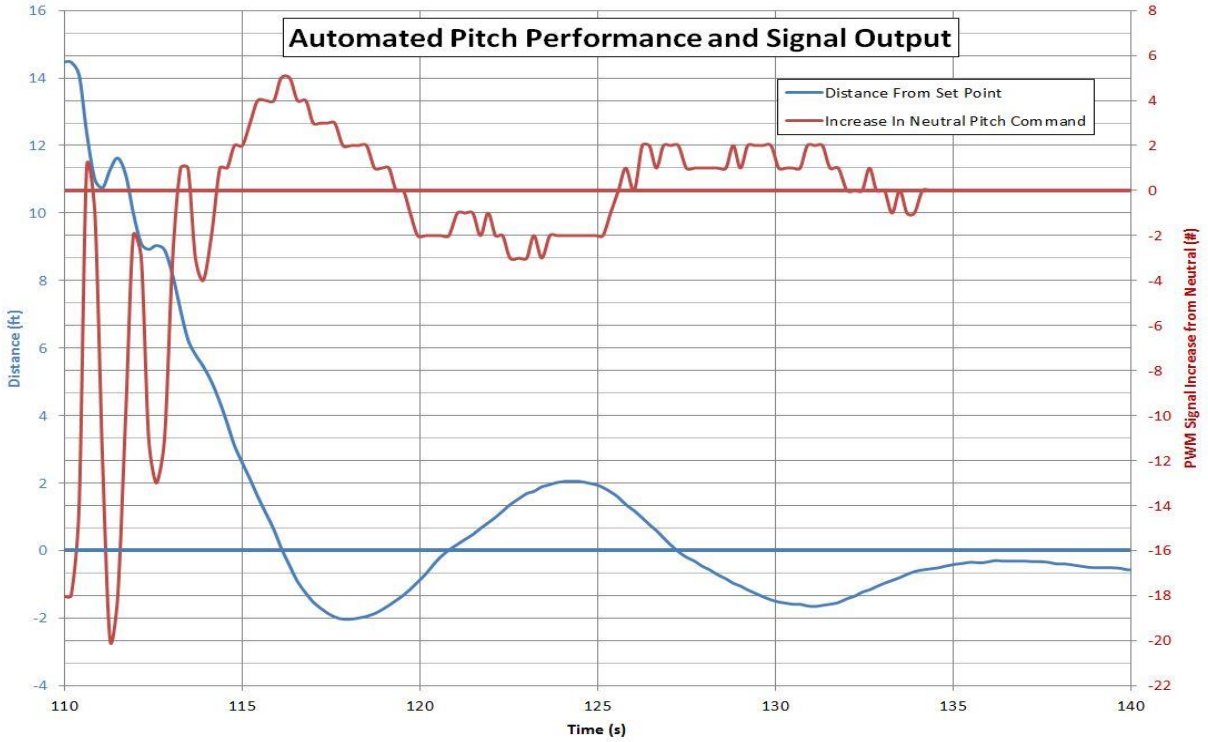

Figure 52 – Performance Graph Comparing Aircraft Distance Offset from Set Point and Pitch Control Signal Output
Image courtesy of Ryan Sass, Santa Clara University

