MicroCART 2018-2019

PROJECT PLAN

Team #20

Client/Advisor

Dr. Phillip Jones

Team Members

Tony Bertucci - Ground Station Lead
Sarah Koch - Controls Lead
Tina Li - Quad Software Lead
Nina Moriguchi - Flight Simulation Lead
James Talbert - Hardware Lead

Team Email

sdmay19-20@iastate.edu

Team Website

http://sdmay19-20.sd.ece.iastate.edu

Revised: Sept. 24 / Version: 1.0

Table of Contents

1 Introductory Material **8**

1.1 Acknowledgement 8

1.2 Problem Statement 8

1.2.1 Problem 8

1.2.2 Purpose, Goals, and Approach 8

1.3 Operating Environment 9

1.4 Intended Users and Intended Uses 9

1.5 Assumptions and Limitations 10

1.5.1 Assumptions 10

1.5.2 Limitations 10

1.6 Expected End Product and Other Deliverables 10

1.6.1 Quad Hardware and Software 10

1.6.1.1 Vivado Upgrade Testing 10

1.6.1.2 Second Quad 10

1.6.1.3 Quad Hardware Upgrade 11

1.6.1.4 Quad Software Upgrade 11

1.6.2 Ground Station 12

1.6.2.1 Communication Adjustment 12

1.6.2.2 Updated GUI 12

1.6.2.3 Multiple Object Tracking Capabilities 12

1.6.2.4 Manual Assist (Ground Station Side) 12

1.6.2.5 Generic Vehicle Integration to Backend Capabilities 12

1.6.3 Controls Systems 13

1.6.3.1 Model Linearization and LQR Controllers 13

1.6.3.2 System Parameterization Instructions 13

1.6.4 Continuous Integration 13

1.6.4.1 Quad Simulator 13
1.6.4.2 Upgrade Testing Framework 13
1.6.5 Documentation 13
1.6.6 Demos 14
2 Proposed Approach and Statement of Work 14
2.1 Objective of the Task 14
2.2 Functional Requirements 14
2.2.1 Quad Hardware and software 14
2.2.2 Ground Station 15
2.2.2 Flight Simulator 15
2.3 Constraints Considerations 16
2.4 Previous Work And Literature 16
2.5 Proposed Design 17
2.5.1 Quad Hardware and Software 17
2.5.2 Controls 17
2.5.3 Ground Station 17
2.5.4 Continuous Integration 18
2.6 Technology Considerations 18
2.7 Safety Considerations 19
2.8 Task Approach 19
2.8.1 Hardware Development 19
2.9 Possible Risks And Risk Management 20
2.10 Project Proposed Milestones and Evaluation Criteria 20
2.11 Project Tracking Procedures 21
2.12 Expected Results and Validation 21
2.13 Test Plan 21
2.14 Interface Specifications 21

2.14.1 Ground Station Interface Specifications 22
2.14.2 Ground Station GUI Specifications 22
2.14.3 Ground Station CLI Specifications 22
2.14.4 Ground Station Access Point Specifications 22
2.15 Hardware and Software Testing 22
2.15.1 Automated Unit and Functional Testing 22
2.15.2 Flight Simulation 23
2.15.3 Flight Test 24
2.15.4 Controls Testing 24
2.15.5 Latency Testing 26
3 Project Timeline, Estimated Resources, and Challenges 26
3.1 Project Timeline 26
3.2 Feasibility Assessment 27
3.3 Personnel Effort Requirements 27
3.4 Other Resource Requirements 28
3.5 Financial Requirements 28
4 Closure Materials 28
4.1 Conclusion 28
4.2 References 30
4.3 Appendices 30
4.3.1 Operating Manual 30
4.3.1.1 Setup the Quad(s) 30
4.3.1.2 Setup Camera System 34
4.3.1.3 Configure the Backend 34
4.3.1.4 Start ground station software 35
4.3.1.5 Final Checks 35
4.3.1.6 Connect the motors and start flight. 36

List of Figures

Figure 1-1: Parts Order for 2nd Quad 10

Figure 2-1: Ground Station flow diagram 21

Figure 2-2: Diagram of automated testing 22

Figure 2-3: Quadcopter altitude PID testing results obtained from the quad simulator 22

Figure 2-4: Flight Test with CLI 23

Figure 2-5: Setpoint error using two different controllers 24

Figure 3-1: Project Timeline 25

Figure 4-1: Zybo board 30

Figure 4-2: Voltage Monitor Connection 30

Figure 4-3: Battery Placement in the quad 31

Figure 4-4: Quads with controllers 32

Figure 4-5: RC Receiver when controller is connected 32

Figure 4-6: Quad tethered 33

Figure 4-7: Deans connectors 35

List of Tables

Table 1: Task Time Estimates 26

List of 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 Introductory Material

1.1 Acknowledgement

MicroCART is a project that is assisted by graduate students working in controls under Dr. Phillip Jones. 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 has been and will be provided by:

- Matthew Cauwels
- Robert Buckley
- Matt Rich
- Dr. Phillip Jones
- Dane Larson
- Matthew Kelly
- Austin Rohlfing
- Eric Middelton

1.2 Problem Statement

We will improve upon an existing platform that is used for research in controls and embedded systems and for departmental demos. The platform will be improved by adding increased, reusable testing of all systems, adding documentation to increase the speed for new users to get started, and by adding new system features.

