MicroCART mini

DESIGN DOCUMENT

Team Number: sdmay25-32

Client/Advisor: Dr. Philip Jones

Team Members & Roles:

Ryan Lowe: Technical Advisor

Daniel Zaucha: Client interaction, Communications Lead

Jonah Upah: Hardware Lead, Team Secretary

Yi Hang Ang: Software Lead

Team Email:

sdmay25-32@iastate.edu

Team Website:

https://sdmay25-32.sd.ece.iastate.edu

Revised 4 May 2025

Executive Summary

The name of this project is MicroCART, which stands for Microprocessor Controlled Aerial Robotics Team. This project is divided into two parts. The first part is optimizing and improving a quadcopter called CrazyFlie, which will be used to conduct MP-4 (Lab 4) of CPRE 488: Embedded Systems Design. This is important because the team is providing a better and more convenient environment for CPRE 488 students to learn. The second part of this project is implementing and improving on the firmware of a quadcopter called FlyPi, which is a product of last year's MicroCART team that will be used in future live demonstrations during Scholar's Day at ISU to attract prospective students. The long-term goal of this project is to learn from what was and will be implemented on the CrazyFlie and further improve on MicroCART's heritage, the FlyPi.

The key design requirements would be to improve the backend-to-frontend communication and the graph logger of the CrazyFlie to reduce latency by implementing a new packet in Python that will contain graph logging values. As of the first semester, the team will need to gain a better understanding of the backend communication before implementation. If implemented perfectly, this design requirement will provide students with a smooth and clearer GUI graph logger, which will help students have a better understanding of the graph.

Besides that, the team plans on implementing a new component called a test stand for the CrazyFlie. This test stand is an Arduino connected to a rotational sensor that provides third-party roll, pitch, and yaw sensor values to assist in tuning the PID controller. The test stand provides the students with an additional set of sensor values in addition to the onboard sensor values from CrazyFlie, which will assist them in deducing the parameters of the PID controller. The team's plan was to connect this test stand to the backend.

For the FlyPi, the team plans on implementing a global positioning tracking system that will keep track of the current position of the FlyPi while it is autonomously moving around the test field. This requirement will be implemented last as the team does not currently possess the knowledge regarding the details of this implementation, but the idea would be to calculate the distance the FlyPi has traveled in x,y, and z coordinates through the aid of the PID controller and the IMU (Inertial measurement unit).

Overall, the team managed to complete the test stand implementation, such as connecting it to the backend via a serial connection, and successfully plotting the test stand values to the MicroCART GUI used by students in MP-4. Besides that, the team improved the overall performance of the ground station, backend, and GUI applications by reducing CPU consumption from 100% to 60-70%, which led to a drastically reduced number of VM (Virtual Machine) and application crashes. The team also improved the documentation presented in MP-4, such as the lab manual, by removing legacy documentation that is not present in the current implementation and reorganizing the verbose document. Due to the team’s effort and commitment to focus more on the MP-4 part of the project, the team was told by their advisor, Dr. Jones, that this is the smoothest deployment of MP-4 by far in the history of MicroCART.

Learning Summary

Development Standards & Practices Used

Since this project is mainly software-focused embedded programming in C, C++, and Python, the team would prioritize following standard software development standards.

- Variable assignments should not be made from within sub-expressions
- Class names should comply with a naming convention
- Local variable and function parameter names should comply with a naming convention
- Failed unit tests should be fixed
- The resulting code, causing failed pipelines on Git, should never be deployed

Engineering Practices

Engineering standards are essential because they ensure that the products produced by engineers meet users' expectations. This means that the product is safe for users, has a standard or consistent quality, and is environmentally friendly. Adhering to engineering standards increases work efficiency and speeds up the engineering process. Therefore, the existence of engineering standards is beneficial to both users and engineers, and these are the engineering standards that the team thinks are applicable to the project.

- IEEE 802.11s-2011: IEEE Standard for Wireless LAN Medium Access Control (MAC)

  - This standard is relevant because it focuses on wireless communication. In the project, the team was given a backend that communicates the remote-controlled quadcopter with the ground station, which means wireless communication and data transmission.
- IEEE 1687-2014: IEEE Standard for Access and Control of Instrumentation Embedded within a Semiconductor Device
  - This standard is about accessing instrumentation embedded within a semiconductor device, which is precisely what the project focuses on integrating and improving a quadcopter's embedded systems.
- IEEE 1936.1-2021: IEEE Standard for Drone Applications Framework
  - This standard is about frameworks for the support of drone applications. The project emphasizes working with drones (quadcopters) and their flight control systems.

Summary of Requirements

1. Mini Quadcopter should be able to:
   - Fly smoothly
     - Flight stabilization
     - No sudden "random" movements
     - Quick reactions to directional inputs
   - Connect to remote equipment for data analysis
   - Be able to connect to remote sensors and utilize the information to fly

  - Can be utilized easily even by someone with no prior experience in controlling any remote control vehicle

2. Frontend/Backend should be able to:
   - Be accessible through the current method
   - Display data from the flight information
   - Be able to enable an uncontrolled flight via sensor data
3. Documentation must:
   - Explain the steps throughout the project
     - Plan of action, task breakdown, time taken, changes made from the previous project, what the team could have done better, and what will be left for the following year's group to complete.
   - Document any problems that came up throughout the development process and record how the team solved them for future project groups or, when applicable, by teachers, TAs, and students.
   - How to solve issues that come up frequently (FAQ Sheet)
   - Catch users up-to-speed on the programming project, depending on their role
     - Student, TA, Advisor, Teacher, Successor team, or the general public
   - Show and explain how the project connects to various other fields and draw interest from observers to look deeper into it

Applicable Courses from the Iowa State University Curriculum

1. CPRE 2880: Embedded Systems 1: Introduction
2. CPRE 3080: Operating Systems: Principles and Practice
3. CPRE 4880: Embedded Systems Design
4. CPRE 4890: Computer Networking and Data Communications

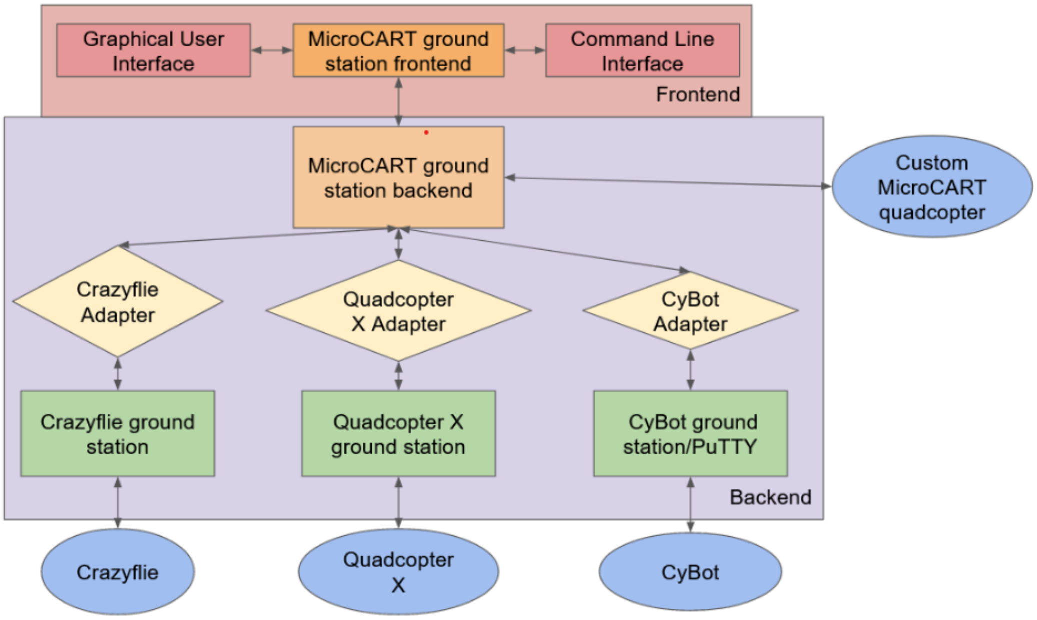
New Skills/Knowledge acquired that were not taught in courses

1. Control Systems Theory
2. Socket Programming
3. Qt Creator

Table of Contents

List of figures/tables/symbols/definitions

1. MicroCART system overview:



2. Glossary of Terms:

2.1.CPRE 488 MP4/Lab 4 - The fourth lab of the CPRE 488: Embedded Systems Design course. The team will be in charge of optimizing the equipment and software used to conduct this lab.

2.2.CrazyFlie - The small drone used for the CPRE 488 MP4 Lab. It is manufactured by Bitcraze[1] and has open-source firmware that can be easily written. When the term drone is used, the CrazyFlie is usually the drone that is referenced.

2.3.FlyPi - The larger custom quadcopter built by previous MicroCART teams. Currently in a complete state, but would require some optimizations.

2.4.GUI - C++ based Graphical User Interface that is created with the application QT, which allows a user-friendly display of the frontend.

2.5.CLI - Command Line Interface, an interface through the command line which allows the user to interact with the system using specific commands.

2.6.Ground station - Name used for the group of software components that lie in between the quadcopters and the GUI: Backend, CrazyFlie Adapter, and CrazyFlie Ground station.

2.7.Backend - Software module written in Python that handles incoming packets from the frontend and sends them to the necessary destination. Also handles data from cameras and other sources.

2.8.Crazy radio/dongle - USB radio stick that sends packets to and from the CrazyFlies.

2.9.Test stand - A device used to hold the CrazyFlie in place while fine-tuning its parameters, a port is located at the bottom of the test stand that allows an Arduino to connect to it.

2.10. Test stand - An Arduino connected to a sensor that will collect positional data from the CrazyFlie through the port and send it to the PC directly via a USB cord.

1 Introduction

1.1 Problem Statement

Quadcopters, and drones in a broader sense, are seeing more day-to-day usage across many fields such as agriculture, transportation of goods, the military-industrial complex, and so many more! This team's project is known as MicroCART mini, and the team is designing/iterating new software for a mini-quadcopter that will be used as learning materials for Iowa State University's Department of Electrical and Computer Engineering students, in addition to actualizing a quadcopter into flight via designing hardware and software. As MicroCART is focusing on creating small, remote-controlled devices for both educational and non-specific usages, the focus of the team's design will primarily be on sustaining controlled flight. Uncontrolled flight is a hazard not only to the quadcopter but also to the environment around it, which means the team will have to make the controls adaptable for the mini quadcopter to be used by untrained non-professionals and for an automated program to be able to utilize sensors to obtain information from around the quadcopter and through a program complete a flight in new terrain while minimizing damage from or outright preventing any crashes. Since the quadcopters are constrained by their small sizes, quadcopters with remote sensors to absorb information were connected by the team, such that the quadcopters will be able to utilize it for flight navigation. Sensory navigation opens up the possibility for unnavigated routes to be flown, such as in a disaster scenario for search and rescue, to have new route information recorded, and to optimize a flight path in new terrain safely.

1.2 Intended Users

1. **CPRE 488 Students**

   a. Senior/Graduate-level students taking CPRE 488 in Spring 2025.

   b. Must have completed CPRE 381 or COMS 321

   c. Must be able to perform with Mini-Quadcopters after 4 intro labs

   d. A limited amount of time they can dedicate solely to this class

Needs: CPRE 488 students need an operational and improved platform to work on Lab 4. The team can improve the prior hardware, systems, and framework to provide students with a more convenient environment to work on Lab 4.

Benefits: Lab 4 for CPRE 488 students will be conducted smoother, increasing their productivity and making better progress. Students will also learn how to fine-tune control systems like the PID Control in Lab 4 and implement that into an RC quadcopter.

2. **Successor Project Team**

   a. Senior-level students working on the MicroCART Senior Design Project in the future.

   b. Senior-level knowledge base

   c. Multiple Disciplines (i.e., CPRE, EE, SE)

   d. Will be working off of what the team left off

Needs: Successor Senior Design teams would need tutorials, like a step-by-step guide or video tutorial, on complicated parts of the project. Besides that, they would need proper and updated documentation on the project based on what the team changed and improved from past projects. They would also need code that is easy to understand and to make changes to.

Benefits: With improved information "library", successor project teams will be able to find the information that is associated with the different parts of the project they will be working with in a shorter period of time, and be able to catch up or surpass the progress that the team has made in comparison. It will also give an outline of the order to go about the project when they are starting out, to give themselves a longer period of time to optimize their own progress.

3. **CPRE 488 Teacher/Advisor/TAs**
    a. Course Instructors for CPRE 488
    b. High-level course knowledge
    c. May have seen previous projects done and performed
    d. Observing to see if the project and students' work meet project requirements
    e. Have limited time and more responsibilities

Needs: A high-level overview of what the project is and how the team has organized the project. Separation of different presentations and the expectations that the team was trying to meet for all. Take note of the detail where the team found trouble and how it was overcome. General instructions that detail the processes the team went through, explaining why certain methods were chosen.

Benefits: A reduced time period for reading necessary items and ignoring the details that they have seen before, and can otherwise ignore. An enhanced ability to find where groups/individuals are struggling and to be able to quickly tell them possible solutions to issues that arise. Able to take up less time than would otherwise be without the pre-recordings

4. **Potential Incoming College Students**
    a. High school tour groups
    b. High school-level knowledge
    c. Want to attract them to be like us
    d. May have interests in other engineering fields
    e. Need to show them how this connects to other ISU disciplines

Needs: Potential incoming college students need to be interested and drawn into the project, and be able to see what they could learn if they were to become students here. The incoming students need to have an explanation of the project that will make sense to them, given that they will not be familiar with much of the material/technologies that were used by the team.

Benefits: Potential students may be able to base their decisions on colleges by seeing what engineering students at Iowa State can accomplish.

2. Requirements, Constraints, And Standards

2.1 Requirements and Constraints

- Mini Quadcopter should be able to:
    - Fly smoothly
        - Flight stabilization
        - No sudden "random" movements
        - Quick reactions to directional inputs (For example, to stop turning when the turning button is no longer pushed)
    - Connect to remote equipment for data analysis
    - Be able to connect to remote sensors and utilize the information to fly
    - Can be utilized easily even by someone with no prior experience in controlling any remote control vehicle

- Frontend/Backend should be able to:
    - Be accessible through the current method
    - Display data from the flight information
    - Be able to enable an uncontrolled flight via sensor data

- Documentation must:
    - Explain the steps throughout the project

        - Plan of action, task breakdown, time taken, changes made from the previous project, what the team could have done better, what the team did not get to, and what will be left for the following year's group to complete.
    - Document any problems that came up throughout the development process and record how the team solved them for future project groups or, when applicable, by teachers, TAs, and students.
    - How to solve issues that come up frequently (FAQ Sheet)
    - Catch users up-to-speed on the programming project, depending on their role
        - Student, TA, Advisor, Teacher, Successor team, or the general public
    - Show and explain how this project connects to various other fields and draw interest from observers to look deeper into it

2.2 Engineering Standards

The sub-category that is appropriate for this project would be Computer Technology. The three IEEE standards that apply to this project are:

1. IEEE 802.11s-2011: IEEE Standard for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications Amendment.

- This standard is relevance because it focuses on wireless communication. In this project, the team had a backend that communicated with the remote-controlled quadcopter with the ground station, which means wireless communication and data transmission.

