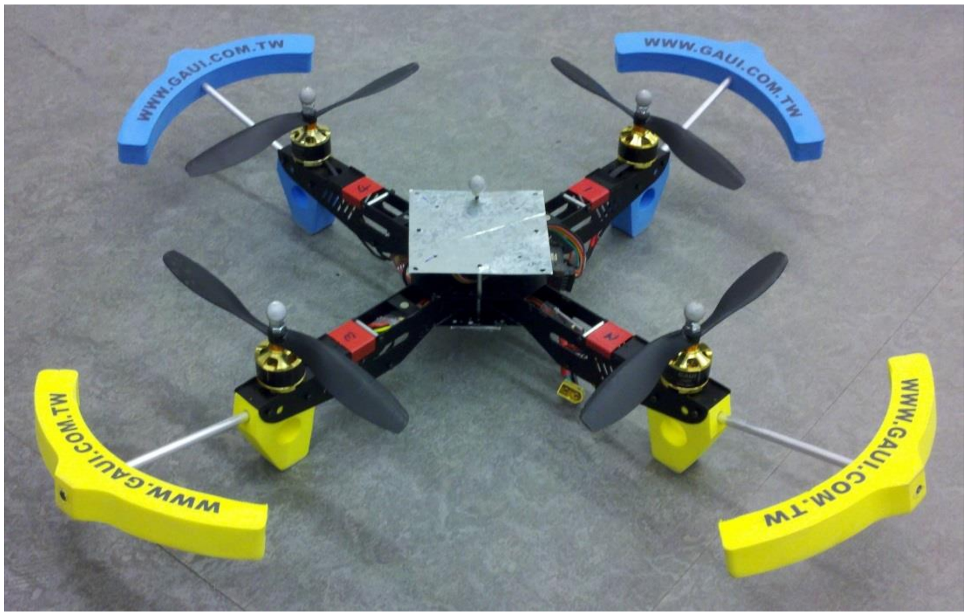
Iowa State University

MicroCART Final Documentation
May 14-10



Team: Kevin Engel
Nathan Ferris
Willam Franey
Michael Johnson
Kelsey Moore
Lucas Mulkey
Aaron Peterson

Advisor: Dr. Phillip Jones
Client: Dr. Nicola Elia

The goals of our project were to convert the system from being completely controlled by an external computer and camera system, to one that is almost entirely onboard the UAV. This will require using Bluetooth to communicate between a laptop base station and the FPGA, an onboard inertial measurement unit (IMU), a pseudo-GPS using the camera system, and a new light weight GUI for sending commands. We also took out the GAUI 344 and replaced it with a simple motor controller.

Table of Contents

List of Figures

MicroCART Final Documentation May 14-10

1 General Definitions

1.1 Abbreviation list

MicroCART - Microcontroller Controlled Aerial Robotics Team

IMU - Inertial Measurement Unit: Contains an accelerometer and gyroscope for accurate roll, pitch, and yaw

GPS - Global Positioning System

Pseudo-GPS - The VPN camera system used as a way to determine location on the x, y, and z plane

PID - Proportional-Integral-Derivative controller

FPGA - Field Programmable Gate Array

Quadcopter - The Micro-Controlled Aerial vehicle used for the project

ESC - Electronic Speed Controller

CRC - Cyclical Redundancy Check

LABET board - a custom made FPGA board for MicroCART, with the name based on Iowa State's Make to Innovate (M2I) program: HABET: High Altitude Balloon Experiments in Technology. 'L' in LABET: Low.

1.2 Hardware Descriptions

Base Station
The base station PC is used to grab data from the camera system and send the information to the FPGA. It also takes user commands to control the quad. Currently, it has both a low-level terminal-based interface and a full graphical interface.
Inputs: Camera System Data, User Commands
Outputs: Position data to the FPGA
Interfaces: GUI, LABET FPGA

High-Speed Camera System
The high speed camera system is our current method of GPS. This system tracks four small infrared reflectors located on top of the quadcopter and quantifies the information

(x,y,z) and outputs via the VRPN protocol to the base station.
Inputs: Location of infrared balls
Outputs: Position data to the base station via VRPN
Interface: OptiTrack Software

Bluetooth XBEE module

To utilize wireless communication functionality, we have two XBEE Bluetooth modules that communicate through UART. We are using the first module to send the position data and commands from the base station to the second module located on the LABET FPGA board.
Inputs\Outputs: Position data from camera system and commands from interface
Baud rate: 19200
Interface: Base station GUI

LABET FPGA

The LABET FPGA board, located on the quadcopter, is used to stabilize and control the quadcopter.
On the board: IMU, Bluetooth module
Inputs: Position data from the Bluetooth module, Acceleration and orientation from the IMU
Outputs: PWM signal to quad's motors.