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 been made more

reliable so that anybody, even those 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 aspects of the project started with becoming familiar with the current system. This included reading existing documentation, updating it to increase clarity and completeness, 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 simulation using the quad's software. During this whole period we demoed new features to show our client and advisors progress and get feedback regarding functionality of the platform.

1.3 Operating 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 Intended Uses

The primary set of end users is composed of future MicroCART members and controls graduate students at Iowa State. We also have 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 alter 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 characterization 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 behavior of the quadcopter.

1.5 Assumptions and Limitations

1.5.1 Assumptions

- Our VRPN camera system as it exists provides sufficiently accurate position data
- Our VRPN camera system as it exists provides sufficiently frequent updates
- The quadcopter will be used within the camera system

1.5.2 Limitations

- The quadcopter must use a wireless link to the ground station
- 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
- The quadcopter must be physically tethered to the lab floor

1.6 Expected End Product and Other Deliverables

The quadcopter system consists of three major subsections: the quadcopter hardware/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 Hardware and 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. In addition, we added unit tests for the hardware modules in simulation as part of our continuous integration pipeline.

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 1-1: Parts Order for 2nd Quad

1.6.1.3 Quad Hardware Upgrade

The client has requested we upgrade the quads to the latest Zybo board that includes a connection for a pi camera. The current quads have many long wires and loose connections that are potential points of failure. Our team will create a custom pcb board to replace those connections, as well as install the newer zybo boards.

1.6.1.4 Quad Software Upgrade

Currently the quad can only fly reliably within the VRPN Camera system. By integrating the GPS and improving controls, a quad that will reliably fly outside will be aimed for. The improved communication and multi-client capability developed last year opens up the possibility of networking and quad coordination. To take advantage of the newer zybo boards and cameras the client requested, a version of Linux with OpenCV, to be run on the second processor on the board, will be explored. Software surrounding these new functionalities will be developed.

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. Last years team developed the hardware but did not get a chance to fully test it, therefore, a range of tests will be developed and implemented.

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 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. Our team will focus on improving the simulator model and integrating 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 the tests more standardized and provide more flexibility in writing the tests, the tests were ported 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

The year before, 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. Last year's team made it their goal to have documentation for all existing demos, documentation consistent in all code, and documentation for the research done during their time on the team. To follow up on that goal, our team will continue adding

documentation. This year, the areas of controls model and simulation, groundstation, the CI Testing Framework, along with pure hardware plans need improvement and organization.

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 Proposed Approach and Statement of Work

2.1 Objective of the Task

At the conclusion of this year's iteration of the project, we plan to have a stable, testable FPGA hardware platform; a functioning, modular software application capable of maintaining stable flight; a ground station capable of coordinating multiple generalized autonomous vehicles and a flight simulator that can test the software application in an automated fashion. All of these systems will be documented for the next year's team, with automated tests to verify changes as they are made. This will allow the project to meet its goals of being a research platform, and an interesting demonstration for prospective students.

2.2 Functional Requirements

2.2.1 Quad Hardware and software

The quad's system can be split into structures, sensors, actuators, and the Zybo Z7 control board. The structures provide the frame that holds the actuators and sensors in place, and connects the units together with wires. The Zybo Z7 control board needs to be able to communicate with sensors, getting data from each one within the control loop. Additionally, the Zybo Z7 needs to be able to control the actuators based on sensor data to affect stable flight.

The structures need to be rigid in flight, so that the behavior of the quad is predictable. This means that the motors should be in fixed positions (but free to rotate the rotors) as well as the sensors, so that the sensor data and actuation have a fixed relationship with the

quad as a whole. The wiring needs to be tightly connected such that in-flight vibrations do not lead to disconnection.

The Zybo Z7 control board contains a Xilinx Zynq FPGA, capable of instantiating any sensor/actuator interfaces we need. In order for the board to interface with the sensors and actuators, we will need to design, verify, and test new and existing interface modules. The testing and verification process should be automatic, so that changes do not go untested. The software on the control board needs to use these interfaces to implement a feedback control system. This control system will control the quad's motion to keep it hovering in a desired location.

The sensors and actuators should be selected such that they are capable of giving the control board enough information and control to maintain stable flight. This means that the sensors need to have low enough noise levels to give the control board accurate information about the quad's motion. The actuators need to be sufficiently responsive to allow the control board to react to disturbances in the environment.

2.2.2 Ground Station

The ground station is the hub through which data flows between separate subsystems. The ground station is responsible for routing position and orientation data from the camera system to the quad or other vehicle(s) and for exposing flight data to the user(s). The ground station needs to route these messages very quickly, particularly from the object tracking system to the quad. The object tracking camera system is capable of 100 updates per second, and the quad is capable of 200 control loop updates per second, so the bandwidth and latency requirements are important.

2.2.2 Flight Simulator

The purpose of the quad simulator is to create a safe environment to test out code before using it on the real quad.

- The simulation will replace the soft/hardware components from the drivers onward, isolating the controls of the quad. These drivers are standardized.
- The simulator will be integrated into GitLab CI with tests that can be run every time the controls software is updated.
- The parameters that the simulation is based on need to be accurate. Calibration needs to be done after every major hardware change.
- The Ground station communication to the quad will also be simulated or bypassed. This communication is standardized.
- The VRPN cameras will not need to be simulated. Mathematical noise will be added to the calculated position of the quad and latency will be added to the simulated wifi signal. Same can be done for the Lidar, Optical Flow, and GPS. Their noise ratio needs to be sampled.
- The code for the 2 cores on the zybo board will be kept separate and will be compiled separately to test them. The core that is not linux will be simulated in

real-time and not-real-time. The core that is linux is simulated in not-real-time. The linux code will be tested by itself, there is no need to simulate Linux itself, just run the code that would run on Linux. Simulation is needed to create fake data for the code to use.

