MicroCART

Design Document

Team Number: 50
Client: Dr. Phillip Jones
Advisers: Matt Cauwels, James Talbert
Team Members:
Evan Blough -- Technical Team Lead, Embedded Software Lead
Kynara Fernandes -- Ground Control Station Lead
Aaron Szeto -- Controls Lead
Joe Gamble -- Embedded Hardware Lead
Shubham Sharma -- Crazy Fly Implementation Lead, Website Manager
Jacob Brown -- Physical Hardware Lead
Team Email: sdmay20-50@iastate.edu
Team Website: http://sdmay20-50.sd.ece.iastate.edu/
Revised: 04/26/2020

Executive Summary

Engineering Standards and Design Practices

1. Follow the Principles of Programming by adhering to IEEE software development practices

- IEEE 1233-1996 IEEE Guide for Developing System Requirement Specifications
    - Used for determining system requirements in relation to client needs and functional/non-functional requirements
- IEEE 12207-2017 International Standard: Software life cycle processes
    - Used for involving our stakeholders in the development process and production of maintainable software.
- IEEE 1008-1987 Standard for Software Unit Testing
    - Used as a model for developing unit testing for our control software
- IEEE 802.1-2010 Standard for Port-based network controls
    - Used as a model for communicating with the drone

2. IEEE Code of Ethics provided by the IEEE organization

Summary of Requirements

1) Integration of a **secondary** quadcopter into the high-speed camera system,

2) Design and develop capabilities to support the implementation of the real world application by improving the GCS functionalities

3) Integrate Crazyflie platform into system

4) Implement GUI widgets to improve user interaction

5) Design tuning stand to easily tune Crazyflie platform

6) Configure existing system to fly multiple quadcopter and crazyflies simultaneously

7) Make an adapter that converts existing control commands to equivalent functionality in the MAVLink telemetry protocol.

Applicable Courses from Iowa State University Curriculum

List of Iowa State University courses which were applicable towards the MircoCART project:

- CPRE 288:
    - This course:
        - Goes over elementary embedded design flow/methodology
        - It gives students a basic understanding of micro-controllers.
        - Applications laboratory exercises with embedded devices.
    - We work with microcontrollers a lot with the project as evident from the project title. Hence, this course is a great foundation to assist us in the project.
- EE 230:
    - This course:
        - Is an overview of circuitry design and analysis
        - Gives students experience working with laboratory instrumentation and measurements
    - This is an applicable course as the project consists of circuit design knowledge from 230.
- CPRE 381:
    - This course:
        - Is an introduction to computer organization and evaluating the performance of computer systems
        - Datapath and control, scalar pipelines, and introduction to memory and I/O systems
    - The project consists of ARM-processors we would need to work with. And since this course is centered heavily around processors it is helpful towards the project.
- EE 330:

    - This course:
        - Goes over semiconductor technology for integrated circuits.
        - Analysis and design of analog building blocks.
        - Laboratory exercises and design projects with CAD tools and standard cells.
    - The project would require an understanding of circuit issues such as unmatched impedance and circuit devices such as current mirrors. Which is why this course is applicable for the project.
- EE 224:
    - This course:
        - Introduction to using Matlab for EE work
        - Analysis of Signals and various signal systems like LTI
    - This class teaches how to use Matlab and quadcopter controls are based in Matlab hence it is helpful for the project.
- EE 321:
    - This course:
        - Continuation of EE 224
        - Focuses more on modulation and data transmission
    - Focuses on data transmission which applies to how the quadcopter communicates with the controls and ground station, which is heavily used in the project.
- CPRE 488:
    - This course is about:
        - Embedded microprocessors and embedded memory
        - Component interfaces communicating to embedded software
        - Platform-based FPGA technology w/ hardware synthesis
        - And Real-time operating system concepts
    - This project requires knowledge with VHDL and C development on Xilinx platforms, multidisciplinary projects, embedded system design, and control systems on drones. Hence, why the course is applicable to the project.
- SE 339:
    - This course:
        - Software architecture.
        - Uses a raspberry pi to simulate a fleet tracking application.
        - Teached about how different components in a project should communicate and the best practices to do so.
    - This project requires knowledge of engineering and software architecture. Since various components of this project must communicate with each other like the camera system, the drones and the ground station. This course is applicable to this project in choosing the best architecture for our system

New Skills/Knowledge acquired that was not taught in courses

Qt software libraries - Qt is an open source GUI development framework built to run on C++ applications. Qt was used to create our graphical user interface to meet the needs of our research/development users.

RC communication - RC receivers and transmitters use defined methods of communication. Our application interfaces with an RC receiver. The receiver puts out 6 channel PWM signals. These signals are used to control drone behavior.

PID Controls - PID Control is not generally taught in depth without taking specific controls classes. PID control uses feedback to account for present, past, and anticipated error. These errors are assigned different weights depending on gain values, and the overall output to the actuators is recalculated. PID is one of the control algorithms our drone platform uses to correct its position.

Table of Contents

List of figures/tables/symbols/definitions (This should be similar to the project plan)

Term	Definition
CLI	Command-line interface
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
GCS	Ground Control Station, the application that runs on a host computer that communicates with the quad via a Wi-Fi connection and sends its 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	A 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 the 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

Shield Board/Breakout Board	PCB board designed by a previous MicroCART team that integrates with the Zybo board to interface with sensors and actuators used for autonomous flight.
MAVLink	MAVLink is a binary telemetry protocol used by several autopilot and ground control station platforms. MIT-licensed. Has several supported languages.
AV	Autonomous Vehicle
SITL	The software in the loop testing/simulation environment that served as the MAVLink adapter's test environment

1 Introduction

1.1 Acknowledgement

The development of this project has been aided by ECPE Faculty and graduate students at Iowa State. They offered valuable technical advice and insight. We would like to acknowledge their contributions to this project below.

- Matthew Cauwels
- Dr. Phillip Jones
- James Talbert
- May 2020 MicroCart Team

1.2 Problem and Project Statement

1.2.1 Problem Statement

MicroCART is a drone platform that will be used by graduate students to perform research on embedded control systems. The platform will also need to be used for demonstrations to professionals or future students to showcase the utility of skills obtained through the course of a EE/CPRE/SE degree. This platform will also need to be maintainable and modifiable by future senior design teams. Currently, the MicroCART quadcopter platform is functional, and it was to be extended to a copy of the current platform to implement simultaneous flight of two drones. Before working on the bigger drones we used smaller drones called Crazyflies to test swarm flight capabilities as well as drone axis rotation with our created 3-D printed tuning stand. In order to achieve these objectives, we needed to copy, refine, and modify the current platform to have a more intuitive interface and more accurate flight performance.

Figure 1.2.1: Current drone implementations

1.2.2 Solution Approach

To implement our solution , we furthered our knowledge of the previous development done on the project. We read the documentation of the drone's subsystems and inferred the functionality of portions that weren't documented. We also elected focus areas for each team member to specialize in areas of drone development. Once each of us were familiar with our respective subsystems, we started building the second platform.

There is also a smaller drone platform that we used to familiarize ourselves with navigation called Crazyflies. We hoped to implement swarm flight with the smaller Crazyflie drones, and once we had experience with the Crazyflies we would have been able to move on to navigating the larger drone. By the end of the first semester, we hope to have both large drones operationale.

The second semester consisted of us adding new features to the drone to fulfill the purposes outlined above in the problem statement. The GUI and Embedded Software leads worked on extending the functionality of the current GUI tool by adding new widgets like the datalog visualizer, PID sliders, and Draw to Flight. The team worked to implement swarm flight for the Crazyflie and quadcopter platforms, but the COVID restrictions on entering the 3050 lab halted progress to full completion. A MAVlink adapter for the existing backend software was developed to make up for loss of progress on the swarm flight deliverable. As a team we worked to add Linux to the second core to promote development on the quadcopter platform. This second Linux core

could have high level C++ libraries like OpenCV that could be used for soft real time object tracking. The Embedded Hardware, Controls and Physical Hardware Leads worked on implementing a small drone flight tuning stand to help future MicroCART teams and CPRE 488 students tune smaller drone platforms like the Crazyflie.

1.3 Operational Environment

The operational environment for the quad is the inside of Coover 3050, and this room has a VRPN camera system that can be used for drone navigation. For indoors operation there will not be any hazardous environmental factors like extreme heat or cold. The environment in the lab will have people in it, and the drone could collide with expensive equipment in the lab, so failsafes for flight faults are important considerations. Due to the fact that the operation environment is identical to its use environment, we expect little design difficulty caused by the operational environment

1.4 Requirements

1.4.1 Functional Requirements

The functional requirements for the project are different for each feature being implemented into the existing system. The functional requirements for the swarm flight were being able to connect both quadcopters to the backend software at the same time via TCP Socket, where the quadcopters are configured as servers. The quadcopters have the requirement of handling setpoints and parameter requests from the backend commands getparam and setparam. Both quad copters and crazy flies with swarm flight should be able to be given a grouping of setpoints that have no intersection. The quadcopters and crazy flies were to navigate to these setpoints and avoid collision while operating simultaneously. The extensions to the existing ground control station also have functional requirements.

The functional requirements for the PID Sliders are as follows. The PID Slider widget was added into the existing GUI tool as a Qt Widget Class. This widget generates formatted commands (Format Described in Appendix 6.3.8) to set the corresponding PID gains on the quad's control graph (Control Graph Figure in Appendix 6.3.3). These commands are sent over to the quad using a TCP socket maintained by the Backend software. The Datalog Visualizer widget takes in flight logs generated in the quad's data logging format (Format Described in Appendix 6.3.2). These log files are parsed and the resulting data is displayed to screen based on user input, which will be expanded on in the UI requirements section. The Draw to Flight Widget reads a pattern drawn in the widget and converts this pattern into corresponding setpoints for two axis navigation. These setpoints are then used to generate a flight script which can be saved as a bash script and be executed later.

The functional requirements for the mini-quad tuning stand are as follows. There are multiple types of stands, one type of stand will support single axis rotation another will support multi-axis rotation. These stands should be able to hold small drones and endure stress induced by their movement. The stands have sensors that will read information from the drones current orientation. A Microcontroller unit will process the sensor readings to display movement statistics over a command line interface.