USB Controller

The USB controller allows the quadcopter to be flown as originally intended, but through bluetooth and the LABET board.
Outputs: Pitch, roll, yaw, throttle
Interface: USB/GUI

Quadcopter

The signals that are output from the LABET board travel to the ESCs, which tell the motors how fast to spin. The IMU is on the quadcopter and provides the orientation data that was previously provided by the camera system. The ESCs connect directly to the battery for power, and the motors are connected to the ESCs. The ESCs determine how much voltage to send to the motors by the duty cycle of the PWM. The higher the voltage to the motor, the higher the revolutions per minute (RPM).
On the Quadcopter: Speed Controller x4, Motor x4, LABET Board, 2200mAh Battery, LABET battery(Don't know its size)

1.3 Project Definition

At the beginning of the year, the system was trapped indoors and was heavily reliant on a high-speed camera system, and all the computing was done on the base station. The PPM signals for throttle, pitch, roll, and yaw were then transferred through an FPGA and a stock wireless controller to the quadcopter. They were then translated by the GU 344 into PWMs and sent to the ESCs and motors. This severely limits mobility and reduces the potential for real-world applications. The mission we decided to implement was to convert the system from being completely controlled by an external computer and camera system, to one that is almost entirely on-board the vehicle. This required using bluetooth to communicate between a base station and the on-board FPGA to give directions for the quadcopter to follow. The FPGA would then use appropriate sensors to keep the quadcopter in a stable hover or move to the desired location. These changes will allow us to fly outside and demonstrate some autonomous functionality.

2 Specifications

2.1 Users and Uses

The design of our project is intended to be used for demonstrations of embedded systems and controls integration on an autonomous flying vehicle. It also adds to the continued research and development of autonomous flight vehicles. The intention is to continue improving its ability to fly with the least amount of human interaction as possible.

For this reason, the intended users for the initial setup and understanding are those trained for the system. The interface requires the ability to program the FPGA board from the code design computer, debugging and fixing the quadcopter code and hardware when something doesn't go accordingly, and a list of commands currently used for flight. After the quad flies stably and the initial setup is complete the average American population should be able to use the GUI to control the quadcopter.

2.2 Operation Environment

The goal for the project is to enable the quad-helicopter to fly autonomously in a pseudo-outdoor environment that is very different from where the quad-helicopter currently flies for all of its testing. The test environment will be located indoors where wind is negligible and the climate is controlled. Tests will take place in a three dimensional grid defined by the IR camera system that is about 27 m^3 , with a thin carpet landing surface. It will be able to operate in temperatures between 40^oF to 90^oF; the operating environment will contain no obstacles. The only obstacles for it to not fly completely outdoors is the pseudo-GPS and a lack of account for wind speeds.

2.3 System Requirements

Functional Requirements
The quadcopter has many functional abilities and interactions. The quadcopter can be controlled manually by two different manual controllers, a RX controller and a USB controller connected to the base station. To use the RX controller you would just disconnect the FPGA board. To fly with the USB controller you would click on the manual control box located on the GUI shown in Figure 2.

The quad also has a working PID control system, a new motor signal-mixer, and a bluetooth communication system. The bluetooth communication allows the user to send information from the base station computer to the board mounted on the quadcopter. The

PID system keeps the quadcopter in a stable hover when a throttle is inputted. The motor signal-mixer is the replacement for the GU 344 and translates the PWMs into an input the ESCs can read.

The quadcopter is also able to use the old GUI and flight system from 2013. In order to use this functionality, you would need to remove the bumpers, legs, and FPGA board from the quad. You would also need to reconnect the GU 344 to the proper motors and connect the RX controller to an FPGA board located at the current base station. Once those are connected, you bring up the old GUI and turn on the camera system to fly the quad. You can use the drawing function to fly the quad in a specific formation.

Non-Functional Requirements

An important non-functional requirement we implemented is an easy way to shut down the motors as a safety precaution for those observing and flying the quadcopter. Safety, while not specifically required for the project to work, is a good non-functional requirement to have in order to make the project usable and demonstrable in a normal setting. Another non-functional requirement is to have an easy to use GUI. The system redesign doesn't need a GUI at all, but providing one will allow users with less experience to control the quadcopter more easily.

