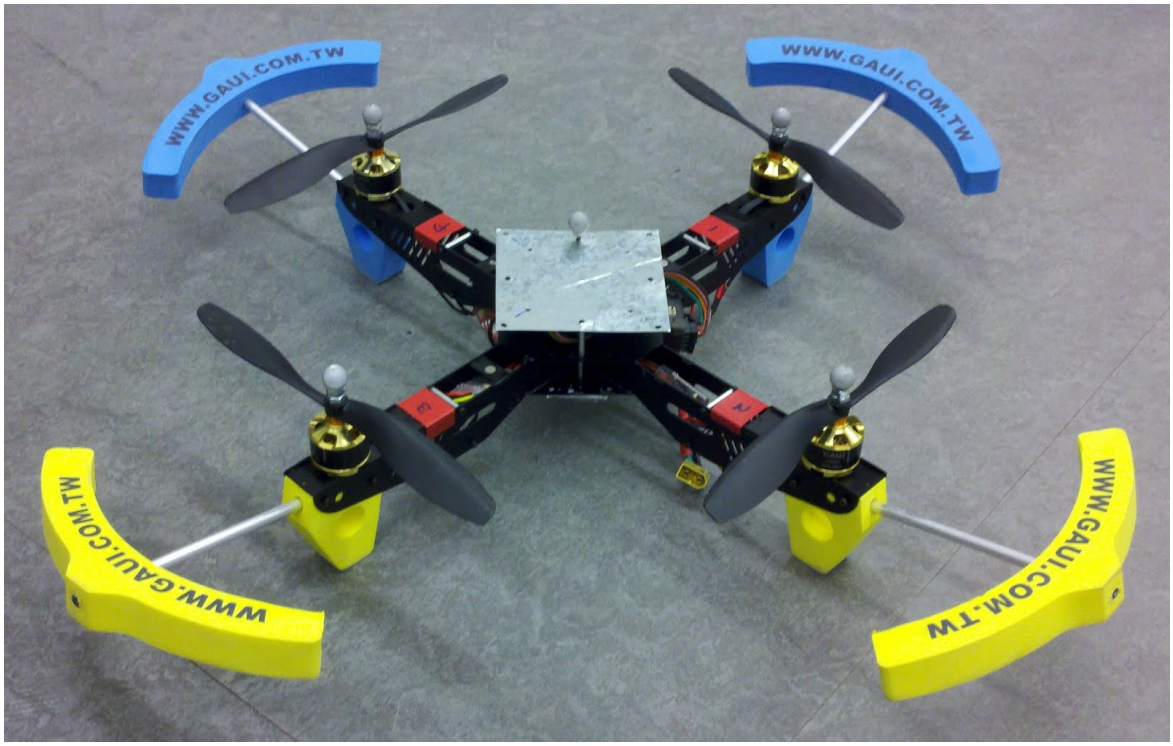
Iowa State University

MicroCART Final Design Document
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 lightweight GUI for sending commands. We also took out the GAUI 344 and replaced it with a simple motor controller.

Table of Contents

Ground Testing

Flight Testing

Timeline

Closing Statements

What went well

What didn’t go well

Results and Deliverables

Closing Summary

General Definitions

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.

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:

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)

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.

Specifications

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.

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^{o}F to 90^{o}F; 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.

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.

Design Description

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.

