MicroCART

Design Document

sdmay24-32
Dr. Phillip Jones
Trevor Friedl, Justin Kenny, Steven Frana, Clayton Kramper, Will Maahs, Travis Massner
sdmay24-32@iastate.edu
https://sdmay24-32.sd.ece.iastate.edu

Revised: 11/28/2023

Executive Summary

Development Standards & Practices Used

SCRUM

TCP/IP Networking Protocol

Summary of Requirements

- Improve infrastructure walkthrough documents to support future teams' beginning on the project.
- Implement bug fixes and improve the design of the GUI used to control the drones, to make it more usable for future MicroCART teams and students taking CPRE 488.
- Improve the CPRE 488 MP4 lab and administer it to a lab section.
- Continue work on last year's large quadcopter by adding a raspberry pi with a four-core processor to control the drone and perform other high-level functions. The large quadcopter should be capable of carrying payloads between locations.
- Add a camera to the larger quadcopter to allow it to perform an interesting demo.

Applicable Courses from Iowa State University Curriculum

CPRE 288

CPRE 381

CPRE 488

New Skills/Knowledge acquired that was not taught in courses

PCB Design

Control Systems Theory

Linux Device Driver Development

Cross-Compiling

Table of Contents

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

Glossary of Terms:

Crazyflie - The small drone used for the CPRE 488 MP4 Lab. It is manufactured by Bitcraze, and has open source firmware which can be easily written.

FlyPi - Larger custom quadcopter built by previous MicroCART teams. Currently in an incomplete state.

Pycrocart GUI - Python-based GUI used for interacting with Quadcopters, which currently is stand-alone and connects to the drones on its own accord.

Groundstation- Name used for the group of software components that lie in between the quadcopters and the GUI: Backend, Crazyflie Adapter, and Crazyflie Groundstation.

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

cflib - Open-source Python library created by Bitcraze used for communicating with the Crazyflies.

cflib Groundstation - Proposed new software module that will communicate with the Crazyflies using cflib and will replace the Crazyflie Adapter and Crazyflie Groundstation.

Crazyradio - USB radio stick that sends packets to and from the Crazyflies.

FreeRTOS - Free real-time operating system, which will be used on one of the cores of the raspberry pi.

Bootcamp - Document with links to necessary resources designed to introduce people new to the infrastructure in six weeks or less.

1 Team

1.1 Team Members

Justin Kenny

Steven Frana

Trevor Friedl

Clay Kramper

Will Maahs

Travis Massner

1.2 Required Skill Sets for Your Project

- C
- C++
- Python
- Linux
- Networking
- Sensor System Integration
- GUI Design
- AutoCAD
- Sensor Data Visualization
- Controls System Theory
- PCB Design
- Using FPGAs
- Linux Device Driver Development
- Cross-Compiling

1.3 Skill Sets covered by the Team

C - Justin, Steve, Trevor, Clay, Will, Travis

Python - Justin, Steve, Trevor, Clay, Will, Travis

AutoCAD - Justin

Vitis/Vivado - Clay, Steve

VHDL - Justin, Steve, Trevor, Clay, Will

Linux - Steve, Justin, Clay, Trevor, Will, Travis

GUI Design - Steve, Justin, Travis

Networking - Justin, Will, Travis

Control Systems Theory - Steve, Will

PCB Schematic Design - Trevor

1.4 Project Management Style Adopted by the team

SCRUM

1.5 Initial Project Management Roles

Project Manager - Trevor Friedl

Meeting Scribe - Justin Kenny

Technical Lead(s) - Clayton Kramper and Steven Frana

Team Organization Lead - Will

UI/UX Designer - Travis Massner

2 Introduction

2.1 Problem Statement

The current MP-4 lab delivered to CPRE 488 students to teach them about control systems theory using quadcopters, can be improved to provide a better learning experience for students. On top of this, the FlyPi needs to be finished and more functionalities added for research purposes and to demo to prospective students.

2.2 Requirements & Constraints

MP-4 Lab:

Functional:

- The GUI does not crash or freeze and communicates with the backend.
- Communication between the processes utilizes a new packet structure that relies solely on TCP packets rather than file read/writes.
- Combine the Crazyflie Groundstation and Crazyflie Adapter using cflib.

Non-Functional:

- Improve video and documentation for the lab and upload them to MicroCART YouTube channel.
    - Better introduction to enable next year's team to start work faster.

