MicroCART 2017-2018
Final Report

Team # 17

Clients / Advisors:
Dr. Phillip Jones
Dr. Nicola Elia

Team Members:
Jakub Hladik - Test Lead
Tyler Imboden - Quad Software Lead
Matthew Kelly - Webmaster and Documentation Lead
Dane Larson - Ground Station Lead
Blake Pries - Communications Lead
Austin Rohlfing - Controls Lead
Peter Thedens - Repository Lead
Kyle Trost - Team Lead

Team Website: **http://sdmay18-17.sd.ece.iastate.edu**
Team Email: sdmay18-17@iastate.edu

Revised: April 22, 2018

Table of Contents

List of Figures

Definitions

Term	Definition
CLI	Command line interface
Continuous Integration	automated process of running tests on every commit to the repository
Demo	Short for demonstration; this is one of the deliverables of the project: a demonstration of the quad's capabilities, for example, doing a backflip with the quad, finding an object and following it, communicating with a second quad to perform flight patterns
FPGA	Field Programmable Gate Array; this is a board that can be programmed to simulate electrical hardware components. Used often in reconfigurable computing
Ground Station	The application that runs on a host computer that communicates with the quad via a Wi-Fi connection and sends it coordinates to the quad
GUI	Graphical user interface
IR	Infrared wavelengths of light longer than visible light; used in the VRPN system to determine the position of the quad
LIDAR	Light Detection and Ranging; this is a system for determining the altitude (z) of the quad using the onboard sensor
Optical Flow	system using pattern of motion of objects, surfaces, and edges caused by the relative motion between the and the scene to determine position; used by the quad to calculate position (x, y) when not in the lab using the VRPN system
PID	Proportional-integral-derivative control system; standard control algorithm used on the quad
Quad	Short for quadcopter; this is the hardware platform we use in this project
Setpoint	In a control system, the target value for an essential variable
VRPN	Virtual-Reality Peripheral Network; this is the system used to determine the position (x, y, z) and orientation (ϕ,θ,ψ) of the quad in the lab using a set of 12 stationary cameras and an IR transmitter on the quad

1 Introduction

Microprocessor Controlled Aerial Robotics Team or MicroCART project is centered around the development of a quadcopter (see Figure 1 below) and tracking system. This project has been in development since 1998 and the current system has been passed down since 2006. The project aims to create a stable and easy to use platform for researching control theory. The quadcopter flies primarily in the Distributed Sensing and Decision Making Laboratory within a twelve camera infrared tracking system.

Figure 1-1: MicroCART quadcopter

1.1 Acknowledgement

MicroCART is a project that is assisted by graduate students working in controls under Dr. Nicola Elia and Dr. Phillip Jones. Additionally, as this is an ongoing project previous team members will also be providing help in understanding the current system. As such, we would like to acknowledge the assistance that will be provided by:

- Matthew Cauwels
- Robert Buckley
- Matt Rich
- Dr. Phillip Jones
- Dr. Nicola Elia
- May 17-16 MicroCART Team Members

1.2 Problem and Project Statement

We expanded upon an existing platform that is used for research in controls and embedded systems and for departmental demos. The platform was expanded by adding and improving functionality for more multiple quads or generic trackables, expanded controls, and simulation tools.

1.2.1 Problem

The MicroCART platform designed in previous years had many flaws that hindered its use for research and demo purposes. The previous platform failed to familiarize the user(s) of the system in a time horizon that would make it viable for research. This is because the system did not have ample documentation available for the users of the system to learn about the platform and its uses. From the viewpoint of a user running demos the area within the VRPN system the platform as it stood is most stable when confined to flying within the VRPN system as the optical flow navigation can not hold position during flight. This means that the areas available to the quadcopter for demos can only utilize the small amount of space in the lab for a demo. Lastly, the quadcopter is controlled using a PID controller that requires logical guessing and checking to tune, we now have a new linear controller that can be computed faster and be tuned around multiple points on the non linear model.

1.2.2 Purpose, Goals, and Approach

The MicroCART senior design team serves several purposes. One purpose of the team was to improve on the existing quadcopter design in order to give graduate students a more stable platform to use for research and testing of control theory. Another purpose is to showcase the skills that a student in the ECpE department can gain throughout their time at Iowa State by creating an impressive demo that the quad can perform. The quad has also become more reliable so that anybody with little knowledge of the project should be able to read some documentation and feel comfortable performing the demo. We built upon the previous MicroCART team's platform by improving the stabilization, designing new demos, updating the ground station GUI, and building upon the virtual quadcopter software.

Our approach to the various projects started with becoming familiar with the current system. This included updating documentation, reading existing documentation, and running previous demos. Upon gaining enough knowledge we started building upon the existing platform to add demos and functionality that is currently capable. We then built an improved second version of the quadcopter. Lastly, the testing team will be finishing implementation of the virtual quadcopter to support full flight. During this whole period we demoed new features to show our clients and advisors progress and get feedback regarding functionality of the platform.

1.3 Operational Environment

In order to fly using the VRPN software for position and orientation data, the quad must be inside of a small area (less than 10 m^2) inside of Coover 3050. This lab is designed to cause very few environmental impacts on the quadcopter. Through the use of ventilation, window shades, and Coover's heating and air conditioning, the lab has a nearly constant light and temperature with little to no accumulated dust to affect air quality.

Additionally, the project relies on two Linux computers. One is used to control the ground station software and the other contains build tools for the FPGA on the quadcopter itself. This provides a platform for development that is consistent across team members and easier to demo as there are not issues with building or ensuring correctness of the various communication aspects used for the project.

1.4 Intended Users and uses

The primary set of end users is composed of future MicroCART members and controls graduate students at Iowa State. For the goal of creating demonstrations for prospective students, someone from the two categories (the user) will be running the demo for them (the audience). This means that the users can be assumed to have competence in using multiple forms of programs (for example, either GUI or CLI) and in reading general technical documentation.

The other goal listed above regards the modular implementation of new control algorithms as a research opportunity for graduate students. These users have three primary needs from our product. The first is a robust and reliable system to decrease variation in test results. This includes having sturdy quad hardware, low communication latency, and a bug-free user interface. The second need is to have modular software with complete documentation to allow for them to achieve the implementation themselves, without the need for significant system rework or intervention of the MicroCART design team. Finally, these students will need the data from the system identification in order to form their models. This includes information about mass, moments of inertia, motor resistances, rotor areas, and many other properties that determine the true actuation of the quadcopter.

1.5 Assumptions and Limitations

1.5.1 Assumptions

- Our VRPN camera system as it exists provides sufficiently accurate position data.
- The quad will flies without significant external disturbances.

1.5.2 Limitations

- Area limitation for multiple quadcopters within the VRPN camera system.
- Accuracy of onboard sensors (e.g. optical flow, LIDAR, IMU, GPS).
- Latency and range of the wireless link between the quadcopter and the ground station.

1.6 End Product and Deliverables

The quadcopter system consists of three major subsections: the quadcopter software, the ground station, and the control systems. Each of the subsections is essential to meet the desired objectives and fulfill requirements. Documentation and demos are also a major deliverable for our project and will be discussed.

1.6.1 Quad Software

1.6.1.1 Vivado Upgrade Testing

The entire quad software platform relies on the Xilinx toolchain. The software we were using to develop the Quad software is the Xilinx toolchain. Specifically XPS, which is known to be a very non-user friendly and dated piece of software. One of the major goals of this project was to transition to the new and improved Vivado to program the FPGA hardware.