The MAVLink adapter also has functional requirements. This adapter will operate as a server for a TCP socket. This socket will establish connection to the backend based on the configuration of the config.c file. The MAVLink Adapter will handle getparam, setparam, and getnodes commands from

the client interface. The MAVLink adapter will act as a TCP server for a MAVLink device. The MAVLink adapter will convert the client commands to appropriate functionality to set waypoints and navigation behavior for the MAVLink platform. The MAVLink adapter will also be able to set on-board parameters with the getparam and setparam commands.

1.4.2 Non-Functional Requirements

The Non-functional Requirements for the swarm flight features are that they should communicate with the GCS wirelessly with an average latency of 2.2 ms (See References [2]). The vehicle platform's stable points should not vary from the setpoint coordinates by more than 1.5 feet to quantify accurate performance and maintain safety regulation. The GUI widgets should respond to user input within at least a second to ensure usability.

1.4.3 UI Requirements

The UI requirements for the Datalog Visualizer are that a user should be able to select a log file from a file select window. A user should be able to specify the min and max bounds for each axis. The user should be able to select multiple axises to be displayed. The user should be able to view the name and units of each parameter using a drop down menu. The Draw to flight widget should allow a user to click and drag patterns on the screen. The user should be allowed to clear this pattern at any time. The PID Slider allows a user to slide between minimum and maximum values for the parameter.

1.5 Intended Users and Uses

The platform will be used to test different control systems and their varying efficiencies for students. Students will require quick, simple interfaces to maximize learning and reduce time use for lab. The system will also be utilized in future senior design projects, requiring our team to practice good documentation and written communication techniques. Finally, the project will be utilized in college demonstrations as a means of showing off to our employers, alumni, and visitors what our college can do as well as an attempt to recruit future students.

1.6 Assumptions and Limitations

1.6.1 Assumptions

- We wouldn't have more than two large quadcopters in our operating environment
    - Any further maybe too hazardous/complicated
- The operating environment indoors wouldn't contribute any significant hazards
    - Humans acting in a hazardous manner might complicate this
- We assumed that standard components we buy from manufacturers come in working condition
    - This assumption may not be valid, but would save time for excessively testing every single electrical component
    - This assumption cost us time because we assumed an IMU reading was valid, but severely impacted flight performance

1.6.2 Limitations

- Drone flying environment limited to the area of the camera sensors within Coover 3050
    - The flight is dependent upon cameras tracking drone
- The computational power of the drone

    - Limited in processing time
    - Limited in memory access speed
- Feedback delays in control systems by the camera systems
    - Delay in ms expected
    - From Wi-fi communication
- Battery to payload ratio imposes limitations on overall runtime
    - Battery technology and weight characteristics of drone limit flight to appr. 5 minutes
    - Frequent replacement of battery systems necessary

1.7 Expected End Product and Deliverables

The product is academic in nature so there aren't commercialized descriptions.Detailed Documentation has been provided through gitlab markdowns and documentation generators like doxygen.

1.7.1 Implementation of Second Drone

We are going to make an exact copy of the current drone platform from the previous year. Most of the work done on this portion will be in the form of research on the existing system. In order to make another platform we will need to research the existing documentation. We are making this deliverable, because we need to have one stable platform to demo our solution to people and one for a senior design team to do development on. This deliverable was completed on October 25th.

1.7.2 Swarm Flight of Crazyflies

After we have a second drone, we will hopefully have access to several crazy flies. We can use these platforms to test out extending our GCS software to multiple platforms. This is an intermediate deliverable. We will use this deliverable to get experience with programming swarm flight with inexpensive platforms and move on to our permanent solution for swarm flight. Crazyflie flight was achieved midway through the first semester and swarm flight was worked on for the rest of first semester and about half of the second semester before Covid-19 stopped us.

1.7.3 3-D Printed Tuning Table

For testing our Crazyflie drones, we will design a 3-D printed tuning table. This table should be able to rotate on various axes so we can test our Crazyflies rotation abilities without worrying about crashes. Work on it began during the second semester and was finished about 2 months into it.

1.7.4 Swarm Flight of large drones

After we have experience with swarm flight on the crazy flies and have the second drone built, we will work on having two drones operate simultaneously. This deliverable relates to our client's need of demoing to students and faculty. Showing multiple vehicles operating simultaneously is more impressive than just a single vehicle. This showcases the usefulness of skills obtained with an ECPE degree. Controls students also will want to test control algorithms with multiple platforms interacting. This was planned to be finished by the end of second semester but Covid-19 stopped our progress.

1.7.5 Draw to Flight

The Draw to Flight feature was implemented into the existing GUI interface. This widget is responsible for reading a user drawn pattern and converting it into equivalent setpoints. These setpoints are converted into bash scripts that can be executed to send commands to the quadcopter platforms to get them to move to the desired setpoints.

1.7.6 Data Log Visualizer

The Data Log Visualizer uses the QCustomPlot open source widget to construct graphs of the recorded flight data from the quadcopter platform. The flight data will be in a specific format shown in the (See Appendix 6.3.2). The user will be able to select which parameters to be graphed. They can set the min and maximum bounds they want graphed. Dropdown menus will contain a list of parameter names and units. A user can select multiple y axis parameters to be plotted. This widget will help future MicroCART teams and Graduate Students debug complex flight issues.

1.7.7 PID Slider

The PID Slider Feature allows a user to send new PID gain constants to the quadcopter platform. The PID gains correspond with the parameters shown in the (See Appendix 6.3.3). The Slider Widget uses the MicroCART communication protocol to update the parameters of the quad. (See Appendix 6.3.9) This widget will help future MicroCART teams and Graduate Students be able to tune the PID coefficients on the drone's control algorithm to improve flight performance.

1.7.8 MAVLink Adapter

The MAVLink Adapter converts the existing client commands (getparam, setparam, and getnodes) into a format that is compatible with the MAVLink open source project communication protocol for autonomous vehicles. The Adapter supports the MAVLink microservices used for getting and setting on-vehicle parameters, sending waypoints, and selecting flight modes. This deliverable will be evaluated for completeness using the Ardupilot open source project's Software in the Loop(SITL) testing environment.

2. Specifications and Analysis

2.1 Completed Design

The work completed during the second semester was reduced due to the closing of the labs. Before this happened development was made towards swarm flight, testing apparatus for crazyflie, and functionality of the groundstation. Expectations were discussed with our client with limited resources and milestones were set accordingly. The methods in which these tasks were accomplished involved using Agile methods to coordinate coding work and discord for communication. During the later half of the semester Zoom was used extensively for face-to-face interaction.

2.2 Design Analysis

During the first semester design portion of the project we have worked to understand the current implemented system. Each component of the system has been built over many years and

passed through many teams. It takes time to understand each portion of the system to an extent at which it can be utilized; therefore, the majority of the time has been spent reviewing documents.

Our ability to process the current documents has progressed well into the second semester. Each person on the team has demoed the drone and has read documentation. We had begun to complete initial work including soldering, control system analysis, gui interfaces, and initial design document creation. Such completed portions of the design were a network setup to communicate with multiple drones, modularity of widgets within the GUI interface, and design of testing systems for crazyflie control development.

Strengths

- Team comradery and communication
- Good mix of disciplines w/ large knowledge base
- Contacts with previous MicroCART users to answer questions
- Large group to distribute work

Weakness

- Large number of systems with intricacies
- No control theory knowledge base
- Poor documentation
- Limited timeframe to implement
- Little actionables with resources cut-off due to Covid-19

2.3 Development Process

After being assigned our project, we met with our customer and advisor to discuss the requirements for our project. Since we're working on an embedded systems project, our system has multiple components that would have different requirements that would change as we move through the school year.

We have completed the design phase of our project. Research was made in subjects control theory etc. to help us understand our customer/users better and in turn, produce a better tailored product.

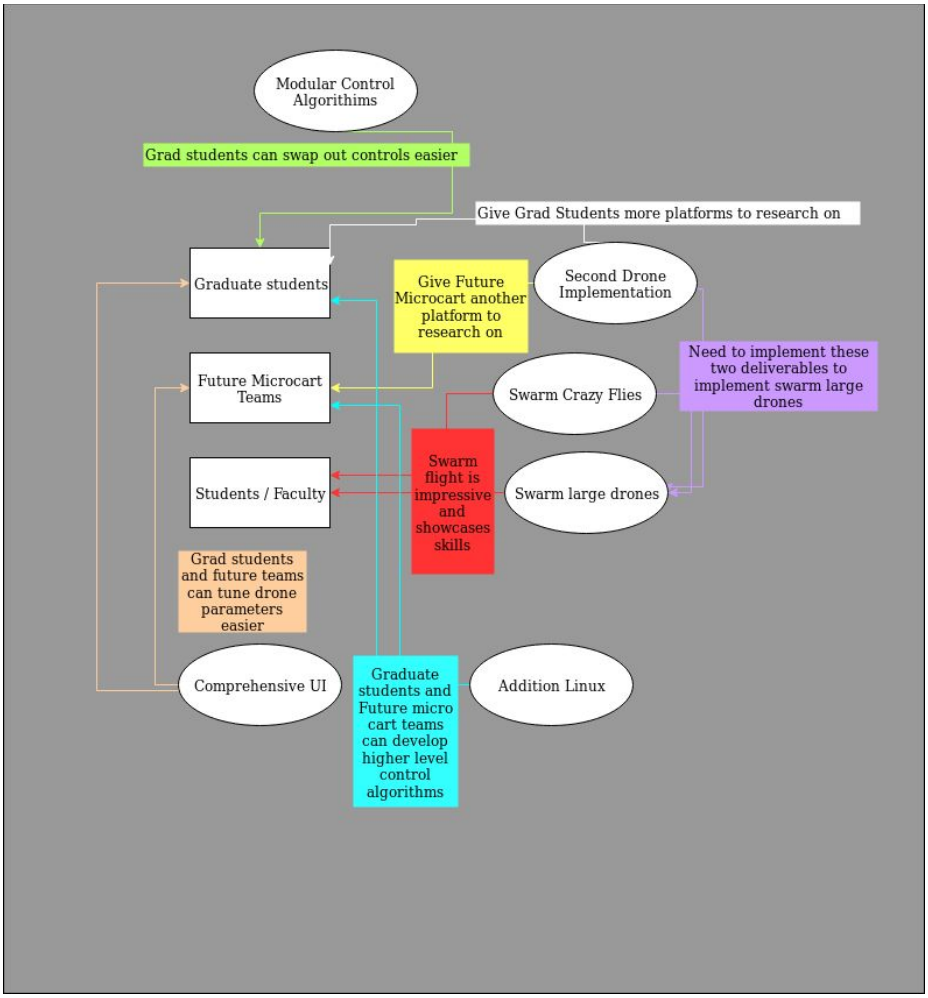
After the design phase, the implementation phase began. The implementation phase involved taking design and making them a reality. This was done using the research gained during the design phase, expected tools for the tasks, and solutions along the way. Unfortunately, we have not gotten out of the implementation phase as many of the pieces in development remain so due to restrictions in what can be worked on.