FlyPi:

Functional:

- Add Raspberry Pi to the FlyPi, and have it running FreeRTOS on one core and Ubuntu on the other three cores.
    - Securely Mounted
    - Sufficient Power Supply **(constraint)**
    - Communication with control board
- Attach a camera to the drone. **(constraint)**
    - Securely Mounted
    - Camera output can be streamed to another PC.

General/Other:

Non-functional:

- All development by year-end is successfully merged into the main branch of the repository; personal branches of code are deleted.
- New developments are documented and any old documentation is updated as needed.
- The introductory materials (Bootcamp) are clear enough so that future teams/researchers can be introduced to the system in 6 weeks or less.
- New VM file is hosted as an artifact on the Git repository and is available for access for students and researchers.

2.3 Engineering Standards

TCP/IP Networking Protocol - Used to communicate with drones.

2.4 Intended Users and Uses

Dr Jones - The FlyPi and any improvements made to the infrastructure will allow him to get future microCART teams and researchers acclimated to the infrastructure quicker.

CPRE 488 students - The improved MP-4 will make it easier for students to learn control systems theory and remove any frustrations with the current infrastructure.

Prospective ISU students/Alumni - The FlyPi will be an interesting demo that showcases the degree program at ISU.

Graduate researchers using the Controls Lab - The FlyPi will allow them to deploy and test control algorithms quicker on a platform with more computing power.

3 Project Plan

3.1 Task Decomposition

MP4 Improvements:

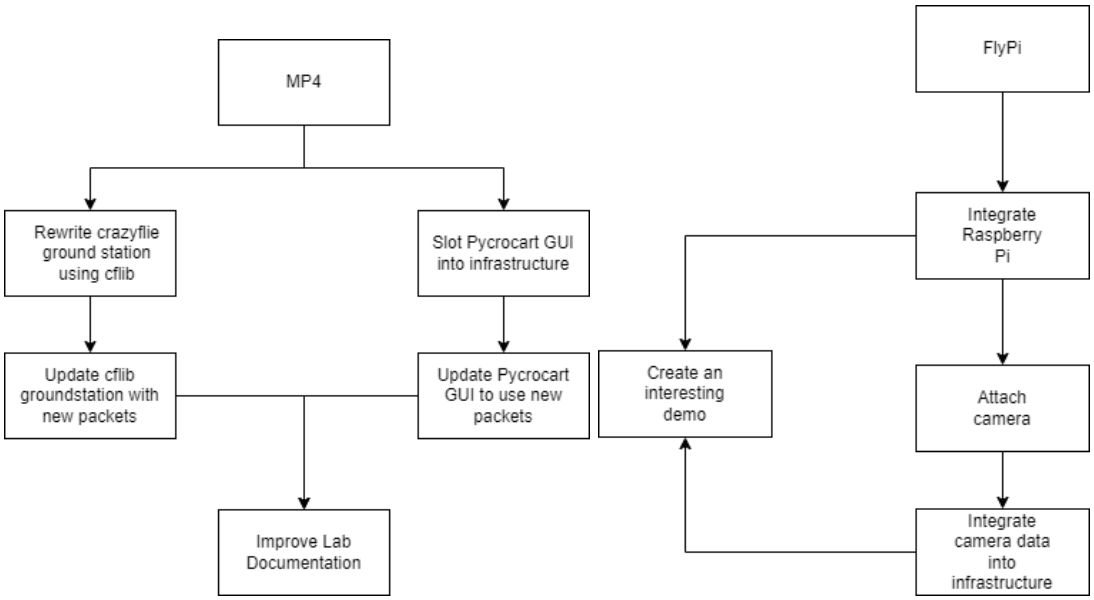
1. Combine the Crazyflie Groundstation and Crazyflie Adapter using cflib
    a. Get familiar with the Groundstation, Crazyflie Adapter, and cflib
    b. Write functions to use cflib to write and read desired data from the Crazyflie
    c. Write functions to use sockets to decode incoming packets, call the appropriate function, and send back encoded packets.
    d. Test that it is integrated with the current infrastructure by using CLI to send packets.
2. The GUI does not crash or freeze and communicates with the backend.
    a. Remove cflib functionality from the existing Pycrocart GUI
    b. Update the Pycrocart GUI to interface with CLI commands
3. Communication between the processes utilizes a new packet structure that relies solely on TCP packets rather than file read/writes.
    a. Update cflib Groundstation to accept new packets.
    b. Update CLI with new commands that correspond with the new packets.
    c. Update Pycrocart GUI to connect directly to Backend's TCP socket and send the new packets.
4. Improve written and video documentation used for the MP4 Lab.
    a. Update written documentation in order to fit standards of the lab plus any additional changes and refinements made from new discoveries.
    b. Improve the "introduction phase" of the boot camp in order to provide more concise and understandable instruction for next year's team by cleaning up difficult sections.