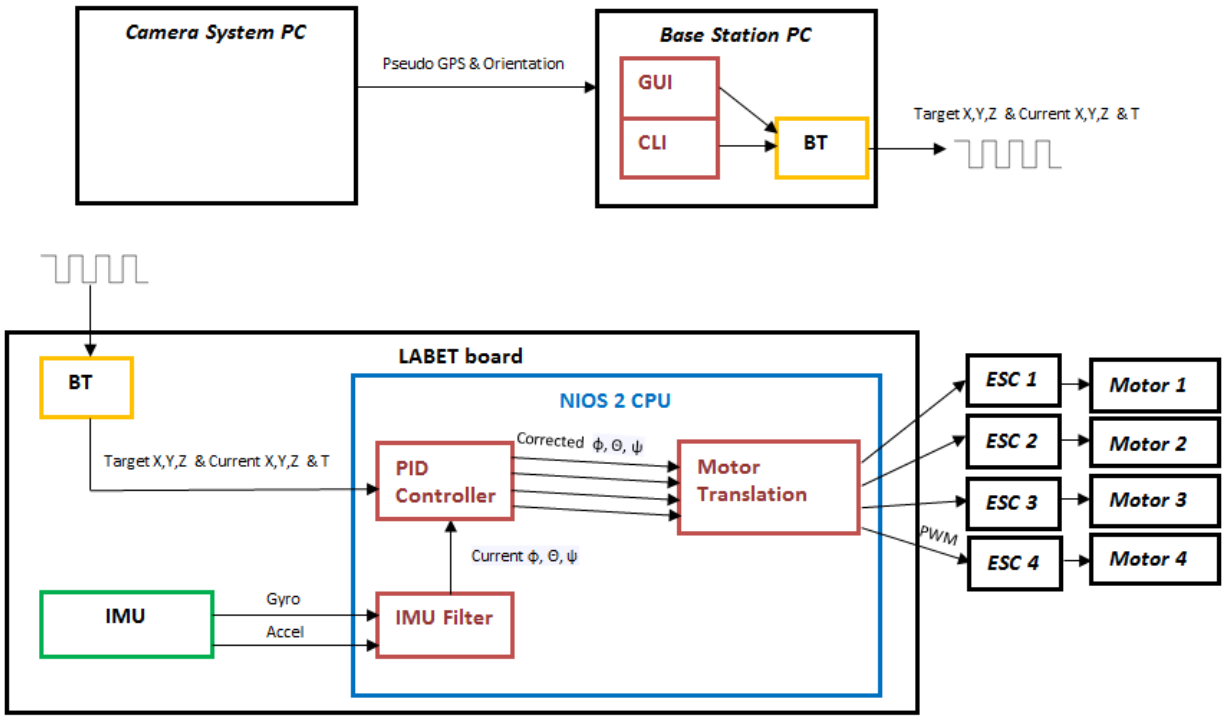


Figure 1: Overall system diagram

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.

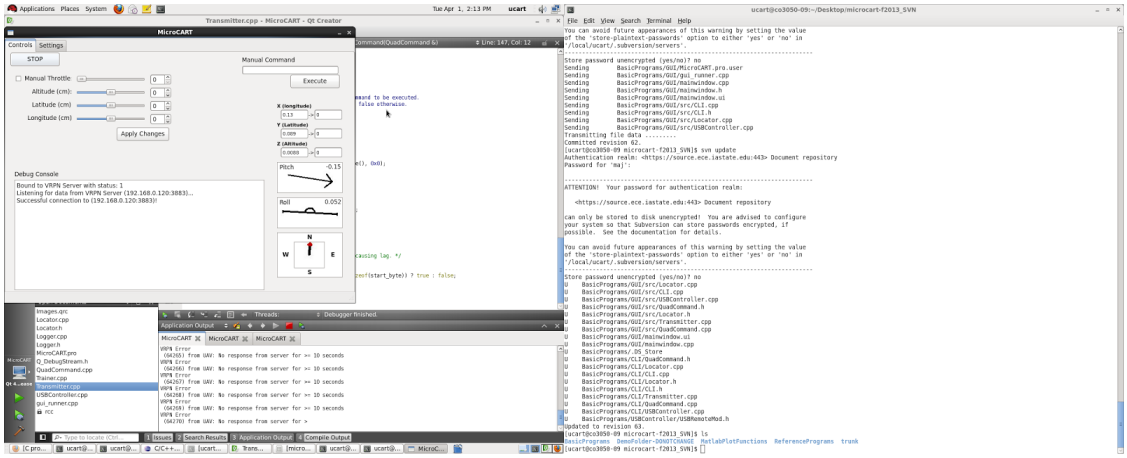


Figure 2: GUI interface with sliders and Pitch and Roll indicators

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.

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.

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.

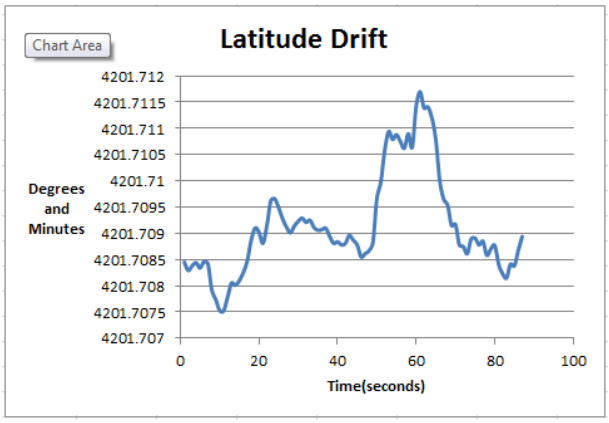
The PID starts with a series of constants that are determined based on the mass and dimensions of the quadcopter. This is changed and created by tuning them using the current code and seeing which way the quad wants to drift, and by using a simple CAD design within MATLAB.