Through the pieces of change during the phase of the year, we have utilized an agile development process. The plan is as such:

- **Tickets:** We have used Trello to handle our tasks and keep track of the tasks that need to be done, the tasks that are in progress and our accomplished tasks.
- **Meeting with stakeholders:** We have met our customers, users, and advisor once a week on Mondays to discuss our accomplished tasks and the concerns or goals they have for the coming sprint or sprints.
- **Sprints:** We have used 1-week sprints in which we will go through one iteration of standup, grooming, retrospectives and stakeholder meetings.
- **Grooming:** After our stakeholder meetings, we regrouped, and created and prioritized tickets on Trello for the new sprint.
- **Stand-up and Retrospective:** We started our weekly team meetings with scrum, where we talk about what we did, what we planned on doing and if we had any blockers for that

sprint. Since we are full-time students, we compromised on doing this weekly instead of daily.

2.4 Design Plan



3. Statement of Work

3.1 Previous Work And Literature

Microcart is an ongoing project at ISU that has been worked on by generations of students. Previous teams have already built quadcopter drones, testing platforms, and supporting control software. The previous year's quadcopter design was used as a starting point for how to build and modify a new version of the quadcopter. Having an existing system to build off of benefited us by abstracting work needed to implement deliverables. For example, instead of having to write a

function to open a socket between the backend and a connected device, a previous team's function could just be modified to meet our needs.

There were also times where having an existing system had pitfalls. The system is very large and complex. It took a very long time to learn a subset of the relevant work done by previous teams. Even after researching the code in the repository, there were some issues where code was in a not working state. Some versions of the backend were missing some crucial source files that prevented compilation. This resulted in rolling back the software for the backend. Working within a large repository passed down between different teams has led to several redundant functions within the source.

3.2 Technology Considerations

Highlight the strengths, weakness, and trade-offs made in technology available.

Discuss possible solutions and design alternatives

Strengths

- High performance processing
    - Complex control integration
    - Visual data computation
    - General quickening in data processing
- Flexibility of using FPGA with dedicated processor over just using a micro controller unit for embedded controls
- New batteries with improved battery life and energy densities
- High performance motors
- Large existing software libraries
- Open source software
- Camera system with high accuracy positional data

Weaknesses

- Cost of new material and parts
- Weight added to quadcopter resulting in lower battery life and risk of loss of flight
- Controls and flight performance are very dependent on
    - Quadcopter weight
    - Motor resistance
    - IMU calibration
    - Properly tuned PID gains
- Increased dependency on a large volume of different technologies results in a convoluted mashing of technologies that is not easily digestible by future teams/students

Trade-offs (Summary)

- Cost and complexity for more advanced solutions with return on drone abilities
- Usage of new technologies tends to increase the abilities, but increases weight, power, and generally reduces performance of the drone

3.3 Task Decomposition

3.3.1 Drone Construction

Leveraging of current drone hardware buildup and extra parts to create a new, fully functioning drone. The drone buildup will require the use of knowledge in electrical hardware skills and some mechanical understanding. The information on how to build-up the new drone will come from observations of the previous drone.

3.3.2 Multi-Drone Flight

The new drone built from the drone assembly process will be flown in tandem with the current working drone. The crazyflies in development are also going to be flown with the large drones. This will all require code for the drones to recognize each other's position and ensure no collisions occur. We will also have to modify the current networking scheme. The GCS and drone interact only with each other, so to add multiple drones we will need to reconfigure the networking scheme to an appropriate type [2].

3.3.3 Draw to Flight Widget

The draw to flight widget was a subtask under the comprehensive UI deliverable category. This task involves using the Qt framework and existing GCS GUI to implement a widget that allows a user to draw a flight pattern on the screen. The resulting drawing coordinates were converted into a flight bash script that could then be executed to transmit setpoints to the quadcopter. The quad would then fly to these setpoints. One assumption of this feature, is that the quadcopter controls were tuned enough to accurately navigate between points. This assumption was invalid because the controls on the quadcopters were only accurate to extremely simple patterns.

3.3.4 PID Slider Widget

The PID Slider Widget was a subtask under the comprehensive UI deliverable category. This task involved creating another widget in the GCS GUI that would be used to easily set the values of the P, I, and D gains of the quadcopter control algorithm. This feature would ease the tuning process for users and developers. This task was implemented by testing the UI elements separately to confirm reading and setting values. After the UI portion was finished, the backend functionalities were tested by checking reads and writes to the quadcopter board. Finally, integration tests were done to confirm the whole widget worked as intended.

3.3.5 Data Visualizer Widget

The Datalog Visualizer Widget is a subtask under the comprehensive UI category. This task involved making a graph for showing recorded flight logs of the drone. The flight data contains information about orientation, and resulting outputs to the actuators. Logs can also be used to quantify flight performance by viewing rise time, overshoot, settingling time, steady state error and stability between the real position and setpoint positions.

This task was implemented using the QCustomPlot software with Qt. QCustomPlot is under GNU general public license. This functionalities defined for this task are as follows, the widget reads in log data formatted in the drone's log format(appendix), the widget displays this data to the screen depending on which axes and min max bounds the user sets for the x and y axes, and finally the widget should be able to display multiple plots on the y axis and select a log file from a log file selection window. This task was added later in the semester to account for lack of progress on the multiple drone flight due to COVID restrictions.

3.3.6 MAVLink Adapter Set and Receive Waypoints and Flight mode

The implementation of setting and receiving waypoints from a mavlink device is a subtask of the MAVProxy adapter deliverable. This task involved writing C functions that used the Mavlink 2 C library to communicate with the platform to send navigation commands. The adapter maintains a two dimensional array of coordinates and transfers and edits these coordinates when a user uses the ./getparam and ./setparam commands of the ground control station. This task shares dependency with the set and receive MAVParamters, because they both will be controlled with the getparam and setparam command line args.

3.3.7 MAVLink Adapter Set and Receive MAVParameters

The implementation of setting and receiving MAVParameters is a subtask of the MAVLink adapter deliverable. This task included functionality for getting and setting the MAVParameters of the Mavlink platform which could be used for tuning navigation behavior. These parameters could be something like sensor, motor, or compass offsets.

3.3.8 Mini-quad Tuning Stand Model

Designing the model for the stand is a subtask of the Mini-quad Tuning Stand deliverable. The stand can be used for the tunning of the crazyfly and CPRE 488 drone platforms. The stand was designed with Autodesk inventor and had to have correct proportions to allow freedom on 3 axises of rotation.

3.3.9 Mini-quad Tuning Stand Microcontroller and Sensor Processing

Work was performed on the design of a turntable system to adjust the control system of mini-quads with focus on usage for the crazyflies. Two systems were generated: a single axis system and a gyroscopic system. Both would utilize the TIVA C-series Launchpad. Encoders that would measure the rotational speed and angle would interface with the microcontroller via the onboard ADC and QEI (Quadrature Encoder Interface). Unfortunately code was not fully developed as the sensors remain in the working lab. 3D models have been generated and printed.

3.3.10 Crazyflie Adapter

Crazyflie Adapter was partially worked on by a previous senior design team. This semester we used the old version of the adapter and modified the code to build on our new machines. The adapter was set up to work with the newest Crazyflie version. We also have set up a VM to run the Crazyflie CLI on Ubuntu so we have a sustainable release of the working project. We have then set up port forwarding in the VM for passthrough for communication between the adapter and the camera system.

3.4 Possible Risks And Risk Management

3.4.1 Equipment Risks

There are several things that have impeded the progress on deliverables. A large amount of time was spent trying to get the computers in Coover 3050 updated to ensure future teams could develop easier. One of the machines in the lab was 15 years old.

A large amount of progress was lost on the multiple drone flying deliverable because an assumption was made earlier that the calibration of the IMU unit on the drone would not significantly impede flight performance. This assumption was false. A large amount was spent trying to get the second drone to fly with stability and accuracy, but the IMU readings were off, so it would lurch forward and flip.

One of the RC Controllers broke part way through the semester, so the second drone needed a failsafe because there was only one working RC Controller. We resolved this issue by using one of the CPRE 488 controllers instead, but this was time consuming because the mappings for the controller and receiver didn't line up.

The Multiple drones flying deliverable was dependent on access to the 3050 lab, so COVID restrictions resulted in these deliverables not getting completed.

3.4.2 Knowledge Area Risks

Using the previous team's software introduced several knowledge risks throughout the semester. The code that was given to us took a long time to understand. We had to use some frameworks we were unfamiliar with like Qt, QCustomPlot, Mavlink C library, and the CrazyFlie library. We also had to use development environments we weren't familiar with like Vivado, XSDK, and Code Composer. We had limited knowledge on control tuning for the Quadcopter platform. Some of our deliverables were dependent on the quad's performance, so that significantly impeded progress.