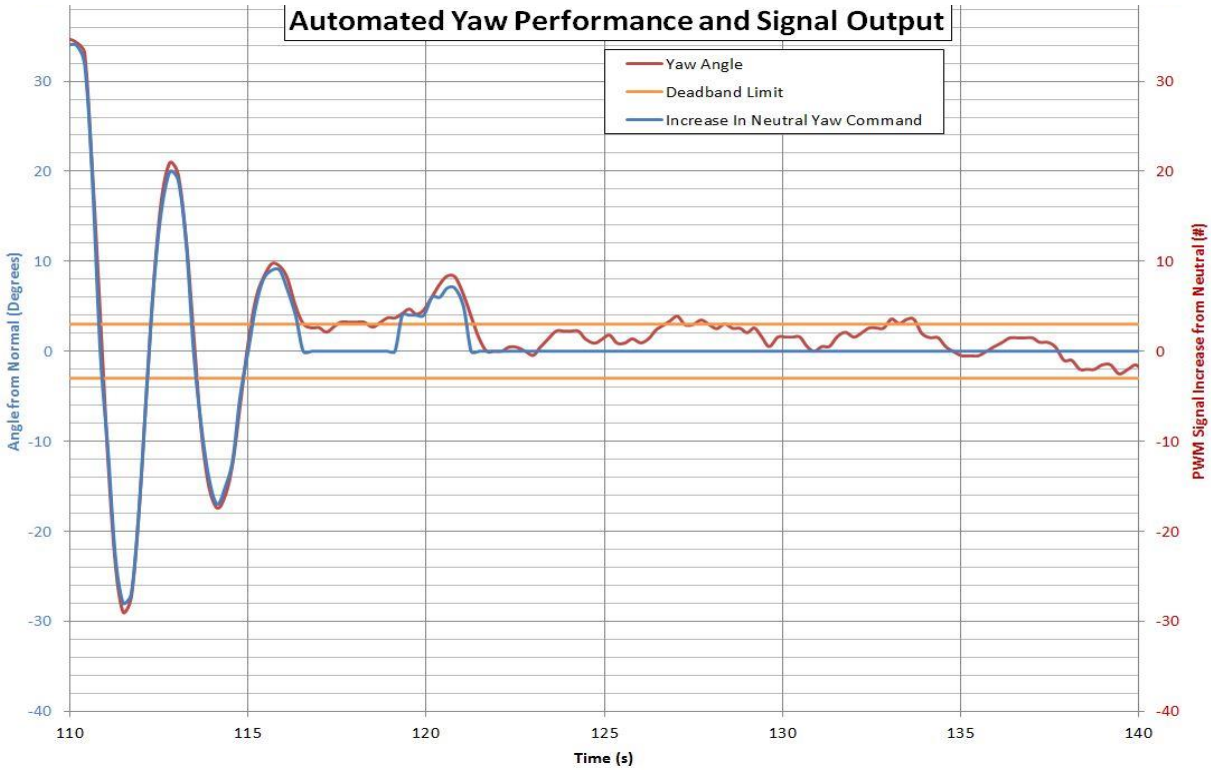

Figure 53 – Performance Graph Comparing Aircraft Yaw Angle Offset from Set Point and Yaw Control Signal Output
Image courtesy of Ryan Sass, Santa Clara University

The graph in Figure 52 shows the yaw performance, in red, with the normalized yaw control output value, in blue, overlaid. The left vertical axis displays the values for normalized distance from the set point. The right vertical axis displays the normalized value for control response as added to the neutral output command. The yaw angle input deadband bars are also shown as horizontal line shown in orange, and the yaw set point is the horizontal line shown in red.

4.6. Overall Test Results

The completed result of this project and the battery of testing is the successful demonstration of a drone that can automate pitch and yaw control via an Arduino with repeated, stable, and reliable results. The aircraft can record its data of performance and can exhibit "plane lock" flight as intended, moving up, down, left and right, while maintaining a given distance from the wall. From the test results, it can be seen that the current resolution of distance control performance from a flat wall is within ±2 feet, maintaining yaw control within ±3° from normal. The drone also has built in signal error rejection for safer operation, and the ability to return to manual flight control or automated landing for further margins of safety.

To review and comment on the Project Performance Objectives listed in Section 1.6:

- Demonstrate Aircraft Control with the use of yaw and pitch control loops. : Success. The aircraft can successfully control the pitch and yaw axes using the Arduino control assembly developed for this project.
- Demonstrate Yaw Control within ± 20° from normal. : Success. From the test result data, yaw control was maintained within ±3° from normal.
- Demonstrate Pitch Control within ± 3 ft. of a given set point. : Success. From the test result data, pitch control was maintained within ±2 feet of a given set point.