1.6.1.2 Second Quad

Our client advisor requested that we develop a second quad for available flight. The difficulties in building a second quad revolve around the lack of documentation of parts used on the quad, along

with the availability of the previous generation of parts. The new quad needed to be able to run the same software on a different set of hardware.

Kit Includes (Currently Model # F450 ARF Kit V1) FW665827020:	**We have:**
Motors (DJI 2212, we currently are using 2312 motors, but my understatnding is that the only difference is a 1mm difference in stator size, also the 2212 is 920rpm and 2312 is 960 rpm/min)	some parts, but not a whole kit
Frame	
Speed Controllers (Currently 30 A Opto) (Kit comes with DJI 420 LITE 4S 20A BLDC)	
Propellers	
Will also need:	
Legs (PHM-LG001) (previously ordered from "ALL e RC, LLC", sold as 1 leg, need 4 per quad)	enough for one quad
Zybo Board	2, which have been used for other purposes
Micro SD Cards	0
Wifi Module (ESP8266 Wifi Module)	0
Lidar (Garmin LIDAR-lite V3, old LIDAR is no longer available)	0
Optical Flow (PX4flow) (We have version 1.3)	1
IR Reflecting Globes and mount (Model # MCP1090)	0
Voltage Divider	0
I2C Hub Grove	0
Motor Disconnect Connectors (Male and Female)	0
Standard Lipo Connectors	0
Radio Receiver	0
Remote Controllers?	
Batteries	2
Battery Monitors	1
Need to Make:	
Top clear plastic cover over zybo board	
adapter plates between frame and zybo board (2 for each quad)	

Figure 1-2: Parts Order for 2nd Quad

1.6.2 Ground Station

1.6.2.1 Communication Adjustment

Communication between the ground station and quad is over WiFi in a configuration that sets up the quad as an access point (AP) to connect to from the ground station. This works well when there is only a single quad but allowing an arbitrary number of quads simultaneously flying requires a change to make the quad, and other trackables using WiFi, clients that connect to a AP setup by the ground station.

1.6.2.2 Updated GUI

The GUI was created to originally run with a single object, being the quad, and with the addition of flying multiple objects the GUI must be updated. The starting process will allow users to setup the network enabling demos and testing. The navigation window of the GUI will also be updated to allow for switching between objects in flight and to allow for demos with multiple quads flying together.

The GUI is not fully functional in its current state, and does not do checking for incorrect data entered by the user. The new GUI will add all missing functionality and allow users to switch modes of navigation during flight (if conditions are met). The GUI will also perform checks when

sending coordinates to make sure that the user does not try to have the quadcopter accelerate into the ground or perform other tasks that may break a component on the quadcopter. Lastly, the controls graph generated currently is an image, we would like this to be capable of interaction so the user can change PID values from with the GUI.

1.6.2.3 Multiple Object Tracking Capabilities

To allow the quad to track an object we first need to get the camera system to recognize a second object as trackable. The VRPN system has the capability to track more than one object and will send that information to the backend. The backend needs to send this new information to the quadcopter for the quadcopter to track the object.

1.6.2.4 Manual Assist (Ground Station Side)

We were given two main modes of operation autonomous or manual mode. A third mode called manual assist mode that uses a usb controller to move an x, y, z coordinate setpoint that the quad flies to using the quad's autonomous controls. It will provide a means to getting familiar with quad control in a manner that is less likely to hurt the quad as there are added safety features.

1.6.2.5 Generic Vehicle Integration to Backend Capabilities

The previous platform was only fit for use with our specific quad that accepts our defined commands. We implemented an adapter framework to easily attach new types of vehicles to our system. We also used this framework to create an adapter implementation of the Crazyflie vehicle as well as the Cybot platform used in the embedded systems class.

1.6.3 Controls Systems

1.6.3.1 Model Linearization and LQR Controllers

The primary deliverables of this year's team were a modular linearization of the system model and a pair of LQR controllers. The linearization is a script that uses symbolic MATLAB derivation of the nonlinear model provided by Matt Rich in [1]. This allows a future user to change the nonlinear model and immediately recompute the system linearization. Similarly, the linearization is also dynamic on the measured system parameters, so no further work needs to be done to account for potential future changes of physical properties (e.g. using bigger rotors or a frame with a greater mass).

1.6.3.2 System Parameterization Instructions

Because there are now two quadrotors, it is more important than ever to be able to measure and track the physical properties of of each quadcopter. As such, the controls team aggregated parameter measurement procedures from both Rich's [1] and McInerney's [4] research, as well as from un-versioned documentation from the previous year's team. These were formed into a series of four parameter identification instruction documents, written in Markdown and stored on git, that contained straightforward instruction, consistent variable usage, and (where necessary) example MATLAB scripts. Additionally, a Markdown document was created to track all relevant parameter values and instruction sets.

1.6.4 Continuous Integration

1.6.4.1 Quad Simulator

The new quad simulator models a virtual flight dynamics environment for various flight tests. The current established model in the simulator does not model rotor dynamics; however, it still offers a reliable platform for performing sanity checks of the changes in controls and quad software. The current simulator uses a slightly modified version of the actual quadcopter controls. The simulator also offers input and output through sockets which enables control to be running outside of the simulator. Future teams may integrate the simulator with the automated environment of GitLab.

1.6.4.2 Upgrade Testing Framework

Continuous Integration is the system that tests changes to code using the virtual quadcopter software. To make our tests more standardized and provide more flexibility in writing the tests, we ported the tests from a custom barebones testing framework to a standard testing framework, Unity [5]. This provides a fully developed set of testing functions that can be used by future teams.

1.6.5 Documentation

Many areas of the code, especially those relating to ground station and quad software, were lacking documentation. The ground station contains four main components that are separated well but adding functionality was not explained nor is it mentioned that this is custom communication between the ground station and quad. The quad software is designed in a way that makes it so external directories must be used in build tools and there is also no explanation of the hardware running on the quad. We made it our goal to have documentation for all existing demos, documentation consistent in all code, and documentation for the research done during our time on the team.

1.6.6 Demos

As one purpose of this project is to showcase the talents within this department, new demos needed to be developed to showcase yearly changes. These demos are performed to controls classes as well as to undergraduate students. We plan to implement the following major demos:

1. Have a quad that tracks an object on the ground, or in air, and maintains a set distance away from it.
2. Have multiple quads perform synchronous movements
3. Have multiple types of quads running at the same time flying together.

2 Design and Implementation

2.1 Proposed Design

2.1.1 Quad Software

In order to effectively upgrade our Xilinx Toolchain software to the latest version we determined that creating a new project, re-develop the same hardware, and export that to the old hardware workspace using the old software and project to reduce the amount a variable that we introduce into the system.

In terms of the second quad our approach was to duplicate the wiring, hardware locations and orientations of all the components. This again was to make as many parts of the software reusable, so we didn't have to maintain a software build for each quad.

2.1.2 Controls

The controls for the quad is currently implemented using nested proportional-integral-derivative (PID) controllers. There is a set of PIDs for each of the three Cartesian components of position (x, y, z) and one for yaw (rotation around the z-axis). These were chosen because they achieve a very configurable approach to quadcopter controls, as modifications to the quad can be accounted for by simply adjusting the various PID constants.

The problem with PID controllers is that they contain almost no information about the system physics, and once tuned to reasonable values control cannot be reliably improved except through modifying the coefficients by hand to meet qualitative judgements. The primary change we wanted was to create a controller based on a physical model of quadrotor actuation, which can serve as non-trivial starting point for future controls research on this platform. Specifically, the plan was to implement an LQR controller capable of flying the quad to prove the correctness of our model and its computed linearization.