FlyPi:

1. Integrate Raspberry Pi with FlyPi.
    a. Securely mount the Raspberry Pi to the drone.
    b. Run proper power and data wires throughout the drone.
    c. Partition the cores of the Raspberry Pi into Free RTOS (1) and Ubuntu (3).
2. Attach a camera to the FlyPi and allow it to have first-person view (FPV) capabilities.
    a. Find a suitable camera and attach it to the drone..
        i. Power and Communications wiring.
    b. Update the drone firmware to stream camera output to a remote computer.
3. Create a demo for the FlyPi
    a. Find a computer vision task that can be completed with the camera and raspberry pi.
    b. Export OpenCV to use in a linux driver.
    c. Create an interesting demo for prospective ISU students.

Bootcamp Improvements:

1. Add a new VM Link to the bootcamp document.
    a. VM .ova file is hosted as an artifact on the gitlab repository
    b. The bootcamp document is updated with a link to the artifact.
    c. The VM is downloaded on the Coover 2041 Lab computers.
2. Improve the Bootcamp document. Make it easier for next year's team to get introduced to the infrastructure.
3. Record instructional videos related to the lab and the FlyPi and upload them to the MicroCART YouTube channel.



3.2 Project Management/Tracking Procedures

Agile - Gitlab issues

Since our project is a multi-year project, each component is in a different state in respect to the software development life cycle. Thus agile will be most useful for picking up each of these components and taking it as far as we can in the time of our senior design project. We can work incrementally on developing, testing, and documenting, and hopefully leave a good starting point for next year's team.

3.3 Project proposed milestones, metrics and evaluation criteria

MP-4:

1. 100% of previous features can be done with the combined cflib Groundstation component.
2. Pycrocart GUI relies solely on sending and receiving the new packet types

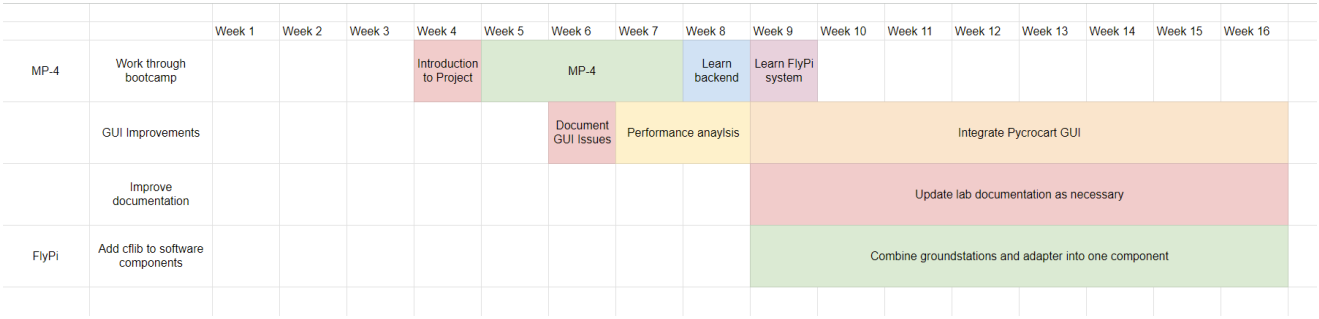
3. Pycrocart GUI handles scheduling of packets without crashing or freezing

FlyPi:

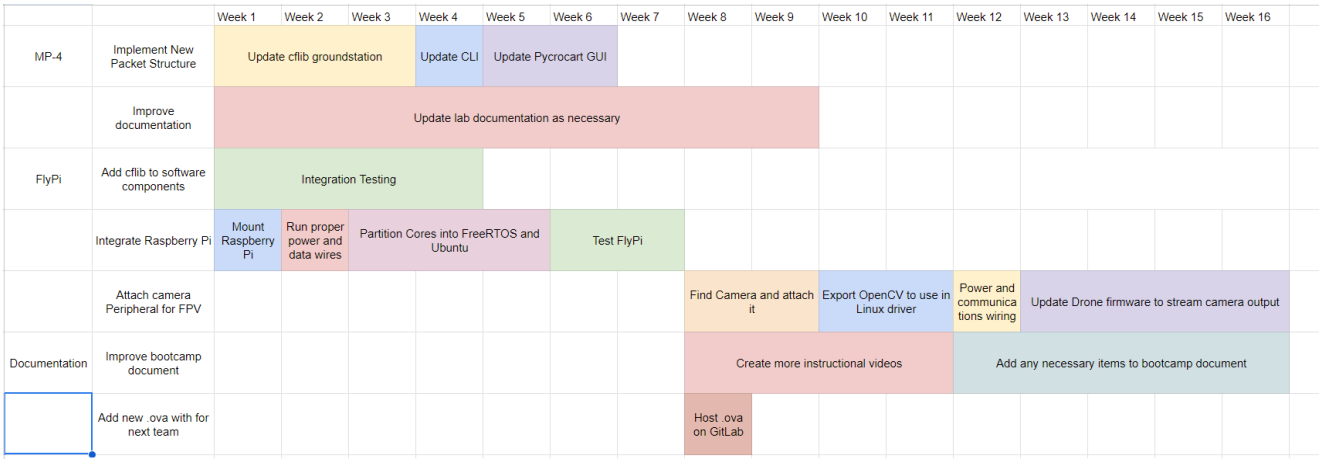
1. Raspberry Pi allows the drone to be controlled in the same way as the current mounted crazyflie.
2. A camera is attached securely to the drone and powered correctly.
3. The camera output can be viewed from another computer.

3.4 Project Timeline/Schedule

Semester 1:



Semester 2:



3.5 Risks and Risk Management/Mitigation

Risk	Probability	Mitigation
Injury due to poor flying/tuning	10%	Tether all test flights or use test stands when trying new things. Practice with flying quadcopters will also reduce this risk.
Damage to equipment due to poor flying/tuning	50%	Tether all test flights or use test stands when trying new things. Practice with flying quadcopters will also reduce this risk.
Battery Fire	1%	Batteries will not be left changing unattended. LiPo batteries that expand or produce excessive heat will be retired.
Camera is not received	5%	Search for several camera options and order a different option if one is not able to be sent.

3.6 Personnel Effort Requirements

Task	Explanation	Hours
Communication between the processes utilizes a new packet structure that relies solely on TCP packets rather than file read/writes.	The CLI and GUI will need to be able to assemble and send the new packets, and the cflib Groundstation needs to be able to receive and parse these packets, and then call the appropriate cflib function.	30
The GUI does not crash or freeze and communicates with the backend.	The current state of the GUI makes it crash and freeze often during normal use. Fixing this would make it much easier for students to complete the lab.	60
Improve written and video documentation used for the MP4 Lab	Better documentation would make it easier for students and future teams to get introduced to the lab and infrastructure.	15
Combine the Crazyflie Groundstation and Crazyflie	Combining the Crazyflie Groundstation and Crazyflie Adapter should make the Groundstation	30

Adapter using cflib	more stable to use.	
Integrate Raspberry Pi with FlyPi	Adding a Raspberry Pi to the FlyPi would be a continuation of last year's team's FlyPi drone where the Raspberry Pi would be used as the flight controller.	70
Attach a camera to the FlyPi and allow it to have first-person view (FPV) capabilities	Add a camera peripheral to the FlyPi. The end goal is for the first-person view of the drone to be displayed to a screen such as a projector or monitor.	100
Add a new VM Link to the bootcamp document.	There is a VirtualBox image that is used to complete the MP4 Lab. We will need to add a new link to the bootcamp document to be able to download that .ova file.	5
Improve bootcamp document	Better documentation will allow next year's team to get acclimated to the project more quickly.	40

3.7 Other Resource Requirements

- VirtualBox
    - Visual Studio Code
- Crazyflie Drones
    - Flow Deck
    - Lighthouse Control Deck
    - Batteries/Chargers
- Test Stands
    - 3D Printed drone holders
    - Test stand data collectors (Arduinos)
- FlyPi
    - Raspberry Pi
    - Variety of drone parts
    - Crazyflie Big Quad Deck
    - Batteries/Chargers

4 Design

4.1 Design Content

The design aspect of our project is to redesign certain parts of the infrastructure to improve the experience of the MP-4 for students, and to design a system to add to the FlyPi drone that can support a camera with FPV capabilities.