3 Design Description

3.1 Design Overview

The system design we created is located in Figure 1. The system starts at the camera system which locates the position of the quad in a pre-defined cartesian system. The information is sent to the base station and is relayed, along with commands, to the LABET board via bluetooth.

The PID controller on the board takes in the current position and new position command and adjusts the orientation to make the current position equal to the position command. Orientation must be updated more frequently than position to prevent crashing, instability, and improper positioning. The current orientation is input from the IMU and processed by a filter since the gyroscope and accelerometer have some error. The PID sets an orientation to make the quad change its position. The set orientation changes based on how far the quad is from its new position.

The values outputted by the PID are used to adjust the current values of throttle, pitch, roll, and yaw. These signals input to the motor signal-mixer and adjust the value for each motor. The set value for each motor changes the duty-cycle of the PWM for that motor. The ESC receives the PWM and changes the RPM of the motor.

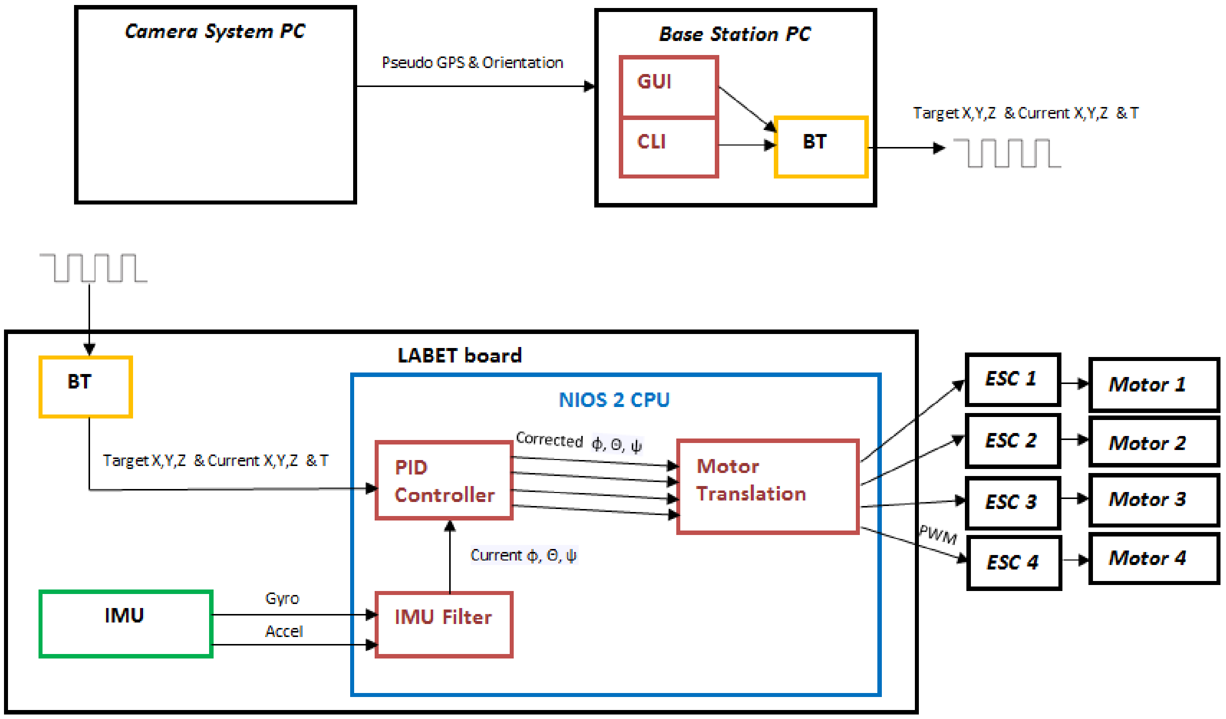
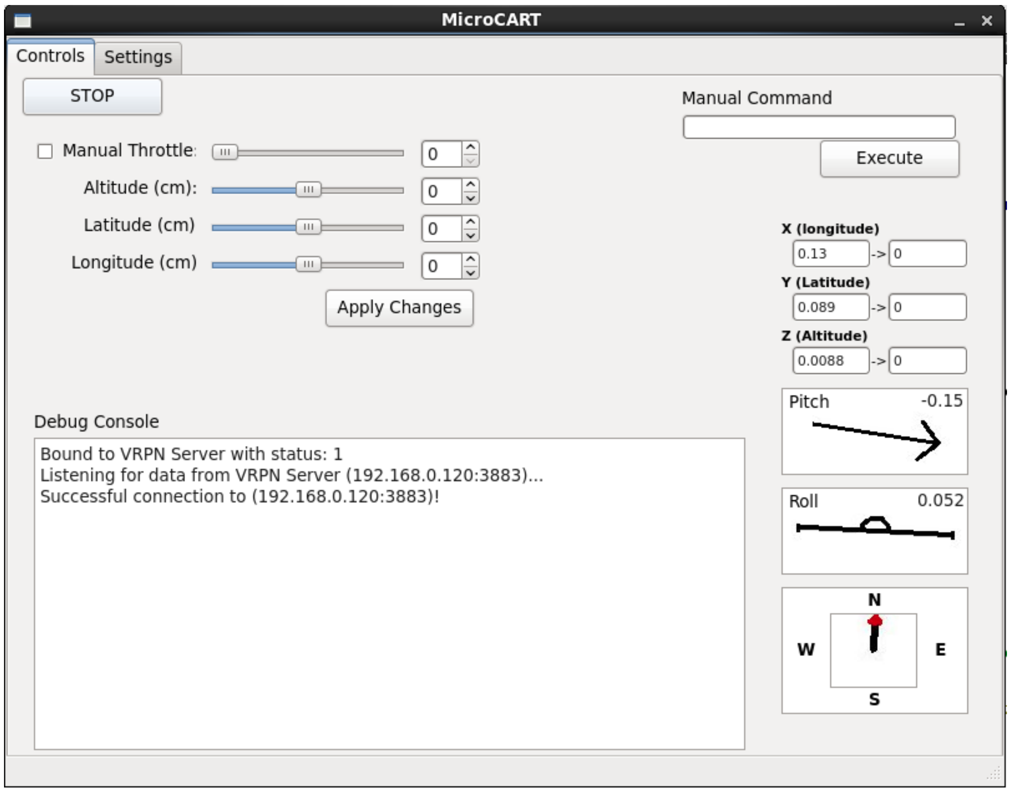