3.4.3 Costs Risks

We had several dependent hardware components that we needed to buy to support our drone applications. Buying these components halted our progress because we have to go through a purchasing procedure and justify our need for these components. Some components broke or malfunctioned, so we had to buy new ones. We had to buy custom circuit boards, RC controllers, Lithium Polymer batteries, etc. We have to wait for these things to be purchased in order to develop. We had to buy several new materials to construct the mini-quad stand.

3.5 Project Proposed Milestones and Evaluation Criteria

Key milestones for our project are

1. Having multiple autonomous Crazyflies controlled with our GCS software
2. Build our own Quadcopter drone
3. Swarm flight with the Quadcopter
4. Implement the Mini-Quad Stand
5. Implement Linux to second core
6. Improve UI
    a. PID Slider Widget
    b. Draw-to-Flight Widget

    c. Datalog-Visualizer Widget
7. MAVLink Adapter
    a. Set and Receive Waypoints
    b. Set and Receive MAVParams
8. Mini-Quad Tuning Stand
    a. Physical Model
    b. Microcontroller and Sensor Processing

The testing procedure for evaluating the success of each milestone is below.

3.6 Project Tracking Procedures

Our team has multiple ways of tracking our project's progress. We use Trello to communicate what needs to be done, their deadlines, and by whom. Our client also requires us to write weekly reports to track our one week sprint's progress and a meeting to discuss the sprint. We have a scrum-style stand-up every week to ensure that our team members are making significant progress, that no progress is being blocked and to plan for collaboration.

3.7 Expected Results and Validation

The desired outcome of this project is to implement the deliverables described in section 2 to service the needs of the user groups shown in the design plan diagram. The user groups are future Microcart teams, controls graduate students, and flight demonstration viewers touring the university. Each task can be mapped to the needs of a specific user group.

Building our own quadcopter serves both future teams and graduate students. This has been evaluated by doing integration tests with the existing control software of the ground control station, unit tests with the XSDK with the debugger tools to ensure sensor reading fidelity, and the final system test has been repeatedly running flight scripts with the drone. The feature has been considered a success because the quad is able to fly within the same degree of accuracy as the previous year's drone. This degree of accuracy can be described as a stable point being reached within 2 minutes and max variance from setpoints of less than 3 feet under normal flight circumstances.

Implementing swarm flight for the large quad copters was never completed, but it would have shown the value of an ECPE degree to the flight demonstration viewers. We were partially successful because we were able to perform an integration test with two drones that proved that they could receive flight instructions with the getparam and setparam commands. There is a url link to a video of the test listed in appendix 6.3.10. The drones can be seen reacting to individual commands without props on. Flight tests with simultaneous connection failed because of instability. We were unable to analyze flight logs to diagnose the issue before the COVID restrictions stopped our lab access.

Implementing a mini-quad stand would benefit CPRE 488 students and future Microcart teams. We tested the physical model by attaching a Crazyflie and rotating it to ensure the printed model could handle the stress and allow the Crazyflie to move freely. The stand passed this evaluation.

The UI additions we made all service the needs of future Microcart teams, controls graduate students, and flight demonstration viewers. The PID sliders will help future Microcart teams and graduate students tune flight performance. The PID slider had the frontend and backend unit tested. The frontend test was that the GUI is able to use the slider bars to set values. The backend unit test tested setting and receiving these values to the quad platform. This was verified by the xsdk debugger in a video in the appendix.

The Datalog Visualizer helps future MicroCART teams and Controls students by allowing them to solve complex flight issues. Graphing actual versus setpoint values allows both groups to tune the drone's PID easier. The Datalog visualizer was tested at a high level by using the application to look at current and previous years log files to ensure compatibility. The datalog visualizer passed it's evaluation.

The Flight to Draw Widget shows the value of an ECPE degree to flight demonstration viewers. It is something that can be shown to impress people that visit the university. We tested it by drawing a simple line and watching the flight follow the path. It worked for simple paths only. We also confirmed the setpoints it generates are correctly formatted. The accuracy of the drone bottlenecks the performance of the widget.

The MAVLink adapter will give future MicroCART teams that may want to test a new platform a very rich library to start with. Instead of hand designing hardware, firmware, and software from scratch, they can use a MAVLink platform in conjunction with the backend to do some autonomous navigation. This will allow them to implement higher level functionalities like GPS navigation or image processing navigation. This feature has working getnodes, getparam, and setparam commands for the adapter. The adapter has worked with the Ardupilot SITL testing.

4. Project Timeline, Estimated Resources, and Challenges

4.1 Project Timeline

First Semester Timeline

Task/Week	1	2	3	4	5	6	7	8	9	10	11	12	13
Drone Assembly													
Board Assemble													
Chasis													
Testing boards													
Zybo refitting													
Interconnect assembly													
Crazyflie development													
Documentation of research													
Autonomous flight													
Integrating with GCS													
Multiple drone flight													
Ground Station Development													
Implement new PID features													
Documentation for future teams													
Reimplementing lost work													
Features for different controllers													
Project clean-up													

Second Semester Timeline

Task/Week	1	2	3	4	5	6	7	8	9	10	11	12	13
Multi-Quadcopter Flight													
Restructuring network													
RC transmitter setup													
Configuring GCS as AP													
Testing and tweaking													
Crazyflie Development													
Develop GCS adapter													
Swarm communication													
Crazyflie firmware update													
Virtual Machine Integration													
Crazyflie integration													
Port forwarding													
Ground Station Development													
Draw flight path													
Data-logger visualizer													
MAVLink													
Miscellaneous													
Rotational Stands													
Project clean-up													

4.2 Feasibility Assessment

On planning our tasks for the full Spring semester, we would have been able to complete all our tasks. However, with the circumstances of the pandemic at hand, we were unable to physically be in our lab and complete our tasks. We also didn't have much of a fall back plan and ended up finishing miscellaneous tasks, documentation and tutorials for future teams.

4.3 Challenges

Since we had our lab access revoked due to the recent pandemic, we needed to put a halt on some of our major requirements such as multidrone flight etc. These features were not fully completed.

5. Testing and Implementation

5.1 Interface Specifications

Our project requires a wide array of interface specifications. Another specification is an improved GUI to be more user friendly for future MicroCART teams. There is a module in the GUI to allow the user to plot out a flight path for more dynamic flights. Another specification are the Matlab and GUI data analysis tools. They take flight data from a text file in a specific format. (See Appendix 6.3.2)

5.2 Hardware and software

There are multiple software interfaces used for this project. This means multiple interfaces are used for debugging, interacting, and developing on these projects. The software worked on includes the C/FPGA controls application on the quadcopter, C/C++ code on the GCS, and C++ Qt Creator environment with GDB debug tools.

- The Matlab interface outputs errors and expected values easily to the screen. Matlab interface specifies debug functions that can be used to trace computational errors in our simulation software.

- Vivado and XSDK are interfaced with the embedded C and Programmable logic on the Zybo 7020 board. Both Vivado and XSDK have comprehensive testing functionality. XSDK can use JTAG/USB interface to debug embedded software on the FPGA board.

- There is an interface with the C++ GUI application with the QTCreator IDE. This IDE has the Qt Test library, which supports a testing interface similar to JUnit tests. This tool also allows the user to set breakpoints in the code and use GDB to debug segfault errors.

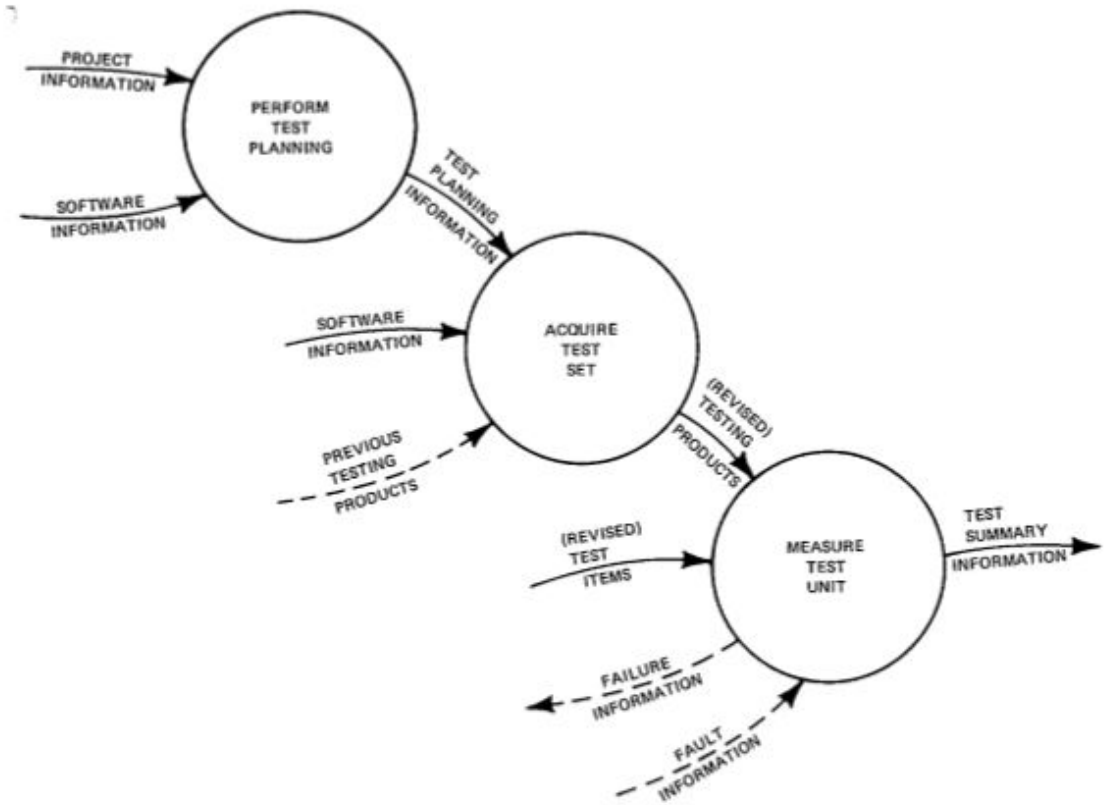
- The project consists of hardware that would test/debug interfaces. This is achieved as the printed circuit board we use has several copies of outputs. This will allow us to interface the hardware with oscilloscopes to take traces to resolve complex issues.