5. Results, Conclusions, and Future Work Suggestions

5.1. Results of Development

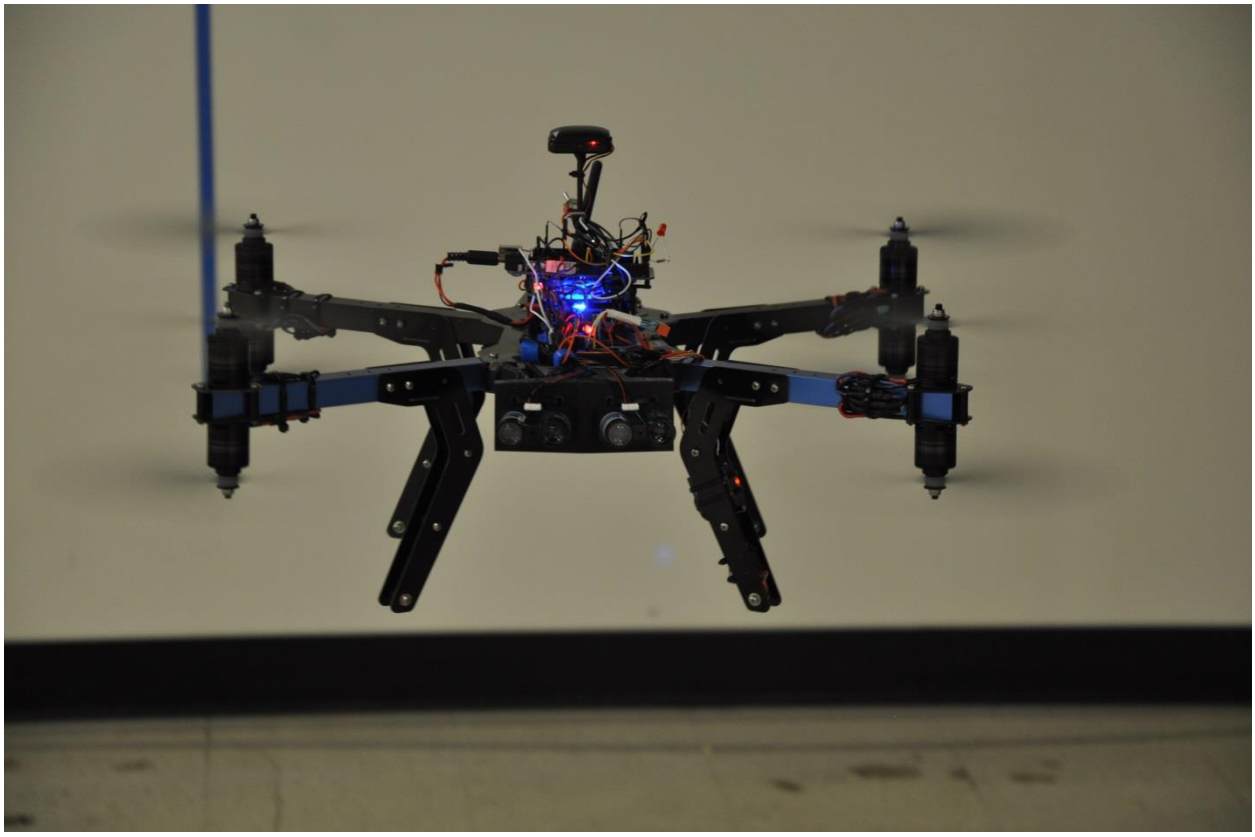

Figure 54 - Capstone Project Drone in Flight
Image courtesy of Ryan Cooper, Santa Clara University

The results of the development of this project are very successful in demonstrating the ability for a microprocessor to take control of an aircraft and exhibit closed loop control. With the exception of a few minor and one major flight incidents, the development of this project went smoothly from paper drawings and thoughts to an automated flying aircraft in three quarters of development. Each successive quarter resulted in successful advancement that continued the project forward in a relatively safe manner to prove successful operability.

The end result of the project development is a drone capable of automated simultaneous yaw and pitch control, resulting in "plane lock" with a wall. With the added error rejection and failsafes built in, the drone is a very prolific demonstrator of not only the capabilities of automated drone flight, but the potential in the level of control able to be exhibited over a drone. This was the first iteration of feedback control with a drone, and the resolution was ±2 feet from a set point, a little shy of twice the size of the craft itself, and ±3° in yaw control. With better, more sensitive sensors and PWM generators, the level, agility, and resolution of control could be vastly

improved. The end result of this project certainly opens the door to further development in many different areas.

5.2. Conclusions of Operability

From these results, there are a few conclusions we can draw from its performance. It is reasonable to conclude that complete flight automation with an Arduino - with the addition of more sensors - is possible. It is also reasonable to conclude that the Arduino can execute multiple control loops and multiple output signals at an acceptable state of control and resolution. This would lead to consideration that flying a drone from a slanted flat surface, such as a roof or flying at a given yaw angle from a wall is very possible to implement.

It can also be concluded that with some precautions, planning, and safety measures, that automated object recognition and avoidance control is possible, further giving reason to continue development of flying automated drones indoors or outdoors without GPS.

5.3. Suggestions for Future Work and Development

While this capstone project unfortunately must come to an end, it does open the door to future development, either for this project or similar projects wanting to utilize similar styles of control. Below are some of the suggestions for future development of automated drones.

5.3.1. Suggested Hardware Improvements

While the hardware used for this project was substantial to achieve project success and produce adequate results, there are some comments regarding some of the hardware.

The Arduino Mega board felt like it was the correct board to use for the maturity of this project, however, smaller and/or faster boards might be worth considering. The same capabilities may be accomplished with an Arduino Micro or Nano. Or, switching to an Intel Edison board may be a better choice for finer control of the aircraft.

The PWM board vastly improved the abysmal resolution performance the Arduino itself attempted to do - by a factor of 15 - however even the resolution of the PWM board was almost not fine enough. Because the duty cycle of the RC signal ranges from 5% to 9% to indicate 0% to 100% input, the full resolution of the PWM board is not used. Add to this the limiting of each control signal in software, and this reduces the usable resolution by almost another 30%. Therefore, much finer resolution of output signals may provide more accurate control, especially when the aircraft is near the set point and finer control input is required.

The Lidar sensor package had an offset of 7.5°; however this value should be tooled with to find the most optimal offset. Also, while the sensors themselves proved very accurate and fast, perhaps a better way of sensing a wall might be considered that can incorporate more information regarding the environment the drone is in.

Finally, powering the Arduino was performed with a 9V battery; however there is a plug for power directly from the aircraft battery that was never implemented to power the Arduino. While it was never determined if this power was regulated, it did not prove reliable power when a direct connection was attempted. Also, because the power module that connects to the aircraft battery that powers the Pixhawk only outputs 5V; this was not deemed a suitable choice for the Arduino. Developing a regulated power source from the aircraft's battery to power the Arduino and peripherals would be beneficial, as dying 9V batteries were a common problem, and resulted in a few odd behaviors during test flights.