- This simulator will also be integrated with a testbench for the ground station. The testbench will be a command line script/program/CI that will run set tests to make sure the code is doing what we want.
- To be easy to use, view and analyze, a good CLI, GUI, 3D rendering of the quad and analysis tools are needed.
- To be accessible to many users any code not written by the team needs to be open source if possible.

2.3 Constraints Considerations

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-178C, 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-178C 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-178C 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.

2.4 Previous Work And Literature

All of our work is based upon the efforts of those who came before us. MicroCART is an ongoing project that has been undergoing iterations since the 1980's. Previous teams have made great strides, especially in the quadcopter software architecture. It has been surprisingly easy to understand and follow their design. Their work is lacking in documentation in some areas, especially in tool setup instructions.

2.5 Proposed Design

2.5.1 Quad Hardware and Software

In order to effectively upgrade our Xilinx Toolchain software to the latest version of Vivado we determined it was best to create a new project in Vivado, re-develop the same hardware, and export that to the old hardware workspace using the old software and project to reduce the amount of variance that we introduce into the system.

Upgrading to the newer Zybo boards may need some hardware port changes. To increase reliability, more robust physical ports will need to be soldered on and the wiring replaced with a custom designed PCB board. The current hardware power specifications and wiring need to be mapped out/ documented before designing can begin.

2.5.2 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.

2.5.3 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.

2.5.4 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 plan 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. Automated tests that integrate with this simulator will also be made to test the ground station. In addition, we plan to replace the testing framework currently in use with a more powerful C testing library, Unity [5]. To do this we will work to convert the existing tests to use Unity.

2.6 Technology Considerations

This project makes use of several technologies, including a 12 camera object tracking system, a 9 degree-of-freedom Inertial Measurement Unit, a LiDAR unit, an optical flow sensor, and an FPGA. These subsystems give the system greater awareness of its position and motion, and increase its' flexibility with regards to adding new sensors/technologies to the system.

The object tracking system is capable of providing accurate, high frequency updates about the quad's position. This can the be used by the control algorithm to provide actuation to maintain stable flight. The use of the camera system restricts the quadcopter's operating area significantly. It is possible to use the Inertial Measurement Unit to achieve near-stable flight without the camera system, this can be further improved with the use of a LiDAR sensor to maintain a fixed height off of the ground, though that approach is more susceptible to environmental disturbances such as an uneven floor. The addition of an optical flow sensor provides additional data about the quadcopter's horizontal motion, but is again more susceptible to environmental disturbances, and requires a visible pattern on the floor.

We also make use of several technologies that are not part of the system at runtime for development and testing. All code and technical project files are stored on a departmental GitLab server, making them available to all members and any advisors. This platform also provides tools for automated testing whenever changes are made to the contents. We use the Xilinx FPGA development toolchain (Vivado and XSDK) to build the hardware platform for the control board, and to develop the application software on the quad. These tools are well maintained, but not without annoyances, they often take actions that make the project less able to be worked on from different machines, reducing the options for development.

The current controls model is created in Simulink in MATLAB, a proprietary software, making this model not easily accessible to many users but very powerful. Any CI testing utilizing this model will also be unavailable to the user. Xcos in Scilab is a good open source alternative that is suitable for controls applications. There would be a learning curve and it would take time to replicate the model as far as it currently is.

Simulink has its own 3D rendering capabilities but a recent release has a specific module set up for easy integration with FlightGear (which is free), which has an easier UI, but because it is fairly recent the model may have issues if it is opened in older versions of MATLAB.

The virtual quad currently works on Unix systems. It may or may not be worthwhile to create a version to run on Windows systems.

The Groundstation is written in QT and compiles for a specific system. It may be worth developing a lighter version that can run on android systems that will make the ground station portable when the quad will eventually be used outside.

2.7 Safety Considerations

This project does carry physical safety concerns. The quadcopter has blades that spin very quickly, and are capable of causing bodily harm. We mitigate the risk by using a tether, limiting the operational area of the rover to a hemisphere that can be avoided by people while the quad is in operation. There are application-level kill-switches that can be used to terminate quad function as long as the software is working correctly. The worst case of failure is the quad setting all motors to full output and then getting stuck in software. In this case, the quad will fly around the tethered area rapidly, until it runs out of battery. As long as no-one attempts to catch the quad, all personnel should remain safe even under this condition.

2.8 Task Approach

2.8.1 Hardware Development

The development of hardware in vivado is a process that involves a (relatively) long process to verify changes on the physical system. As such it is advantageous to test submodules in simulation or to work with single-purpose designs to verify a block until

there is reasonable certainty that the system will function. This implies a heavier dependency on reading documentation as compared with the software project. We will make efforts to keep the hardware design modular, so that individual components can be tested in isolation, where build times are faster.

2.9 Possible Risks And Risk Management

Many team members are working on tasks which they have no previous experience with. To mitigate the risk of falling behind schedule or being unable to complete these tasks due to lack of knowledge, our team has been working with members from previous teams to set up knowledge-sharing meetings and ease the learning curve as we take ownership of the project.