2. IEEE 1687-2014: IEEE Standard for Access and Control of Instrumentation Embedded within a Semiconductor Device
    - This standard is about accessing instrumentation embedded within a semiconductor device, which is precisely what this project focuses on: integrating and improving a quadcopter's embedded systems.

3. IEEE 1936.1-2021: IEEE Standard for Drone Applications Framework
    - This standard is about frameworks for the support of drone applications. This project emphasizes working with drones (quadcopters) and their flight control systems.

These standards were chosen due to how this project, which is building and/or implementing a control system into a mini quadcopter, utilizes remote control through wireless devices, accessing the quadcopter itself for data, and reiterating the fact that this is a drone device that the team is using.

Some of the other possible standard choices that the team did not choose were battery standards. The team did not use these standards due to them being rather broad and rather nonspecific to this project, for the most part, due to them being a far more secondary aspect, in comparison to the chosen standards, which are undoubtedly more related

3 Project Plan

3.1 Project Management/Tracking Procedures

The team adopted a waterfall-agile hybrid methodology. The project is broken down into different phases to guide the general path for the rest of the project. The different phases include MP4, Backend, and Frontend. To effectively distribute work and manage deadlines for these different phases, the team adopts the Agile methodology to allocate tasks and issues.

Progress throughout the course of this project has been documented through the use of GitLab issues. GitLab issues were used to designate tasks for each team member and provide a timeline for what was needed to work on. This is how previous teams for this project have tracked progress, and the team will follow suit. It is also helpful to track deadlines and motivate/keep track of team members to work on a specific issue before it is due.

3.2 Task Decomposition

- Documentation
  - Progressive throughout
- MicroCART

    - CPRE 488 - MP 4 (aka Lab 4)
        - PID Research
        - CrazyFlie
    - Backend
        - CPRE 488- Framework
    - Frontend
        - GUI & CLI
    - Communication
        - CrazyFlie Adapter
        - CrazyFlie Ground Station
    - Global Positioning Control
    - Test Stand
- Semester End Presentation

3.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

Milestones:

MP-4 compilation:

- Since it is based on an existing lab, it is easy to compile all the documents. The goal is 100% completion, but not making a lab document that a student must submit.

Understanding the backend (Big picture):

- This involves understanding how the system works from a high-level perspective. It is quantifiable by the ability to explain what each component is and its purpose.

Dive into sub-components of the communication pipeline:

- Expanding on the last milestone, the team then needs to gain a deeper understanding of each subcomponent. This is quantifiable in a similar way.

Optimize issues with the GUI:

- There are issues within the GUI that affect the ease of use. While it is functional, there are prominent bugs that should be addressed. This is measurable by the number of bugs encountered during a session. The team aims to attack the most common ones to reduce the debugging time for students.

Add Global Positioning:

- This is a new feature the team will be adding to the 488 lab. This would need some involvement from Dr. Jones to tie it to the 488 curriculum. To measure this would lay the groundwork for other teams to build upon. The best outcome would be to implement the feature and thoroughly add it to the lab.

Implement Test Stand

- The test stand is an Arduino that connects to the backend via a serial connection that provides third-party roll, pitch, and yaw sensor values to assist in tuning the PID controller. The test stand provides the students with an additional set of sensor values in addition to the onboard sensor values from CrazyFlie, which will assist them in deducing the parameters of the PID controller. The team's plan was to connect this test stand to the backend.

Performance Optimizations

- The team improved the overall performance of the ground station, backend, and GUI applications by reducing CPU consumption from 100% to 60-70%, which led to a drastically reduced number of VM (Virtual Machine) and application crashes.

Lab Documentation

- The team also improved the documentation presented in MP-4, such as the lab manual, by removing legacy documentation that is not present in the current implementation and reorganizing the verbose document.

Explore FlyPi:

- This is a stretch goal in the experimental portion of the project, the previous teams were doing some fairly complex stuff. A reasonable goal for the team would be to organize better what already exists. The best outcome would be to expand upon what the last group left.

Pick up where the last group left off (FlyPi):

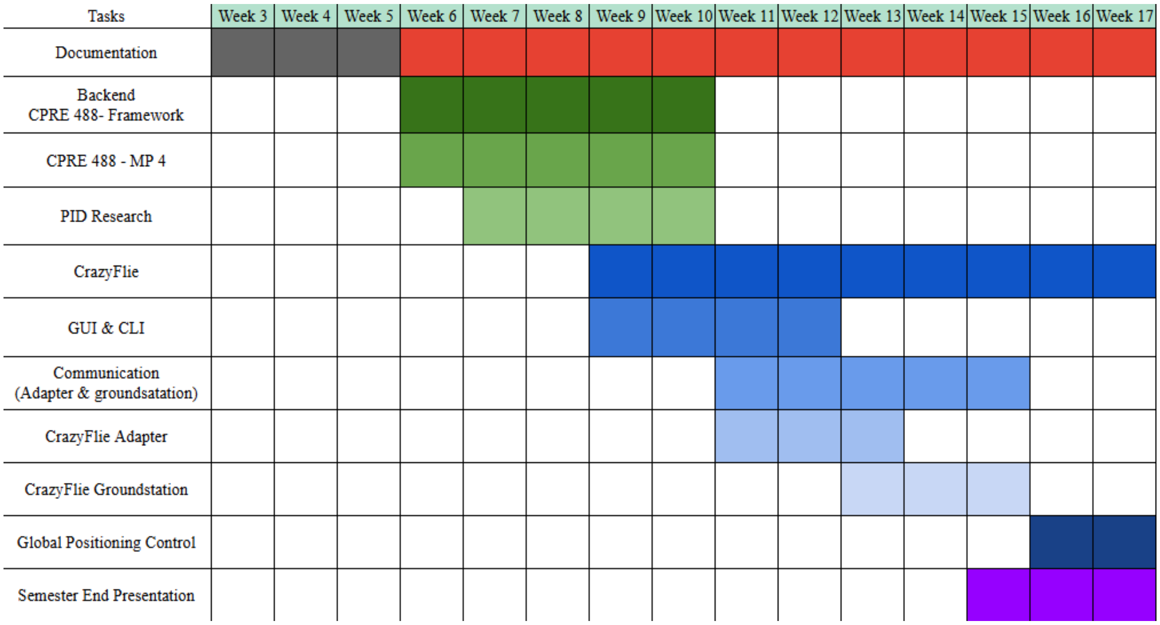
- This is expanding on the last milestone. The actual contents of what is achievable are unknown at this point, as the team has not completed the exploration yet.

3.4 Project Timeline/Schedule

Note:

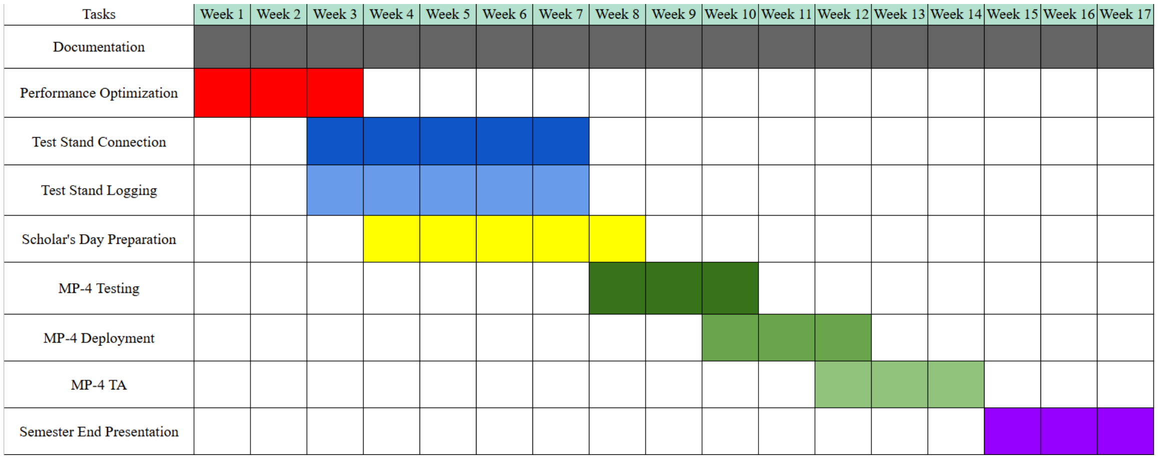
- Subtasks are other colors of the same group
- Associated tasks are worked on while working on the task itself

First Semester



Tasks	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11	Week 12	Week 13	Week 14	Week 15	Week 16	Week 17
Documentation	⬛	⬛	⬛	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
Backend CPRE 488- Framework				🟩	🟩	🟩	🟩	🟩							
CPRE 488 - MP 4				🟩	🟩	🟩	🟩	🟩							
PID Research					🟩	🟩	🟩	🟩							
CrazyFlie							🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
GUI & CLI							🟦	🟦	🟦	🟦					
Communication (Adapter & groundsatation)									🟦	🟦	🟦	🟦	🟦		
CrazyFlie Adapter									🟦	🟦	🟦				
CrazyFlie Groundstation											🟦	🟦	🟦		
Global Positioning Control														🟦	🟦
Semester End Presentation													🟪	🟪	🟪

Second Semester



Tasks	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11	Week 12	Week 13	Week 14	Week 15	Week 16	Week 17
Documentation	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛
Performance Optimization	🟥	🟥	🟥														
Test Stand Connection			🟦	🟦	🟦	🟦	🟦										
Test Stand Logging			🟦	🟦	🟦	🟦	🟦										
Scholar's Day Preparation				🟨	🟨	🟨	🟨	🟨									
MP-4 Testing								🟩	🟩	🟩							
MP-4 Deployment										🟩	🟩	🟩					
MP-4 TA												🟩	🟩	🟩			
Semester End Presentation															🟪	🟪	🟪

3.5 Risks and Risk Management/Mitigation

Risk Scale: 1 (Low) - 10 (High)

Backend CprE 488 framework/ CprE 488 MP-4:

(2) Low risk:

- The team is working to optimize the existing solution. There are some bugs present that could hinder students' progress in MP-4. The risk is low because the team can always revert to the previous version that contains minor bugs.
- Mitigation: do incremental solutions so that if the team has trouble with one aspect, it doesn't affect others.

Ground Station and Adapter:

(1) Low risk:

- This was requested from the previous senior design team. They wrote the software to communicate with the crazy file through the crazy radio. In that pipeline, there is an intermediate component, the adapter. They mentioned that the ground station should absorb this. It works as is, but it might help speed up the communication pipeline.
- Mitigation: This is more just ensuring that performance does not diminish and that the result is more readable to next year's team.

Global Positioning:

(7) Medium risk.

- This is a familiar feature; therefore, the teams would be building it from the ground up. It would be a rewarding aspect of the project. But the risk is that the team does not get it done in the time allotted, and then would have to pass it to the next team, which could lead to miscommunication or abandonment of the feature.

- Mitigation: makes sure that the team documents their intentions and progress to a degree where, if the team does not finish, the next group will be able to pick up where was left off quickly.

Deployment:

(9) High risk

- This is the final deployment of the VM containing the revisions. At the end of the year, the team will want changes to be deployed across all 488 lab machines. There is a reasonable concern that if there is an issue, there would be little time to fix it. Therefore, none of the changes would take effect for the 488 lab.
- Mitigation: Do a test deployment beforehand to identify potential issues.

The highest risk task that the team had identified was the deployment of the MP4 lab. Now that this has passed, the team has been able to mitigate the risk and has not had too many issues come up. The way that the team has done this has been by acting as TAs during the lab sections to help fix any bugs or issues that a student may run into. Another way is by making sure that the team is quick in sending out updates through the Discord channel that the students use and by making small revisions to the MP4 Lab manual that may be confusing or misleading. The team has also been able to do a good job at maintaining the health of the quadcopters and ensuring that there are enough quadcopters in the lab for everyone to be able to have access to a drone. An additional risk for deployment was the addition of the test stand tracker and how this was going to be used. This is a risk because there is an error where if the test stand is not disconnected or dismounted from the virtual machine before closing the GUI, the whole system will crash. This is something that the team has been able to mitigate by emphasizing in the lab manuals that this will lead to a crash, and also went over this in the lab section before students started using any of the technology.

3.6 Personnel Effort Requirements

4 Group Members - Expecting a minimum of 6 hours per week from each person

Note: The estimated hours that are used are the value of the largest amount of time expected to be spent on a task, even though from the Gantt chart on 3.4.) Overlapping of multiple tasks on various weeks can be seen, so even though estimated hours/week/individual is 6 per say, then during the following weeks, less time will be dedicated to the task.}

[Hours Total Formula]: Duration x Estimated Hours/Week x Number of individuals

Estimated hours Formula: [Hours Total] / 4 / 12

(Rounded up due to 3 weeks of solely research)

First Semester

Tasks	Duration (Weeks)	Estimated (Hours/week) [Per individual]	Hours Total (Hours) [Sum of all team members]
Documentation & Research	14	2	96
Backend CPRE 488- Framework	5	–	–
CPRE 488 - MP 4	5	6	120
PID Research	4	2	32
CrazyFlie	9	–	–
GUI & CLI	4	3	48
Communication (Adapter & ground station)	5	-	-
CrazyFlie Adapter	3	4	48
CrazyFlie Ground station	3	4	48
Global Positioning Control	2	3	24
Semester End Presentation	3	3	48

Estimated Total Duration	12	~10	464 ~116 per individual

[While the above is a calculated estimate, a more likely approximation is 80 total hours and 8 hours/ week after accounting for time overlaps, division of work, and breaks]

Second Semester

Tasks	Duration (Weeks)	Estimated (Hours/week) [Per individual]	Hours Total (Hours) [Sum of all team members]
Documentation	17	2	136
Performance Optimization	3	5	60
Test Stand Connection	5	3	60
Test Stand Logging	5	3	60
Scholar’s Day Preparation	5	2	40
MP-4 Testing	3	2	24
MP-4 Deployment	3	2	24
MP-4 TA	3	4	48

Semester End Presentation	3	1	12
Estimated Total Duration	14	~8	464 ~112 per individual

[While the above is a calculated estimate, a more likely approximation is 80 total hours and 8 hours/ week after accounting for time overlaps, division of work, and breaks]

3.7 Other Resource Requirements

Some non-financial resources that this project has utilized include knowledge from the preceding groups and the code repository, virtual machine, and a BitCraze CrazyFlie information sheet that will record the state of lab equipment. Quality assurance is checked by both the team and their advisor to ensure that it works to the expected specifications.