5.3 Functional Testing

5.3.1 Unit Testing

During our development and testing process, new and existing software modules that we created were tested in accordance with IEEE software unit testing standard.We will follow the flow

diagram below for developing comprehensive unit tests for modules both on the GCS software, embedded platform software, and Matlab simulation software.



This unit testing scheme was revised to include unit testing for critical hardware components. Unit testing has already been done for many of the hardware components on the drone. Most of our software development has been in C/C++, C functions were unit tested in separate executables before moving on to integration tests within the existing software systems. Typically during the designing and testing process, c functions could be individually tested to confirm expected results and then integrated into the existing system for integration testing. Some code was hard to develop unit tests for because the code's inputs were not easy to replicate outside of the existing system.

5.3.2 Integration Testing

After a software/hardware module made it past successful unit tests, we assumed that a module works, but the only way to verify its functionality after this point is to perform integration tests. Some integration tests had already been conducted for system submodules. We ran unit tests on our Breakout Circuit board and Zybo board, so after they were verified we ran integration tests with the two programmed boards interfacing with one another. These tests confirmed functionality for the Lidar and IMU hardware modules within the board integration environment.

5.3.3 System Testing

When substantial progress had been made on the development of the project, we ran system testing to confirm that the drone solution has fulfilled our design requirements as stated earlier in this document. Another one of the tests we developed was a test script that generates flight paths for the drones. When multiple drones fulfill this task independently, they were to be integrated together. Due to the embedded nature of the project it is difficult to describe the exact inner workings of a system test entirely. There are several videos of system tests for most deliverables shown in the videos section of the appendix (Appendix 6.3.10). The MAVLink adapter's end to end functionality was tested in the SITL. The system tests were considered a success if the correct outputs were produced by the three test scripts while the MAVLink adapter was running. There was a system test done for the PID Slider that has a video. The system test involved starting up the widget and doing a write and read to confirm the drone PID values were updated.

5.3.3 Acceptance Testing

Acceptance testing was performed up to the progress we had made during the first half of this last semester. Although not all of our expected tasks have been completed and passed acceptance testing, we were able to complete acceptance testing for connecting multiple drones to the network, creating new GUI interface tools, and a draw-to-flight tool and the adapters.

5.4 Non-Functional Testing

The non functional requirements for our design were as follows: improving the usability of our GCS, improving drone flight responsiveness/latency, enhancing flight accuracy of the drone during autonomous navigation. These requirements will each have separate tests to ensure that their performance metrics are met.

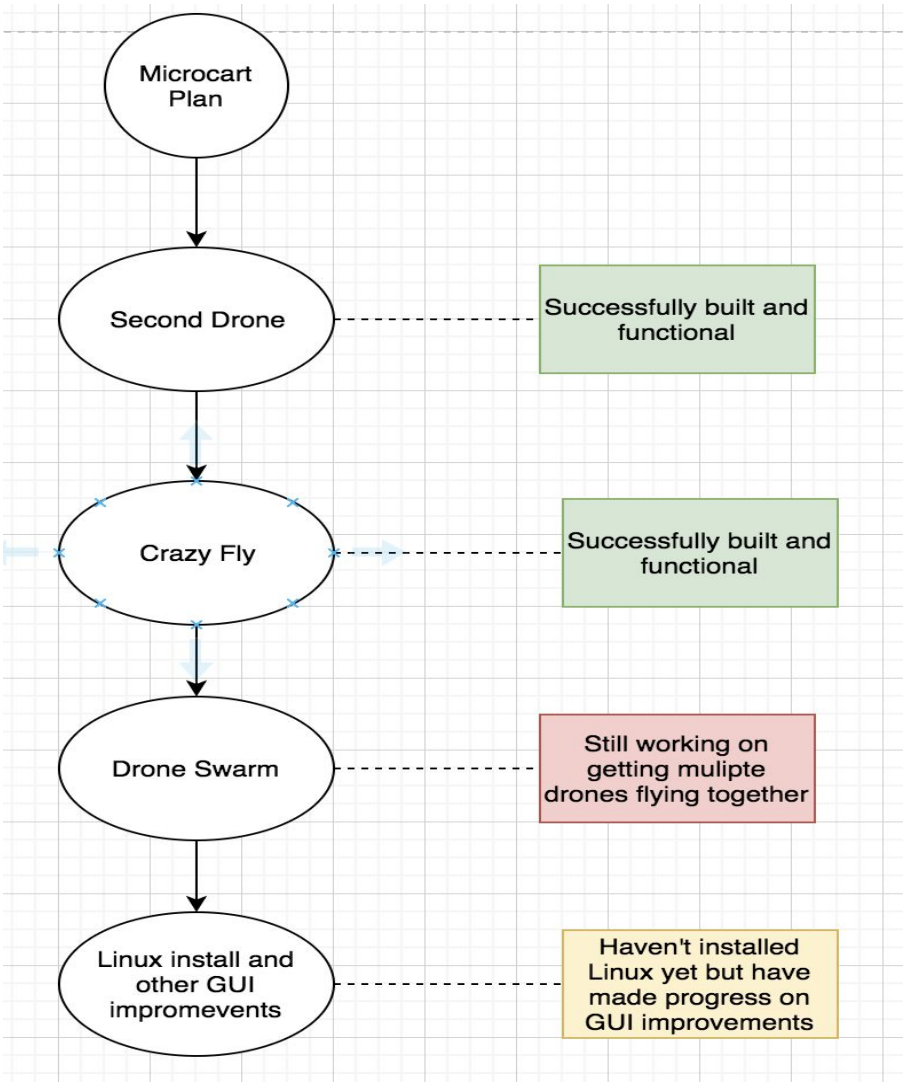
The utility of our GCS software had been evaluated by our primary user group. The GCS station has various functional requirements like the ability to send parameters to the drone, and the ability to receive measurements / data from the drone. We have been in contact with a graduate student that is in charge of emulating the needs of our primary user base. He evaluates the non-functional requirements for our GCS software like being able to find functionalities easily.

In order to fulfill the nonfunctional requirement of latency, we have had to specify exactly what type of latency is being examined. The drone platform currently has log files that record sensor data such as, actuator output and user inputs. If a setpoint is given to a drone platform like the quadcopter or Crazyflie, the latency can be examined by observing the time between when a command was sent from the GCS and when the command was received by the quad platform. The latency of the current system with a networking configuration of one drone is on average 2-4 ms from the ground station and back. We made progress as we connected them through the client and were able to have two on the network, but never confirmed throughput to the two drones during flight.

Flight accuracy metrics were analyzed with the drone's data logging capabilities. The VPRN camera system will give an absolute reference to compare actual location to setpoint values. A drone navigating on an autonomous path shouldn't vary more than half a foot from its intended location on any axis. Any more variance in flight path could introduce danger during swarm flight, and it would make the project ineffective for autonomous navigation research.

5.5 Process

We used a variety of tests to check on our second drone. Some examples of this include testing the motors individually and all at once and testing if the drone worked with the given scripts. We ran similar tests on the Crazyflie but with manual flight instead of scripts. The other methods like drone swarm and GUI improvements are still a work in progress and we were still in the process of testing them when the lab got closed.



5.6 Results

– List and explain any and all results obtained so far during the testing phase

- While testing for the current draw, I discovered that we are able to test across the entire drone power draw and divide by the number of motors to get the current draw of each motor. Remembering from my circuits class, this makes sense. since the controllers only use a couple milliamps, It is a negligible amount when considering the 10 amps RMS drawn per motor during flight.

- During the creation of the draw-to-flight PID widget, the program sent the coordinates to the drone that correctly defined the picture that was drawn, confirmed using the drone debugging platform that would list the coordinates.
- The PID allowed the user to select commands to send to the drone. The drone's debugging mode shows all the commands sent to it for easier verification. This is how the functionality of the PID controller was confirmed
- The Datalog Visualizer was able to read from old and new log files in the format specified in the appendix. It also fulfilled desired UI functionalities.
- The MAVLink adapter was able to process getparam, setparam, and getnodes commands from the MicroCART backend. The adapter was able to set GPS setpoints for the simulation vehicle and direct it to travel.

We've managed to make good progress on our Micro Cart project during this first and second semester. Our successful results so far include building a second working drone, soldering an FPGA, implementing some GUI improvements like sliders, building several Crazyflies and flying one of them, making a data analysis tool, connecting multiple drones to the network, design a draw-to-flight feature, and further network setup for multi-drone flight.

The results we are working on improving is getting two drones flying at the same time, getting a Crazyflie swarm working, and implementing more GUI improvements.

Some implementation issues we are running into is the lack of documentation for previous years code. This is a problem because whenever we try to run their code we get error messages so a good chunk of our workload this semester was trying to understand and fix the code. Similarly, we also have some issues on the hardware side of the project because there's no documentation or instruction on what we're supposed to do. We also had issues with our lab space since we need admin access on the computers to continue working and we're still waiting on that.

6. Closing Material

6.1 Conclusion

This project had the goals of implementing the second drone, being able to fly multiple crazyflie platforms simultaneously, developing a tuning stand for the crazyflie platforms, flying multiple large quadcopters simultaneously, implementing the draw to flight widget, implementing the datalog visualizer widget, implementing the PID slider widget, and implementing the MAVLink GCS adapter. Several of these deliverables have passed system testing phases, and others have had progress made but no completion. Some lack of progress on deliverables can be attributed to COVID-19 restrictions. Each of these features can be mapped back to projected user needs for all 3 of our user groups.

Implementing a second drone gives future MicroCART teams and graduate students a testing platform to develop on while maintaining another stable quadcopter that can be used for flight demonstration. This reduces future project overhead. Partial work and documentation done on the crazyflie adapter can be leveraged by future teams to fully support the crazyflies with the GCS software. The development of a tuning stand is not complete, but the physical model is complete. This physical model provides an adequate testing environment for future MicroCART teams and graduate students.