Ordering hardware components for the quad presents another possible source of delay to our schedule. According to a previous year's group, this process can be very slow. The best method for avoiding delay, in this case, is to order parts well before they are needed so that there is time for a re-order if necessary. However, introducing a reimbursement-style system that allows students to order parts would be preferred and greatly increase efficiency.

Due to FAA regulations, and with winter covering a significant portion of possible testing time, it will be difficult to test the quad outside. Consulting with our advisor about appropriate testing locations and rigorously careful scheduling will be needed.

2.10 Project Proposed Milestones and Evaluation Criteria

Our team will implement CI testing through Gitlab, so that every new software addition will have a test written for it and can be run against all the previous tests to check it doesn't break the system.

First Milestones

- Familiarizing ourselves with the available documentation
- Improving documentation and testing for the project.

Second Milestones

- Updating hardware to the newer Vivado
- Mapping the physical quad circuit system
- Robust zybo ports soldered
- Updating quad parameters for simulation
- Successful communication between virtual quad and Simulink simulator for CI testing

Third Milestones

- Designing of PCB
- Zybo board switched out
- Designing controls that accept matrices
- Updated Simulink simulator to include other sensors

- Successful communication between virtual quad, Simulink simulator and FlightGear for CI testing
- Linux integration

Fourth Milestones

- PCB designed, built and integrated
- Successful communication between virtual quad, Simulink simulator, FlightGear, and Ground station for CI testing
- OpenCV support

2.11 Project Tracking Procedures

Our team will meet weekly to discuss progress with our client and as a team. Each week we will set goals and then report on the progress of those goals. We will be using issues in GitLab to track work items and bugs. Any time a demo fails, we will attempt to determine a cause and create an issue with any available information. As we work further to resolve the problem, the issue thread provides a dedicated space for discussion that is permanently recorded for documentation purposes.

2.12 Expected Results and Validation

The list of deliverables are as follows:

- Continuation of the Documentation Policy
- Vivado Upgrade Testing
- A Fully Functional Second Quad
- Upgraded Zybo boards on both quads
- Custom PCB boards on quads
- Real time Ground Station data logging
- Multiple Object Tracking Capabilities
- Multi Quad Synchronization
- Linux Integration and OpenCV functionality
- Model Linearization and LQR Controllers
- System Parameterization Additions
- Quad Simulator
- Upgraded Testing Framework
- New Demos

2.13 Test Plan

2.14 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.

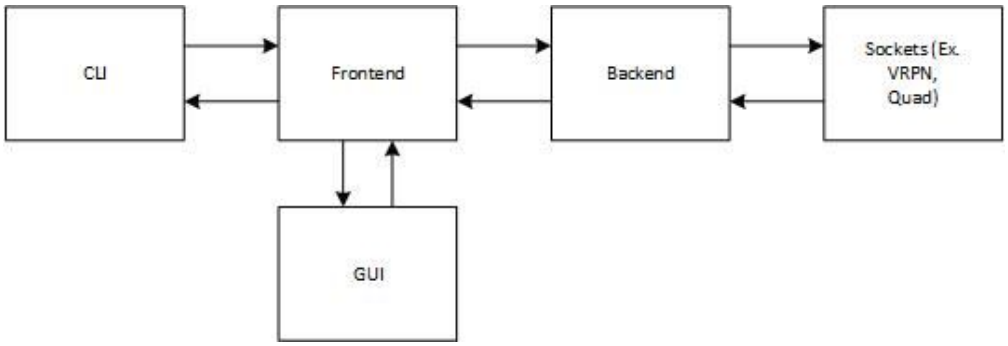


Figure 2-1: Ground Station flow diagram

2.14.1 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.

2.14.2 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.

2.14.3 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.

2.14.4 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].

2.15 Hardware and Software Testing

2.15.1 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.

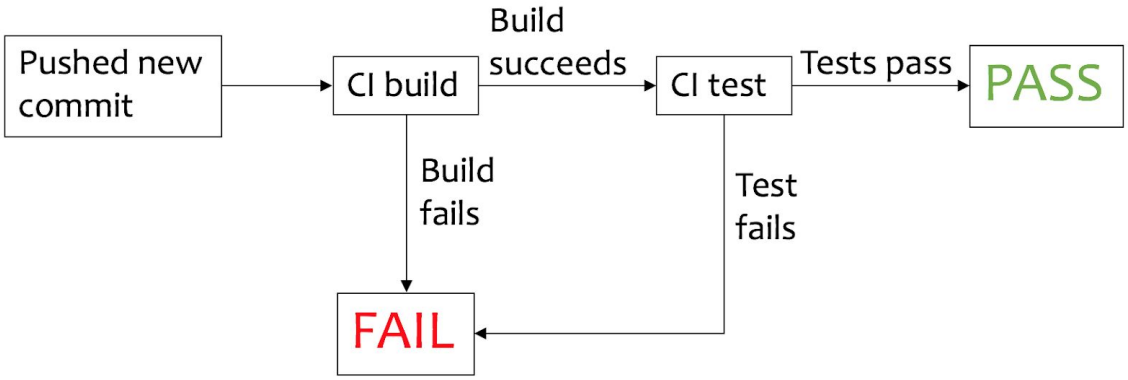


Figure 2-2: Diagram of automated testing

2.15.2 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.