4 Design

4.1 Design Context

4.1.1 Broader Context

The MicroCART team set out to improve the usability of the MicroCART application for the CprE-488 lab. This became a priority for the team due to the ramifications of a faulty application that is intended for students to use in the lab. When the team first started to get acquainted with MP-4 issues, the team immediately noticed some small quality of life issues that could be forgiven, but a couple of major issues that directly hindered the team from doing the lab. It was the major issue that the team focused on. From a student perspective, the course material is hard enough, but to also have to battle faulty software is unacceptable. There were two main issues that the team set out to fix.

The first issue noticed was that the application would bog down the virtual machine, which led to the loss of communication between the application and the drone. This lab involves the flight of the drone, so constant disconnections resulted in the inability to fly the quadcopter confidently. Even on the less consequential portions of the lab, where the drone is tethered down and the students are meant to observe the drone's movement and tune the drone's PID values accordingly, led to an external source of error that was not the fault of the students.

The second issue was the unfinished implementation of the test stand. Previous teams had designed a 3D-printed test stand that contained an integrated rotational sensor. This sensor enables the student to check the drone's self-estimated parameters with an external sensor. But the final piece of the puzzle was missing. To fully utilize this, two connections needed to be made. The first was to use an Arduino to digitize the analog data from the rotational sensor. The second connection was from the Arduino that packaged the rotational data and relayed it to the backend via a serial connection. This would then enable students to use the test stand sensor. This was one of the team's biggest priorities due to the progress the previous team had made.

Area	**Description**	**Examples**
Student Considerations	The design is intended for students who are learning a new subject, and for them, it is frustrating when, along with the challenge of learning, they are also having to do so on unstable software.	If a student were in the middle of one of their first test flights and the drone lost connection momentarily, it could lead to unexpected behaviour, no fault of the student's.
Servisablity	As this design project is an ongoing project, a future team will likely have to work with or, at the very least, read the code. So it is the team's duty to make sure it is readable.	When looking through the code base for the first time, it can be quite daunting, there is likely a lot of stuff never seen before. But if there are no comments or

		supporting documentation, this compounds the issue.
Safety	This design is centered around the use of a drone. There is the potential of bodily harm to students. This makes it vital to clearly instruct the student on safety procedures.	If the lab document is not clear about how the drone should be tethered down during initial testing, the drone could fly into someone's face.

4.1.2 Prior Work/Solutions

The MicroCART project has been ongoing for many years, but the MicroCART mini aspect of the project has been active for about six years, when the CPR E 488 class was revamped due to technological advancements and started to move away from large and dangerous quadcopters. The teams from previous years have designed many of the class materials, ranging from the classroom GUI used in their MP-4 to the lab documents and even the custom 3D-printed mini-quadcopter testing stands, which are used frequently in the later parts of the lab.

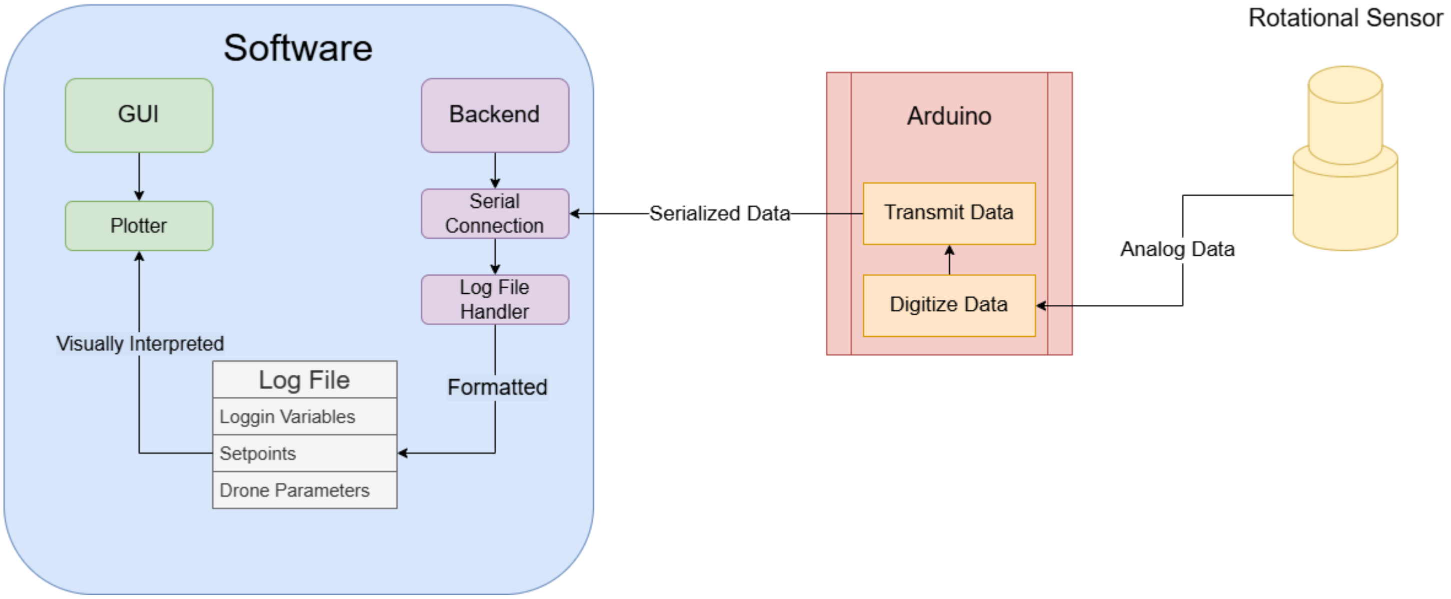
But how does their project compare to similar products (mini quadcopters)? To answer that, one must first examine some of the mini quadcopters available on the market. The products utilized for this explanation include the Kopis Freestyle 4-inch FPV

Drone[5], the DJI Mini 4 Pro Drone[6], and an “Open source ESP32-based quadcopter made from scratch”[4], which was assembled from individually sourced components by a professional in the fields of computers and robotics. The Kopis Freestyle 4-inch FPV Drone[5] is a mini quadcopter equipped with a first-person camera, though it has a short flight time of five to six minutes unless upgraded with a better battery. The DJI Mini 4 Pro Drone[6] is a relatively large mini quadcopter with a longer battery life of 34 minutes, or 45 minutes with the battery upgrade, a more crash-resistant frame, and a camera with night vision. However, these advantages come at the cost of a significantly higher price ($759 without upgrades) and a weight of 249 grams, just one gram shy of the 250-gram threshold, at which point any drone must be registered with the FAA (Federal Aviation Administration)[3]. The mini quadcopter most similar to their project is the “Open source ESP32-based quadcopter made from scratch”[4], which is composed of individual parts that are all detailed alongside sourcing information. This drone prioritizes stability and was demonstrated to fly quite steadily.

Some aspects that make the MicroCART project unique compared to these mini quadcopters include how their GUI interacts with the CrazyFlie to transmit commands and receive data, as well as the presence of a custom test stand that allows for the testing and configuration of the drone’s flight capabilities while preventing accidents. Most drones on the market are connected to a limited piece of software, with only the open-source drone allowing for easily customizable firmware. Unlike the other open-source drones, however, CrazyFlie is an open-source quadcopter from BitCraze[1], a company that sells both the drones and the equipment required to communicate with them, while also providing troubleshooting software that can be freely downloaded to repair firmware in case of microprocessor errors caused by crashes.

4.1.3 Technical Complexity

The addition of the test stand data at the students' disposal was the most complex feature the team implemented. This feature allowed students in the lab to pull from multiple sources to more confidently work through the lab. Piping data from an analog sensor to a graphical display on a separate system is what leads to the complexity of this feature. Below is a diagram of the pipeline.



Above is an overview of only the test stand data pipeline. In this one pipeline, many disciplines are tested. Start at the rotational sensor, which could be thought of as a high-precision, low-friction potentiometer where Vdd and GND are supplied to the sensor, then through the use of the Arduino's analog pins, a third SENS line can be sampled to determine the rotation.

Now the Arduino takes this voltage reading from the sensor and passes it through a function to convert the voltage to radians. This function only provides the angle of the sensor, but a key data point needed during the lab is the rate of change. This is where the Arduino's onboard timer plays a vital role, with the ability to calculate the delta time, the rate is then able to be calculated. The Arduino now contains both data points, but must wait for a request from the backend before sending the most current data points.

Within the backend, a dedicated thread is assigned to making connections to external sources, primarily the drone. Responsibility for binding to the Arduino was given to this thread. Once a secure connection is made, the thread first requests drone data, then requests the test stand data from the Arduino. These two sources of data are not received in the same format for the GUI to read. This is where the log file handler takes over and formats the requested data into a single line and writes it to a log file. After data is written to the log file, the GUI reads the new set of data points and plots them to a graph that utilizes QT Creator's plotting framework.

This pipeline tests multiple disciplines by containing 3 coding languages, both analog and digital interpretation, and many different development and testing techniques. But by far the most challenging aspect is the timing. This whole pipeline is on a strict timeline for visual clarity; the drone and test stand data need to be aligned so that they are both representing the same point in time.

4.2 Design Exploration

4.2.1 Design Decisions

One of the initial ideas the team had was to streamline the data to the plotter, This would require reworking most of the existing framework that had been built over the last couple of teams. This would allow the data pipeline to have a greater data rate, resulting in a smoother plot. The next big design decision was not to scrap the previous team's work but rather improve upon it so as not to create more work for the team than needed. There are many negatives to doing this, but the benefits are that the team is able to focus on key issues rather than building up a new code base with potentially more issues. A more focused design decision was hardware-oriented. It was to solve a known issue with the Bitcraze drone; the solder joints of the battery connector are very prone to breaking. This would extend the lifetime of the drone before maintenance is required.

4.2.2 Ideation

For the decision to address the flaw in Bitcraze's CrazyFlie design. This pertains to the eventual break of the battery connector's solder joint. This break will occur due to regular use in the lab far quicker than seems acceptable. The first solution is to cut the broken cable, clean the old solder off the PCB, and re-solder the same cable back to the drone.

The next solution was to buy new cable assemblies that allow for longer cables that would take some of the strain off the solder joint.

The team explored the idea of designing a 3D-printed sled system that would contain the battery; this sled system would remove the need for a wired connection in favor of a guided connection.

Next, the team simplified the previous idea and looked to see if any of the header pins on the drone were Vdd and GND. This would then allow for daughter boards that would carry the battery and supply the drone power through the header pins.

The last idea was to use the same specification of male connector as the wired version, but get a through-hole component that we could solder directly to the board, which would have a much stronger connection that would stand up to wear and tear.

4.2.3 Decision-Making and Trade-Off

This problem started to become evident at an inopportune time for the team. Time was a big concern for the solution we picked because the lab that needed the drones was two weeks out from when the idea was picked.

First idea, to just cut clean and re-solder the same cable back to the drone. This was by far the quickest solution, and likewise was held fairly high on the list. But it came with many drawbacks, the first being that the final cable is smaller, resulting in more strain on the solder joining, which results in it breaking faster. The next drawback is that there is potential damage that can be caused to the drone's PCB.

The idea to buy new cable assemblies that would provide the drone with a longer battery connector is ideal; the cables are available online, and the implementation would be staggered. Whenever a solder joint fails, a longer cable would be attached. To meet the time constrain, the cables were available on Amazon Prime, with 2-day shipping.

The sled idea didn't get much traction. It would involve quite a bit of CAD time to first design and more than likely multiple revisions of 3D printing. This would be a huge time sink that wouldn't be guaranteed to work. This would be more suited for next year's team when they have months to develop the design.

The daughter board is a good idea in concept, but none of the header pins were mapped to Vcc, so it was instantly out.

The last idea was to acquire a through-hole connector that would be soldered directly to the drone's PCB. This had some traction because of the ruggedness it provided, but the availability of the connectors wouldn't meet the time constraints present. Therefore, this solution is relegated to a future team.

4.3 Final Design

4.3.1 Overview

As this project is built upon year by year from each MicroCART team, the entire design from the ground up is not completed in one year. The current revision of the project started in 2019. The first 3 years of this revision were spent building the core framework of the application and designing the CprE 488 lab. The next two teams expanded functionality and worked to make the framework more robust. The current team they are picking up the project in the second half of its lifespan. This is where the final implementation of features and the polish is applied.

There are two large-scale aspects that the team accomplished. The first is the integration of an external test stand to provide data that isn't directly reported from the drone itself. The reason this is important is that when writing software for the drone, the drone's self-reported data, such as its rotational speed and angle, is not always reliable. This may stem from an error in the student software or a faulty sensor on the drone. But having an external source to validate the drone's self-reported data allows the students to confidently determine if the software on the drone is working accordingly. The second large-scale aspect the team accomplished fell into the polish category of work. This entailed improving the efficiency of the application that students will have to use. Through the use of rigorous code evaluation, many sections of the code were altered to use fewer resources. The functionality of the code remained intact, but it lowered its consumption of vital computing resources. The reason this is important is that the application runs on a virtual machine, which is analogous to running a computer simulation on a computer. This means the virtual machine is allotted far fewer resources than the machine which the virtual machine is running on.

4.3.2 Detailed Design and Visual(s)

Starting with the test stand data, the team's goal was to display the data from the integrated test stand to the applications plotter. While this would be trivial to do in a stand-alone application, the real challenge was navigating the existing framework that was set up to only display data from the drone. Taking a deeper look at the pipeline that was constructed, first, look at the sensor.

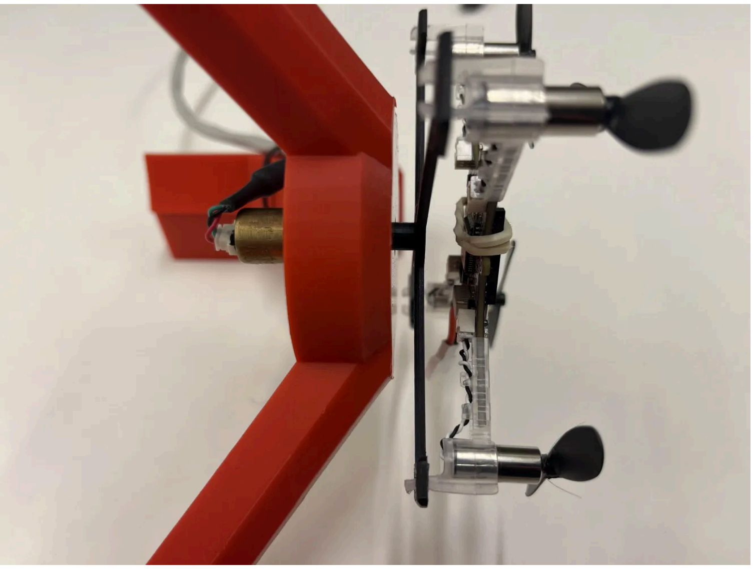


Figure 4.1: Test stand sensor

This rotational sensor is embedded into the test stand. The sensors are interacted with through the use of a drone holder that locks the drone to a single axis of rotation.

The drone holder fits over the sensor probe and allows the sensor to read the exact rotation of the drone on a given axis.