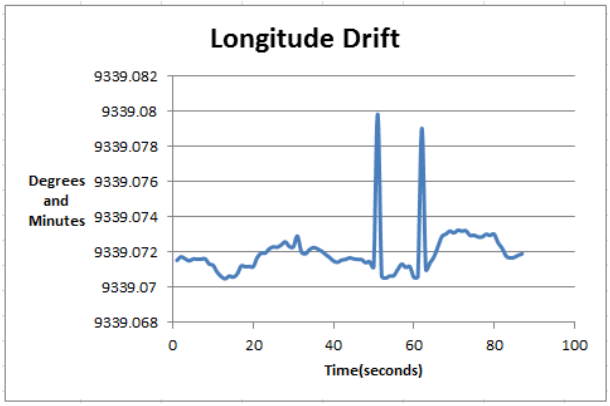
The PID also needs information from the IMU in order to determine the orientation and position. The target position and current position is taken in by

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.

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.





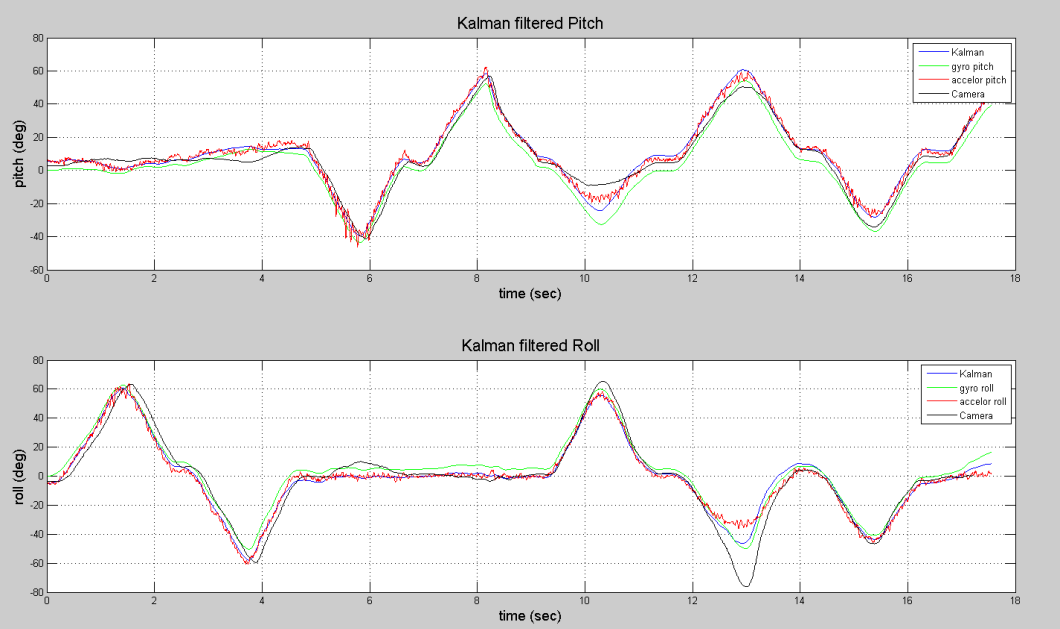
Filters and Translations

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.



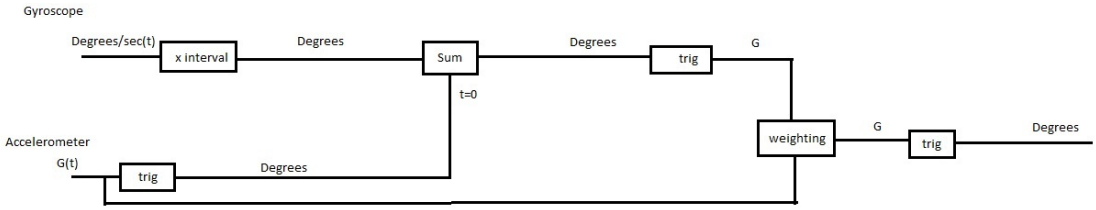
Kalman filter averaging of accelerometer and gyroscope data

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.