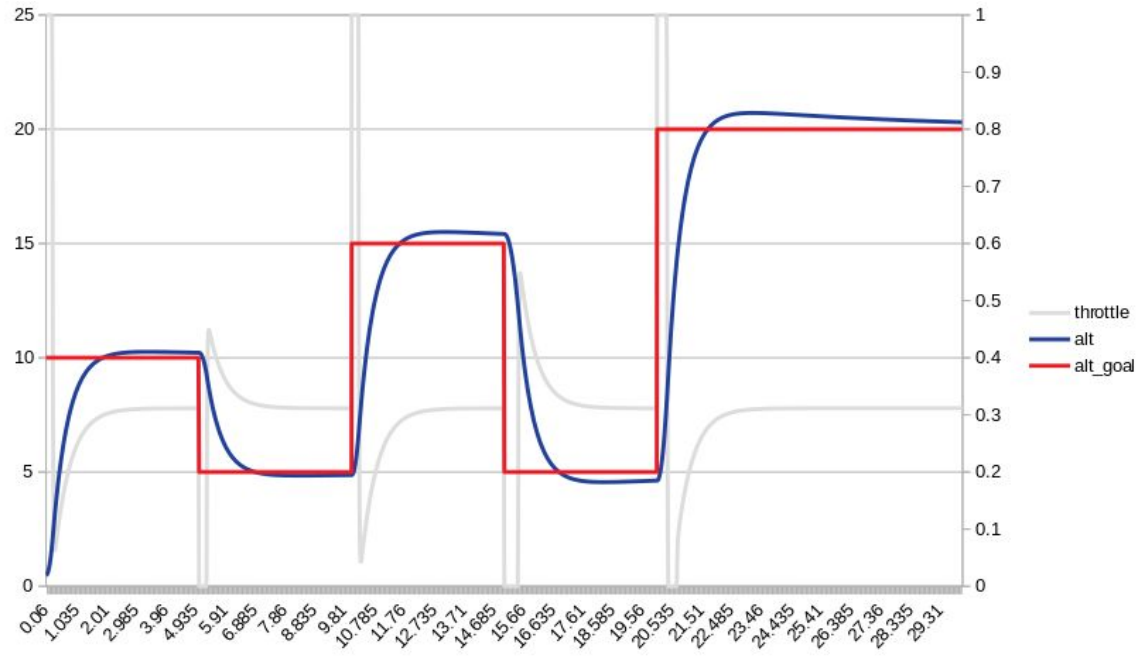


Figure 2-3: Quadcopter altitude PID testing results obtained from the quad simulator

2.15.3 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.

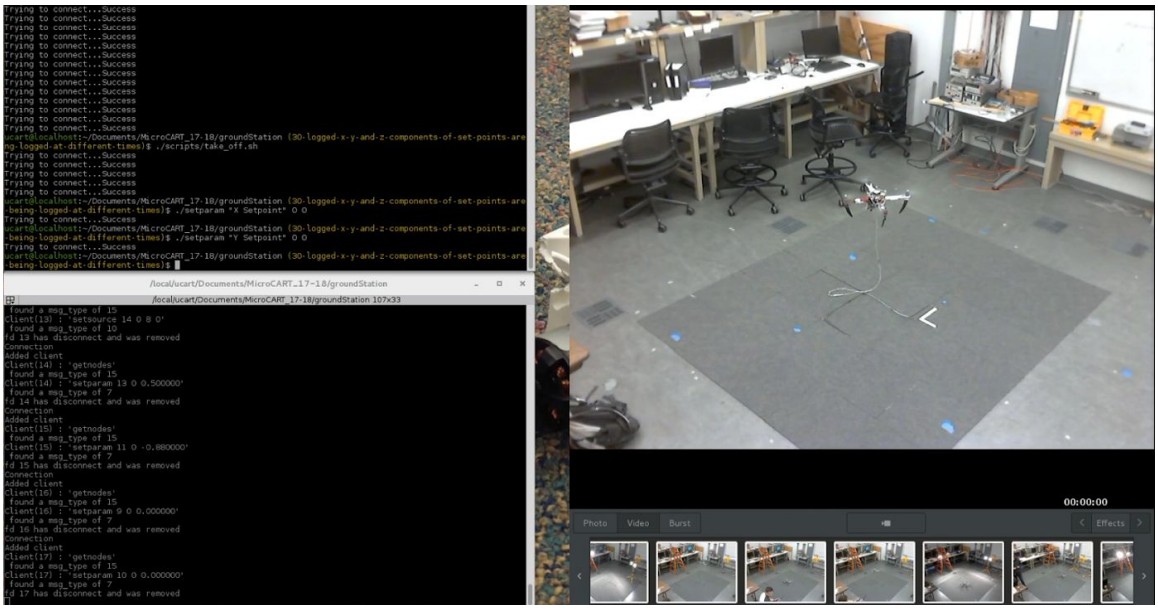
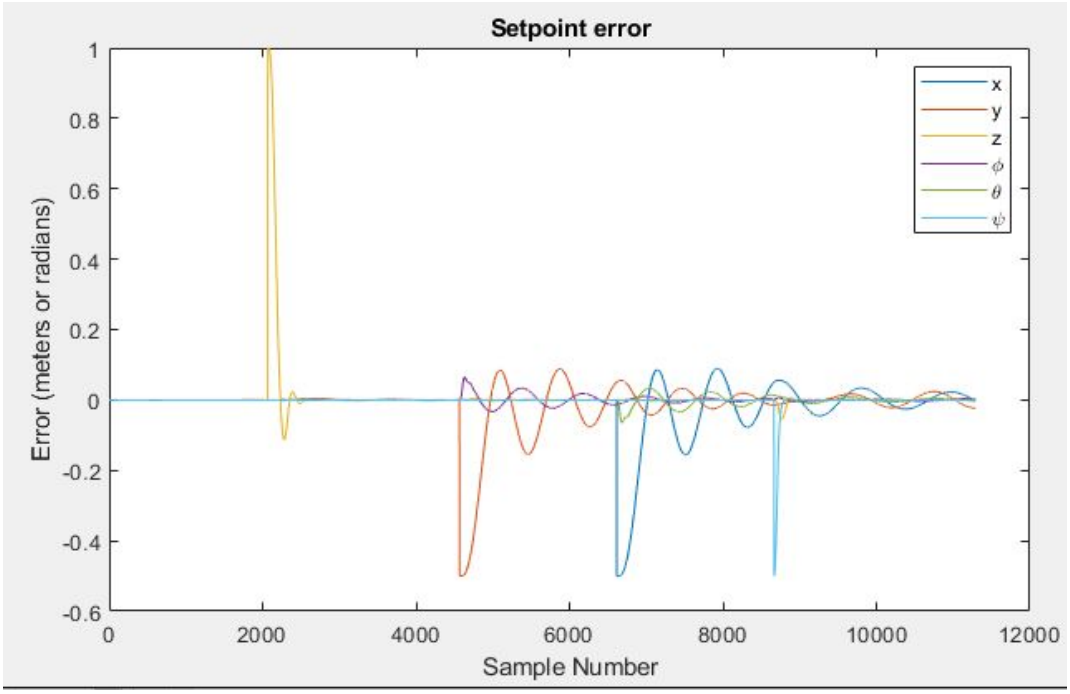