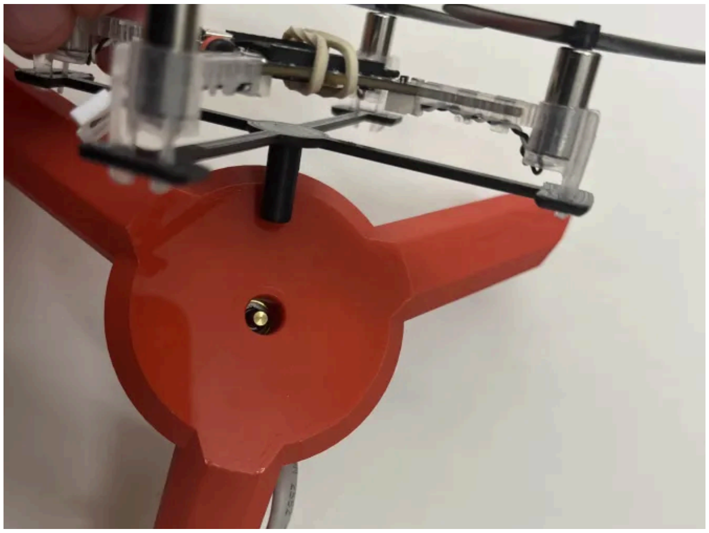


Figure 4.2: Drone holder

This sensor is analogous to a potentiometer in how it behaves electronically. Mechanically, the sensor has very low friction to not introduce large amounts of error into an already delicate system. The sensor has a three-pin interface like most potentiometers: Vcc, GND, and Sense. Vcc is the upper bound voltage, and GND is the lower bound voltage. This sets the range for the Sense pin, which is where the rotational data is read.

To interface with this sensor, the easiest way is to use an Arduino Nano, the Arduino has all the features necessary to provide Vcc and GND as well as read an analog signal.

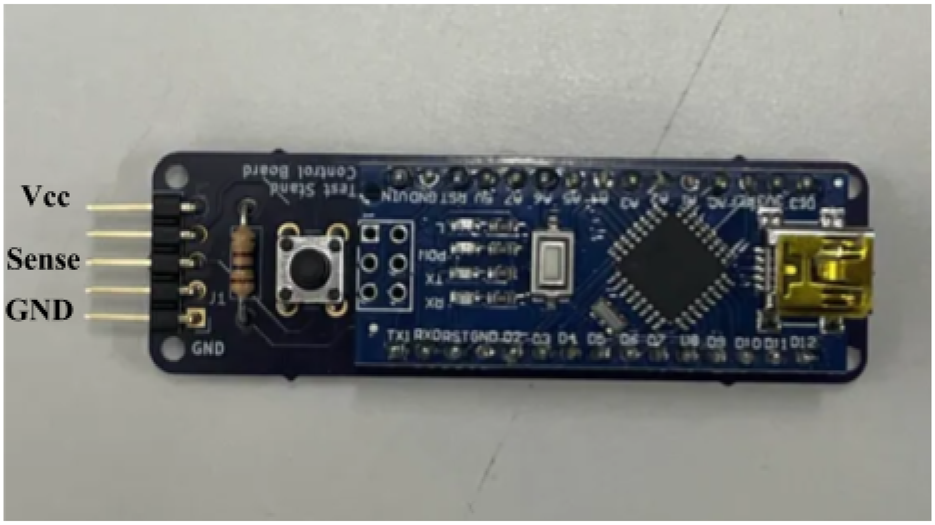
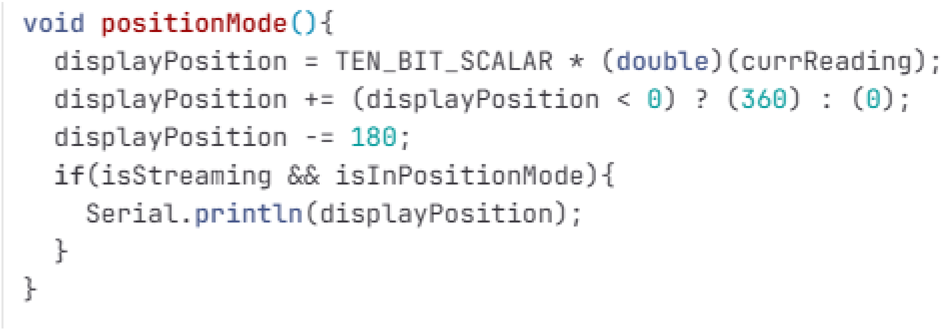


Figure 4.3: Arduino Nano with analog pin VCC, GND, and Sense

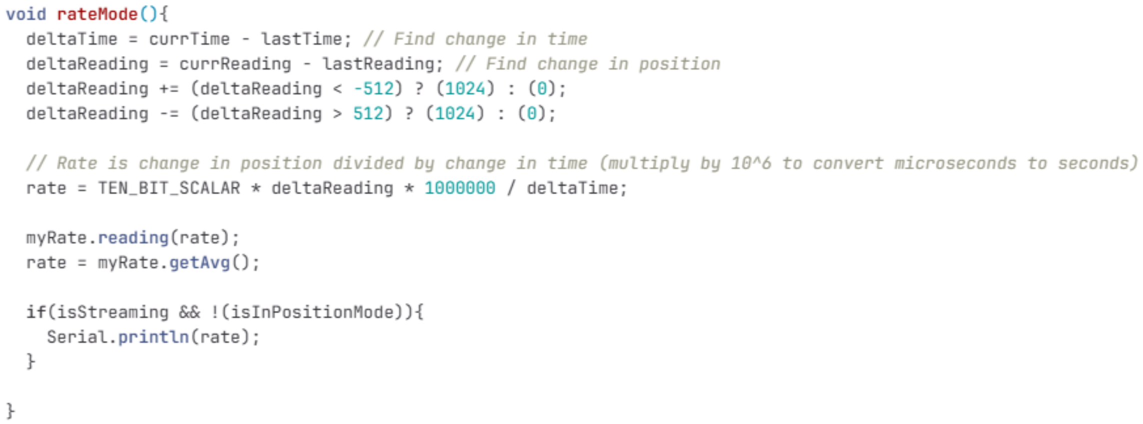
The Arduino contains just enough computational power to act as an interpreter for the sensor. An analog signal can't be given directly to a standard PC, this is why it is a vital component. The Arduino is able to take the analog data and convert it to digital data that can be transmitted via a USB serial connection. To convert the analog signal to digital. By enabling the analog pin in the firmware, it passes voltage through its integrated ADC. This allows the team to write firmware for the Arduino. Now, within the Arduino firmware, the voltage Sense pin is a float primitive that interacts nicely with code. A simple C function is used to convert the voltage to radians. The radians are then converted to degrees for easier human readability.



```
void positionMode(){
  displayPosition = TEN_BIT_SCALAR * (double)(currReading);
  displayPosition += (displayPosition < 0) ? (360) : (0);
  displayPosition -= 180;
  if(isStreaming && isInPositionMode){
    Serial.println(displayPosition);
  }
}
```

Figure 4.4: Function to convert voltage to radians

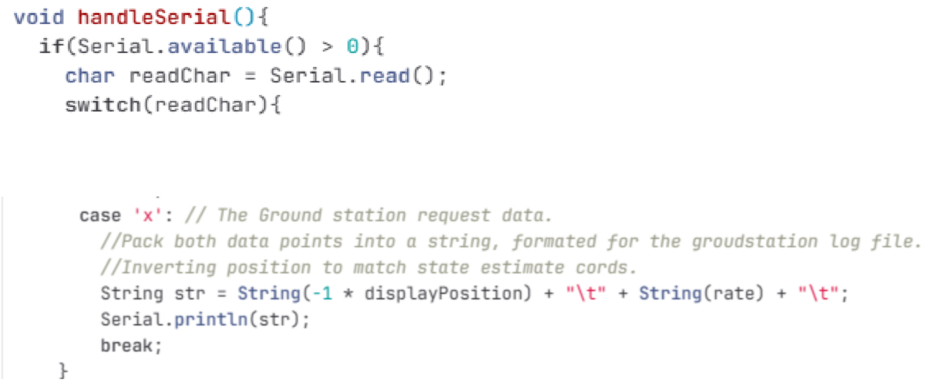
The next piece of data needed is the rate of change in degrees per second. This requires a time component. Another reason for using an Arduino is that it contains an onboard timer that allows for the use of delta time. In this case delta time is the time in seconds for one loop. By having the difference in time between two angle measurements and the difference of the two angles, the rate of change is now available to report.



```
void rateMode(){
  deltaTime = currTime - lastTime; // Find change in time
  deltaReading = currReading - lastReading; // Find change in position
  deltaReading += (deltaReading < -512) ? (1024) : (0);
  deltaReading -= (deltaReading > 512) ? (1024) : (0);

  // Rate is change in position divided by change in time (multiply by 10^6 to convert microseconds to seconds)
  rate = TEN_BIT_SCALAR * deltaReading * 1000000 / deltaTime;

  myRate.reading(rate);
  rate = myRate.getAvg();

  if(isStreaming && !(isInPositionMode)){
    Serial.println(rate);
  }

}
```

Figure 4.5: Function for rate calculation

These calculations happen as fast as the Arduino can process them to communicate over a serial port. During each loop, the Arduino checks to see if a request has been made from the ground station; if so, it packages the two most current data points and transmits them over the serial port.



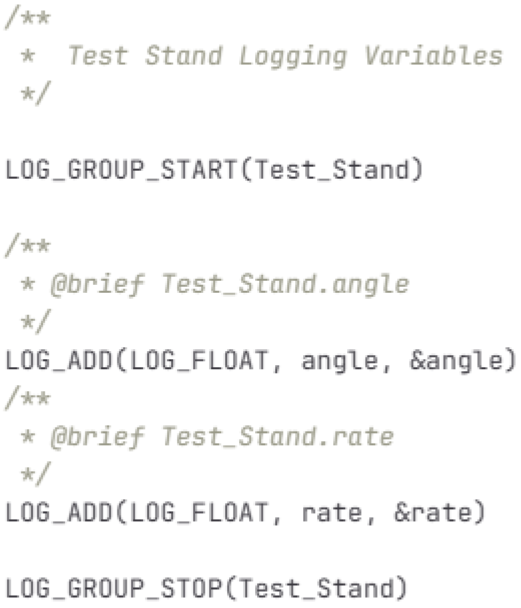
```
void handleSerial(){
  if(Serial.available() > 0){
    char readChar = Serial.read();
    switch(readChar){


      case 'x': // The Ground station request data.
        //Pack both data points into a string, formated for the groudstation log file.
        //Inverting position to match state estimate cords.
        String str = String(-1 * displayPosition) + "\t" + String(rate) + "\t";
        Serial.println(str);
        break;
    }
```

Figure 4.6: Requesting and sending data

The data has now been packaged into a universal format, the MicroCART application can now be addressed. To reiterate, the team was working with a framework that was not intended for this type of integration. In lieu of rewriting the ground station that facilitates the transmission and reception of data to the radio, which in turn communicates to the drone, the team decided to make due and fit it into the existing framework.

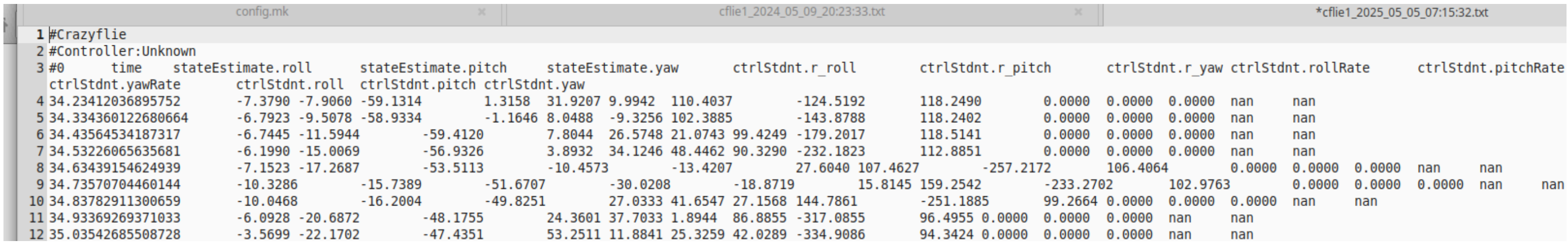
Starting with the ground station as a whole, the two threads that pertain to this are the connection thread that is used to communicate with the drone radio and the logger thread that is used to read the drone's reported data and format it into a log file. The initial thought is to set up the serial communication in the connection thread, but this thread specializes in communication with the drone. So the serial connection was created in the log handler thread. When thinking in terms that the serial connection is just another set of data points it makes sense to set it up there.