4.2 Design Complexity

As detailed in the diagrams below, the current MicroCART infrastructure consists of an array of different components and subsystems, making designing and implementing changes to the architecture difficult and complex. Some of these systems and their associated challenges are listed below.

1. Crazyflie- The small RC quadcopter used for the MP4 lab uses its own packet structure which any components developed by the team must adhere to. Since it is a movable component, special safety considerations must be taken into account to prevent damage to the quadcopters or bystanders.
2. FlyPi- This will involve many different disciplines to complete, such as PCB design, Embedded programming, and bare metal CPU operations. The challenging requirements related to this aspect are detailed further below.
3. Groundstation- The Groundstation is the C-based software component that connects all the different aspects of the infrastructure together, and actually consists of three different components: Backend, Crazyflie Adapter, and Crazyflie Groundstation. These components all interact over TCP connections, and they are connected to the quadcopters via radio. Due to the complexity of the interaction between these components, modifying or adding components will represent a significant challenge.
4. GUI- Two GUIs currently exist at the moment: one written in C++ and one written in Python. One is connected to the Groundstation infrastructure, but needs improvement, and the other is not connected to the wider infrastructure. Using and improving either one will involve complex architectural and design decisions.

Some challenging requirements with justification:

- Add Raspberry Pi to the FlyPi, and have it running FreeRTOS on one core and Ubuntu on the other three cores
    - Getting one core to run FreeRTOS on the CPU bare metal and regular Ubuntu running on the other three cores is a unique challenge that goes beyond any undergraduate courses at ISU.
- Attach a camera to the FlyPi
    - FPV (First-Person-View) drones are becoming a more popular hobby product, but

completing this task on a custom system is an equivalent task to creating one of these FPV drones.

- Communication between the processes utilizes a new packet structure that relies solely on TCP packets rather than file read/writes.
    - Implementing a new packet structure will involve coordinated modification of all components in the system. The new packet structure will also have to be built in such a way that documentation is complete and adding new packets is easy for future teams.

4.3 Modern Engineering Tools

- VS Code: The development environment that will be to make software changes.
- Virtual Box: Virtual Machine software used to run a custom linux environment for development and testing.
- FreeRTOs: Real-Time Operating system for our drone firmware
- KiCAD 6.0/7.0: PCB Design Software

4.4 Design Context

Area	**Description**	**Examples**
Public health, safety, and welfare	Not particularly relevant to our project directly, since it mainly deals with research. However, in a broad sense, the research facilitated by this could be used to positively impact the public.	Using the PID control algorithms tested on the system to use drones to perform rescue operations.
Global, cultural, and social	In an indirect way, the improvement of the MP4 lab will make it easier for students to focus on learning instead of struggling with the system.	A student could spend 4 hours learning about PID control rather than trying to reboot the GUI after it freezes every time.

Environmental	Little to none, since this is a small scale project for research purposes that doesn't particularly interact with the environment.	No particularly good examples.
Economic	The research done via the infrastructure could potentially have many industrial applications. An interesting demo could potentially attract new students to ISU.	A greater number of new students at ISU would increase the amount of funding going towards its research.

4.5 Prior Work/Solutions

Most of the previous work done for this project comes from the work done by previous teams, and the work done by Bitcraze, which provides the quadcopters and open-source software on which the infrastructure is built. The advantages of the previous team's work is that they have left a slew of documentation and resources, and they left a system that is highly configurable and customizable, making it suitable to improve and add to. The disadvantage of their work is that it was completed by many independent groups that were forced to start from scratch on the infrastructure. Thus, many motivations behind architectural decisions are lost and some parts of the system are incomplete or created in an inexplicable way. Additionally, the accumulation of multiple years of MicroCart Teams has led to a somewhat "messy" codebase that has been passed down to us.

Pros of our proposed changes:

- Using the Python-based library provided by Bitcraze will simplify the code base and use a library that has more documentation and support.
- The Pycrocart GUI is faster and more reliable, making it better for students completing MP4.
- A camera on the FlyPi will be an interesting demo for prospective ISU students.

Cons of our proposed changes:

- Adding more to the infrastructure and rewriting it further could potentially make things more complicated for future teams.
- If we are unable to fit the Pycrocart GUI into the current infrastructure, then it will not be able to be integrated with some of the other backend functions or the FlyPi.
- Since the FlyPi is a somewhat open-ended project, it may not be possible to use a camera with it at all.