5.3.2. Improved Response Time or Narrowing the Deadband

Improving the response time of the aircraft to reach the set point and improve control performance will require some more serious hardware to be able to more carefully tune parameters, as well as more accurate ways to read data in order to characterize the performance. However, judging by the overall responsiveness of the aircraft, this ability to move faster to the set point should be possible either by further tuning the existing hardware or improving the hardware as well as retuning.

To narrow the deadband would be to improve the ±2 feet of slop in the position hold. To be clear, the deadband of the implemented control loop was very narrow, but the performance resulted in a deadband of ±2 feet. Again, finer resolution of the control signals, as well as refined, or possibly different control characteristics when close to the set point might improve the ability for the aircraft to maintain control.

5.3.3. Altitude and Lateral Control

The next natural step for development of the capabilities of the current hardware of this drone would be to include roll and throttle control. The Pixhawk has an ALT HOLD mode; however this mode was never tested due to odd aircraft behavior of this mode during testing. While tethered to an anchor, it appeared the drone attempted to 'fly off', revving the motors at alarmingly high speeds. After an attempt to demonstrate safe operation while flying on a 5' rope tether were unsuccessful, due to a perceived lack of safety in the event of a fly off (it appeared the drone would hit the end of the tether and proceed to swing along the tensioned tether into the ground during light simulations of a fly off), the decision was made to not perform any indoor test, tethered or untethered, of altitude hold. Proper testing of this mode should be performed outside where there is no ceiling to contend with.

During most of testing, roll correction was a constant issue, and the drone would drift from side to side. Another flat wall combined with a roll control loop would easily solve this drift, as the added loop's implementation would be nearly identical to the control loop implemented for pitch control.

It would be great to see automation of all 4 axes of flight, allowing complete "hands off" of the controls.

5.3.4. Adjustable set points

The RC receiver has a wonderful infinitely adjustable auxiliary dial on it, which appeared to be a great way to adjust the set point of the distance from the wall while in flight. Because this project was focused on implementing operational success of the aircraft itself, there was not enough time to develop a method to implement receiving this signal for this project. It would be great to see the aircraft reach a set point and then re-correct to a new set point while in mid-flight.

5.3.5. Pattern Following in "Plane Lock"

Now that the drone has exhibited "plane lock" ability, perhaps programming the altitude and lateral control to allow the drone to fly in an up-down / left-right style pattern, in order to demonstrate automation for things like wall scanning, surveying, or perhaps photography.

5.3.6. Slanted Plane Control

Another great development would be to test the control of the aircraft exhibiting "plane lock" with respect to a slanted wall, either a roof style plane, or the drone controlling pitch control while holding a yaw angle from the normal to the wall. A roof-style plane would require implementing simultaneous throttle and pitch control (as well as yaw control still), and a slanted wall would require roll and pitch control (as well as yaw control still). Either would require a combination of mixing signals to achieve performance.

5.3.7. +6 Sensor Alignment Control

This would be an expansion suggested in Section 5.3.2, by adding sensors in both directions for roll, pitch, and altitude. Perhaps allowing the drone to shift between controlling distance from one side vs. the other (like controlling distance on its left then right side as it travels from one area to another), would allow a drone to start safely navigating down hallways and start to achieve successful object avoidance by keeping away from the closest object.

5.3.8. Multi Drone Formation

Set up two drones to maintain distance lock from one another while performing flight maneuvers. This would require much more sophisticated sensing techniques, as well as multiple people for development.

References

1. **X8+ Operation Manual** – https://3dr.com/wp-content/uploads/2015/04/X8-Operation-Manual-vC.pdf
2. **Pixhawk Autopilot Information** - https://pixhawk.org/modules/pixhawk
3. **Lidar Information** - http://pulsedlight3d.com/products/lidar-lite-v2-blue-label.html
4. **Pixhawk Reference** - http://ardupilot.org/copter/docs/assembly-instructions.html#
5. **Adafruit 16-Channel Servo Driver Library Reference** - https://learn.adafruit.com/16-channel-pwm-servo-driver/library-reference

Appendix

Project Flight Procedures

When operating the drone in automated flight mode, it is suggested that the following general procedures are followed for safe operation of not only the drone, but the test area. The procedures include:

1. Preflight
    1.1. Observe the flight area. Ensure there is enough area and height in which you want to fly, and that user ability will allow for flight error.
    1.2. Determine “no go” or “mandatory manual control and/or land” zones, this will help to know when the aircraft still has error margins but must action must be immediately taken in order to avoid an accident.
    1.3. Ensure aircraft is properly calibrated, motors are connected properly and motors are connected to the correct motor connection. Ensure all components on the aircraft are secure for flight and that the propellers can spin freely.
    1.4. Ensure RC Transmitter is properly calibrated, and signals correspond to proper flight controls. Ensure all auxiliary signals behave as expected prior to flight.
    1.5. Check battery voltage for aircraft and Arduino. Ensure sufficient voltage, and if using a multi-cell battery, cells are balanced.
    1.6. Have a video recorder set up. In the event of a flight incident, video review can assist in helping to understand the chain of events that led to an incident.
    1.7. Consider personal safety equipment. Foot and eye protection should be considered.
    1.8. Consider a neck tether for the RC Transmitter, it will allow for better input control by not having to support the transmitter weight.
2. Starting and Liftoff.
    2.1. Have on proper safety equipment for flight, including eye and foot protection
    2.2. Start video recording
    2.3. Plug in Battery
    2.4. Wait for Pixhawk to initialize. A tone and a blue or green blinking light means the Pixhawk is ready.
    2.5. Ensure throttle is in the low position and all switches are in their proper place, and then power the RC transmitter.
    2.6. Ensure the RC transmitter is commanding the Pixhawk to be in a safe flight mode, such as LAND.
    2.7. Ensure switch for SD card recording is in the “LOG” position.
    2.8. Switch power to the Arduino ON

2.9. Confirm operation of the automation/manual control switch by toggling. Feedback will be a red light illuminating in automation mode, and extinguishing for manual mode. Leave in manual mode.

2.10. Press and hold red flashing safety switch until it turns stable red. The ESC's will make the motors briefly jump, and a tone can be heard. Motors now have power, but are not yet armed or should be spinning.

2.11. Stand back a good distance from the aircraft. Standing directly behind the aircraft is recommended.

2.12. Switch the flight mode on the RC transmitter to a flyable mode, such as STABALIZE.

2.13. Hold the left control stick (throttle and yaw) in the lower right position. Once tone is heard, release yaw control to neutral. Copter is now armed and will fly upon advancement of the throttle.

2.14. To takeoff, SLOWLY advance throttle until the drone comes off the ground.

2.15. Upon liftoff, back off the throttle a little to reduce climb rate.

2.16. Proceed to flying!

3. Manual Flight

3.1. In manual flight, roll, pitch, yaw, and throttle are all under user control.

3.2. Any flight adjustment to one axis will affect the aircraft in such a way that another input is usually needed. For instance, if pitching forward, more throttle is required to maintain the same altitude or the drone will pitch forward and lose altitude.

3.3. The switch for manual flight was defaulted to be in a position where if the RC transmitter is grabbed or bumped, it will push the switch into the manual position.

4. Preparing for Automated Flight

4.1. To prepare for automated flight, fly the drone to an area where the drone's Lidar sensors appear to still be observing the wall. It is suggested to not exceed 35° – 50° of yaw from the wall. Remember, the further away the drone is, the more wall required for more extreme angles of yaw.

4.2. Ensure the aircraft is relatively stable before switching to automated flight.

4.3. Flying less than 8 feet or more than 40 feet from a wall is not recommended for automatic flight. Ensure the aircraft is within this tolerance.

4.4. When ready for automated flight, switch the automated / manual control switch outward to allow manual flight.

5. Automated Flight

5.1. Once in manual flight, control authority over pitch and roll will not be exhibited by the transmitter. The Arduino has full control of these flight axes.

5.2. Allow the drone to drift a little. It will drift under automated control, especially the further away it is. Large amounts of yaw or pitch are a sign of incorrect operation. In this case, consider returning back to manual control by pulling the control switch back in to manual mode.

5.3. Remember that roll and throttle control are still required for safe flight. This means observing the aircraft at all times during this flight mode as well. While pitch and yaw movement may not necessarily require corresponding throttle or pitch input, consider focusing on maintaining these two axis of flight in a controlled, slow, and stable manner.
5.4. When ready to return to manual mode, pull the control switch back in to manual mode. Pitch and yaw control authority will be restored to the RC transmitter.

6. Landing
   6.1. While landing the drone can be performed in STABALIZE mode, the simplest method is suggested to switch to LAND mode.
   6.2. Find a suitable landing area. Hopefully the area of flight is all suitable for landing on.
   6.3. When in position, switch to land mode and maintain directional control.
   6.4. Maintain direction control through touchdown of the aircraft.
   6.5. Reduce the throttle to its minimum position.
   6.6. For advanced fliers, consider landing on a padded area for reduction in shock and possible landing damage.
7. Post Landing and Shutdown
   7.1. Hold the left control stick in the lower left position to disarm the aircraft. A tone can be heard when held. Land mode will automatically disarm the aircraft and play the tone.
   7.2. Ensure the aircraft is in LAND mode or disarmed before approaching the aircraft and that the throttle position is in the full down position.
   7.3. Press and hold the solid red light button until it starts blinking, this indicates the ESC's no longer have power routed to them and it is safe to be near the propellers.
   7.4. Switch the SD card switch to OFF and wait 1-2 seconds. It is suggested to repeat this again for thoroughness.
   7.5. Switch off the Arduino power.
   7.6. Disconnect the battery
   7.7. Power off the RC transmitter
8. Emergencies
   8.1. In the event of an unsuitable flight condition, the best options are to switch to manual control, and switch to LAND mode. When in doubt, LAND!
   8.2. If manual control disorientation occurs, switch to LAND mode! Even if it results in a rough landing, it is better than a failed attempt to save the drone and crash it hard.
   8.3. Ensure that a battery failsafe is in place and that the accelerometers exhibit proper calibration prior to liftoff. Never attempt to fly if the accelerometers exhibit poor calibration.