2.1.3 Ground Station

The overall architecture of the various components for the ground station will stay consistent but the network architecture, as well as backend functionality, will be improved. The ground station is currently well designed allowing for a backend server, a frontend for clients to use for communication with the backend, and various clients such as the GUI or CLI, more details on each interface can be found later in the report. The benefits of the system as it stands is that the communication and server are kept with the backend so that clients do not need added complexity to deal with the different objects that are connected. The frontend provides a simple interface that clients can pass data to and get a response as needed. This again hides the backend implementation from the clients and this interface is simplified and provides all functionality that the quad and backend have to offer.

The network architecture for the MicroCART system consists of the quad setting up an access point for the ground station pc to connect to. This allows for only a single object, but with the addition of a second quadcopter and usefulness of the MicroCART ground station tools, it is a drawback of the system. The ground station will be updated so that it can setup as a AP so that an arbitrary number of other trackables such as quads can be communicated with simultaneously. A DHCP server will be needed as well to provide objects an IP address for communication. There will be static IPs for any MicroCART quad or other support object to allow users the most control when flying and confidence that they will be communicating the with intended object. There will also be a range of IPs randomly assigned if a different approach is wanted for another application. This change allows for the added functionality to the backend for supporting multiple trackables.

The next major change involves the integration of multiple objects into the backend. This will involve separating the backend from the dependence of a single communication scheme. The backend will be configurable from a config file that will define what a basic trackable is from the ground station side. Meaning that the user could configure as many objects as desired to all be controlled by the ground station and get update from the new single threaded VRPN tracker. This

tracker will loop through all objects and provide them each with position information assume they are using the VRPN system. This will also bring about changes in the GUI which will consist of a means of quickly switching between trackable objects which it will load from the same config as the backend.

2.1.4 Continuous Integration

The original Continuous Integration (CI) system ran a suite of tests that performed checks on parts of the quad software, using a set of sockets to simulate the drivers used on the quad. It relied on a basic testing framework, created by a previous team member, consisting of a single assert function. To address these limitations, we planned to add an additional part to the testing procedure to test the controls themselves. This would involve interfacing with a flight simulator and connecting the controls used on the quadcopter to the simulator, with the output of the simulator connected as inputs to the control model and the outputs of the control model connected to the inputs of the simulator to provide throttle levels to the motors of the quad in simulation. In addition, we planned to replace the testing framework currently in use with a more powerful C testing library, Unity [5]. To do this we we worked to convert the existing tests to use Unity.

2.2 Final Implementation

2.2.1 Quad Software

2.2.1.1 Second quad

The initial thought to this was to provide a more stable platform for graduate student to do research on by removing the experimental components on the old MicroCART quad. But as we progressed we decided that the second quad would be essential for a synchronous movement demo. In order to support flight on the second quad we had to adapt our quad software to accommodate for the new quad hardware. Additionally we worked to improve the old layout of the quad.

2.2.1.2 Vivado Build

The amount of problems we ran into while trying to get a Vivado build working is too many to list out on this document. What is important to note is that there is a platform that is building and failing in runtime at the moment. The problems we ran into are documented well and the current problems are also in a state that future teams and resolve in a reasonable time. without fighting the same compatibility issues that this year's team has. A majority of these issues sprout from the project files being forced to updated from the 2014 to 2017 version of XSDK(another wonderful Xilinx toolchain product). The major goal of the Vivado project changed scope throughout the course of this project from getting a duplicated build to getting a workable build that others can fix/improve upon.

2.2.2 Controls

Using the nonlinear model of quadrotor flight provided by Matt Rich in [1], we implemented a modular script for model linearization that will be of use regardless of future project directions. Our linearization script uses symbolic differentiation in MATLAB of the nonlinear model, which means that changes to the model written as modifications of the existing vector functions will immediately be reflected in the general linearization. Additionally, the linearization script follows

one containing all system parameters, so if any physical parameters change (for example if the quad mass increases with new components), the linearization will again immediately reflect the change.

To prove this model and linearization, we implemented LQR controllers to fly the quad in our existing Simulink model flight simulation. The only difference between these controllers was their choice of a weighting function, whose value the LQR is intended to minimize. Rich in [1] outlines a method for picking which weights in the matrix (when the cost function is visualized as a matrix multiplication of the current state vector) should be included, but did not give values.

Because of the lack of a cost function (and without sufficient controls experience to cleverly design one), we implemented a MATLAB script that could determine the optimal weighting matrix to minimize setpoint error, given the list of setpoints and their corresponding times. These values were put into the Simulink model, which was wrapped in a function that returned the sum of the norms of the setpoint errors. Then, a large set of random weights were generated, and each was run as an initial input to MATLAB's built-in constrained minimization function, which perturbs the input weights repeatedly to attempt to minimize the total error estimated by the simulation. With this approach, we were able to achieve impressively stable simulation results with nearly no knowledge of the proper choice of weighting.

Once the linearization was computed and a set of weights were given, MATLAB has built in functions to take the four matrices that comprise these two inputs and construct an LQR controller, so very little work was required in dealing with specifics of this type of control.

2.2.3 Ground Station

2.2.3.1 Communication Adjustment

As the AP is moving away from the quad onto a linux platform it allows the use of well known services to provide the network. We choose to use both hostapd as well as dhcpd for creating an AP and dhcp server on the ground station computer. This is done by creating custom configuration files for both services to run on startup. It also allows for the use of a password protected network that the ESP8266 ESP-01 chip supports on board the quad supports. In order to make the the installation of said services as easy as possible for future users of the system that are scripts created that will check and install the services needed. Next, it was decided that we want to be able to enable and disable the network so scripts were also created that allow for this process that can be run stand alone whether the end user is using the GUI or CLI.

The main issue with the access point setup was trying to get the services running on a school setup computer. Services that worked on a laptop running Fedora linux did not work out of box on the Red Hat linux school configured computer.

2.2.3.2 Updated GUI

We choose to keep the current GUI and add the new functionality to it. This started adding a means of network setup from either the scripts option or a button for users. The next step involved adding in the configuration similar to the backend by having the GUI aware of the trackables we can make it easy for the users by using the names specified so that is clear which object is being controlled. In order to specify this option a box that allows the user to select objects can be seen from the navigation window for quick switching of objects. The last change was one to enable

double object mode to fly the two objects together. This will take away the ability for the user to control each object independently and allow the GUI to control the movement for the user.

Issues arose with this implementation and the GUI has multiple threads that make synchronization an issue when attempting to modify many areas. Also, in order to test the GUI we have to run through the actual tests which is time consuming as there are no automated tests setup for this as of yet.

2.2.3.3 Multiple Vehicle Tracking Capabilities

To support multiple vehicles we had to make major changes to the current backend, frontend, and network topology. As described in section 2.2.3.1 The network was switched from an access point being hosted on the quad to an access point being hosted on the ground station computer. And in 2.2.3.2 the GUI was also updated to support this functionality. The main changes with this involve the meaning of creating multiple tracker objects from the backend. In order to support this the tracker had to become an object and there is now a handler thread that will loop through all trackers provide updates to all quads. This had to be single threaded as TCP issues arose when multiple trackers were running in different threads.

2.2.3.4 Manual Assist (Ground Station Side)

To implement manual assist mode we utilized the controller software that was created for the simulator (implemented by the continuous integration team) to read from a USB controller. We then read values of joystick and switches at a set intervals and translated those values to the movement of a set point in 3D space. The quad control algorithm then moves to the set point and attempts to stay at the set point. As an added feature we also implemented a security function that limits where the set point can go which is bounded within the VRPN camera system.