4.6 Design Decisions

1. We have decided to use the PycroCart GUI built by last year's team instead of the current GUI in the infrastructure. Although it will take more work to get it interfaced with other components, the performance benefit is considerable.
2. We have decided to rewrite the Adapter and Crazyflie Groundstation in python with cflib, which although will require more man hours, will lead to better performance and ease-of-use for future teams.
3. We have decided to select a camera as our chosen peripheral on the drone, since it has many opportunities to provide an interesting demo to prospective ISU students.

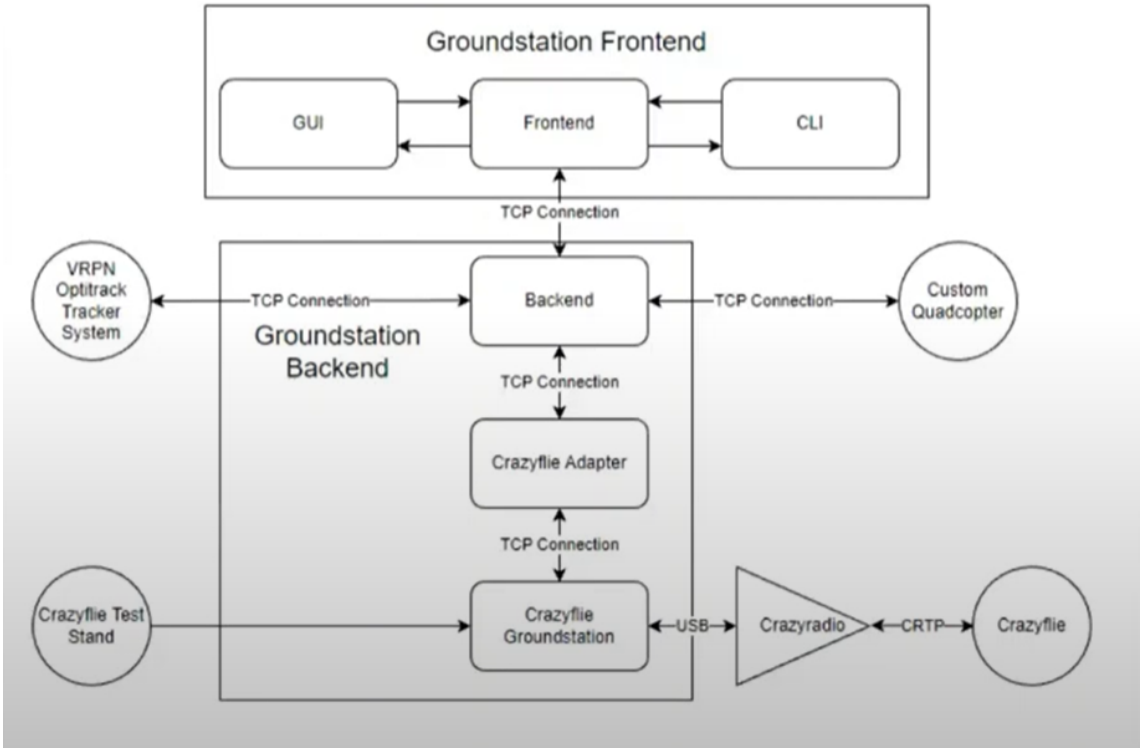
4.7 Proposed Design

4.7.1 Design 0 (Initial Design)

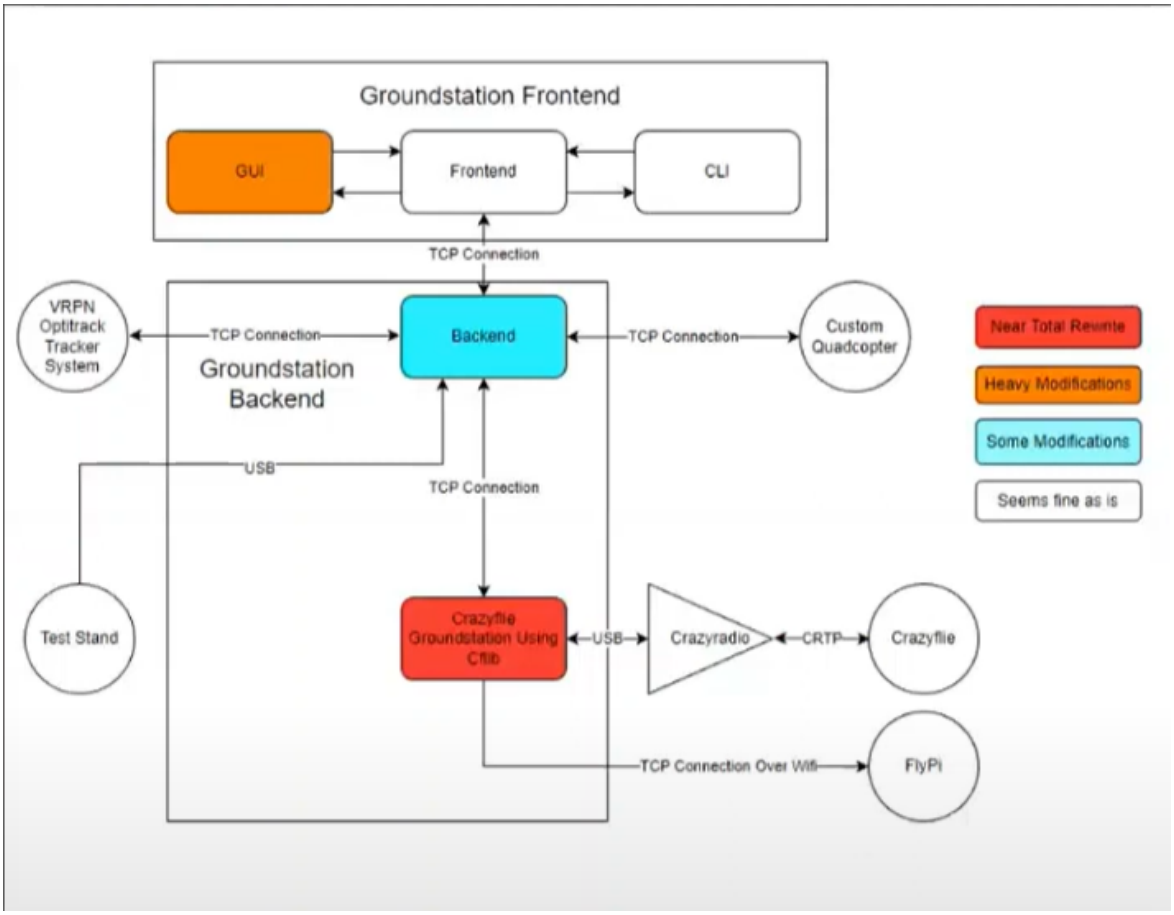
Design Visual and Description