This is where things get a little tricky. The logger was never intended to receive external data other than the data of the drone; therefore, the format of the log file is entirely dependent on what the drone says it is reporting. So, to allow for this external source, we had to tell the drone to log the test stand data. The drone has no concept of what that data is other than it being a float defined in its firmware. The drone has no way to get this data; therefore, it responds with empty data. This was necessary for the log file handler to reserve space for the test stand data.



```
/**
 *  Test Stand Logging Variables
 */

LOG_GROUP_START(Test_Stand)

/**
 * @brief Test_Stand.angle
 */
LOG_ADD(LOG_FLOAT, angle, &angle)
/**
 * @brief Test_Stand.rate
 */
LOG_ADD(LOG_FLOAT, rate, &rate)

LOG_GROUP_STOP(Test_Stand)
```

Figure 4.7: Creating test stand logging variables

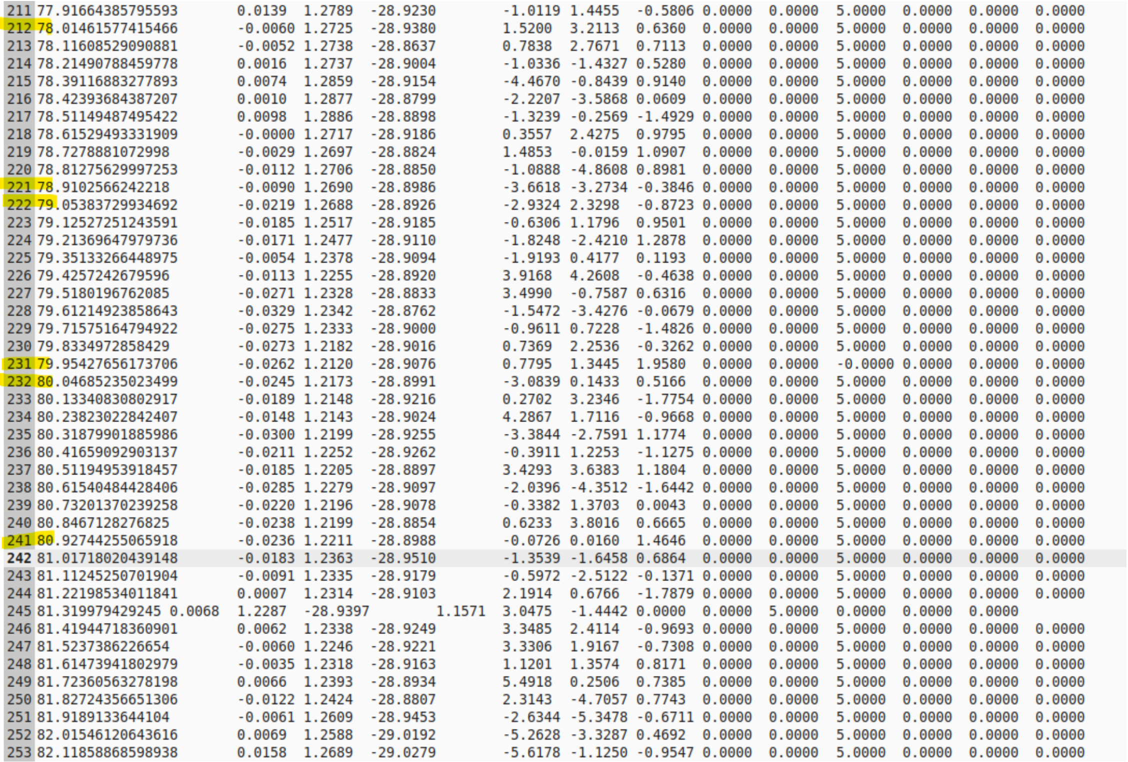


config.mk
cflie1_2024_05_09_20:23:33.txt
*cflie1_2025_05_05_07:15:32.txt

```
#Crazyflie
#Controller:Unknown
#0      time    stateEstimate.roll      stateEstimate.pitch     stateEstimate.yaw       ctrlStdnt.r_roll        ctrlStdnt.r_pitch       ctrlStdnt.r_yaw ctrlStdnt.rollRate      ctrlStdnt.pitchRate
ctrlStdnt.yawRate       ctrlStdnt.roll  ctrlStdnt.pitch ctrlStdnt.yaw
34.23412036895752       -7.3790 -7.9060 -59.1314        1.3158  31.9207 9.9942  110.4037        -124.5192       118.2490        0.0000  0.0000  0.0000  nan     nan
34.334360122680664      -6.7923 -9.5078 -58.9334        -1.1646 8.0488  -9.3256 102.3885        -143.8788       118.2402        0.0000  0.0000  0.0000  nan     nan
34.43564534187317       -6.7445 -11.5944        -59.4120        7.8044  26.5748 21.0743 99.4249 -179.2017       118.5141        0.0000  0.0000  0.0000  nan     nan
34.53226065635681       -6.1990 -15.0069        -56.9326        3.8932  34.1246 48.4462 90.3290 -232.1823       112.8851        0.0000  0.0000  0.0000  nan     nan
34.63439154624939       -7.1523 -17.2687        -53.5113        -10.4573        -13.4207        27.6040 107.4627        -257.2172       106.4064        0.0000  0.0000  0.0000  nan     nan
34.73570704460144       -10.3286        -15.7389        -51.6707        -30.0208        -18.8719        15.8145 159.2542        -233.2702       102.9763        0.0000  0.0000  0.0000  nan     nan
34.83782911300659       -10.0468        -16.2004        -49.8251        27.0333 41.6547 27.1568 144.7861        -251.1885       99.2664 0.0000  0.0000  0.0000  nan     nan
34.93369269371033       -6.0928 -20.6872        -48.1755        24.3601 37.7033 1.8944  86.8855 -317.0855       96.4955 0.0000  0.0000  0.0000  nan     nan
35.03542685508728       -3.5699 -22.1702        -47.4351        53.2511 11.8841 25.3259 42.0289 -334.9086       94.3424 0.0000  0.0000  0.0000  nan     nan
```

Figure 4.8: Log file header

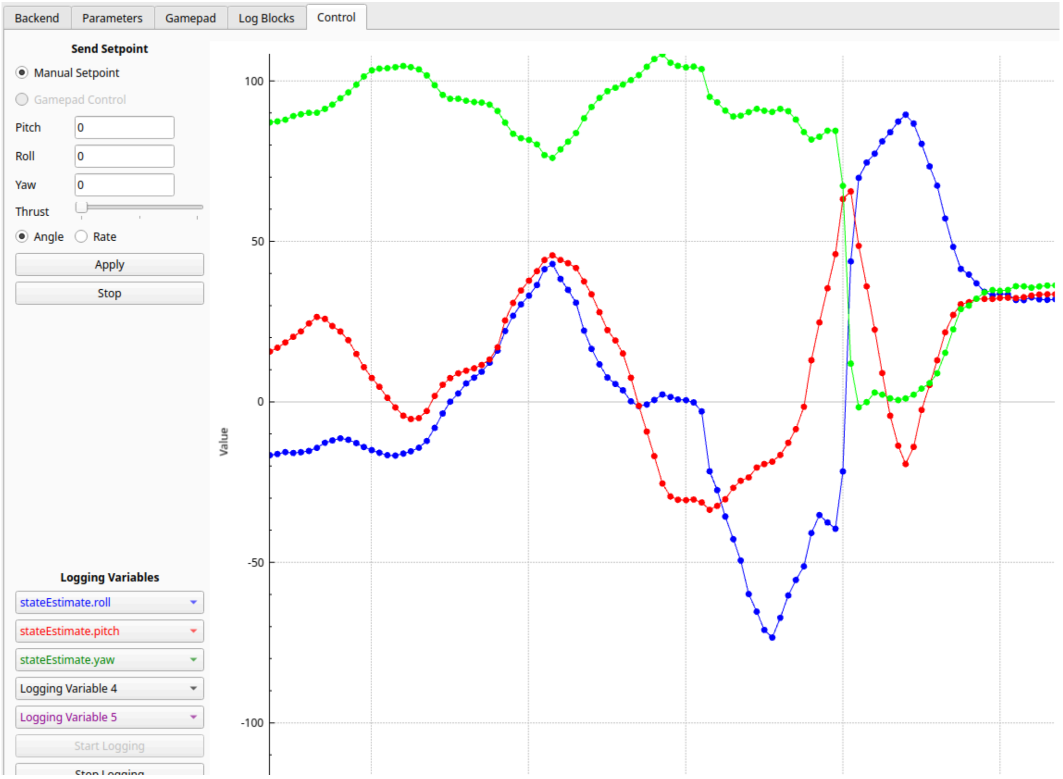
Now that space is reserved in the log file, the data is intercepted before it is formatted, and the serial port is polled, and the data is applied there, overwriting the empty data from the drone. The formatted string is then written to the log file.



```
77.91664385795593       0.0139  1.2789  -28.9230        -1.0119 1.4455  -0.5806 0.0000  0.0000  5.0000  0.0000  0.0000  0.0000
78.01461577415466       -0.0060 1.2725  -28.9380        1.5200  3.2113  0.6360  0.0000  0.0000  5.0000  0.0000  0.0000  0.0000
78.11608529090881       -0.0052 1.2738  -28.8637        0.7838  2.7671  0.7113  0.0000  0.0000  5.0000  0.0000  0.0000  0.0000
78.21490788459778       0.0016  1.2737  -28.9004        -1.0336 -1.4327 0.5280  0.0000  0.0000  5.0000  0.0000  0.0000  0.0000
78.39116883277893       0.0074  1.2859  -28.9154        -4.4670 -0.8439 0.9140  0.0000  0.0000  5.0000  0.0000  0.0000  0.0000
78.42393684387207       0.0010  1.2877  -28.8799        -2.2207 -3.5868 0.0609  0.0000  0.0000  5.0000  0.0000  0.0000  0.0000
78.51149487495422       0.0098  1.2886  -28.8898        -1.3239 -0.2569 -1.4929 0.0000  0.0000  5.0000  0.0000  0.0000  0.0000
78.61529493331909       -0.0000 1.2717  -28.9186        0.3557  2.4275  0.9795  0.0000  0.0000  5.0000  0.0000  0.0000  0.0000
78.7278881072998        -0.0029 1.2697  -28.8824        1.4853  -0.0159 1.0907  0.0000  0.0000  5.0000  0.0000  0.0000  0.0000
78.81275629997253       -0.0112 1.2706  -28.8850        -1.0888 -4.8608 0.8981  0.0000  0.0000  5.0000  0.0000  0.0000  0.0000
78.9102566242218        -0.0090 1.2690  -28.8986        -3.6618 -3.2734 -0.3846 0.0000  0.0000  5.0000  0.0000  0.0000  0.0000
79.05383729934692       -0.0219 1.2688  -28.8926        -2.9324 2.3298  -0.8723 0.0000  0.0000  5.0000  0.0000  0.0000  0.0000
79.12527251243591       -0.0185 1.2517  -28.9185        -0.6306 1.1796  0.9501  0.0000  0.0000  5.0000  0.0000  0.0000  0.0000
79.21369647979736       -0.0171 1.2477  -28.9110        -1.8248 -2.4210 1.2878  0.0000  0.0000  5.0000  0.0000  0.0000  0.0000
79.35133266448975       -0.0054 1.2378  -28.9094        -1.9193 0.4177  0.1193  0.0000  0.0000  5.0000  0.0000  0.0000  0.0000
79.4257242679596        -0.0113 1.2255  -28.8920        3.9168  4.2608  -0.4638 0.0000  0.0000  5.0000  0.0000  0.0000  0.0000
79.5180196762085        -0.0271 1.2328  -28.8833        3.4990  -0.7587 0.6316  0.0000  0.0000  5.0000  0.0000  0.0000  0.0000
79.61214923858643       -0.0329 1.2342  -28.8762        -1.5472 -3.4276 -0.0679 0.0000  0.0000  5.0000  0.0000  0.0000  0.0000
79.71575164794922       -0.0275 1.2333  -28.9000        -0.9611 0.7228  -1.4826 0.0000  0.0000  5.0000  0.0000  0.0000  0.0000
79.8334972858429        -0.0273 1.2182  -28.9016        0.7369  2.2536  -0.3262 0.0000  0.0000  5.0000  0.0000  0.0000  0.0000
79.95427656173706       -0.0262 1.2120  -28.9076        0.7795  1.3445  1.9580  0.0000  0.0000  -0.0000 0.0000  0.0000  0.0000
80.04685235023499       -0.0245 1.2173  -28.8991        -3.0839 0.1433  0.5166  0.0000  0.0000  5.0000  0.0000  0.0000  0.0000
80.13340830802917       -0.0189 1.2148  -28.9216        0.2702  3.2346  -1.7754 0.0000  0.0000  5.0000  0.0000  0.0000  0.0000
80.23823022842407       -0.0148 1.2143  -28.9024        4.2867  1.7116  -0.9668 0.0000  0.0000  5.0000  0.0000  0.0000  0.0000
80.31879901885986       -0.0300 1.2199  -28.9255        -3.3844 -2.7591 1.1774  0.0000  0.0000  5.0000  0.0000  0.0000  0.0000
80.41659092903137       -0.0211 1.2252  -28.9262        -0.3911 1.2253  -1.1275 0.0000  0.0000  5.0000  0.0000  0.0000  0.0000
80.51194953918457       -0.0185 1.2205  -28.8897        3.4293  3.6383  1.1804  0.0000  0.0000  5.0000  0.0000  0.0000  0.0000
80.61540484428406       -0.0285 1.2279  -28.9097        -2.0396 -4.3512 -1.6442 0.0000  0.0000  5.0000  0.0000  0.0000  0.0000
80.73201370239258       -0.0220 1.2196  -28.9078        -0.3382 1.3703  0.0043  0.0000  0.0000  5.0000  0.0000  0.0000  0.0000
80.8467128276825        -0.0238 1.2199  -28.8854        0.6233  3.8016  0.6665  0.0000  0.0000  5.0000  0.0000  0.0000  0.0000
80.92744255065918       -0.0236 1.2211  -28.8988        -0.0726 0.0160  1.4646  0.0000  0.0000  5.0000  0.0000  0.0000  0.0000
81.01718020439148       -0.0183 1.2363  -28.9510        -1.3539 -1.6458 0.6864  0.0000  0.0000  5.0000  0.0000  0.0000  0.0000
81.11245250701904       -0.0091 1.2335  -28.9179        -0.5972 -2.5122 -0.1371 0.0000  0.0000  5.0000  0.0000  0.0000  0.0000
81.22198534011841       0.0007  1.2314  -28.9103        2.1914  0.6766  -1.7879 0.0000  0.0000  5.0000  0.0000  0.0000  0.0000
81.319979429245 0.0068  1.2287  -28.9397        1.1571  3.0475  -1.4442 0.0000  0.0000  5.0000  0.0000  0.0000  0.0000
81.41944718360901       0.0062  1.2338  -28.9249        3.3485  2.4114  -0.9693 0.0000  0.0000  5.0000  0.0000  0.0000  0.0000
81.5237386226654        -0.0060 1.2246  -28.9221        3.3306  1.9167  -0.7308 0.0000  0.0000  5.0000  0.0000  0.0000  0.0000
81.61473941802979       -0.0035 1.2318  -28.9163        1.1201  1.3574  0.8171  0.0000  0.0000  5.0000  0.0000  0.0000  0.0000
81.72360563278198       0.0066  1.2393  -28.8934        5.4918  0.2506  0.7385  0.0000  0.0000  5.0000  0.0000  0.0000  0.0000
81.82724356651306       -0.0122 1.2424  -28.8807        2.3143  -4.7057 0.7743  0.0000  0.0000  5.0000  0.0000  0.0000  0.0000
81.9189133644104        -0.0061 1.2609  -28.9453        -2.6344 -5.3478 -0.6711 0.0000  0.0000  5.0000  0.0000  0.0000  0.0000
82.01546120643616       0.0069  1.2588  -29.0192        -5.2628 -3.3287 0.4692  0.0000  0.0000  5.0000  0.0000  0.0000  0.0000
82.11858868598938       0.0158  1.2689  -29.0279        -5.6178 -1.1250 -0.9547 0.0000  0.0000  5.0000  0.0000  0.0000  0.0000
```

Figure 4.9: Log file data

As the log file handler is writing lines to the log file, the GUI application follows along, reading each new line and parsing it to find the selected variables for plotting, then takes the values in the position of that desired variable, the position is defined by the header. The QT plotter framework then takes over from there to plot the data on the graph in the GUI.