2.2.3.5 Generic Vehicle Integration to Backend Capabilities

To allow for easy addition of new vehicle types we created an adapter framework. The adapter is a separate process that works as an intermediary. It decodes commands coming from ground station and calls an appropriate callback function which controls the new vehicle type. This narrows down the work of implementing a new vehicle to modifying only a set amount of callback functions. To show this worked we developed an adapter that uses the crazyflie vehicle.

The main issue we faced with this was implementing the crazyflie adapter. Other students were working with the crazyflies in parallel using them for different projects. These students updated firmware and caused the crazyflies to not be stable using know working software outside of our adapter. We were able to prove that our adapters worked by insuring the commands sent to the outside software were correct but could not get stable crazyflie flight.

2.2.4 Continuous Integration

2.2.4.1 Quad Simulator

The new quad simulator uses a JSBSim open-source flight dynamics model which creates a virtual sandbox for flight testing. The current development stage of the simulator offers built-in quadcopter controls. User can control the quadcopter manually using a USB joystick, or in autonomous mode using set points. Open-source FlightGear flight simulator offers visualization of the quad simulator state. The simulator offers input and output through sockets which enables the

control algorithm to be run outside of the simulator. Future teams may integrate the simulator with the automated environment of GitLab using scripted test flight case to test changes in quad software as well as the controls.

Figure 2-1: Quad simulator state visualization through FlightGear

2.2.4.2 *Upgrade Testing Framework*

To upgrade the testing framework, we first identified the framework we wanted to move to. We chose the Unity testing framework because it is a well-developed testing framework for C and was suggested by MicroCART 2016-2017 [5]. We then rewrote each of the tests to use the Unity equivalent of the various asserts and checks that they contained. One issue we encountered when porting the tests was getting the testing framework to compile. Since the makefile targets were not gathered in one place and instead distributed through multiple files, it was more difficult than expected to complete this conversion. Once we understood how the distributed makefile definitions interacted, we wrote additional documentation to help future teams.

2.3 Standards

There is not a direct set of standards that is well suited for quadcopter drone software and hardware development. IEEE publishes some high-power electronics safety standards, but they are designed for systems significantly larger than ours. There are also pure software standards, but our project, as seen above, is not purely software. As such, the closest thing we have to a standard to follow is DO-178B, the aviation software standard created by the United States government. This still has its share of shortcomings in relation to our project, however. Given the experimental

nature of MicroCART and its remarkably low risk of serious injury upon a significant software failure (compared to the manned aircraft that the standard was designed for), some of the requirements should be considerably loosened. For example, DO-178B gives an acceptable frequency of failure for each level of significance, and even the lowest level of risk is given an acceptable frequency of one failure per 1000 hours, which is unreasonably (and unnecessarily) strict given the scope and scale of the project at hand. Nonetheless, the standard sets forward a useful sequence of steps in which there is a process to work from requirements to code and then to fully test both for accuracy and completeness. Aside from the DO-178B standard, there are a few other standards that are partially applicable to the MicroCART project. One of these standards is the IEEE 802.11 standard. Due to MicroCART using WiFi to communicate between the ground station and the quadcopter we must adhere to the IEEE 802.11 standard as it relates to wireless communication between devices on a wireless local area network (WLAN). We are also in compliance with the IEEE 1625, ISO 9899, RFC 791, and RFC 793 standards, which are related to Lithium Polymer batteries, the C programming language, Internet Protocol version 4 (IPv4), and Transmission Control Protocol/Internet Protocol (TCP/IP) respectively.

3 Testing

3.1 Interface Specifications

The major interfaces for the MicroCART project involve the ground station and the multiple areas including the Backend, Frontend, CLI, and GUI. Figure 3-1 shows the communication between the various areas and the sockets outside of the ground station computer.

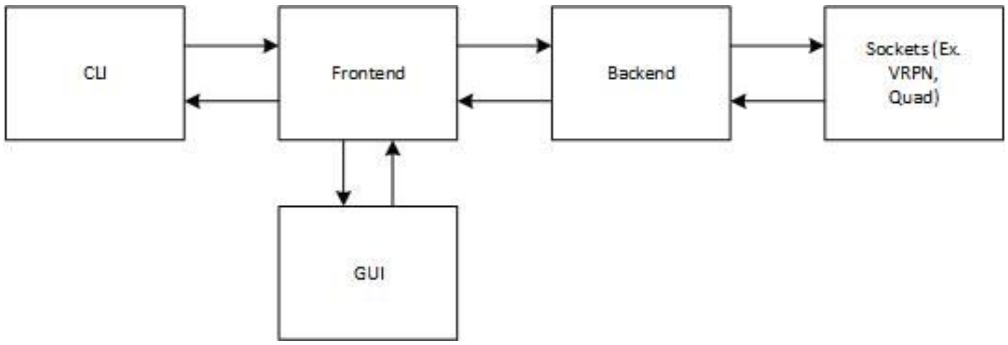


Figure 3-1: Ground Station flow diagram

3.1.1 Ground Station Interface Specifications

The ground station interface consists of two major components including the Backend and Frontend. The Backend provides a server that the VRPN system and user interfaces connect to via sockets for communication. This allows the obtaining of position information from the VRPN system as well as accepting commands from the frontend. Additionally, the backend connects to the quadcopter also via a socket. The frontend provides methods that handle the interfacing with the Backend for both the CLI and GUI.

3.1.2 Ground Station GUI Specifications

The Graphical User Interface will have four tabs that allow for starting of the backend, viewing the control graph, navigation, and real-time graphing. The backend tab both starts the backend and will also allow for the use of the Command Line Interface directly within the GUI. The controls tab allows the user to change the constant values within the controls to tune the PID values during flight. Navigation allows sending of coordinates to the quadcopter and the running of demos. Lastly, the real-time graphing tab will allow a configurable real-time data transmission between the quadcopter and GUI to graph during flight.

3.1.3 Ground Station CLI Specifications

The Command Line Interface provides direct access to the commands that the GUI sends for the user. It does not provide many of the extra features that the GUI provides such as automating setpoint sending, real-time transmission, and viewing the control graph. This is a more lightweight interface that still provides the use of all the same commands sent from the GUI. The CLI also allows for the creating of scripts that can run instead of a C based program.

3.1.4 Ground Station Access Point Specifications

To support multiple quadcopters we setup up a wireless access point on the ground station. This required server software such as hostapd, dhcpd, and a reprogramming of the WiFi chip on the current quad. Once this was implemented we were able to host multiple vehicles, simultaneously, from a single ground station [2].

3.2 Hardware and Software Testing

3.2.1 Automated Unit and Functional Testing

All commits to the Git repository are tested through a suite of continuous integration scripts. These scripts perform unit tests on the software that runs on the quad and higher level functional tests that run on the "virtual quad" which interfaces with a set of Unix drivers. The scripts are run automatically using the GitLab pipeline integration. We will continue to improve the test coverage over the existing code, and as more features are added, tests will be added to cover them. The automated testing flow is shown below in Figure 3-2.