MP-4/Infrastructure

Previous Design:



Proposed Design:



Essentially, the changes made to the proposed design over the previous design are adding the Pycrocart GUI as a frontend and combining the Crazyflie Adapter and Crazyflie Groundstation.

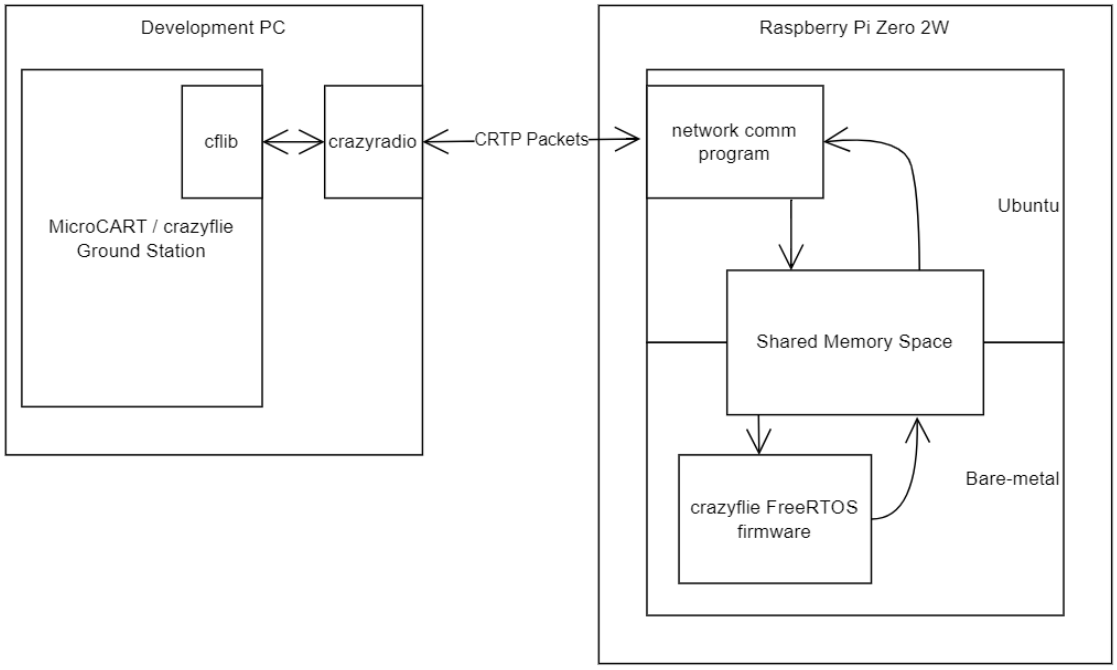
The Pycrocart GUI fulfills the requirement “The GUI does not crash or freeze and communicates with the backend,” since from our evaluation it is much more intuitive and performant than the current GUI. To be able to insert it into the current infrastructure, we would need to configure it to use the CLI command like the other GUI does. Since in its standalone form it is able to connect to the Crazyflies, it could also be touched up to be used for MP-4 without the rest of the infrastructure.