Send Setpoint

● Manual Setpoint

○ Gamepad Control

Pitch 0

Roll 0

Yaw 0

Thrust

● Angle ○ Rate

Logging Variables

stateEstimate.roll

stateEstimate.pitch

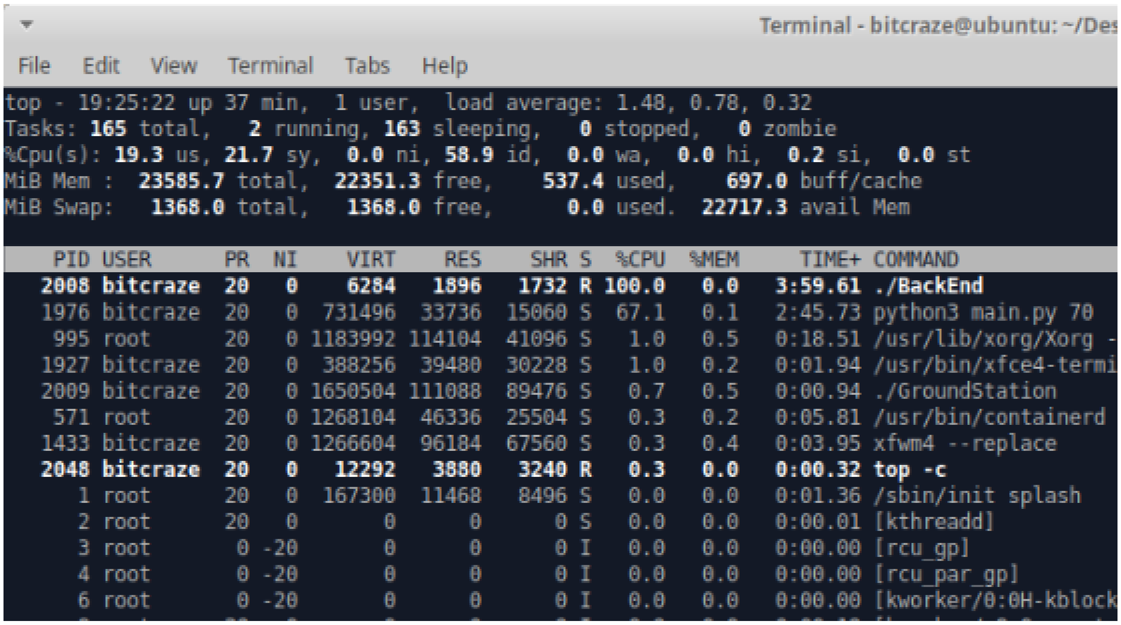
stateEstimate.yaw

Logging Variable 4

Logging Variable 5

Figure 4.10: Plotter

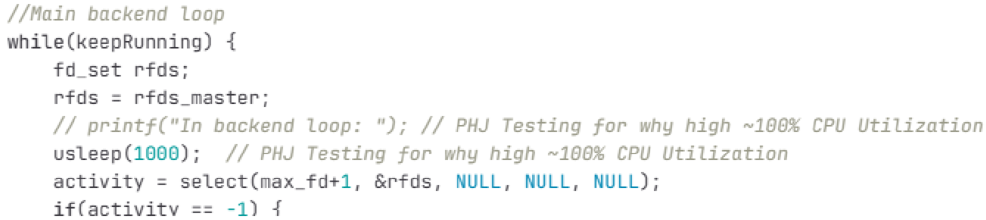
The other large contribution was in the form of improving the efficiency of the code. Originally, the functionality of each sub-system was sound. But when put together is a real-world application, all the minor inefficiencies add up to create an unstable environment. This is primarily due to running on a VM where resources are limited, and depending on the host machine, not enough resources are available to the VM to keep up with its consumption. This led to the VM's CPU utilization consistently reaching over 100%, meaning that processes were being stalled.



```
top - 19:25:22 up 37 min,  1 user,  load average: 1.48, 0.78, 0.32
Tasks: 165 total,   2 running, 163 sleeping,   0 stopped,   0 zombie
%Cpu(s): 19.3 us, 21.7 sy,  0.0 ni, 58.9 id,  0.0 wa,  0.0 hi,  0.2 si,  0.0 st
MiB Mem :  23585.7 total,  22351.3 free,    537.4 used,    697.0 buff/cache
MiB Swap:   1368.0 total,   1368.0 free,      0.0 used.  22717.3 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   2008 bitcraze  20   0    6284   1896   1732 R 100.0   0.0   3:59.61 ./BackEnd
   1976 bitcraze  20   0  731496  33736  15060 S  67.1   0.1   2:45.73 python3 main.py 70
    995 root      20   0 1183992 114104  41096 S   1.0   0.5   0:18.51 /usr/lib/xorg/Xorg -
   1927 bitcraze  20   0  388256  39480  30228 S   1.0   0.2   0:01.94 /usr/bin/xfce4-termi
   2009 bitcraze  20   0 1650504 111088  89476 S   0.7   0.5   0:00.94 ./GroundStation
    571 root      20   0 1268104  46336  25504 S   0.3   0.2   0:05.81 /usr/bin/containerd
   1433 bitcraze  20   0 1266604  96184  67560 S   0.3   0.4   0:03.95 xfwm4 --replace
   2048 bitcraze  20   0   12292   3880   3240 R   0.3   0.0   0:00.32 top -c
      1 root      20   0  167300  11468   8496 S   0.0   0.0   0:01.36 /sbin/init splash
      2 root      20   0       0      0      0 S   0.0   0.0   0:00.01 [kthreadd]
      3 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 [rcu_gp]
      4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 [rcu_par_gp]
      6 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 [kworker/0:0H-kblock
```

Figure 4.11: High CPU utilization

The first improvement to the code was finding out about the overuse of busy waits; the ideal use is none, but it's understandable when testing the sub-systems individually, the busy wait doesn't seem to have a dramatic effect on performance. But when multiple systems are busy waiting, they eat up CPU utilization, literally doing nothing.



```
//Main backend loop
while(keepRunning) {
    fd_set rfds;
    rfds = rfds_master;
    // printf("In backend loop: "); // PHJ Testing for why high ~100% CPU Utilization
    usleep(1000);  // PHJ Testing for why high ~100% CPU Utilization
    activity = select(max_fd+1, &rfds, NULL, NULL, NULL);
    if(activity == -1) {
```

Figure 4.12: Busy-wait

The second improvement was achieved by finding a misconception that had permeated for a couple of years. When logging, there is fine-tuning of how often the data is sent from the drone to the ground station. This variable is the period of the data transmissions. This had been treated as a frequency. So when previous teams were running into this high CPU utilization, they would lower what they thought was the frequency of data, but in fact, they were increasing the frequency of data. Once this was realized, the team corrected the documentation, instructing users to do the opposite of what truly needed to be done. The team also found a stable logging period that shouldn't need to be changed.

4.3.3 Functionality

As this project revolves around the CprE 488 lab, our implementations were directly used during our final semester. This allowed us to get crucial feedback for overlooked errors in the lab manual and many quality-of-life improvements from the students.

As students in the lab, their first interaction with material created by the team was the MP-4 manual that first gives the students a rundown of the hardware involved such as all the features of the drone, how to connect a battery and power it on, as well as the new edition of the test stand. The test stand portion was entirely written by the team because of the new external sensor that could now be displayed in the application.

The success was measured in the lack of negative feedback rather than overwhelmingly positive feedback. It may seem weird, but the student doesn't know how much the overall performance has increased when they never had to experience all the issues that came along with the previous year's revision. As for the test stand sensor, it was put to use and proved useful for some students during part two of the lab, where they were writing their own firmware for the drone.

The actual use case of everything implemented by the team is laid out in the MP-4 lab manual (Appendix 1). Other than the test stand, the interaction with the improved efficiency is the lack of frustrating of having the connection drop mid testing, and according to Dr. Jones, last year this was the source of a major headache. But this Year the problem was gone, it didn't happen to anyone once.

4.3.4 Areas of Challenge

One of the challenges that the team faced in implementing our design was determining which piece of code logs and graphs data, and how it does it. By back-tracing the GUI code, the team was able to follow the execution flow of the log file handler Python script to understand how it was able to log to and graph from a log file. Multiple challenges arose as the team continued with test stand implementation. For example, the log file was unable to log more than 12 logging variables because it was hard-coded by the previous team.

Besides that, the team was struggling to get the test stand angle and rate to show up as a logging variable for the purpose of graphing those values on the GUI. In the end, the team consulted their advisor, Dr. Jones. The team followed his advice and traversed through CrazyFlie's firmware and found a way to add dummy logging variables to the CrazyFlie as a way to get it to show up in the log file, and it succeeded.

4.4 Technology Considerations

In our project, we are utilizing the following technologies:

- Virtual Machine (Software)
  - Project environment and means of deployment
  - GUI & remote connection
  - (-) Prone to connection issues even with simple USB inputs
- CrazyFlie - BitCraze's mini quadcopter model (Hardware)
  - Open source software and hardware for CrazyFlie
- Network technology / Radio Communications (Hardware)
  - The wireless communication method
- Microcontrollers/Low-level programming (Hardware/Software)
  - Drones onboard software
  - (-) Limited potential
- PCB fabrication (Solution)
  - Battery retention
- Test Stand (Hardware)
  - Arduino
  - Rotary encoder

The hardware is a set standard that we are unable to change in the MicroCART project. Save for maybe the exception of the ‘battery retention racks’ or lack thereof, that can be remedied by adding PCB-fabricated designs, which can then be attached to the CrazyFlie. The virtual lab environment that is available on the CPR E 488 class website is what will be changed. Specifically, editing the code that is part of the lab environment download for the CPR E 488 MP-4 lab. Utilizing a custom GUI made by previous groups, we have to look through how they made it while we were editing it. The CrazyFlie mini quadcopter is a relatively inexpensive mini quadcopter and as such has small equipment with limited possibilities. In order to communicate with the lab drones, a USB radio is used to communicate with the CrazyFlie, though the radio can also communicate with other equipment that is available in the lab, such as sensors, which can then be fed back to the CrazyFlie and enable more advanced flying methods. Since it has been decided that this year’s MicroCART team cannot change the hardware, the only changes they could make are through the software, unless they were to choose to start from scratch, but due to the limited timeline, it would be very unwise to choose to do so.

5 Testing

5.1 Unit Testing

Our infrastructure has several software components, allowing us to perform unit tests on each of the parts individually to ensure that the system as a whole is working properly. These components include the front-end GUIs, the backend modules, and the ground station. The GUIs can be tested by entering inputs and verifying that the correct outputs are there and that the correct thing is sent to the terminal. The backend and ground station are more complex to unit test as one component, so this can be broken down into smaller components that can be tested manually through checking terminal output.

Besides that, the team conducted unit testing on the test stand Arduino because the default .ino code given was from a previous team that had failed to implement it. Through streamlined debugging, the team tested multiple test stands to ensure that the rotational angle and rate provided by the sensor are correct. The team then compared these third-party sensor values to the CrazyFlie's onboard sensor values to determine if they are valid.

5.2 Interface Testing

Interface testing would be more critical to our design due to the level of complexity in communication between different components. Since we will not be rewriting the firmware code for the CrazyFlie, we will not need to test the interface between the open-source CrazyFlie and Crazyradio. Therefore, we would need to go through a thorough testing for the rest of the components, which are the backend and frontend communication via UNIX sockets. Our method of testing involves inserting print statements when each process sends and receives a message, as well as outputting the outputs to a file for debugging purposes to ensure that data is correctly getting passed, which can be done easily with IDEs like Visual Studio Code.

Besides that, a method of interface testing that the team has adopted is inserting logging variables into the firmware of the CrazyFlie so that external values such as the test stand angle and rate data could be plotted on the graph. This allowed the team to firmly determine if the overall test stand implementation is working.

5.3 Integration Testing

Integration testing will be similar to our interface testing, such as running the components bit-by-bit/individually and not as an overall large component, and generating tests that will ensure the same output is generated when certain inputs are entered, which can be easily done through the GUI or through some effort through the CLI.

5.4 System Testing

After each component is tested individually, we will move on the test the entirety of the system altogether while running the tests with a CrazyFlie drone and conducting Lab 4 step by step. Through the use of our clearly provided documentation of Lab 4 as well as the GUI, we would be able to complete Lab 4 without encountering any errors or bugs, while improving any quality of life issues that arise. This is primarily done before deployment. The team conducted system testing on three different lab computers in Coover 2041 simultaneously to ensure that conducting MP-4 concurrently would work, as we are dealing with radio and packet transmission.

5.5 Regression Testing

MicroCART is based on updating and fixing up the work from teams as far back as 6 years ago. As the project is in an already working state, what we need to do is improve the project and make sure all pieces are working together, just like, if not better, than before. We are ensuring that new features do not break the old functionality by utilizing observe, test, check results, repeat the first three as needed, and finally push the solutions to a development branch for further testing with the other group members' work. By utilizing this process to test for breakage along with utilizing Git to observe changes that occur in the workspace, and backtrack when an update breaks something despite testing for otherwise, we are able to ensure stable testing and improvement.

5.6 Acceptance Testing

The first step of acceptance testing is to ensure client approval, which is accomplished by consulting our advisor, Dr. Jones, before deploying the software for lab use. If we are given the green light by Dr. Jones, we would continue with the deployment and the actual delivery of MP-4. We would then request feedback from students and compare it with last year's lab feedback to deduce if there was an improvement in the overall quality of the lab and their experience. Overall, the team received mixed reviews from the students, but almost none of the bad reviews were on the features that we have implemented, but multiple other issues that have suddenly popped up or existing issues that the team has yet to solve.

5.7 User Testing

Before deployment, the team would act as if they were CPRE 488 students and perform MP-4 while using the current implementation. If a majority of the team comes to an agreement, they would reach out to the current TA (Teaching Assistant) of CPRE 488 and a past student of CPRE 488 to have them assist the team in performing MP-4 while forcibly finding edge cases that the team has overlooked. Overall, the TA has found several issues regarding the current implementation, one of which is that the team accidentally included answers for the PID values for Part 1 of the lab. These issues were fixed, and MP-4 was deployed smoothly.

5.8 Results

Overall, unit testing allowed the team to avoid making assumptions about the test stand implementation that was created by a prior MicroCART team. The team was able to pinpoint the problem of the current implementation as soon as they were able to set up testing for multiple test stand Arduinos. One of the issues that was uncovered by the team was that the yaw rate and angle were inverted if compared to the yaw rate and angle from the onboard sensors.

Besides that, user testing made a difference in our deployment. The assistance from the TA of CPRE 488 helped us achieve a near-flawless deployment of MP-4, with the issue being that the TA has found several issues regarding the current implementation, one of which is that the team accidentally included answers for the PID values for Part 1 of the lab. These issues were fixed, and MP-4 was deployed smoothly.

6 Implementation

The team was successful in implementing the test stand, which is one of the key components of the final design, and was one of the core features that previous MicroCART teams failed to implement. This test stand provides third-party roll, pitch, and yaw sensor values, and plots said values to a graph to assist in tuning the PID controller, which is the introductory part 1 of MP-4. The test stand assists students in deducing the parameters of the PID controller, such as kd, kp, and ki.

Besides that, the team was able to improve the performance of the VM, which is the development platform for MP-4, CPU utilization of the VM was decreased from 100% to 70%. Performance optimization was also one of our main design requirements because previous teams suffered heavy criticism from past CPRE 488 students for the numerous GUI and VM crashes they experienced. This was a good implementation by the team, as this year's MP-4 was dubbed the smoothest MP-4 in the history of MicroCART.