The draw to flight widget can be demonstrated to prospective ECPE students and donors during tours. The datalog visualizer can be operated to view flight data from the drone. Viewing flight logs helps users see how flight performance can be improved by plotting actual position versus setpoints. The PID Slider widget helps users tune the PID gains that control the quadcopters flight performance.

The MAVLink adapter gives users access to a rich control/telemetry library that is flexible enough to be geared towards several different controls applications. Modifying the existing MAVLink code will reduce the overhead in comparison with implementing an entirely new system. Having multiple quadcopters flying at the same time was not completed. The quadcopters were able to connect to the GCS at the same time, but they could not fly accurately. Multiple quadcopters flying simultaneously could have been used during flight demonstrations.

In conclusion, COVID-19 restrictions limited some progress on deliverables, but important functionalities were added to the existing system that will surely fulfill our user groups needs.

6.2 References

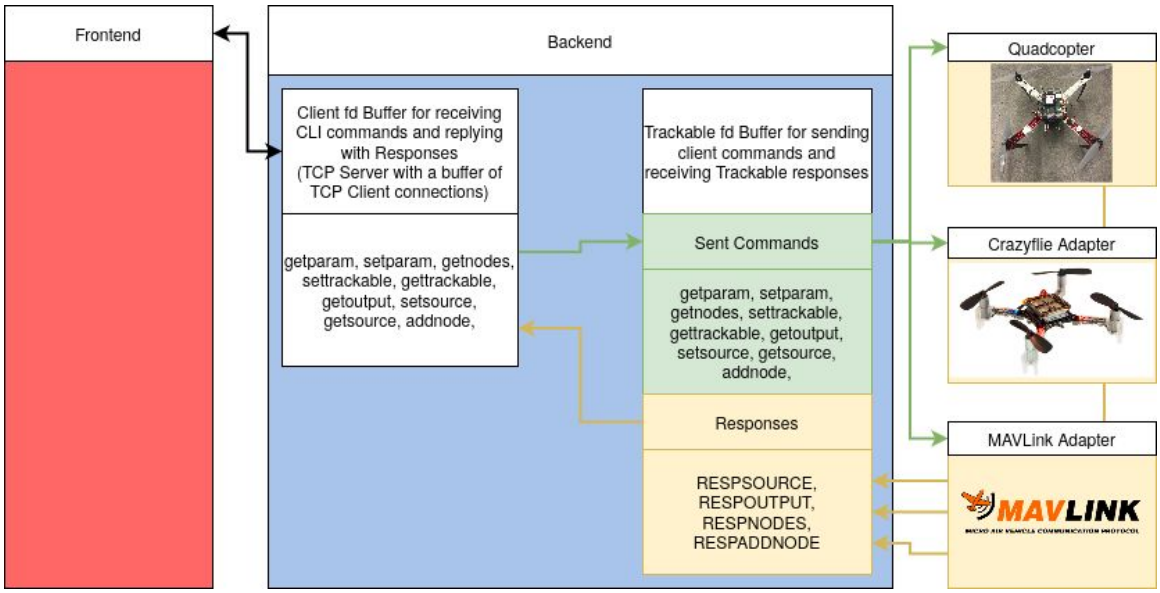
Minimal technical references were made throughout the document. Little technical documents were referenced as the majority of knowledge was pulled from previous work done by previous teams.

[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

6.3 Appendices

6.3.1 Backend



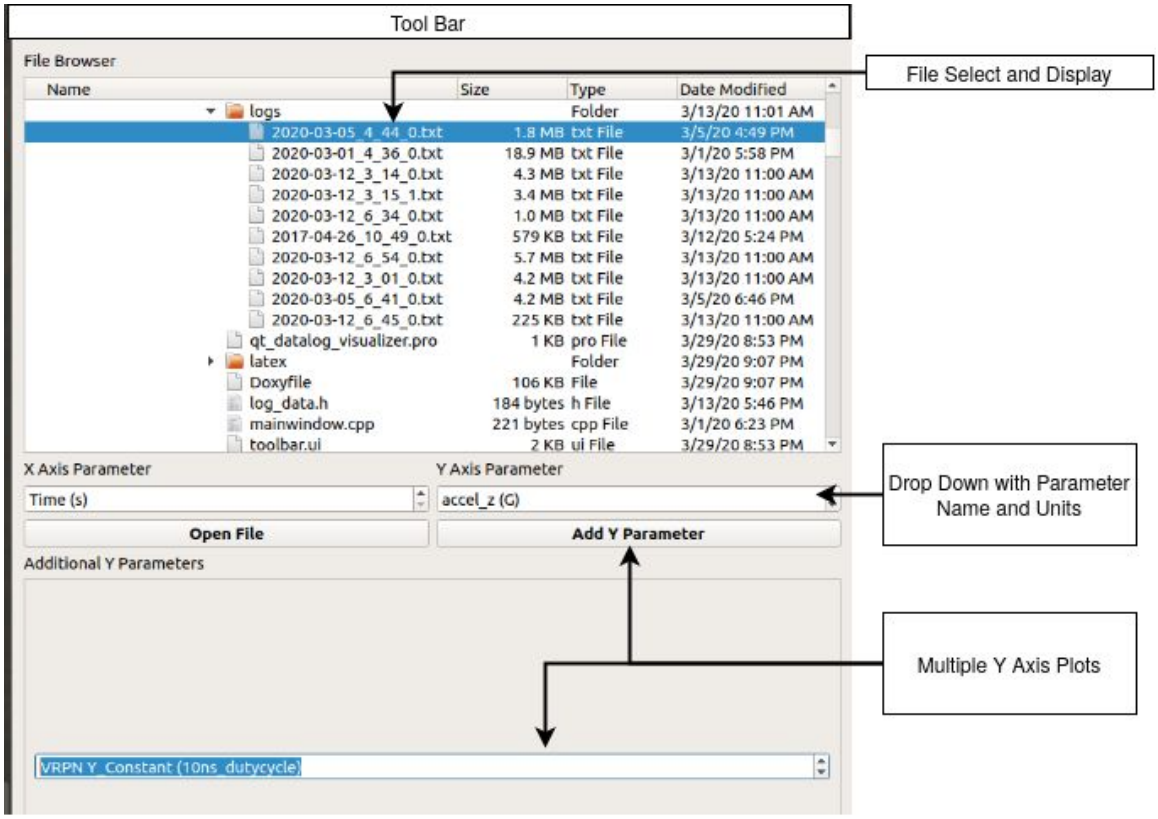
Summary

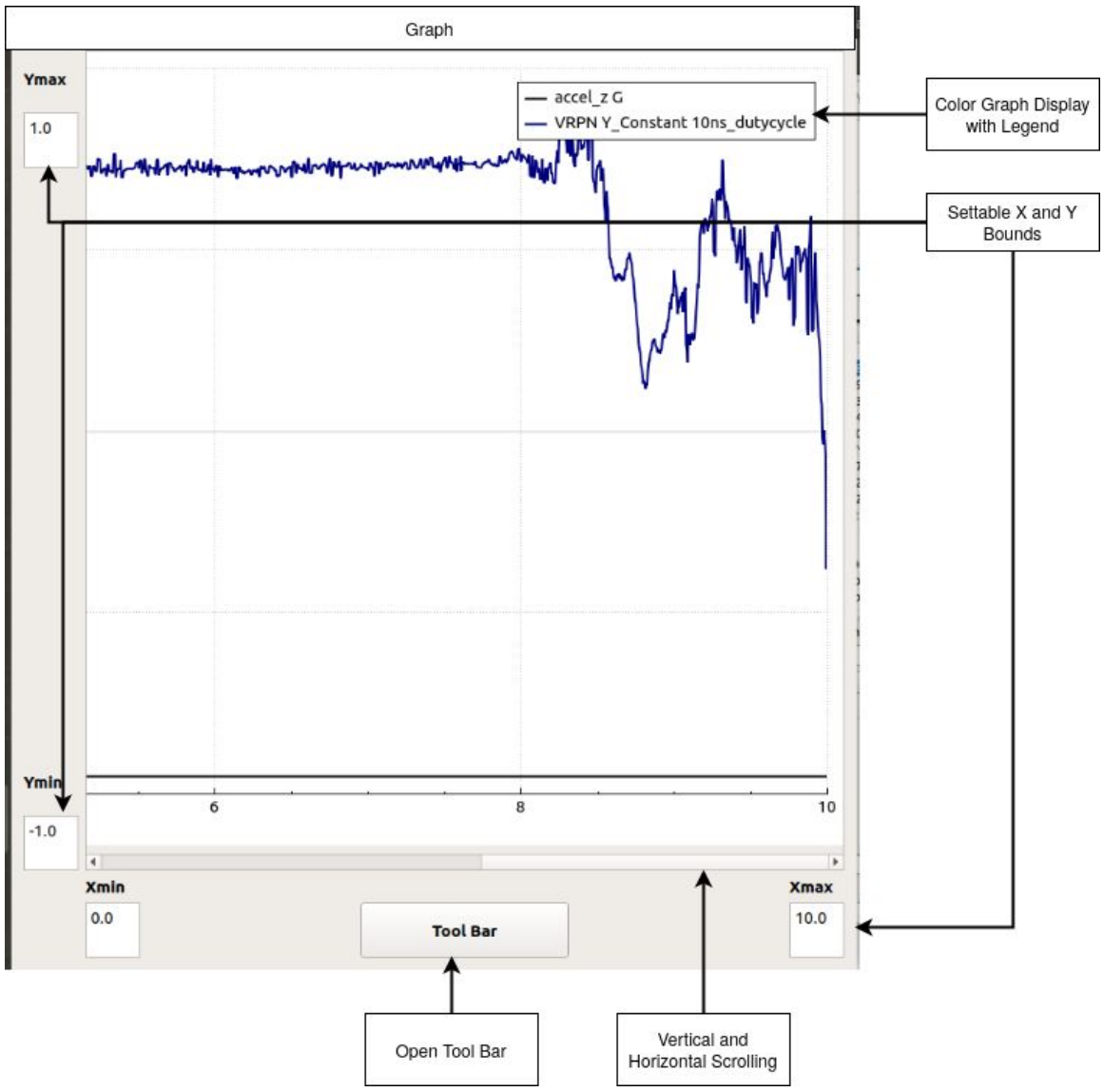
The Backend is responsible for handling front-end command requests.The Back-end acts as a TCP server that will accept requests for connections from the front-end. A request connection is opened every time a user types a command like getnodes or getparam. The backend translates the high level commands received from the front-end socket into a binary message format.