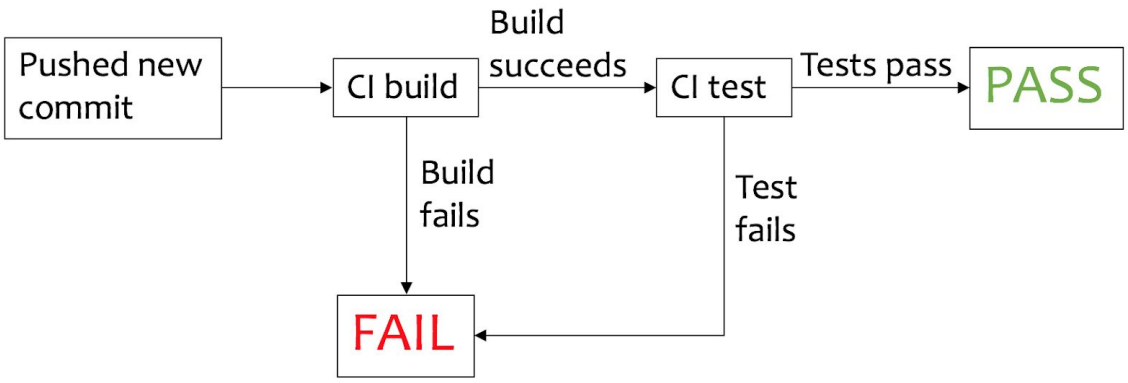


Figure 3-2: Diagram of automated testing

3.2.2 Flight Simulation

The correctness and requirements of the quadcopter software will be tested through the quad simulator and the help of the simulator event test scripts. Different flight regimes will be tested and verified whether the quadcopter position and orientation are within a threshold. In case of an accident, ground contacts are detected and the unsuccessful test is terminated early. An example of the simulation output is provided in Figure 3-3.

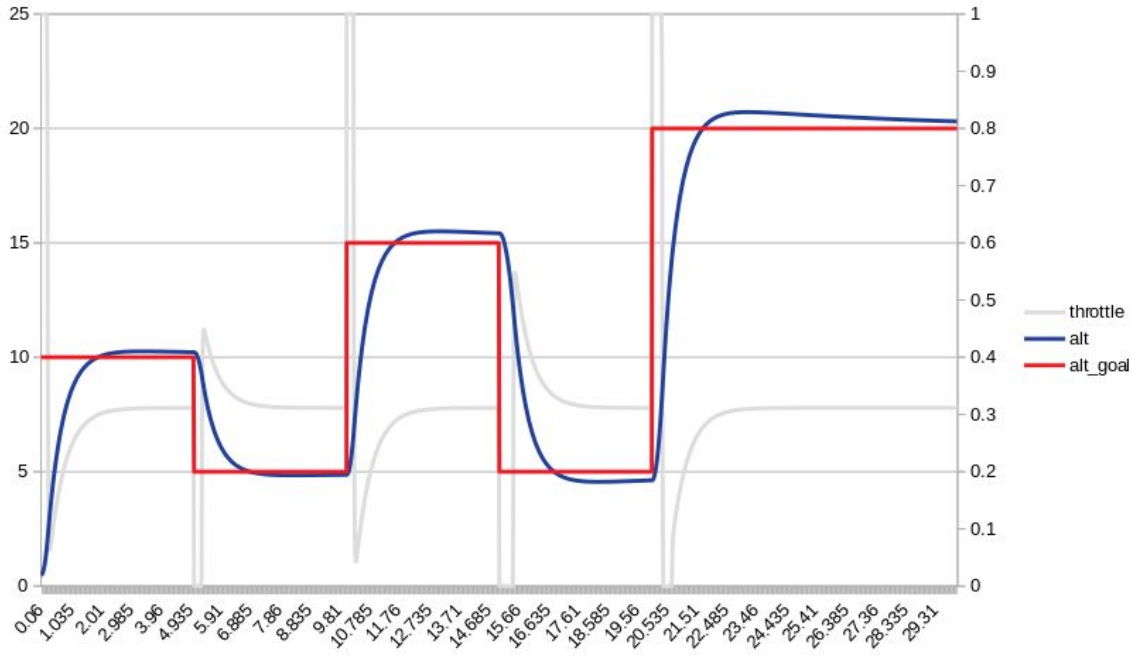


Figure 3-3: Quadcopter altitude PID testing results obtained from the quad simulator

3.2.3 Flight Test

The implementation of the ground station allows for a list of setpoints to be loaded into the GUI to allow for autonomous navigation. The quad will automatically relocate to the next point in the list once it gets within a defined range of the setpoint. We use these set points to test edge cases and to have a flight pattern that will test the more extreme patterns of movement. Another method of testing is to use the CLI to directly call commands that we are testing. In Figure 3-3 the CLI is shown in the top left, Backend in the bottom left and the flight is on the right.

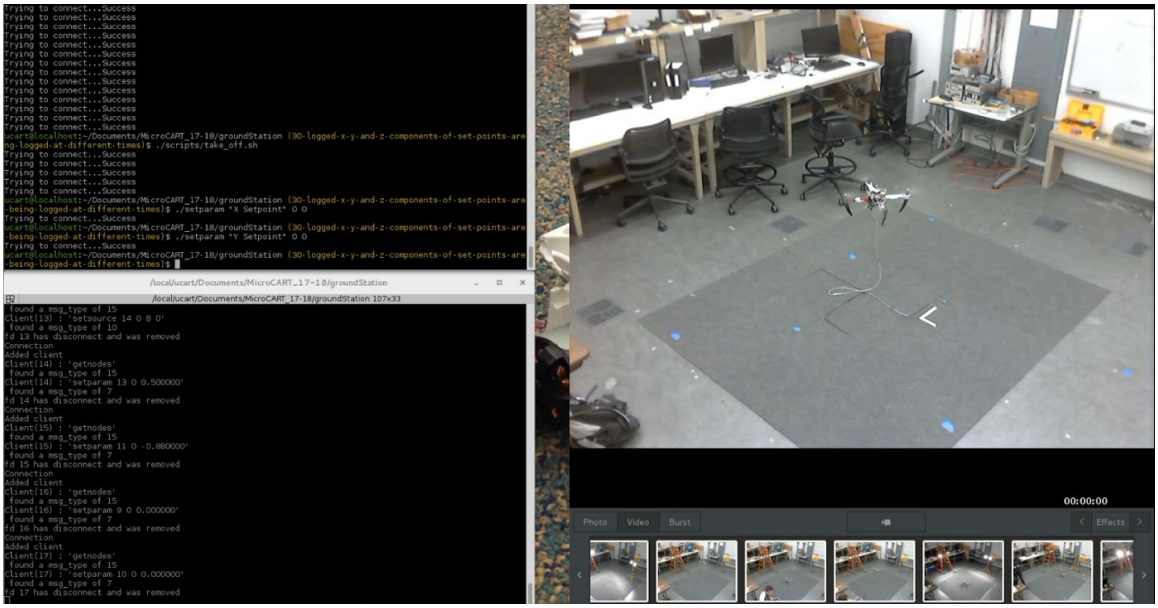
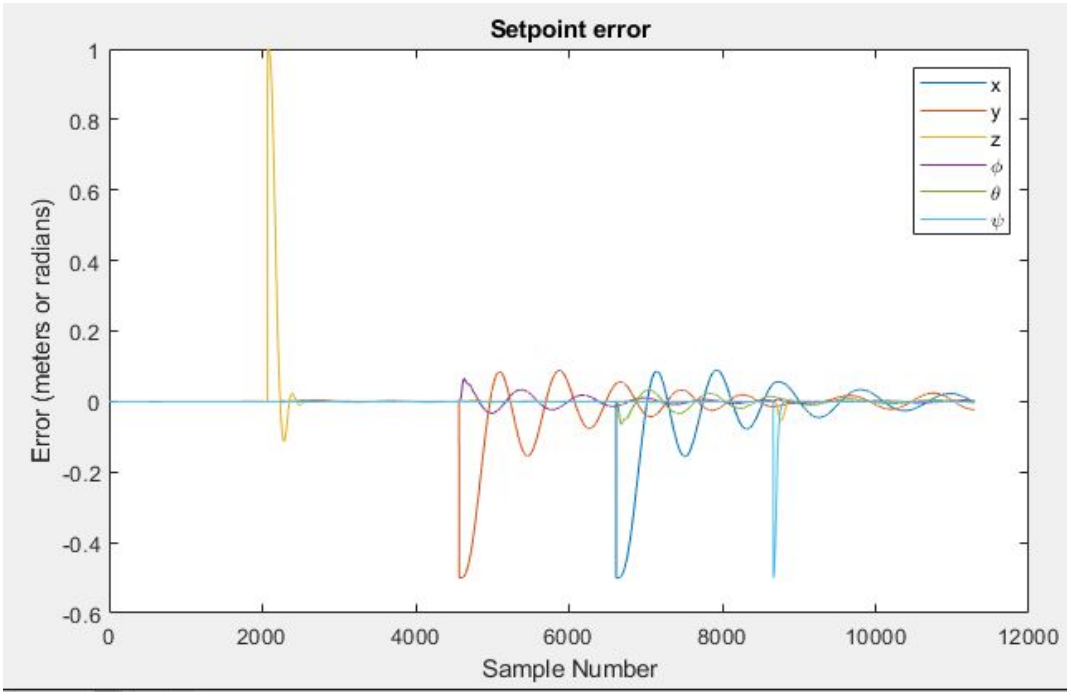