Figure 1: Overall system diagram

3.2 GUI

The GUI is the primary interface to control the quadcopter. The GUI interfaces with a more basic command line underneath, but beyond the ability to enter manual commands into a textbox in the GUI, this is completely hidden from the user. This is intentional. All of the quadcopters basic functionality, such as adjusting throttle or target positions, can be done in the user interface using the sliders as shown in Figure 2. The user interface also displays the quadcopters current orientation and position data in real time to the user. This GUI interfaces to the quadcopter completely through bluetooth, whether using manual or automatic control.



Manual Throttle: 0
Altitude (cm): 0
Latitude (cm) 0
Longitude (cm) 0

X (longitude) 0.13 -> 0
Y (Latitude) 0.089 -> 0
Z (Altitude) 0.0088 -> 0
Pitch -0.15
Roll 0.052

```
Bound to VRPN Server with status: 1
Listening for data from VRPN Server (192.168.0.120:3883)...
Successful connection to (192.168.0.120:3883)!
```

Figure 2: GUI interface with sliders and Pitch and Roll indicators

3.3 Bluetooth

Bluetooth communication is our means of communicating with the quad from the base station. After observing broken packets and odd behavior, we implemented start bytes to ensure the data does not get off the mark on a missed byte. We later implemented a cyclical redundancy check on the board to ensure it received exactly what was sent by the basestation.

3.4 IMU

The IMU contains a combination of sensors (accelerometer, gyroscope). These are used to get the orientation of the quad (pitch, roll). Unfortunately, the data from these sensors has its faults: the accelerometer is noisy, the gyroscope drifts. Thus, it is necessary to use a filter which combines the two sensors to get a more accurate reading.