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.

Design Implementation

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. Because of the rather slow update rate flight was not perfectly stable, but was responsive enough to prove the concept.

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.

Testing

Ground Testing

1.) Testing of motor and ESC performance:

i) Attach IR reflecting tape to the motor, or in the case that no IR tape is available: aluminum foil and transparent tape may also work.

ii) Connect a DC power supply of 10 volts, or whatever your battery supplies, to the battery connection of the ESC.

iii) The function generator will require 3.2 Volts, peak to peak with an offset of 1.6 Volts (or 1.6 Volts amplitude), square wave, with 280 hertz to the ESC inputs. Set the duty cycle to 28%. If setting limits for the motors, move to step iiia, else move to step iv.

a)Set the upper limit with a duty cycle around 45%, but no higher than 50%. Turn on the power to the motor.

b) Once the motor beeps, lower the duty cycle to the lower limit requested.

iv) Observe the rotation speed with respect to duty cycle using a tachometer.

2.) Testing of the PPM outputs from base station computer

i) Turn on oscilloscope and attach positive and negative probes to positive and negative hooks on the PMOD attachment to the Digilent board; adjust the voltage and time scales to obtain a readable image of the PPMs.

ii) Run the GUI or the CLI to send commands from the base station

iii) Verify the commands have changed the PPM values accordingly. Order should be roll, pitch, yaw, throttle

3.) Testing of the PWM outputs from LABET board on quad

i) Turn on oscilloscope and attach negative probe to ground and positive probe to one of the four output pins.

ii) Build and run the code for the LABET board.

iii) Run the GUI or the CLI to send commands from the base station to the LABET.

a) If running camerasystem.c or bluetoothcontroller.c as the main function, outputs are based on GUI/CLI.

b) If running hello_world.c as the main function, outputs will also change based on orientation of the quad.

iv) Based on the code being run, either PWMs for flight controls or for motor controls will be outputted.

Flight Testing

1.) hello_world.c

i) Program the LABET board's hardware and run hello_world.c

ii) Attach the board to the quad with the blue side matching the blue motors.

iii) Wire the board's servos directly to the ESCs

iv) Tether the quad to the ground for safety

v) Plug in the quad

vi) Run the basestation GUI with manual throttle enabled

vii) Slowly increase throttle until liftoff

viii) Adjust pid constants until it is stable

2.) bluetoothcontroller.c

i) Program the LABET board's hardware and run bluetoothcontroller.c

ii) Attach the board to the quad with the blue side matching the blue motors.

iii) Wire the board's servos to the GU-344

iv) Tether the quad to the ground for safety

v) Plug in the quad

vi) Run the basestation GUI with manual control and manual throttle enabled

vii) Slowly increase throttle until liftoff

viii) Adjust gain on the GU-344 until it is flyable

3.) camerasystem.c

i) Program the LABET board's hardware and run camerasystem.c

ii) Attach the board to the quad with the blue side matching the blue motors.

iii) Wire the board's servos to the GU-344

iv) Tether the quad to the ground for safety

v) Plug in the quad

vi) Run the base station GUI with manual throttle enabled

vii) Slowly increase throttle until liftoff

viii) Adjust gain on the GU-344

Timeline

September

- Discussing with advisors to determine overview of project
- Basic flight training with small quadcopters

October

- Acquaintance with software tools (reference guides)
  - Eclipse
  - Quartus
  - QT Creator
- Acquaintance with hardware (datasheets, schematics, wiki)
  - IMU
  - bluetooth
  - LABET board
  - ESC's and motors
  - GU-344

November

- Research core components
  - Bluetooth packet reliability
  - PID controllers

    - IMU data filtering
    - VRPN (pseudo gps protocol)
    - Motor translation
    - GUI

December

- Formulate system design

January

- Implementation of core components

February

- Unit testing of core components

March

- System integration of core components

April

- System integration testing
- Documentation

Closing Statements

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.

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.

Results and Deliverables

While a fully integrated and functioning system was not attained, this years 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

Closing Summary

MicroCART was a very challenging and rewarding senior design team to be a part of. We 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 a bit better, 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 years team to build from, and hopefully they can work out all the kinks for a completely integrated system.