Figure 3-4: Flight Test with CLI

3.2.4 Controls Testing

Testing of the new LQR controller was done entirely through the Simulink flight simulation provided to us by last year's MicrCART team. For the initial design, the simulated parameters were set to the same values that Matt Rich measured on his quad in [1] so that we could directly compare results. Once stable flight had been achieved with that virtual quadcopter, we returned the virtual parameters back to match our own to design a controller for the modern quadcopter.

As mentioned in 2.2.2, a MATLAB constrained minimization function was used to find controller weights, so we required a function that could run the simulation and determine the error of each state from its setpoint at each time sample. Two such examples of this data (plotted as error vs. time sample) are shown below in figure 3-5. These are two of the earlier controllers. Note that the controller on the left – besides having an additional yaw setpoint – is much less stable than the controller depicted on the right. Despite the different plot formats, both of these have a total simulation time of 40 seconds, so the time the y-setpoint error takes to settle in the first (approx. 20s) is more than double the time taken in the second (approx. 8s).



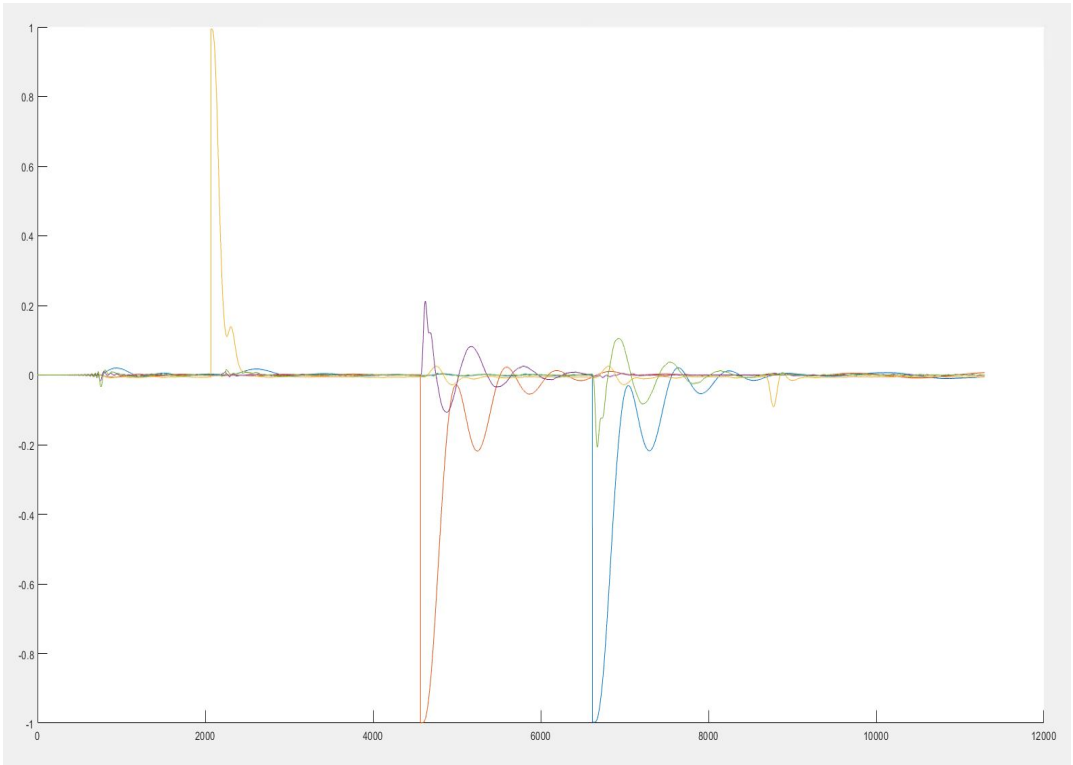


Figure 3-5: Setpoint errors for two separate controllers

3.2.5 Latency Testing

The quad uses WiFi for its communication which it is relying upon for position updates it is very important that the latency in communication is as low as possible. In [2] the WiFi is tested to bound the average latency of a packet within our system. As we changed the network configuration as well as the code controlling the ESP8266 ESP-01 chip latency was tested to ensure that this change and addition of a second quad does not negatively impact the performance of both quads during demos.

3.3 Process Diagram

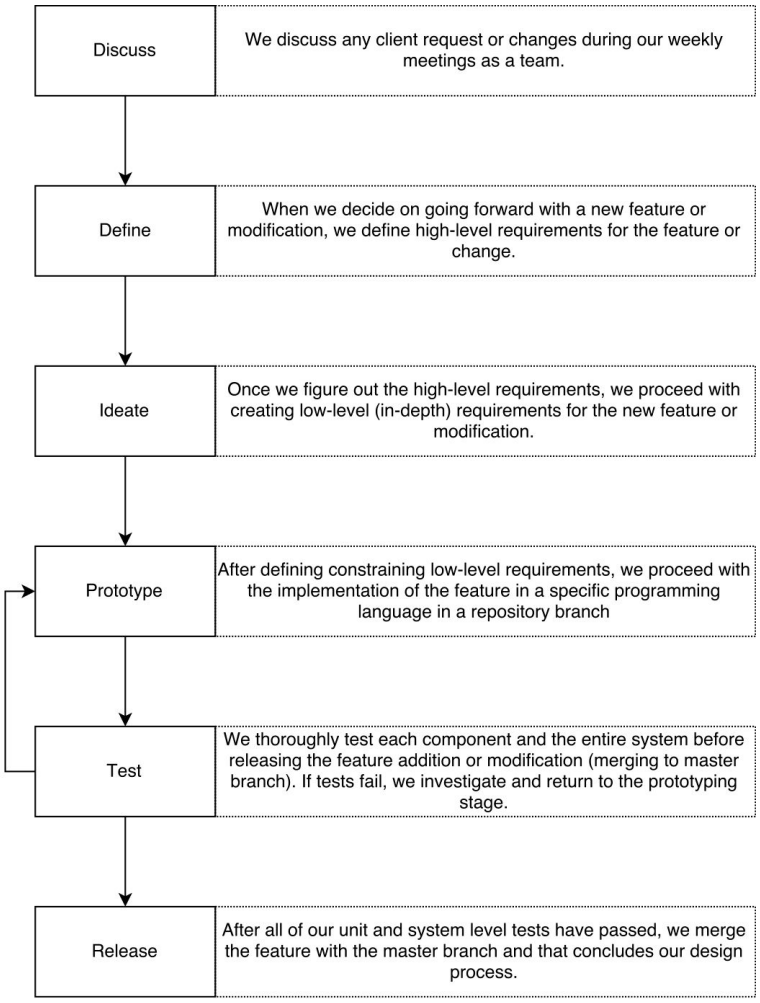


Figure 3-6: Design Process flow diagram, inspired by [3]

The above diagram explains the process we have followed as a guide to development. To effectively produce and improve upon the existing system we first discussed how to approach the problem and determined what must be addressed. After determining the problems we further defined them and came up with solutions for those defined issues. After brainstorming prototyping solutions was the next step. Moving forward, those solutions would be tested using the testing method described in section 3.2. After the correct solutions is determined and confirmed that solution should be

released into our master branch in the repository. It should also be noted that discussion in the form of updates would happen on a weekly basis with the team, client, and partners.