3.5 PID Controller

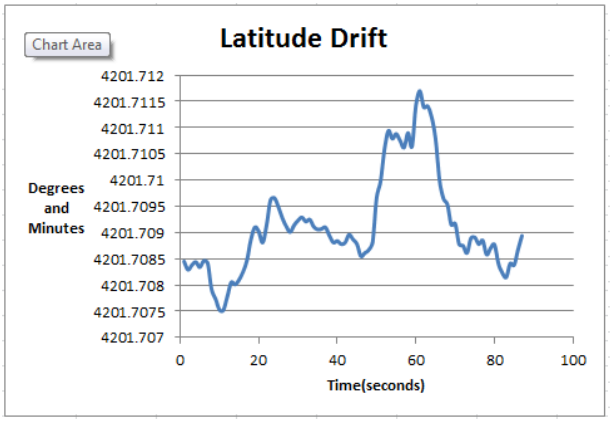
For the quadcopter to be able to maintain a stable hover, a controller is needed to keep it in the proper orientation. The quadcopter uses a PID controller that is implemented on the onboard FPGA board and interacts with the IMU and the information from the bluetooth.

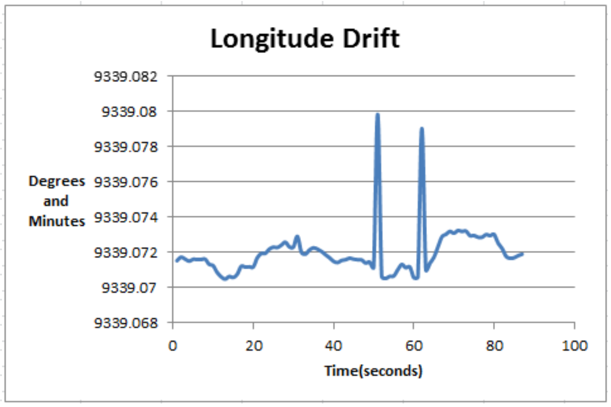
The PID starts with a series of constants that are determined based on the mass and dimensions of the quadcopter. This is changed and created by tuning them using the current code and seeing which way the quad wants to drift, and by using a simple CAD design within MATLAB.

The PID also needs information from the IMU in order to determine the orientation and position. The target position and current position is taken in by bluetooth. From this information the error amount of the current position versus the target position is calculated. The farther it is off, the larger the error it creates. These errors are then used to determine what changes need to be made to the motors. This is calculated within the motor translator.

3.6 Pseudo-GPS

Our GPS module proved to be very imprecise; the output position would drift quite significantly. Since this limitation is hard to use for testing purposes, we use a type of pseudo-GPS. This pseudo-gps is provided via a high speed camera system. It is used to send exact position data over the bluetooth controller to be used by the PID controller.





Figures 3 and 4: Latitude drift and longitude drift of the GPS

3.7 Filters and Mixers

Kalman Filter

The Kalman filter is a real-time algorithm that calculates a very good estimation of the orientation of the quadcopter based on accelerometer and gyroscope data. Since the gyroscope tends to drift over time and the accelerometer is inherently noisy, neither one of these sensors will provide a very good orientation when used alone. The Kalman filter is able to mitigate the weaknesses of each sensor and combine the two to provide a very good estimation of the orientation.

The algorithm used by the Kalman filter starts with a prediction of the quad’s state variables and uncertainties for the next point in time. Then a comparison is made between the prediction and an actual measurement at the time point, and an estimation of the state variables and uncertainties is computed using a weighted average filter. This estimation is is then used as the input and prediction for next measurement and continues on indefinitely. Proper weights of each sensor’s certainty, along with other system variables, were defined through experimentation.