MECH_290_master_code_v2.6_FLIGHT_PITCH_AND_YAW.ino

```
#include <math.h>
#include <Wire.h>
#include <LIDARLite.h>
#include <Adafruit_PWMServoDriver.h>

/*
 * Ryan Sass
 * Santa Clara University
 * rsass@scu.edu or ryan.sass@gmail.com
 *
 * Date Published: 5/16/2016
 * Version Number: 2.6
 * File: MECH_290_master_code_v2.6_FLIGHT_PITCH_AND_YAW.ino
 *
 * Full Control Command Code
 *
 * Intended to be run on Arduino MEGA 2560 r3
 *
 *
 ****** I/O PIN ASSIGNMENTS ******
 *
 *     ~~~ Input ~~~
 * A0 - LIDAR Sensor Left
 * A1 - LIDAR Sensor Right
 * A4 - Receiver Roll Signal
 * A5 - Receiver Pitch Signal
 * A6 - Receiver Yaw Signal
 * A7 - Receiver Gear Signal
 *
 *     ~~~ Output ~~~
 *  03 - [PWM] Roll Signaling
 *  06 - [PWM] Pitch Signaling
 *  10 - [PWM] Yaw Signaling
 *  45 - Control Lockout Signal (tied to Gear Signal)
 *
 */


//VARIABLES TO BE INITIALIZED
/*
roll - roll output pin value
pitch - pitch output pin value
yaw - yaw output pin value
CLS - control lockout signal output pin value
h_val_10 - value between 0 - 1024 that delineates what is 'HIGH'
ctrl_enable - CONTROL ENABLE - allows yaw control to occur. It is a signal
input.
lidar_l - LIDAR SIGNAL LEFT - Signal input of the left LIDAR unit.
lidar_r - LIDAR SIGNAL RIGHT - Signal input of the right LIDAR unit.
lidar_diff - LIDAR DIFFERENCE - subtraction between left and right LIDAR
units.
```

```
lidar_yaw_db  - LIDAR YAW DEADBAND - The deadband window for LIDAR difference
(lidar_diff) that sets to 0.
yaw_error - YAW ERROR - The difference between LIDAR difference (lidar_diff)
value and yaw setpoint (yaw_sp).
yaw_sp - YAW SET POINT - hard coded value of the yaw set point, the
difference between the two LIDAR units.
yaw_p - YAW PROPORTIONAL GAIN - the proportioanl gain for the yaw control
yaw_PWM - YAW PWM SIGNAL - the converted PWM output signal

*/
//variable assignments
bool success1,success2;

//int roll = 3;
int pitch = 2;
int yaw = 4;
int CLS = 45;
int h_val_10 = 612;
int lidar_yaw_db_o = 3;
int yaw_only_db = 40;
int yaw_sp = 0;
int32_t pwm_freq = 45;
int sig_avg = 25;
int yawpitch = 0; //yaw = 0, pitch = 1
int yaw_PWM_neutral = 278;
int lidar_pitch_db = 1;
int pitch_sp = 670;
int pitch_PWM_neutral = 278;
int pitch_trim = 11;
int pitch_error = 0;
int pitch_temp = 0;
int
ctrl_enable,lidar_l,lidar_r,lidar_diff,yaw_error,yaw_PWM,x,pitch_ave,pitch_PW
M,output1,pitch_error_old,pitch_delta,lidar_yaw_db;

float db_grow;
float angle,tx,ty,tz;
float pitch_P = 1.0;
float yaw_P = 1.0;
//float pitch_P = 10.5;
//float yaw_P = 11.5;


int sensorPins[] = {2,3}; // Array of pins connected to the sensor Power
Enable lines
unsigned char addresses[] = {0x66,0x68};
LIDARLite myLidarLite;
Adafruit_PWMServoDriver pwm1 = Adafruit_PWMServoDriver();


void setup() {

  // initialize serial communication at 9600 bits per second:
//  Serial.begin(9600);

  //Turn pins on
```

```
  pinMode(31,OUTPUT);
  digitalWrite(31,HIGH);
  //initializing Lidar signals
  myLidarLite.begin();
  myLidarLite.changeAddressMultiPwrEn(2,sensorPins,addresses,false);
  //start PWM board
  pwm1.begin();
  pwm1.setPWMFreq(48);

  //pinout assignments
  pinMode(CLS,OUTPUT);
  ctrl_enable = 1;
  lidar_yaw_db = lidar_yaw_db_o;
  pwmneutral();
  attachInterrupt(digitalPinToInterrupt(18),stopDataLogging,RISING);
  pinMode(18, INPUT);
  initializeDataLogger();
}

/*******************************************START OF MAIN
CODE*******************************************/
void loop() {
    //process LIDAR sensors
    String s(millis());
    s += " ";
    logData(s);
    lidarprocess();

    yaw_control();//yaw vs pitch control
    //Serial.print("Yaw Pitch \t");
    //Serial.println(yawpitch);
    if(yawpitch == 1) pitch_control();

    //set PWM values to controller
    pwmset();
    logData('\n');

    //print any serial outputs for debugging
    //printoutput();
}
/*******************************************END OF MAIN
CODE*******************************************/

void enab_sig_proc(){
  ctrl_enable = analogRead(A7);

  //character gear signal to equivalent boolean
  if(ctrl_enable >= h_val_10) {
    //enable signal - enable TRUE
    ctrl_enable = HIGH;
  } else {
    //disable signal - enable FALSE
    ctrl_enable = LOW;
  }

  //signal yaw command signal relay to connect
  digitalWrite(CLS,ctrl_enable);
```

```
}

void lidarprocess(){
  //LIDAR signal input & averaging
  x = 0; lidar_r = 0; lidar_l = 0;
  while(x < sig_avg){