3.4 Modeling and Simulation

To test the controls, we have a Simulink model that simulates three things: the quad control logic, the actuation of the quad given the output from the controls (i.e. physics), and the sensor and communication system. These three components tied together in a loop allow us to easily run initial tests on modifications to (or new implementations of) the controls algorithm without the physical risks of running the untested model on the quadcopter itself. This model is based off of [1] and takes into account as many factors as possible (for example, the Sensors component passes the data and simulated noise through quantization and time delay), but it is still not a guaranteed proof of functionality.

Software-In-Loop and Processor-In-Loop simulation will be available through MicroCART Simulator (MCS). MCS backend runs on a flight dynamics simulator called JSBSim which provides environment simulation for aerial vehicles. The controls software is connected to MCS using a TCP socket and the output of the controls software is connected into the simulator to form a closed loop. By running different scripted scenarios, we can simulate different flight regimes and test our controls and the overall correctness of our quadcopter software.

3.5 Results

The list of deliverables are as follows:

- A Documentation Overhaul
- Vivado Upgrade Testing
- A Fully Functional Second Quad
- A New Way to Communicate on Ground Station
- An Updated GUI
- Multiple Object Tracking Capabilities
- Manual Assist Mode
- Generic Vehicle Integration to Backend Capabilities
- Model Linearization and LQR Controllers
- System Parameterization Instructions
- Quad Simulator
- Upgraded Testing Framework
- New Demos

4 Closing Material

4.1 Conclusion

Our MicroCART team has been steadily working to produce a more stable flying quadcopter that can be easily demoed to other students and faculty. We have vastly improved documentation for the entire MicroCART project that has been passed on to us from the previous team. Documentation has provided us a good fundamental understanding for the platform along with a

solid foundation for future teams understanding. The quad software team worked this semester to update the toolflow of the project from XPS to Vivado along with building a second quad. The ground station team improved the backend of the system to allow for simultaneous multiple vehicle flight. They also redesigned the GUI of the system, developed a manual assist mode, and an adapter framework to easily add new vehicle types. The controls team developed instructions describing how to parameterize the system as well as a new LQR controller. The continuous integration team added an interface to a flight simulator to expand the testing capabilities to encompass the control algorithm. Additionally, they ported the existing tests to a more robust and fully-featured testing framework. The subteams combined, worked towards meeting our end goals and deliverables for this semester to provide a platform that can be demoed to potential students, worked on by future teams, and used in research by graduate students.

4.2 References

[1] M. Rich, "Model Development, system identification, and control of a quadrotor helicopter" in *Iowa State University Digital Repository*, 2012

[2] Wehr, David. "ESP8266 WiFi Latency Testing." 17 Sept. 2016, https://docs.google.com/document/d/1VU99wMgkqK2EgbNLdqrdhvj9iikfqk2gtUYQ367K5-Q/edit#heading=h.s00g8emj18jx

[3] Plattner, Hasso. "An Introduction to Design Thinking PROCESS GUIDE." in *Institute of Design at Stanford* https://dschool-old.stanford.edu/sandbox/groups/designresources/wiki/36873/attachments/74b3d/ModeGuideBOOTCAMP2010L.pdf

[4] I. McInerney, "Development of a multi-agent quadrotor research platform with distributed computational capabilities" in *Iowa State University Digital Repository*, 2017

[5] Throw The Switch. "Unity." http://www.throwtheswitch.org/unity

4.3 Appendix

4.3.1 Operating Manual

Below is an in depth explanation of the sets required to properly setup and demo the quad. There are six primary steps required for a demo, as listed below.

- Setup the quad(s)
- Setup camera system
- Configure the backend
- Start ground station software
- Final Checks
- Connect the motors and start flight.

4.3.1.1 Setup the Quad(s)

During a demo the quad should be configured to boot from an SD card and the proper BOOT.bin file is required for the quad that you are using. Navigate to the MicroCART GitLab which is part of the DANC group, the link is listed below as well as at the top of this document.

https://git.ece.iastate.edu/danc/MicroCART

From the GitLab go the Tags section, grab the corresponding BOOT.bin file and place it onto an SD card. Next, ensure that the Zybo board is setup to boot from an SD card, see Figure 1 for the placement of the corresponding jumper, and insert the SD card, this slot as shown in Figure 1 is below the board in the image.

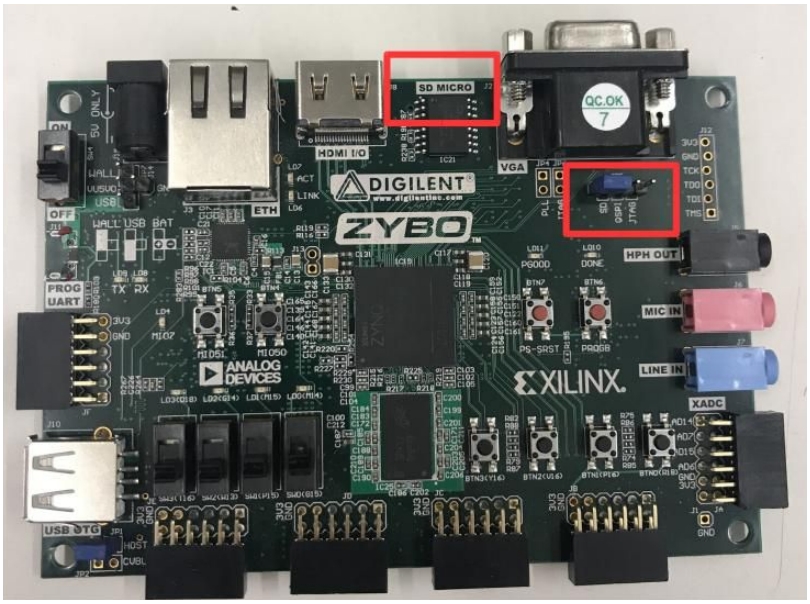


Figure 1: Zybo board

With the Zybo setup grab a battery and voltage monitor, both can be found within the lab near the MicroCART computers, and connect the voltage monitor, the correct connect is the one in Figure 2.

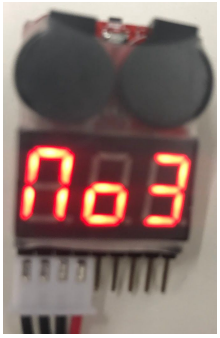


Figure 2: Voltage Monitor Connection

Place the battery on the quad and align it so that the back of the battery is aligned with the hole in the base plate of the quad, see Figure 3. This is approximately in the center of the quad which is the best for the controls that the quad run.