One of the features that the team failed to implement was the global positioning system (GPS) for the CrazyFlie. This was meant to be used in conjunction with the lighthouse system to autonomously fly the CrazyFlie in a 3-D space, maintaining a steady flight while receiving the current Coordinates of the CrazyFlie and the coordinates that it will be flying to. The team was unfortunately unable to complete this implementation because of the difficulty of the implementation and the time constraints faced after implementing the test stand.

A solution for how the battery is held on to the drone quickly became an issue right before the deployment for the CPRE 488 lab. This causes the team to scramble to come up with a solution. The problem stemmed from weak solder joints that would often break on drones with heavy use. The main constraint to the problem was time. Deployment was in two weeks, so a 3D printed carrier was out of reach with the given deadline. This forced the team to use a quicker fix and kicked the real problem down the road. A more permanent solution is in the works, but it will have to be finished by a future team. This is a known problem, so a solution is vital. Hopefully, next year's team can take what has been planned and run with it.

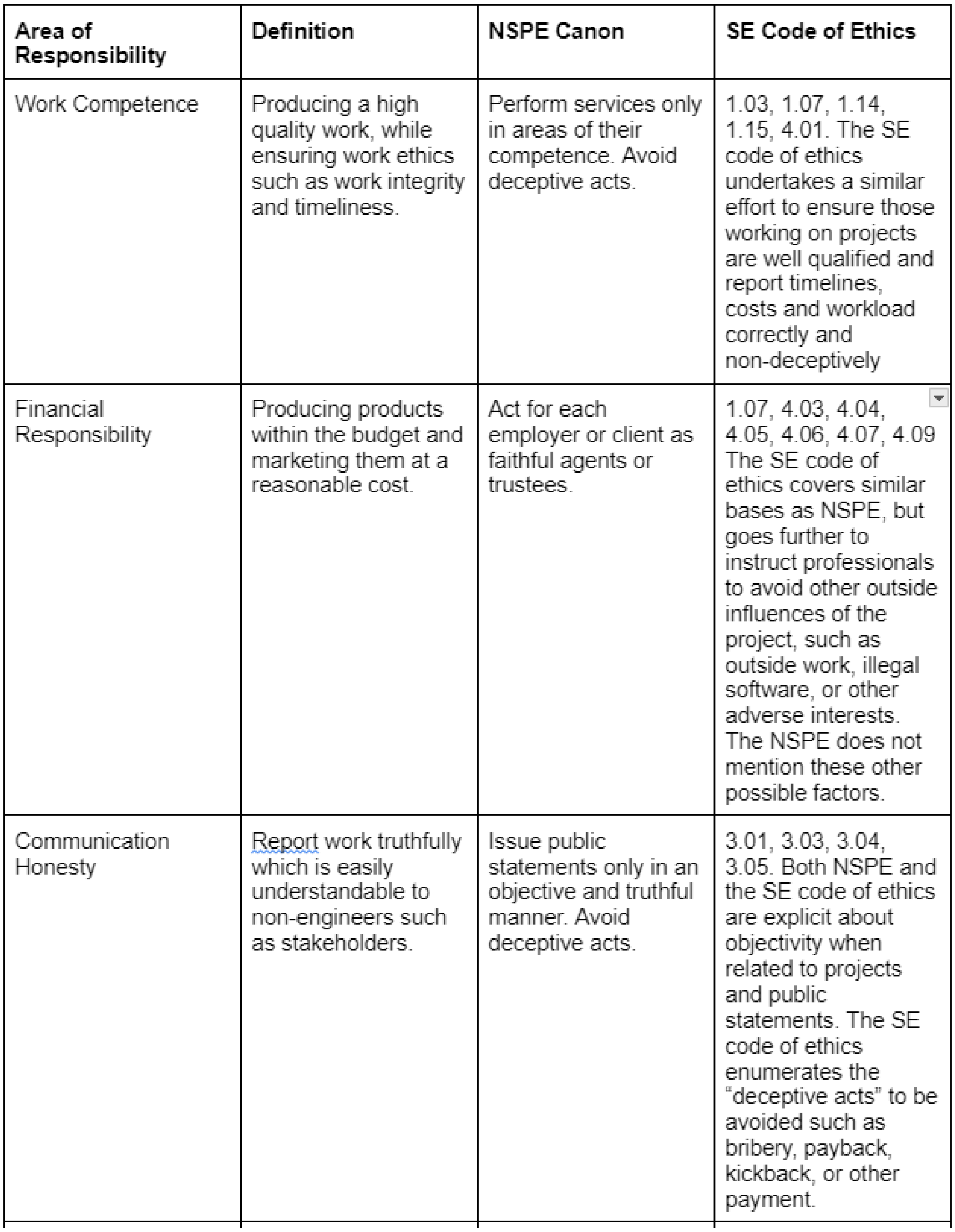
6.1 Design Analysis

The test stand implementation works well as the yaw, pitch, and roll sensor data from the test stand are almost identical compared to the CrazyFlie's onboard sensor. The only issue with this implementation is that students need to disconnect the test stand from the VM before closing the GUI application, or else the entire VM will crash. The team looked into this and found that the VM cannot handle a sudden disconnection of a serial device, which caused it to crash. The team was able to repeatedly mention the idea of disconnecting the test stand first before closing the GUI application during the briefing of MP-4 with the students and in the MP-4 lab manual. Thus, resulting in fewer crashes than normal.

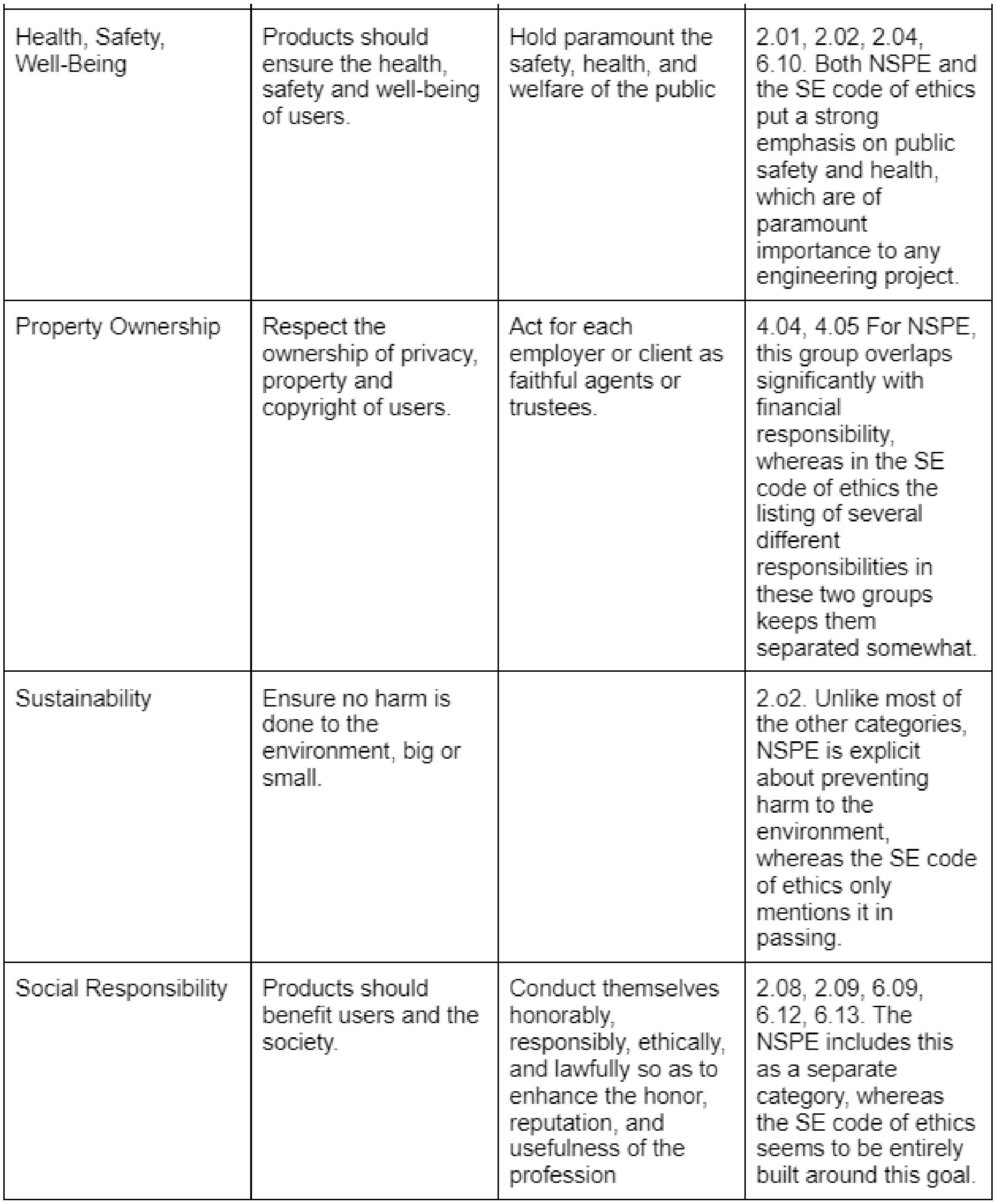
The performance optimization implementation worked extremely well; the CPU utilization of the GUI dropped from 100% to 70%, which reduces the number of crashes due to lag and unresponsive applications. It is revealed to the team by previous CPRE 488 students and their advisor that MP-4 works better than before, as there were significantly less number of GUI and VM crashes.

7 Ethics and Professional Responsibility

7.1 Areas of Professional Responsibility/Codes of Ethics



Area of Responsibility	Definition	NSPE Canon	SE Code of Ethics
Work Competence	Producing a high quality work, while ensuring work ethics such as work integrity and timeliness.	Perform services only in areas of their competence. Avoid deceptive acts.	1.03, 1.07, 1.14, 1.15, 4.01. The SE code of ethics undertakes a similar effort to ensure those working on projects are well qualified and report timelines, costs and workload correctly and non-deceptively
Financial Responsibility	Producing products within the budget and marketing them at a reasonable cost.	Act for each employer or client as faithful agents or trustees.	1.07, 4.03, 4.04, 4.05, 4.06, 4.07, 4.09 The SE code of ethics covers similar bases as NSPE, but goes further to instruct professionals to avoid other outside influences of the project, such as outside work, illegal software, or other adverse interests. The NSPE does not mention these other possible factors.
Communication Honesty	Report work truthfully which is easily understandable to non-engineers such as stakeholders.	Issue public statements only in an objective and truthful manner. Avoid deceptive acts.	3.01, 3.03, 3.04, 3.05. Both NSPE and the SE code of ethics are explicit about objectivity when related to projects and public statements. The SE code of ethics enumerates the "deceptive acts" to be avoided such as bribery, payback, kickback, or other payment.



Health, Safety, Well-Being	Products should ensure the health, safety and well-being of users.	Hold paramount the safety, health, and welfare of the public	2.01, 2.02, 2.04, 6.10. Both NSPE and the SE code of ethics put a strong emphasis on public safety and health, which are of paramount importance to any engineering project.
Property Ownership	Respect the ownership of privacy, property and copyright of users.	Act for each employer or client as faithful agents or trustees.	4.04, 4.05 For NSPE, this group overlaps significantly with financial responsibility, whereas in the SE code of ethics the listing of several different responsibilities in these two groups keeps them separated somewhat.
Sustainability	Ensure no harm is done to the environment, big or small.		2.o2. Unlike most of the other categories, NSPE is explicit about preventing harm to the environment, whereas the SE code of ethics only mentions it in passing.
Social Responsibility	Products should benefit users and the society.	Conduct themselves honorably, responsibly, ethically, and lawfully so as to enhance the honor, reputation, and usefulness of the profession	2.08, 2.09, 6.09, 6.12, 6.13. The NSPE includes this as a separate category, whereas the SE code of ethics seems to be entirely built around this goal.

One area that this team has done well is work competence. This team has recorded the individual workloads and reported the individual contributions to the project honestly and non-deceptively. An area that they have been working on is the health, safety, and well-being of their users, who are students, because the goal of the project is to ensure that no users will be harmed while using it. In order to improve this shortcoming, documentation has been utilized by the team in order for emphasis to be made in regard to certain tasks or risks. In case any other issues come up, this would be the solution that is utilized and recommended by the team for future endeavors.

7.2 Four Principles

Four Principles	**Beneficence**	**Nonmaleficence**	**Respect for Autonomy**	**Justice**
Public health, safety, & welfare	The project helps improve the learning of all who are involved	Design promotes safe practices (ie, Test Stands)	Implementation provides a framework that participants are expected to complete	Design allows for access to all parties
Global, cultural, & social	Brings different communities together to learn	Implementation harms no one indirectly	Design does not affect cultural practices	Benefits are shared equally amongst all parties
Environmental	Mini Quadcopters are small, decreasing potential environmental impact	Rechargeable batteries and non-toxic, minimally processed materials ensure a low environmental impact	Open-source design allows replacement parts to be sourced according to the user's desires	Implementation does not harm the environment
Economic	Project teaches job-applicable skills	The project largely uses pre-existing open-source design parts	CrazyFlie is an open-source drone that can be found outside the school	Custom drone will not infringe upon any private sales; CrazyFlie software only affects our items

A broader area context-principle that this team utilizes positively in this project is how the open-source origin design allows for the replacement of parts non-limitedly. The CrazyFlie mini quadcopter is used by numerous people for testing, ranging from students to teachers, in this project's context specifically. Accidents are bound to occur and can/have caused enough damage that mere tape and glue are no longer enough as solutions. Since there had been some issues that damaged a drone beyond repair prior to the start of the second half of the project, the team has had to look to other locations for replacement pieces, ranging from battery holding implements on top of many of the drones, new wire connectors, and new batteries. It will be up to next year's MicroCART team to find some of these solutions.

One broader context row that this project is largely missing, or rather lacking, is the Global, Cultural, and Social row. Inside this row, many of the pairs involved in this row are largely due to the scope of the project, which is limited, more so to the college. Different communities are coming to the college to learn equally, while our project instead is meant for the audience of a 'student' rather than a 'student of a CERTAIN CONTEXT'. Nonmaleficence, Respect for Autonomy, and Justice are all similarly passive, with not much being put forward in this context. Thus, while this project was not performing in a fashion that hinders this broader context, it is difficult to say that it is supporting it, hence, it is a mostly neutral row.

This section was updated, but no major changes were made, as the neutrality of usage and intention of usage of open-source parts are integral parts of this project. This project was and is meant to help students learn more in regard to the topics covered in

CPRE 488, and the non-limitation of parts allows students to create custom fixes, preventing MicroCART teams from following a fixed path.