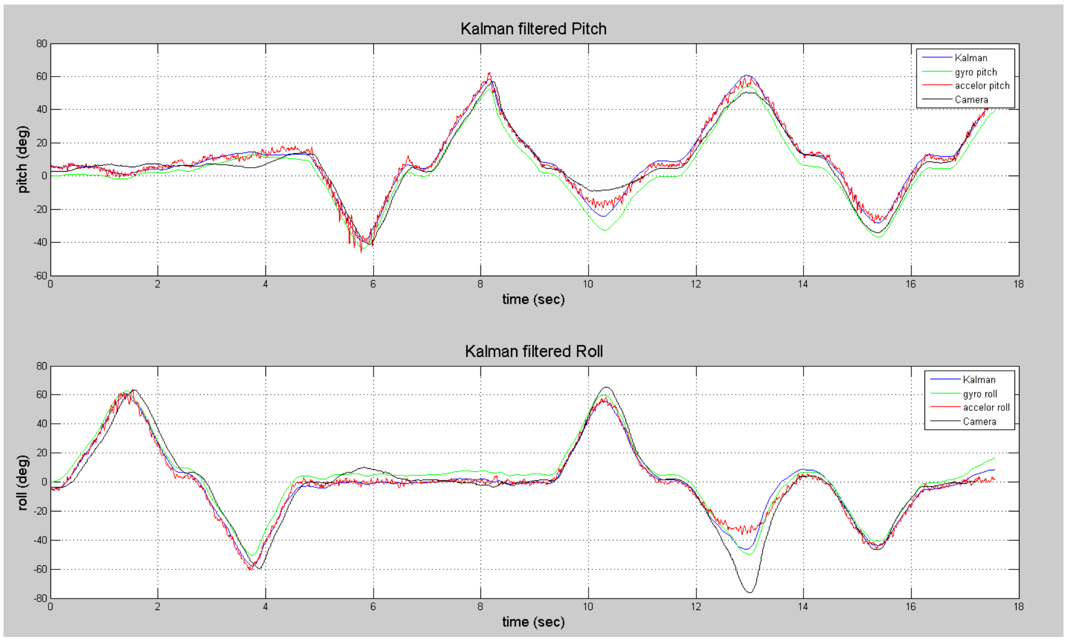


Figure 5: Kalman filter averaging of accelerometer and gyroscope data

Lightweight Filter

This filter is based on the Kalman filter, but involves less math and usually termed a "good enough" filter. Orientation can determined based on accelerometer or gyroscope data. Similar to the Kalman filter, the lightweight filter combines the data from the two sensors to get a more accurate reading as seen in Figure 2 and described below.

Starting with the gyroscope, data is input as degrees/second and by measuring over a period of time, the angle of change about an axis can be determined in degrees. The gyroscope approximation still needs a starting point, which comes from the accelerometer for only the first reading. The overall orientation about any axis for gyroscope reading is then initial accelerometer angle plus the summation of degrees changed from gyroscope at each measurement. The angles are then converted into a unit vector.

The measurements from accelerometer don't rely on previous measurements. A vector is created from the accelerometer data which measures gravity. The two vectors are combined by weighting them together. The more accurate of the two having a larger weight. The vector can then be converted back into angles for orientation.

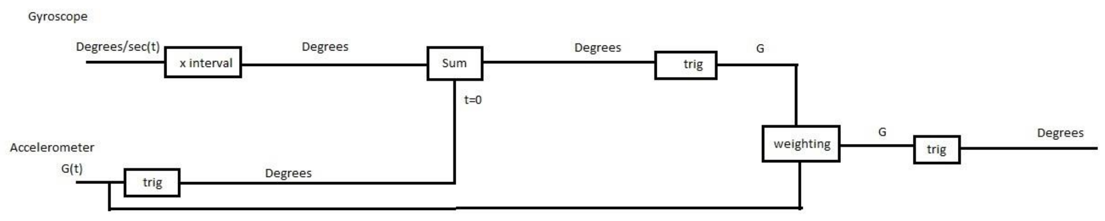


Figure 6: Lightweight filter diagram

Motor Signal-Mixer

To control the quad the same way the GU344 did, motor signal-mixer takes in flight controls of throttle, pitch, roll, and yaw and adjusts the signals for each motor. The flight controls are set as percentages from 0 to 100%. Throttle starts at 0%, whereas pitch, roll, and yaw all start at 50% to prevent rotation about an axis. Going down to 0% will cause movement in one direction and 100% is movement in the opposite direction.

The motor output is also set as a percentage from 0 to 100%. 0% represents the baseline duty cycle for the PWM to the ESC and the motors being off. 100% is the maximum duty cycle that has been set and is the fastest the motors will spin.

All changes in pitch, roll, and yaw require the quad to remain at the same altitude, so the mixer takes in throttle exactly as it is and spins some motors faster and other slower than throttle to achieve the command. As for the other flight controls, their values must be adjusted in the mixer. 50% means no movement for them, so 50 is subtracted from every one of those commands. The output for each motor is then

Throttle ±(Pitch-50)±(Roll-50)±(Yaw-50)

Each flight control that requires rotation is either added or subtracted based on the location of the motor on the quad.

4 Design Implementation

4.1 Bluetooth Manual Control

After the USB controller data has been parsed by the base station, it is sent over bluetooth to the LABET board on the quad. The board uses start bytes and cyclic redundancy checks to ensure the data has successfully been transferred. The data is formatted serially, and includes both the quads current position and orientation as well as the user's target position and orientation. With manual control over bluetooth, the data packet has a flag set that is read by the LABET board. If this flag, called "MANUAL_CONTROL" is set, the LABET board will ignore any automatic flight controls and will just pass all target values from the packet to the motors directly. These values are put into hardware functions which, given an input of 0-20,000, create four unique PWM signals. These signals correspond to throttle, aileron, elevator, and rudder.