    lidar_r += myLidarLite.distance(true,true,0x66);
    lidar_l += myLidarLite.distance(true,true,0x68);
    x++;
    delay(2);
  }
  lidar_r /= x;
  lidar_l /= x;
  logFloat(lidar_r, " lidar_r: ");
  logFloat(lidar_l, " lidar_l: ");
  yawcheck();

  //LIDAR signaling diferencing
  pitch_ave = (lidar_l + lidar_r)/2;
  if(yawpitch != -1){
    anglecalc();
  } else {
    angle = 0;
  }

  db_grow = pitch_sp - pitch_ave;
  db_grow /= pitch_sp;

}

void yawcheck(){
  if((lidar_l > 0) && (lidar_r > 0)){
    if((pitch_ave > 1250) || (pitch_ave < 250)){
      yawpitch = 0;
      digitalWrite(31,HIGH);
    } else {
      if(abs(angle) <= yaw_only_db) {
        yawpitch = 1;
        digitalWrite(31,LOW);
      } else {
        yawpitch = 0;
        digitalWrite(31,HIGH);
      }
    }
  } else {
    yawpitch = -1;
    digitalWrite(31,HIGH);
  }
}

void yaw_control(){

  //set point signal conditioning
  if(lidar_l > lidar_r){
    yaw_error = angle - yaw_sp;
  } else {
    yaw_error = yaw_sp - angle;
```

```
  }
  if(abs(yaw_error) <= lidar_yaw_db) yaw_error = 0;

  //yaw PWM signal conditioning
//  yaw_error = (yaw_error > 45 || yaw_error < -45) ?
(abs(yaw_error)/yaw_error)*45 : yaw_error;
  if(yaw_error > 45) yaw_error = 45;
  if(yaw_error < -45) yaw_error = -45;
  yaw_error /= yaw_P;
  yaw_PWM = yaw_error + yaw_PWM_neutral;
  logFloat(yaw_PWM, " yaw_PWM: ");
  if(yawpitch != 1)  pitch_PWM = pitch_trim + pitch_PWM_neutral;
}

void pitch_control(){
  pitch_error_old = pitch_temp;
  pitch_error = pitch_sp - pitch_ave;
  pitch_temp = pitch_error;
  pitch_error /= 10;

  //signal difference deadbanding
  if(abs(pitch_error) <= lidar_pitch_db) pitch_error = 0;

  //proportional gain
  pitch_error *= pitch_P;

  //yaw PWM signal conditioning
  if(pitch_error > 40) pitch_error = 40;
  if(pitch_error < -40) pitch_error = -40;
  pitch_error /= 2.2;
  pitch_delta = pitch_temp - pitch_error_old;
  pitch_delta *= (1/3.0);
  if(pitch_delta > 30) pitch_delta = 30;
  if(pitch_delta < -30) pitch_delta = -30;
  if(abs(pitch_delta) < 1) pitch_delta = 0;
  pitch_PWM = pitch_error + pitch_trim + pitch_delta + pitch_PWM_neutral;
  if(pitch_PWM > pitch_trim + pitch_PWM_neutral + 15)   pitch_PWM =
pitch_trim + 15 + pitch_PWM_neutral;
  if(pitch_PWM < pitch_trim + pitch_PWM_neutral - 20)   pitch_PWM =
pitch_trim - 20 + pitch_PWM_neutral;
  logFloat(pitch_PWM, " pitch_PWM: ");
}

void anglecalc(){

  if(lidar_l < lidar_r){
    tx = pitch_ave - lidar_l;
  }else{
    tx = pitch_ave - lidar_r;
  }

  angle = lidar_l;
  tz = lidar_r;
  ty = (sqrt( (angle * angle) + (tz * tz) - (2 * angle * tz * 0.9848) ) * 0.5
);
  angle = ( (ty * ty) - (tx * tx) );
  if(angle < 0.05) angle = 0;
```

```
  tz = sqrt(angle);
  if((tx == 0) || (ty == 0)){
    angle = 0;
  } else {
    angle = ( (tx*tx) + (ty*ty) - (tz*tz) ) / (2 * tx * ty);
  }

  angle = (90 - (180 / 3.14159) * acos(angle));
}

void printoutput(){
  //printing output to monitor
  Serial.print("\n\n***** START ****** \n\n\n");
  Serial.print("\n Angle \t");
  Serial.print(angle);
  Serial.print("\n Pitch Ave \t");
  Serial.println(pitch_ave);
  Serial.print("\n");
  Serial.print("\n LIDAR L \t");
  Serial.print(lidar_l);
  Serial.print("\n LIDAR R \t");
  Serial.print(lidar_r);
  Serial.print("\n");
  Serial.print("\n Mode \t");
  if(yawpitch == 0){
    Serial.print("YAW");
  }else{
    if(yawpitch == -1){
      Serial.print("BAD SIGNAL");
    }else{
      Serial.print("YAW & PITCH");
    }
  }
  Serial.print("\n Pitch PWM signal \t");
  Serial.print(pitch_PWM);
  Serial.print("\n Yaw PWM signal \t");
  Serial.print(yaw_PWM);
  Serial.print("\n");
  output1 = pitch_PWM;
  if(output1 == pitch_PWM_neutral + pitch_trim){
    Serial.print("\nNEUTRAL ");
  }else{
    if(output1 > pitch_PWM_neutral + pitch_trim){
      Serial.print("\nBACKWARD ");
    } else {
      Serial.print("\nFORWARD ");
    }
  }
  Serial.print(output1);
  Serial.print("\n\n***** BREAK****** \n\n\n");
  delay(1000);
}

void pwmset(){
  pwm1.setPWM(yaw, 0, yaw_PWM);
  delay(1);
```

```
  pwm1.setPWM(pitch, 0, pitch_PWM);
}

void pwmneutral(){
  pwm1.setPWM(0, 0, yaw_PWM_neutral);
  pwm1.setPWM(1, 0, yaw_PWM_neutral);
  //pwm1.setPWM(2, 0, yaw_PWM_neutral);
  pwm1.setPWM(3, 0, yaw_PWM_neutral);
  //pwm1.setPWM(4, 0, yaw_PWM_neutral);
  pwm1.setPWM(5, 0, yaw_PWM_neutral);
  pwm1.setPWM(6, 0, yaw_PWM_neutral);
  pwm1.setPWM(7, 0, yaw_PWM_neutral);
  pwm1.setPWM(8, 0, yaw_PWM_neutral);
 }
```