The messages are then sent to the current trackable file descriptor that is set using the settrackble command. The trackable file descriptors have the backend connected to a trackable object like an adapter or the quadcopter. Some messages will elicit a response like RESPNODES or RESPARAM from the given trackable. The backend will handle this response and send the message back to the client that called it.

The backend holds a trackable_t struct list in the config.c file, that can be configured to connect to different trackables. If the trackable you want to control uses an adapter a path, then the adapter's socket is supplied in the trackable_t struct. The trackable_t also can be configured to connect to the quads. The trackable file descriptor buffer can hold multiple file descriptors which allows communication with multiple quads at once. This would have been used if we finished work on the swarm flight deliverable.

6.3.2 Data Log Visualizer


Tool Bar
File Browser
File Select and Display
Name
Size
Type
Date Modified
logs
Folder
3/13/20 11:01 AM
2020-03-05_4_44_0.txt
1.8 MB txt File
3/5/20 4:49 PM
2020-03-01_4_36_0.txt
18.9 MB txt File
3/1/20 5:58 PM
2020-03-12_3_14_0.txt
4.3 MB txt File
3/13/20 11:00 AM
2020-03-12_3_15_1.txt
3.4 MB txt File
3/13/20 11:00 AM
2020-03-12_6_34_0.txt
1.0 MB txt File
3/13/20 11:00 AM
2017-04-26_10_49_0.txt
579 KB txt File
3/12/20 5:24 PM
2020-03-12_6_54_0.txt
5.7 MB txt File
3/13/20 11:00 AM
2020-03-12_3_01_0.txt
4.2 MB txt File
3/13/20 11:00 AM
2020-03-05_6_41_0.txt
4.2 MB txt File
3/5/20 6:46 PM
2020-03-12_6_45_0.txt
225 KB txt File
3/13/20 11:00 AM
qt_datalog_visualizer.pro
1 KB pro File
3/29/20 8:53 PM
latex
Folder
3/29/20 9:07 PM
Doxyfile
106 KB File
3/29/20 9:07 PM
log_data.h
184 bytes h File
3/13/20 5:46 PM
mainwindow.cpp
221 bytes cpp File
3/1/20 6:23 PM
toolbar.ui
2 KB ui File
3/29/20 8:53 PM
X Axis Parameter
Y Axis Parameter
Time (s)
accel_z (G)
Drop Down with Parameter Name and Units
Open File
Add Y Parameter
Additional Y Parameters
Multiple Y Axis Plots
VRPN Y_Constant (10ns_dutycycle)

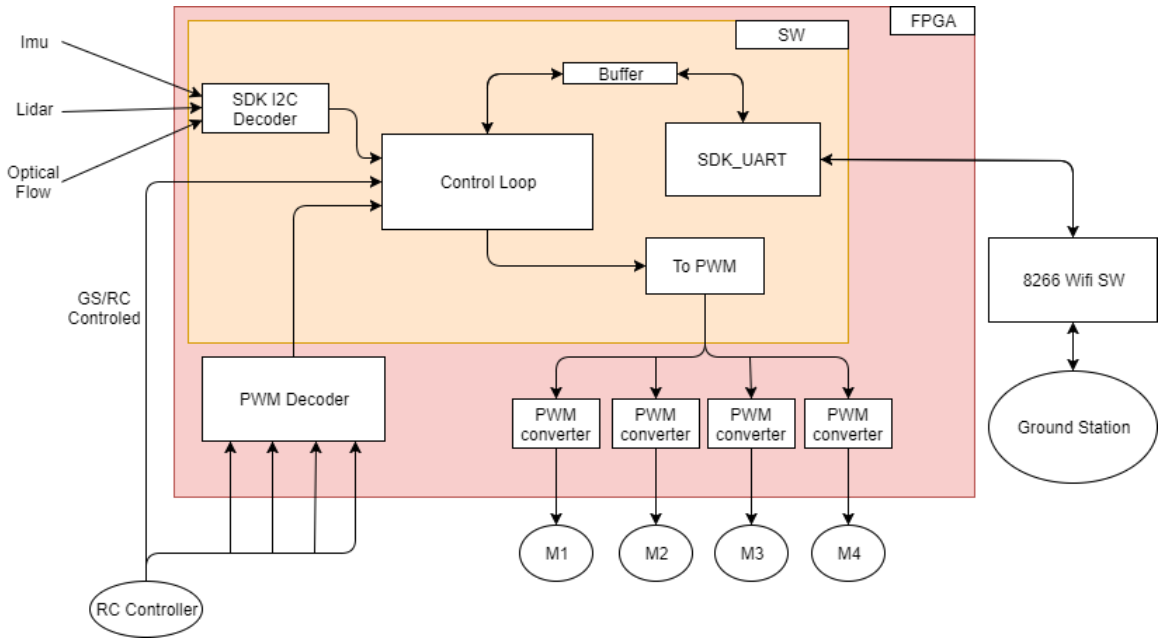

Graph
Ymax
1.0
accel_z G
VRPN Y_Constant 10ns_dutycycle
Color Graph Display with Legend
Settable X and Y Bounds
Ymin
-1.0
6
8
10
Xmin
0.0
Tool Bar
Xmax
10.0
Open Tool Bar
Vertical and Horizontal Scrolling

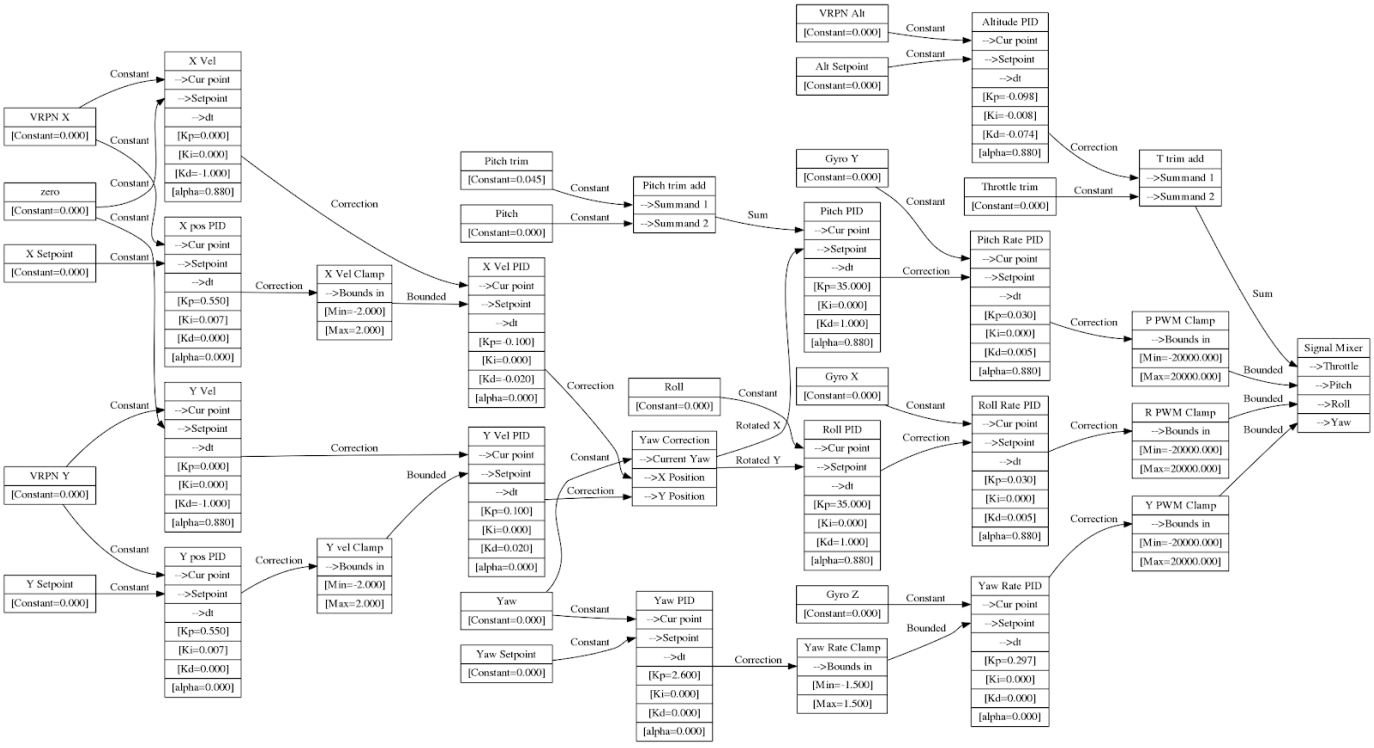
%	Parameter Name	Tab Delimiter '\t'	● ● ●	Newline \n
&	Parameter Units	Tab Delimiter '\t'	● ● ●	Newline \n
	Parameter Value	Tab Delimiter '\t'	● ● ●	Newline \n
● ● ●				
EOF				

Summary

The Datalog Visualizer reads saved flight data from the quadcopter platform. The quadcopter saves its flight data in the format outlined in the diagram above. The beginning of data starts with % which indicates the start of parameter names which are delimited by tabs. The Datalog visualizer reads this format and parses the data into arrays. The arrays are then fed into the QCustomPlot Widget for display. The widget allows the user to select an x and y axis to be plotted. By Default the x axis is set to the first parameter, which is time. A user can select multiple Y parameters to be graphed on the same plot for the same x axis. One theoretical usage of the current setup is using graphs for PID tuning. A user can plot the current position from the VRPN system and compare it with the setpoint location. A user can then determine PID qualities like steady-state error and settling time and change the PID gains accordingly.

6.3.3 Quadcopter





6.3.4 CrazyFlie

Crazyflie development was continued on from last year's team. Single flight was successfully achieved using a smartphone as a bluetooth controller. The development environment needed is now set up and multiple Crazyflies have been built for next year's team.