The signals are passed to the GU-344 which translates the four signals into motor commands which the ESCs can understand. This was implemented as a proof of concept, namely that the bluetooth communication would be fast enough to have stable flight. Since the bluetooh communication had a rather slow update rate flight was not perfectly stable, but was responsive enough to prove the concept.

4.2 Hover Mode

Hover mode incorporates nearly all of the components we have designed throughout the year. When started, the quad takes it's current latitude and longitude which is sent over bluetooth from the pseudo-GPS camera system and sets that as it's start position. It then begins polling the IMU for accelerometer and gyroscope data. Using these values, the lightweight filter estimates roll and pitch angles. These are updated ~100 times per second, and are inserted to the PID controller for processing.

The PID controller then determines how much error is currently in the system and outputs accordingly. It continues this cycle of orientation adjustments in a loop, interrupted only by the occasional bluetooth packet. These packets contain new values of altitude, latitude, longitude, as well as commands for new target positions all of which are also put into the PID controller. The output generated is then taken by the motor translation code where it is converted from throttle, aileron, elevator, and rudder commands to values specific to each individual motor. These values are passed into the hardware functions which generate the PWM signals to be read by the motors.

5 Testing

5.1 Ground Testing

Motor and ESC

One of the first tests we had to do was test if the motors and the ESCs controlling the motors were functioning properly and efficiently. In order to do this test we had to attach a piece of IR reflecting tape to the motor. Since we did not have any IR tape available, we used aluminum foil and transparent tape to create a makeshift IR reflection. We then connected a DC power supply of 10 volts to the battery connection of the ESC to power it. We then used the function generator with a 3.2 Volt peak to peak square wave with an offset of 1.6 Volts and a frequency of 280 Hertz. The duty cycle was 28%.

If we had to set limits for the motors we needed to set the upper limit with a duty cycle around 45% and no higher than 50%. We turned on the power for the motor and once it beeped we lowered the duty cycle using a tachometer. Continuing on whether we had limits or not, we now observed the rotation speed with respect to duty cycle using the earlier tachometer.

PPM Outputs

Another important test we had to create was to test the PPM outputs to from the base station to properly figure out what is being outputted to the GU344. We turned on the oscilloscope located in the lab and attached the positive and negative hooks on the PMOD attachment on the Digilent board. We then adjusted the voltage and time scales to obtain a readable image of the PPMs. We then ran the old team's GUI or the CLI to send commands from the base station to the quadcopter. We then observed the commands being sent from the station as the PPM signals changed. The order for the signals was roll, pitch, yaw, and throttle from left to right.

PWM Outputs

One of the tests we would use in other different testing environments was looking at the PWM signal outputs from the LABET board on the quadcopter. We turned on the oscilloscope and attached the negative probe to the ground and the positive probe to one of the four output pins located on the board. We then would run the code that we wanted to observe the PPM signals from the LABET board. W would also have the GUI or CLI run to get the appropriate signals. If it was running from camerasystem.c or bluetoothcontroller.c for the main function, the outputs are based on the GUI/CLI. If we were running the program hello_world.c as the main function we would be looking at changes based on orientation of the quad and the GUI/CLI.

PID Corrections

To test if the PID corrections were accurate before a flight test, we would use the old outputs for the PID and compare them to the new values we received. We would output the values from the old system into a log for comparison. We also outputted out own values in the command line so that we could easily grab the values for comparison. We had timestamps attached to assure that they were correcting at roughly the same time. We also had a small "fake" quadcopter that was the same shape for the IR sensors that we would use to simulate quad behavior.

IMU Data

To test the IMU data, we would output them into the console of the Quartus software we were using for our programming. We would separate the accelerometer data and gyroscope data and do the appropriate mathematics before outputting the data. We found from these tests that the gyroscope likes to shift over time as expected and using this method we can find the new trim values to use on the quad.

Bluetooth Data Packets

We would sometimes receive bad packets or the packets would give us a CRC. We would test the Bluetooth for many different problems. We would first look to see if they had connected to one another by looking at the modules and seeing if the green light on each of them was solid and not blinking. The next test we would do is if the packets are the same size and if the receiver is getting all the data packets. The final test was if the Bluetooth packets would give the appropriate information or if they were sending gibberish. Based on the answer to those three tests, we could determine where the problem was located.