LidarDataLogger.ino

```
#include <SPI.h>
#include <SD.h>

/*
 * Ryan Cooper
 * Santa Clara University
 *
 * Date Published: 5/15/2016
 * Version Number: 2.6
 * File: LidarDataLogger.ino
 *
 * Intended to be run on Arduino MEGA 2560 r3
  */


// change this to match your SD shield or module;
// Arduino Ethernet shield: pin 4
// Adafruit SD shields and modules: pin 10
// Sparkfun SD shield: pin 8
const int chipSelect = 53;
File dataFile;

void initializeDataLogger() {
//  Serial.print("Initializing SD card...");
  if (!SD.begin(chipSelect)) {
//    Serial.println("Card failed, or not present");
    return;
  }
//  Serial.println("card initialized.");
  int i = 0;
  String file(i);
  while(SD.exists(file)) {
    String temp(++i);
    file = temp;
  }
//  Serial.println(file);
```

```
  dataFile = SD.open(file, FILE_WRITE);
}

bool logData(String data) {
  if (dataFile) {
    dataFile.print(data);
//    Serial.println(data);
    return true;
  } else {
//    Serial.println("error during data logging");
    return false;
  }
}

void logFloat(float f, String descriptor) {
  char c[10];
  dtostrf(f, 9, 4, c);
  String temp(c);
  descriptor += temp;
//  Serial.println(descriptor);
  logData(descriptor);
}


bool logData(char data) {
  if (dataFile) {
    dataFile.print(data);
//    Serial.println(data);
    return true;
  } else {
//    Serial.println("error during data logging");
    return false;
  }
}

void stopDataLogging() {
  if (dataFile) {
//    Serial.println("done logging");
    dataFile.close();
  } else {
//    Serial.println("Failed to log data");
    return false;
  }
}
```