Figure 2-4: Flight Test with CLI

2.15.4 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).



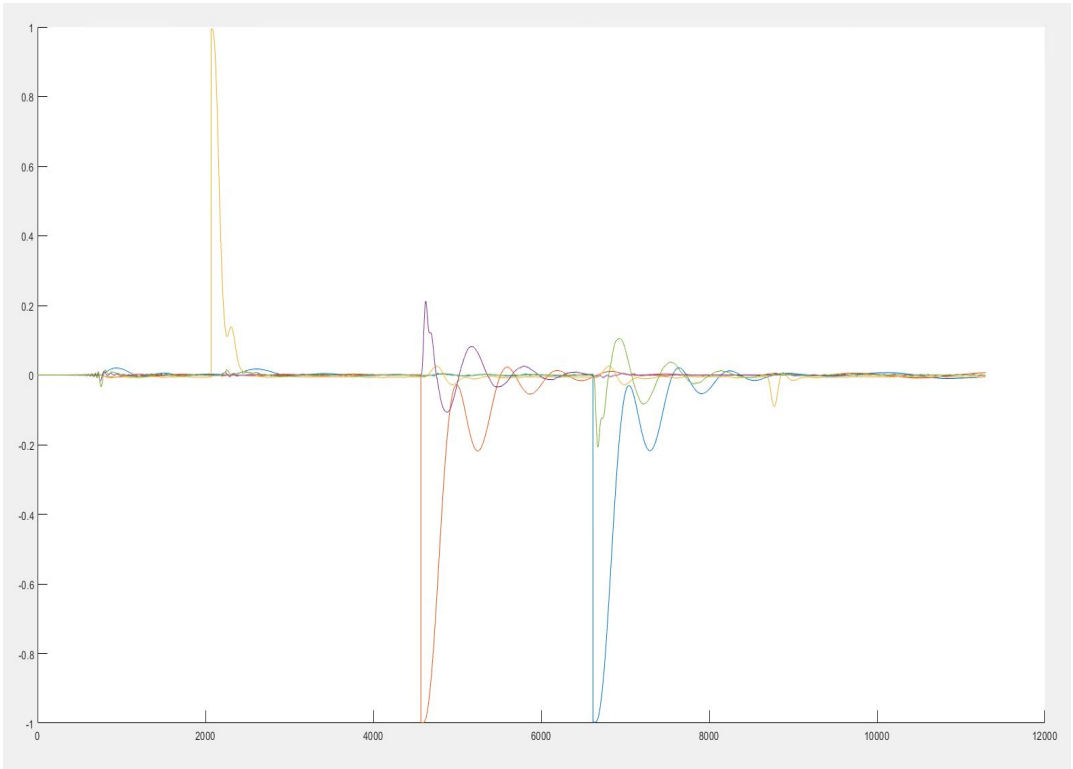


Figure 2-5: Setpoint errors for two separate controllers

2.15.5 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 Project Timeline, Estimated Resources, and Challenges

3.1 Project Timeline

This project is an ongoing project with a lot of existing progress, we plan to spend most of our efforts the first semester familiarizing ourselves with the existing work. As we learn how the system and development tools are setup, we will integrate new features, tests, and improvements into the system moving through the first semester. In the second semester, we will continue to work on the project's technical aspects, while switching the documentation focus from reading documentation to writing it for the next team.