7.3 Virtues

Order, moderation, and resolution[2] are three virtues that are important to our team. To the MicroCART team, order meant how the team was to utilize the AGILE and Waterfall work methods. By working through items in a predefined manner, the team was able to keep track of where they were as a group. Moderation is meant to keep us on track, but not go wild and either do too much or do too little. The team each performed a moderate amount of work on our own, but when it came to working together they were rather lacking, and took more time to learn/check up on what each of us were individually working on when it would likely have been more efficient to work as a group for certain task items. Resolution, or more precisely, resolve, meant to firmly decide on what they would do, before getting into a task and then finishing it, even if it meant having to ask questions or learn new but necessary information to complete the task. In the future, the team as individuals will be continuing this practice, but will ask fellow team members when they come across something that they do not understand, as this would have made many work-halting issues be resolved much quicker.

- Ryan: A virtue that I feel I have demonstrated in this project is perseverance. Perseverance is important in projects such as this because there will be times when a goal may seem impossible or too hard to achieve. This project has brought plenty of challenges, and I have had to get through these challenges to reach our goals. Another virtue that I find important that I have not demonstrated yet is collaboration and working with others. I have found it difficult to find time and opportunity to work with my team members as much as this project requires. This is something that will need to be fixed next semester because it is impossible for engineering projects as large as this to be completed by yourself.

- Daniel: An important virtue that I have demonstrated is clear and thorough documentation. Throughout this project, I have been looking through, organizing, and writing documents in order to better guide future students and project members due to the quite bloated project base that has not really been pruned. In comparison, a virtue that I have not demonstrated well is attentiveness to the needs of others, which is largely in part due to not working deeply with all of my fellow team members, something that our sub-groups will have to work together to do as well. Over the course of the second half of the project, I managed to improve upon this attentiveness, though the documentation began to fall behind as tasks and troubleshooting shot up in importance.
- Jonah: A virtue that has been important to me has been the willingness to learn. This project focused on software development, an area in which I was lacking. The ability to recognize what I did not know and strive to research and gain an understanding of a subject on the fly has been a constant cycle in the development of our project. A virtue that I personally strive for, but regarding this project ultimately fell short was time management. I did not spend enough time on this project throughout the semester. This will need to be fixed for next semester. It will most likely be caused by better personal planning.

- Yi: A virtue that I have demonstrated is inspiration and motivation. This is important because teams require motivation to work, or they will end up with poor work efficiency. What I did was create a Git Issues board and included tasks with a deadline that we will follow as a team. A virtue that I think is important but have not demonstrated is time management. Fall 2024 has been a very busy semester for me. I had bad task prioritization and scheduling that ultimately resulted in less work done for this project. In the future semester, I will be taking fewer classes and managing my time more efficiently through a task scheduler.

8 Conclusion

8.1 Summary of Progress

Overall, many of the reasonably accomplishable goals that were revised for the second semester were able to be accomplished during the second half of this project. These reasonable goals that the team accomplished included fully implementing the test stand, connecting the backend, and updating the MP4 documents in order to ensure a smooth deployment, while working on the FlyPi drone was a stretch goal.

For more context, over the course of the first half of the project, August 2024 - December 2024, the MicroCART team had a late start and many timing issues, which contributed to having a less-than-desirable amount of progress being made. When coming up with a new schedule for the second half, January 2025 - May 2025, the team members were able to achieve much better meeting times and work efforts because of it. These improved meeting times allowed for much better cooperation between the members, and after a meeting with their advisor, Dr. Philip Jones, where the new goals and the stretch goal were established, the reasonable goals mentioned were able to be achieved. As such, the MicroCART team was able to achieve its intended tasks before the deadline.

8.2 Value Provided

The updated MP4 items inside this project are the updated MP4 lab document and virtual machine/lab environment. The updated MP4 lab document is much improved compared to the previous iteration, which had poor navigation (i.e., jumping to the 57th page from page 5 with no return jump available), dead links, and outdated or no longer implemented pieces of information. As the team went through the document and lab environment, they fixed the previously mentioned issues. Updating the lab environment allowed the team to find many bugs due to bad code or overlooked issues from teams from previous years, while also providing them an updated way to fix some accidental/unintentional hardware issues. In a broader context, updating is what is needed to stay up to date with the most improvements while also providing solutions to bugs or things not directly tied to performance. Fixing the lab environment allowed the team to fix the largest and most common hardware issue plaguing the project, which was a firmware error that bricked the system, rendering any drone suffering from it unusable.

8.3 Next Steps

The next step that this team is taking is finalizing the handover documents and videos for next year's team. As stated above in the conclusion, this project group did not have a smooth start, which limited the amount of success they could achieve. Some of the next steps that the team plans to give to future teams are a proper handover document, giving direct access to where things that students are looking for can be found. Even after the wrinkles of the first semester were smoothed out, there were still many issues with finding the information within different parts of the repository and the pre-written student documents that had team members having to wait for their advisor to obtain the information or simply offer an alternative altogether. In order for future teams to have a solid understanding of what needs to be fixed next, such as outdated items in the GUI or missing buttons, any tasks unable to be completed in time by this year's team will be recorded with any thoughts going towards how to fix them written in accompaniment to them to give the next team an idea of how to work on this next. Hopefully, this will enable the future MicroCART teams to be able to complete the FlyPi design that has been in progress for the past few years, but even as of yet, is unable to fly.

9 References

Websites:

[1] "Bitcraze Shop." *Bitcraze Store*, BitCraze, 2024, store.bitcraze.io/. Accessed 17 Nov. 2024.

[2] Franklin, Benjamin. "Benjamin Franklin - 13 Virtues | PDF | Virtue | Science." *Scribd*, 1790, www.scribd.com/document/235479275/Benjamin-Franklin-13-Virtues. Excerpt from Benjamin Franklin's Autobiography.

[3] "How to Register Your Drone | Federal Aviation Administration." *Faa.gov*, 18 Mar. 2024, www.faa.gov/uas/getting_started/register_drone. Accessed 7 Dec. 2024.

[4] Kalachev, Oleg. "Open Source ESP32-Based Quadcopter Made from Scratch." *Arduino Project Hub*, 6 Jan. 2024, projecthub.arduino.cc/okalachev/flix-58fe43. Accessed 26 Sept. 2024.

[5] Liang, Oscar. "Review: Holybro Kopis Freestyle 4-Inch FPV Drone." *Oscar Liang*, 27 July 2021, oscarliang.com/holybro-kopis-freestyle-4-inch/. Accessed 26 Sept. 2024.

[6] Nast, Condé. "The DJI Mini 4 pro Is a Small Drone with Huge Appeal." *WIRED*, 23 Mar. 2024, www.wired.com/review/dji-mini-4-drone/. Accessed 26 Sept. 2024.

10 Appendices

Appendix 1 – Operation Manual

The steps for operating the MP4 infrastructure are laid out best in the version of the MP4 lab document provided to students. The lab document is located here: https://class.ece.iastate.edu/cpre488/labs/MP-4.pdf.

A simplified version of the steps will be provided below:

MP-4 Part 1

1. Log in to the VM using the Bitcraze account name and password of CrazyFlie. Get a CrazyFlie drone and attach a battery to it. Take note of the number on the bottom of the CrazyFlie drone. Using this number, refer to the CrazyFlie Radio Number document provided in the lab document above.
2. Get the USB radio dongle and plug it into the PC you are connected to the VM on. You should then be able to connect this device to the VM by going to the Devices tab at the top of the VM screen and selecting USB. From here, select the Bitcraze AB Crazyradio.
3. Open up a terminal in the VM and navigate to the /MicroCART/CrazyFlie_software/CrazyFlie-firmware-lab-part-1/ folder. This will be the folder where the MP4 lab part 1 is completed, which involves tuning the PID controller.
4. Cd into the /tools/make/config and make or edit the config.mk file. Add in the line CLOAD_CMDS=-w radio://0/<radio_channel>/2M/E7E7E7E7E7, replacing

radio channel with the value found above. Go back to the folder in step 3 and run the make cload command in a terminal. This will flash the CrazyFlie drone with the firmware in part 1. After this is done, run crazycart <radio_channel> to open up the GUI that has been created to assist with the lab.

5. At this point, you have a GUI open that is connected to the CrazyFlie drone. From here, you can navigate through the GUI to set PID parameters, fly the drone through the Controls page, set the logging parameters in the logging blocks page, and assign a controller in the gamepad page.
6. The first step to tuning is to set parameters and alter these by using the logging functionality to fine-tune the correct PID values. The parameters that need to be tuned are in the s_pid_attitude and s_pid_rate groups. You can either set the parameters individually or use the attached JSON file to set the parameters all at once, which is recommended.
7. Now, you will want to attach the CrazyFlie drone to the test stands to be able to tune the PID values without the drone actually flying. Go to the controls page and start setting setpoints and graphing the appropriate logging variables as needed. Set the thrust and press the "Apply" button to start the drone. The "Stop" button will stop the drone.
8. After tuning the PID parameters correctly, you should be able to fly the drone off the test stand and have a stable flight.

MP4 Part 2:

Step 1: General PID

Note, for this section, you should **use the PID constants you found in part 1** for known good values. You can set the default values in the student_pid.h file. However, the constants you discovered earlier may have some assumptions built in, so it may be necessary to re-tune the controller if a significantly different algorithm is used.

1. The first thing that you will write is a general PID function and struct. The PID struct that we provide you will be empty, and you will decide on what should be included in it, which is defined in student_pid.h. You are encouraged to make as many helper functions as you would like in your student_pid.c file to help with roll, pitch, and yaw calculations.
2. The first thing I would recommend writing is the **PidObject** struct in student_pid.h. This struct is used to hold the data that is used for all other PID calculations, so it is required to write many of the other functions.
3. Next, write the basic getters and setters for the PidObject in student_pid.c.
4. Now we can actually write the PID algorithm in the studentPidUpdate function.
5. At this point, you should have filled out everything in student_pid.c and student_pid.h, make sure **all of the "488 TODO" comments have been fulfilled in these files.**

Step 2: Student Attitude Controller

The attitude controller and attitude rate controller have their main functions in student_attitude_controller.c, this is where you should begin working.

1. Complete the initialization function for the attitude controller.
2. Implement the student attitude rate controller PID.
3. Test the rate controller
4. Next, write the student attitude controller PID.
5. Test the attitude controller
6. Then finish the reset PID value helper functions
7. Finally, fill in the logging parameter addresses. Then, make sure **all of the "488 TODO" comments have been fulfilled in these files.**

Step 3: Student Controller, Bringing it all together

Now we need to bring everything together in the controller_student.c file.

1. Start by reading in the setpoints for roll, pitch, and yaw angles as desired values. Forthe mixed attitude mode, the yaw angle should change from the current angle based on the rate given. Also, set the desired thrust.
2. Use the attitude PID controller to set the desired attitude rate to the value calculated. If the controller is in velocity mode, overwrite the attitude rates with the setpoints provided.
3. Next, input the desired attitude rates into the PID controller and use the outputs to set the command variables for roll, pitch, and yaw. Set the output thrust.

4. Copy the values into separate variables for logging purposes.
5. Complete the logging parameters by filling in the addresses.

At this point, you should have filled out everything **in all files**, and make sure **all of the "488 TODO" comments have been fulfilled**.

Step 4: Final Check

- Before you go flying your CrazyFlie for real, it's a good idea to verify everything works as intended on the test stand. Attach the drone to the test stand and briefly check that all axes respond as you expect. For this step, you can use manual setpoints or a gamepad connected to the ground station. See here for details on using a gamepad with the ground station.
- If all looks good, take her for a spin and see how she handles! **Be careful of others** in the lab and try not to crash it too hard!

Appendix 2 – Code

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

Appendix 3 – Team Contract

Required Skill Sets for the Project

- Coding Skills (At least one of: Python | C | C++)
- Networking knowledge
- Embedded systems
- Embedded programming
- Able to utilize Linux / Command Line Interface / Make

Skill Sets covered by the Team

Green = Have || Yellow = Some exposure || Red = No experience

	Coding languages	Networking knowledge	Embedded Systems	Embedded Programming	Linux / CLI / Make
Daniel	🟩	🟨	🟩	🟨	🟩
Jonah	🟨	🟥	🟩	🟩	🟨

Ryan	🟩	🟨	🟩	🟩	🟨
Yi	🟩	🟨	🟩	🟩	🟩

Project Management Style Adopted by the Team

A hybrid method of the Waterfall method and the Agile method for project management.

Individual Project Management Roles

- Daniel Zaucha: Client interaction, Communications Lead
- Jonah Upah: Hardware Lead, Team Secretary
- Ryan Lowe: Technical Advisor
- Yi Hang Ang: Software Lead

Team Contract

Team Members:

1) Daniel Zaucha 2) Ryan Lowe

3) Jonah Upah 4) Yi Hang Ang

Team Procedures

1. Day, time, and location: Face-to-Face

- Wednesday: 11:00 AM - 12:00 PM (Advisor Meetings)
- Monday: 11:00 AM - 2:00 PM
- Friday: 1:00 PM - 5:00 PM
- Sunday: 11:00 AM - 2:00 PM

2. Preferred method of communication updates, reminders, issues, and scheduling (e.g., e-mail, phone, app, face-to-face):
    - Meetings are done in person, unless someone cannot come in person.
    - Discord server set up with all group members to communicate when not in person.
    - Discord, Email, Text last resort
3. Decision-making policy (e.g., consensus, majority vote):
    - Majority votes are required for large-consequential decision-making. If there is no majority or votes are split, we will discuss as a team and try to reach a consensus from group discussion over the issue (OR get Dr. Jones as a tie-breaker vote).
4. Procedures for record keeping (i.e., who will keep meeting minutes, how will minutes be shared/archived):
    - Jonah or Daniel will set up meeting notes and record action items during the meeting. Notes will be kept in a shared google drive folder.

Participation Expectations

1. Expected individual attendance, punctuality, and participation at all team meetings:

    - Everyone will be expected to attend scheduled meetings unless extenuating circumstances come up. If this happens, the team member will inform the group of this.

2. Expected level of responsibility for fulfilling team assignments, timelines, and deadlines:

    - Everyone should contribute to completing team assignments, ideally before deadlines, but preferably a few days before. If someone is not going to get something done because of circumstances, be sure to contact the group for assistance.

3. Expected level of communication with other team members:

    - Discord is how we communicate with each other about meeting and tasks. During the working day responses are expected, subject to change.

4. Expected level of commitment to team decisions and tasks:

    - Everyone puts in an equal amount of time, and thought behind decision making.

Leadership

1. Leadership roles for each team member:

- Jonah Upah: Hardware Lead, Team Secretary
- Yi Hang Ang: Software Lead
- Ryan Lowe: Technical Advisor
- Daniel Zaucha: Client interaction, Communications Lead
- All members work to assist and perform the duties so that the group succeeds as a while. I.e. Software lead can still work on the backend.

2. Strategies for supporting and guiding the work of all team members:
    - The report document will track tasks and time.
    - Meeting will be a place to discuss concerns about the timeline.
    - Report document will also have new issues.
3. Strategies for recognizing the contributions of all team members:
    - Contributions of team members will be noted on the weekly reports and discussed in meetings.