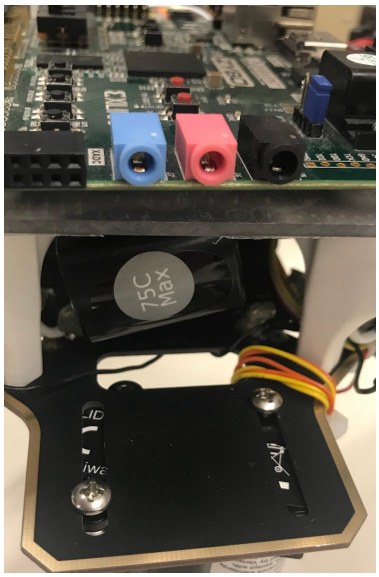


Figure 3: Battery Placement in the quad

At this point the battery can be connected and the Zybo board can be powered on. The “PGOOD” LED should turn red when powered on and the “Done” LED should turn on after the program has completely loaded. These LEDs are directly below the jumper used to configure the Zybo to boot from an SD card, the “PGOOD” on the left and “DONE” on the right, see Figure 1 for location of jumper. At this point grab the controller for the quad that is being demoed, the correct controller for each quad is to the right of the quads in Figure 4, and turn it on. The receiver on the quad should light up when connected as seen in Figure 5.

Figure 4: Quads with controllers (new quad is bottom quad, it has thinner top plate and black ESCs)

Figure 5: RC Receiver when controller is connected

Lastly, ensure that the quad is tethered down as seen in Figure 6.

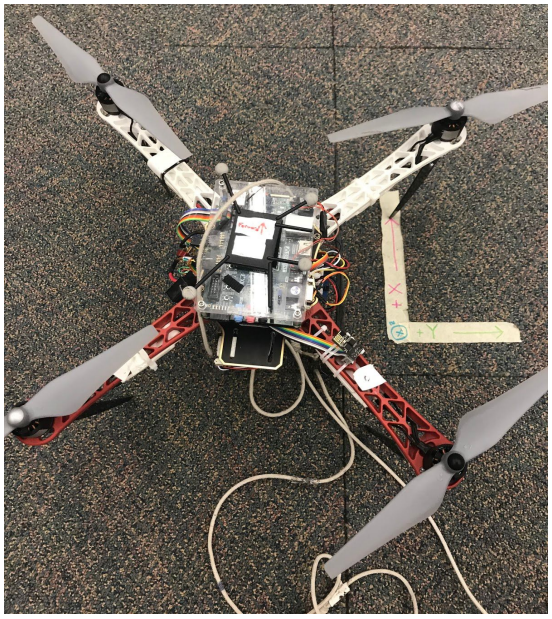


Figure 6: Quad tethered

4.3.1.2 Setup Camera System

If flying inside the lab this step needs to be completed. In the corner of the lab opposite the cabinet sign on the computer with the username and password list below.

Username: camera

Password: Camera

Open the OptiTracking tools software and open and existing project. The project that is most recent should be opened “TrackingToolsProject 2018-XX-XX-XX X.XXpm”, where the X’s are a time and date. Next, open up the corresponding trackable file labeled “MicrocartX.tra”, where X is the quad number that you are using.

At this point the camera system is setup and the real time information will be displayed on the screen.

4.3.1.3 Configure the Backend

Steps to navigate to the ground station directory are below and needed for all following steps.

cdsenior

cd groundStation

If only flying a single quad this step should already be complete. From the base ground station directory on the ground station computer navigate to the backend folder.

cd src/backend

Inside the config.h file ensure that the variable “NUM_QUADS” is set to the desired value. In the config.c check that the “trackables” array is setup and in the correct order for the demo.

4.3.1.4 Start ground station software

Back in the groundStation directory make the VRPN and the backend.

```
make vrpn

make
```

Either start the services for the network, or do so in the GUI, steps to start the network are as follows:

```
cd wifiap

./startAP
```

After the last make the ground station GUI should be available. For more advanced users that would like a command line option the backend can be started and the CLI used.

Directions to start the GroundStation GUI from ground station directory:

```
./GroundStation
```

Directions to start the backend separately and use the CLI from ground station directory, CLI commands are outlined on the GitLab and will not be shown below:

```
./BackEnd
```

The following steps pertain to GUI users only. On start the GUI will show the backend tab, start the backend with the “Start” button (make sure that the network is setup at this point). Go to the next tab labeled “Controller Graph” and click the “Refresh Controller Graph” button. Next, go to the Navigate tab and complete the rest of the steps.

4.3.1.5 Final Checks

At this stage check that all previous steps are complete. On the GUI the current position should be shown and changing over time if you move the quad around. Load and waypoint files if desired with the “Load” button, all files can be found in the corresponding folder in the Documents directory of the ground station computer.

Check that the controller is setup in the “Killed” and “Manual” configuration. The switches should be labeled on both controllers with this message as well. Please note that when the quad is killed it will not respond to any commands from the controller or ground station. When in autonomous mode, which is also labeled on the controller, if at any point a demo goes wrong flip into “Manual” mode and attempt to correct or alleviate damage.

Again make sure that quad is tethered before continuing.

4.3.1.6 Connect the motors and start flight.

Connect the deans connects on the quad, which are the red connectors shown in Figure 7.

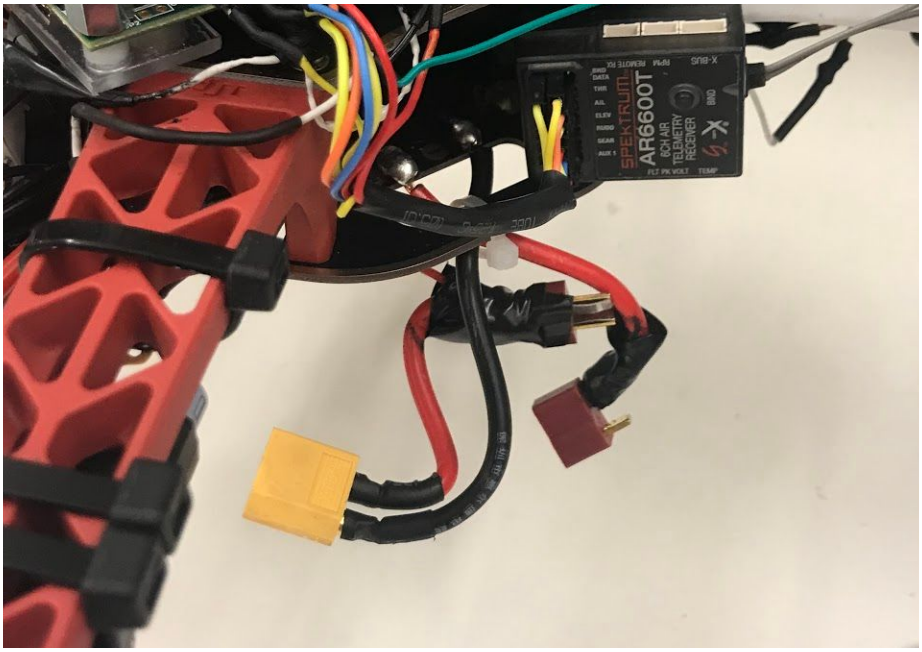


Figure 7: Deans connectors

Turn the quad on and enjoy your flight!

Other Notes:

When in manual mode the controller is providing the input to the quad. Always attempt manual mode before trying autonomous to check for any weird behavior. When in autonomous there are scripts for takeoff and touchdown accessible from the GUI.