			Start	End	Dur	%
	Project	⊖	8/20/18	4/15/19	164	
1	Quad Hardware	⊖	8/20/18	4/15/19	164	
1.1	Vivado Migration	⊖	8/20/18	10/26/18	48	
1.1.1	Single-unit tests		8/20/18	10/12/18	38	
1.1.2	Full hardware system design		10/13/18	10/26/18	10	
1.2	Automate Tests		9/17/18	4/15/19	145	
1.3	Implement automatic sensor interfaces		10/27/18	12/5/18	27	
1.4	Implement Controller in hardware		12/6/18	1/15/19	27	
2	Quad Software/Controls	⊖	8/20/18	4/15/19	164	
2.1	Ensure isolation of drivers		8/20/18	10/19/18	43	
2.2	Migrate app. to new hardware		10/27/18	12/28/18	43	
2.3	Verify LQR control model		9/10/18	11/13/18	46	
2.4	Implement LQR controller		11/14/18	12/27/18	30	
2.5	Write tests to run under the simulator		12/28/18	4/15/19	74	
3	Ground Station	⊖	8/20/18	10/31/18	51	
3.1	Learn GUI Framework		8/20/18	10/17/18	41	
3.2	Improve feedback to user		10/18/18	10/31/18	10	
4	Flight Simulator	⊖	8/20/18	1/31/19	113	
4.1	Create physics simulation of the quad		8/20/18	10/17/18	41	
4.2	Makefile for simulation of quad app.		10/18/18	11/14/18	20	
4.3	Integrate quad app. into simulation		11/15/18	12/31/18	31	
4.4	Simulate ground station commands		1/1/19	1/31/19	21	

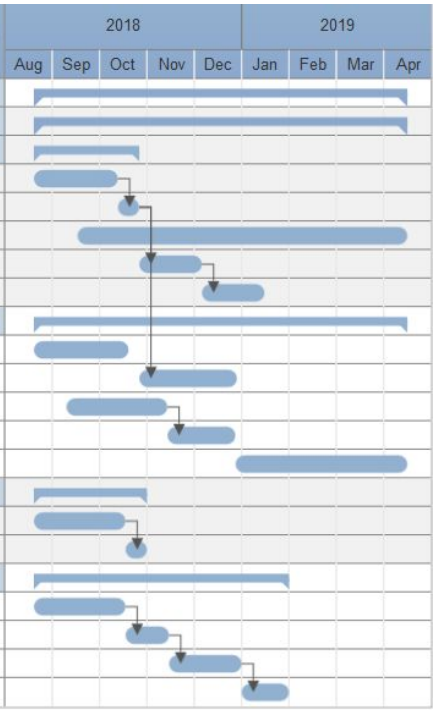


Figure 3-1: Project Timeline

3.2 Feasibility Assessment

From our experience as a team, we fully expect to achieve our goals regarding the unit testing of the system components and the flight simulator. We are unsure of the multi-quad and multi-device support for the ground station. Previous teams have tried and failed to control multiple devices simultaneously in this project. We plan to learn from their experiences, but we are unsure what caused their problems at this time.

3.3 Personnel Effort Requirements

We expect about 7 hours per week from each team member on the technical aspects of the project. See the following table for the expected time/task.

Task		Hours
1	Quad Hardware	110
1.1	Vivado Migration	40
1.1.1	Single-unit tests	30
1.1.2	Full hardware desgin	10
1.2	Automate tests	20
1.3	Implement automatic sensor interfaces	20
1.4	Implement controller in hardware	30
2	Quad Software Controls	125
2.1	Ensure Isolation of Drivers	20
2.2	Migrate app. to new hardware	15
2.3	Verify LQR control model	30
2.3	Implement LQR controller	20
2.5	Write Tests for the simulator	40

3	Ground Station	70
3.1	Learn GUI Framework	30
3.2	Improve feedback to user	40
4	Flight Simulator	100
4.1	Create physics simulation of quad	20
4.2	Makefile for simulation of quad app.	20
4.3	Integrate quad app. Into simulation	30
4.4	Simulate ground station commands	30

Table 1: Task Time Estimates

3.4 Other Resource Requirements

The MicroCART project operates out of the Distributed Autonomous Network and Control Lab in Coover Hall, which is equipped with standard bench electronics equipment as well as quadcopter-specific items. In addition, the lab is often used by graduate students who have expressed a willingness to help us as advisors for the controls problems we will inevitably encounter.

3.5 Financial Requirements

The client will be funding the project as needed.

4 Closure Materials

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. The previous teams documentation policy has provided us with a good fundamental understanding of the platform along with a solid foundation for future teams understanding. The quad software and hardware team will work this semester to update the toolflow of the project from XPS to Vivado along with the hardware on the quads and their functionality. The ground station team will improve the backend of the system to allow for real time tracking and simultaneous multiple vehicle flight. The controls team will continue to develop instructions describing how to parameterize the system as well as a new LQR controller. The continuous integration team will help add new tests for each hardware and software module,

as well as integrating the quad simulator. The subteams combined, will work 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 Appendices

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.