Motor Outputs

One of the final tests we initiated was to see if the motors were giving an appropriate output value for the PID function in the quadcopter's system. To do this we had a print statement in the mixer to print out the values that the motors were getting. We used the "fake quadcopter to check if the motors where given the appropriate output based on the quadcopter's position.

5.2 Flight Testing

We had a couple of different testing methods for flight testing. The different flight testing programs we would use are hello_world.c, camerasystem.c, and bluetoothcontroller.c. Each had their own testing reasons and uses. They will be described below.

hello_world.c LABET Board

We use this as the overall system and our final implementation. This system was the outdoor system test program with Pseudo-GPS. We would first program the LABET board with the board's hardware and run hello_world.c as hardware programming. The board was already attached to the quad, but if it's not the blue side would have to match the blue motors. We then wired the board's servos directly to the ESCs because we had our motor signal-mixer located in this program to override the GU 344.

The most important part of these tests is to tether the quad to the ground so that it doesn't injure people, the surrounding area, or itself. We then plugged the quad into its power supply to activate the motors and the ESCs. The GUI was then brought up and we slowly increase the throttle until liftoff. Based on how steady the liftoff is, we had to correct the PIDs or trim values.

bluetoothcontroller.c LABET Board

We use this as the second test system for our quadcopter implementation. This is the system that runs the quadcopter based off of a USB controller connected to the base station PC. We would first program the LABET board with the board's hardware and run bluetoothcontroller.c as hardware programming. The board was already attached to the quad, but if it's not the blue side would have to match the blue motors. We then wired the board's servos directly to the ESCs because we had our motor signal-mixer located in this program to override the GU 344.

The most important part of these tests is to tether the quad to the ground so that it doesn't injure people, the surrounding area, or itself. We then plugged the quad into its power supply to activate the motors and the ESCs. The GUI was then brought up and we slowly increase the throttle until liftoff. Based on how steady the liftoff is, we had to correct the trim values.

camerasystem.c LABET Board

We use this as a simple tester for our code still using the GU-344. We would first program the LABET board with the board's hardware and run hello_world.c as hardware programming. The board was already attached to the quad, but if it's not the blue side would have to match the blue motors. We then wired the board's servos directly to the GU-344.

The most important part of these tests is to tether the quad to the ground so that it doesn't injure people, the surrounding area, or itself. We then plugged the quad into its power supply to activate the motors, the GU-344, and the ESCs. The GUI was then brought up and we slowly increase the throttle until liftoff. Based on how steady the liftoff is, we had to correct the gain on the GU-344.

6 Closing Statements

6.1 What Went Well

The 2014 MicroCART team was able to implement and debug nearly every component of the goal system. We discovered lots of new issues, many related to hardware in the current implementation. Moving forward with this knowledge, future teams should have a much easier time. We successfully bypassed the stock motor translator (GU-344) which had been on MicroCART's to-do list for some time.

6.2 What Didn't Go Well

The lack of a system level coordinator hurt our projects ability to come together in the final few months. While most of us gained lots of knowledge and depth in our specific area, things didn't integrate until the last few weeks of the semester. This meant we did not have enough time for flight testing and PID tuning.

Due to the scope of the project, the ramp-up time for our inexperienced group was far too high. Because of this, tangible and testable components of the system were not completed until it was very late in the semester.

6.3 Results and Deliverables

While a fully integrated and functioning system was not attained, this year's MicroCART team successfully implemented almost all of the required components:

- Bluetooth communication was new this year, and came with a lot of issues that have been hammered out
- Implemented two filters both of which have been tested extensively and function as expected:
    - A heavy and precise Kalman filter in both Matlab and C
    - A lightweight "good enough" filter in C
- Wrote and tested code to utilize a GPS module
- Wrote motor translation code to get rid of uncertainty caused by the GU-344
- Implemented a multiplexer on the throttle signal for safety concerns
- Created a new flight-ready system using a USB controller and bluetooth communication
- Root-caused hardware issues including an incorrect clock on the board and electrical shorting problems

6.4 Closing Summary

MicroCART was a very challenging and rewarding senior design team to be a part of. We each put in a couple hundred hours and wound up with some of the most important deliverables we set our sights for. Had the hardware cooperated, and we had better communication regarding work integration between team members, I am very sure we could have had a very robust system. None the less, we have left all of the components completed for next year’s team to build from, and hopefully they can work out all the kinks for a completely integrated system.