6.3.5 Tuning Stand

The 3-D printed tuning stand was built for use in this project and another one of Dr. Jones' classes. The tuning table allows for testing Crazyflie rotation around 2 axes. It can also be used to make sure the Crazyflie auto correction is working since the Crazyflie should be correcting itself upright while you spin it.

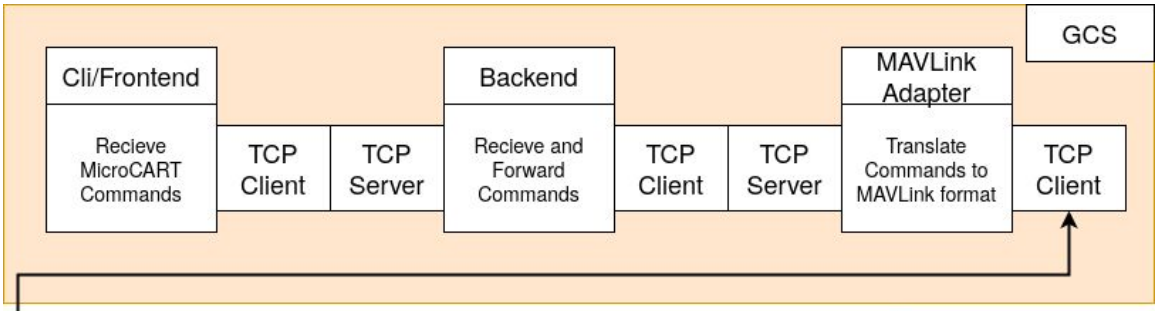
6.3.6 Swarm Flight

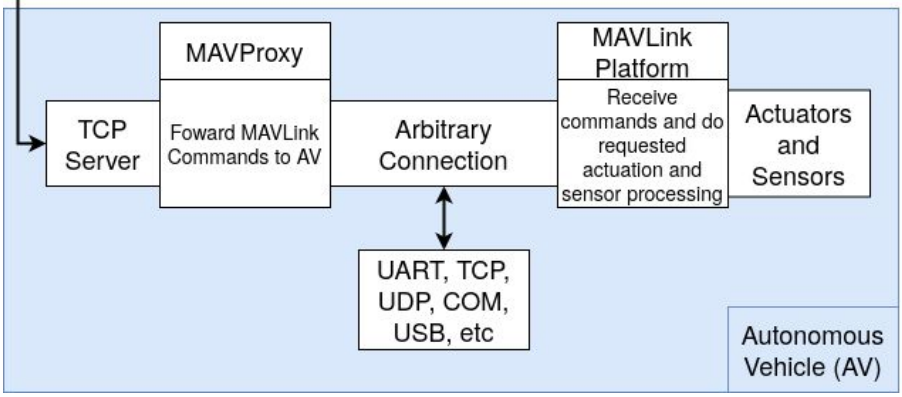
This deliverable was significantly impacted by COVID-19 restrictions. The quadcopters relied on the VRPN system in Coover 3050 to navigate. The restrictions that prevented accessing the lab stopped the completion of this deliverable. There was partial progress made. The second quadcopter platform was built. The platform was then able to fly with control commands with the existing software. The backend and network adapters for the drones were configured to have the GCS as the wifi access point and the quadcopters as client connections.

The vehicles were able to receive control commands from the backend while both were connected simultaneously. There is a video in the appendix that shows the high level test for both of them communicating at the same time. They were not able to fly accurately while connected at the same time. This happened the week before spring break, so this issue was not resolved. A

markdown documentation was created to describe connecting multiple trackables, so a future team could finish this deliverable if they choose.

6.3.7 MAVLink





MAVLink Adapter Node Data Structure

Index	Name	Source Function Calls	
0	Auto/Manual	set_auto()	
1	Arm/Disarm	request_arm()	
2	Mission Items	set_mission_item()	
3	Transmit Mission	request_mission()	transfer_mission_items()
4	MAVParam Start Index		
● ● ● ●	150 MAVParams within current usage window. To view other MAVParams increase the MAVParam Start Index.	get_mavparam()	set_mavparam()
156	Number of MAVParams		

get_node_value(), set_node_value()

Summary

Due to the lack of progress made on other deliverables due to the COVID-19 restrictions, the MAVLink adapter was added to the project because it could be implemented remotely. To

explain what defines the functional and nonfunctional requirements of the MAVLink adapter, a brief introduction will be given. MAVLink is a messaging protocol used to communicate with drones and autonomous vehicles (Rovers, Submarines, etc). MAVLink can be used to control an AV's navigation by sending specific types of commands to an AV with autopilot software. MAVLink is under free software license and the libraries can be modified, which makes it flexible to whatever user group may decide to use it in the future.

The MicroCART GCS supports connecting to adapters that convert the communication protocol listed below into a protocol that is compatible with the AV to be controlled. This deliverable will involve implementing one of these adapters to communicate to a AV that supports MAVLink navigation. The first diagram above depicts how the MAVLink adapter fits within the existing system. The MAVLink adapter connects to the GCS backend as a trackable connection. This establishes a tcp socket between the two softwares. The MAVLink adapter will translate the MicroCART commands into commands with roughly equivalent functionality in MAVLink. The MAVLink library is tens of thousands of lines of code, so only the subset of commands necessary to move the vehicle are supported.

The MicroCART commands that are supported are getparam, setparam, and getnodes. The getparam and setparam commands are used to set constant parameter values in the node data structure shown in the second diagram above. A user can use set and get to change the values of the parameters in the graph, which control how the AV navigates.

The getnodes command has been slightly modified because of overflow restrictions within the MicroCART commands. The getnodes command was used for viewing all the settable parameters within a control graph on the quadcopter. The GCS backend software only supports responses of 4096 bytes at maximum. There are roughly 1000 MAVParameters that can be set on an AV that supports MAVLink. Getting all the settable parameters with their 16 byte names attached is impossible unless you view a sub-window of parameters. The getnodes for the MAVLink adapter looks at 150 parameters starting from the value set in the MAVParam start index node. The start index node is fixed, so a user can change the value to view and set more parameter values.

For example, if a user wanted to set the 200th MAVParam, they could set the start index to 150 and the node id to 55. It is 55, and not 50 because the first 5 nodes are static and always accessible. The last node at node 156 is also static and gives the number of MAVParams available. The first 5 static nodes are used for setting setpoints and enabling autonomous navigation.

This deliverable maps into our user groups' needs. There are several open source projects that use MAVLink for communication (MAVProxy and Ardupilot). These open source projects can have their software changed slightly to fit a future team's needs. When the team modifies the software, they can then deploy it on a physical platform. MAVLink supports GPS navigation and sensor processing very well, so if a team wanted to take a quadcopter platform outside they would have significantly reduced overhead to implement something that could move around outside. Flight demonstrations also impress prospective students more if the control software is flexible enough to control different types of vehicles.

The source code for this deliverable is at this url. Test scripts run to evaluate the adapters are shown in the videos section.

6.3.8 Communication Protocol

Index	0	1	2	3	4	5+datalen	6+datalen
Message Parameter	Begin Char	Message Type	Message Id	Data Len	Data	Checksum	End Char
Bytes	1	2	2	2	var	1	1

Message Type		
Debug 0x0	log_end_id 0x06	get_nodes 0xC
Packetlog 0x1	set_param 0x07	resp_nodes 0xD
GetPacketLogs 0x2	get_param 0x08	add_node 0xE
Update_id 0x3	respparam 0x9	resp_add_node 0xF
beginupdate 0x4	setsource 0xA	
log_id 0x5	getsource 0xB	

Summary

The Microcart Communication protocol can be seen in the diagram above. A beginning character 0xBE is the first index in the packet indicating the start of the frame. After that the next two bytes determine the message type which could be any of the types shown in the table above. The majority of the message types either get or set some configurable value on the platforms that modify navigation performance.

.6.3.10 Videos

Connecting To Multiple Quadcopters Simultaneously

MAVLink Adapter Supporting Missions

MAVLink Adapter Supporting Get and Set Params

MAVLink Adapter Supporting Get Nodes

[Datalog Visualizer in Operation]

[PID Slider In Operation]

[Tuning Stand Implemented]

Fig 1. Successful Quad motor test reading

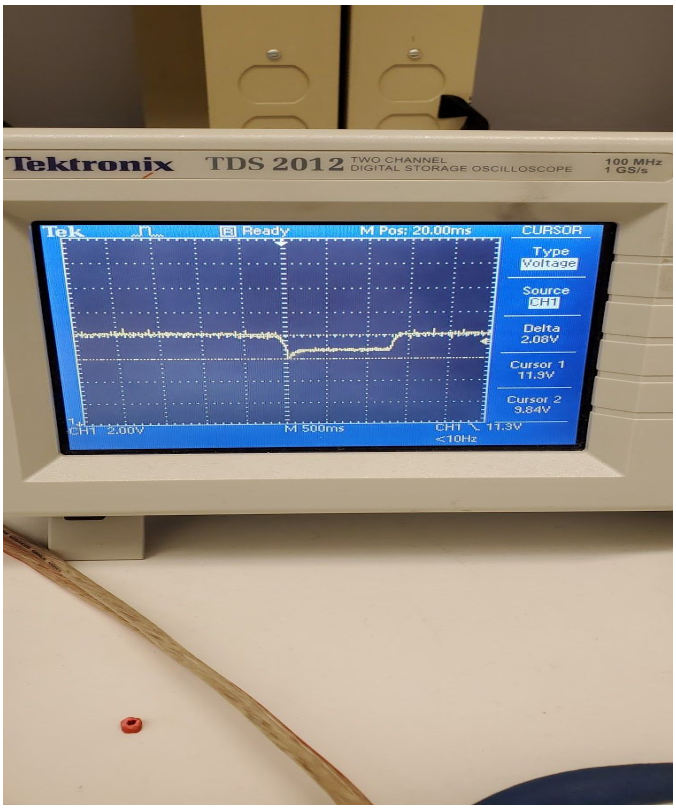


Fig 2 Tuning Table in action