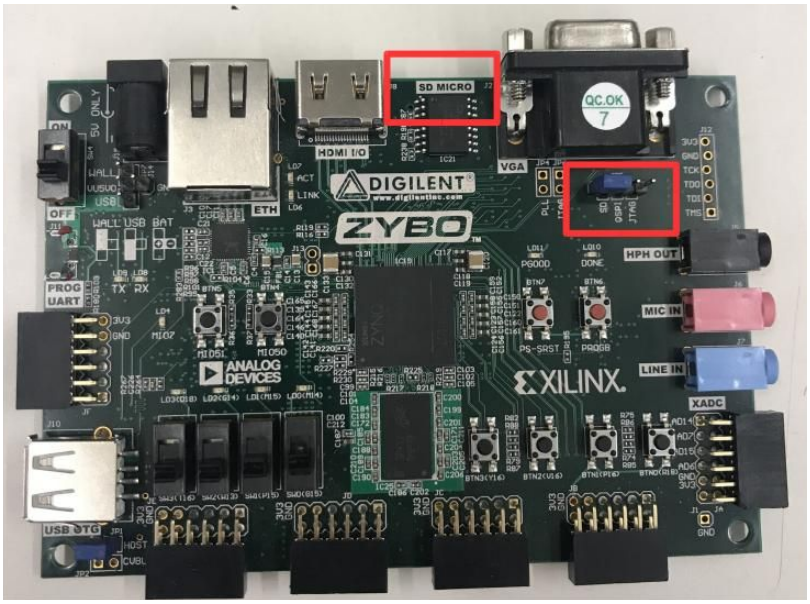


Figure 4-1: Zybo board

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.

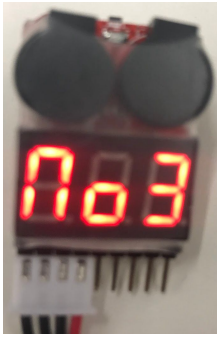


Figure 4-2: Voltage Monitor Connection

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.

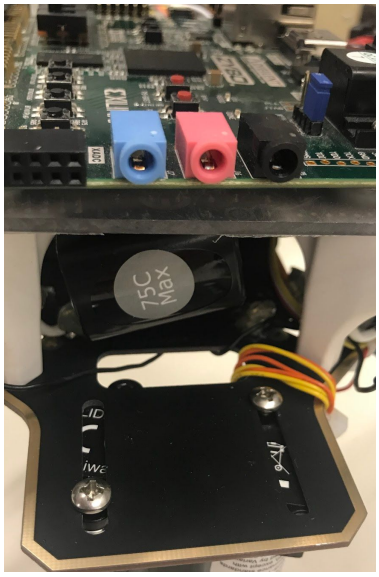


Figure 4-3: Battery Placement in the quad

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

Figure 4-5: RC Receiver when controller is connected

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

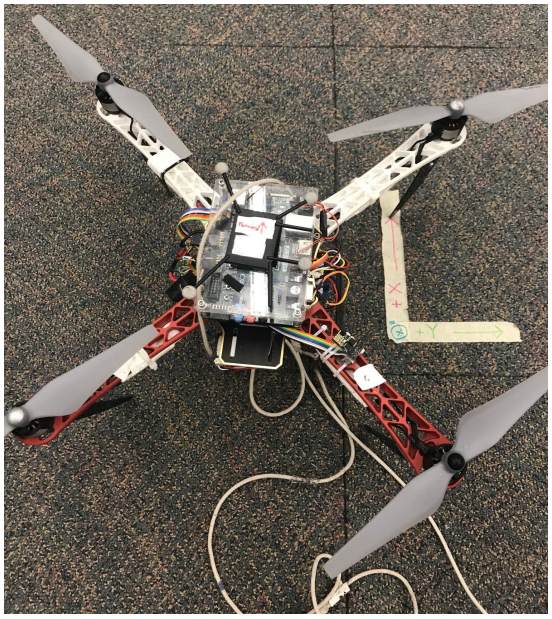


Figure 4-6: Quad tethered

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.

cd senior

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.

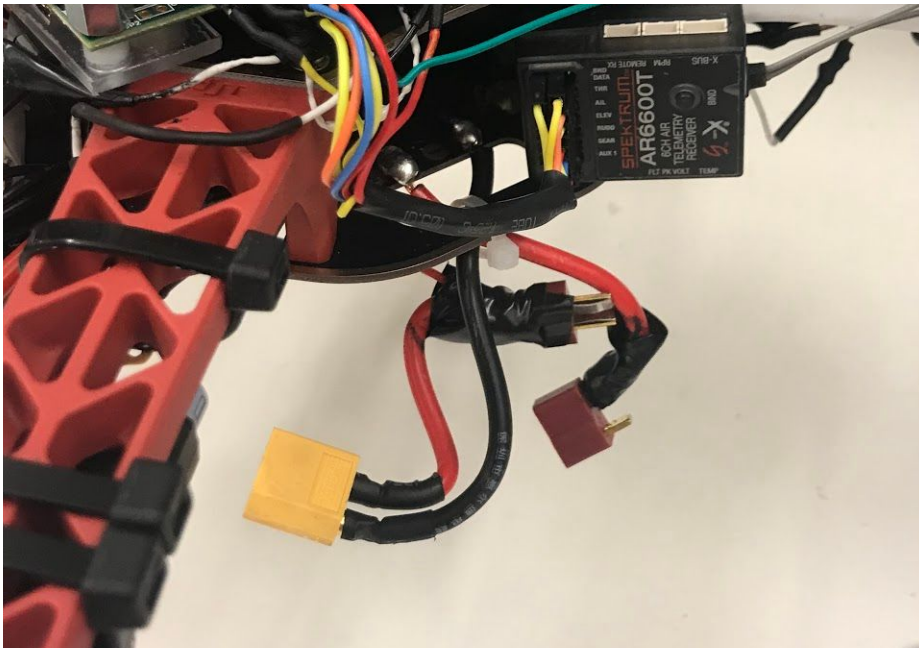


Figure 4-7: Deans connectors

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.