The PycroCart GUI will then run the CLI commands, which sends a packet requesting a specific action (changing setpoint, setting a param, etc.) to the Backend. The backend handles this request by sending another packet over a TCP connection to the desired endpoint (either the cflib roGundstation of the FlyPi quadcopter).

The Crazyflie Groundstation (using cflib) fulfills the requirement “Combine the Crazyflie Groundstation and Crazyflie Adapter using cflib.” Through our observations, these two components were the largest cause of freezing in the GUI, so hopefully rewriting them and combining them will remove one bump in the overall process. Using cflib specifically will be beneficial since it has

provided documentation and support from Bitcraze. When the cflib Groundstation receives a packet from the Groundstation, decodes the packet and calls the appropriate cflib function, which will send the needed data to the crazyflie quadcopter.

FlyPi



This design fulfills the requirements for the FlyPi, since it includes partitioning the Raspberry Pi CPU. Since the FlyPi is a highly complex and technically difficult system, it is highly likely that this design will change and become more detailed as we dive deeper into this part of the infrastructure. Just like with the Crazyflies, the Groundstation will use the crazyradio to transmit command packets to the Raspberry Pi. This will place the necessary data into the shared memory, which will be used by the FreeRTOS core to alter the necessary registers to control the movement of the drone. The camera will return data to the Ubuntu cores (data type is yet to be determined), which will transmit this data back to the Groundstation via aTCP connection.

Functionality

Our proposed designs for the MP-4 Crazyflie Groundstation and the FlyPi are designed in order to communicate directly with the high-level systems that will be in charge of controlling the functionality of each respective system. With respect to the Crazyflie Groundstation, the above diagrams display how a connection is established between the Groundstation frontend and backend using our TCP protocol for data communication. From within the backend, another TCP connection could be open to connect with the OptiTrack tracker system, which should run with low enough latency to prevent any concurrency issues from occurring. Additionally, we will be

connecting our Crazyflie Groundstation now to our proposed FlyPi in development simultaneously while trying to get rid of our Crazyflie Adapter (as seen in the proposed flow chart above).

In respect to the FlyPi design, our biggest task concerns our need to partition the cores into running different instances of operating systems, for instance, one of the cores will need to run FreeRTOS while the other three cores will be required to run Ubuntu. These separate operating systems will have different responsibilities in requirements for speaking with different components of the FlyPi, as the drone controls will be handled by FreeRTOS using the shared memory and the camera and network communication program will communicate with the Groundstation through the three Ubuntu cores.

We can consider our current design implementation to be satisfactory of our current design's previously agreed functional and non-functional requirements, even as we are needing to make changes to our design in order to take in and satisfy new requirements we have with hardware and software. A big focus of ours is to shorten any critical paths from our current design in order to support minimized overhead while delivering accurate and persistent connection to different systems throughout the frontend, backend, and FlyPi.

4.7.2 Design 1 (Design Iteration)

As we explored the inner workings of how the Groundstation handles logging data points for the GUI to plot, we discovered that this was handled by the Groundstation creating files, writing data points into them, and then sending the path to these files back to the frontend. We hypothesized that this process of writing and reading to files was partially responsible for the slow speeds and crashing of the GUI. To rectify this issue, we designed a new packet structure as shown by the link below that will exclusively use the TCP sockets to communicate data between processes. To facilitate this, we needed to add more packet types and modify the existing ones to handle our proposed changes, and we will need to update the CLI to handle these packets as well.

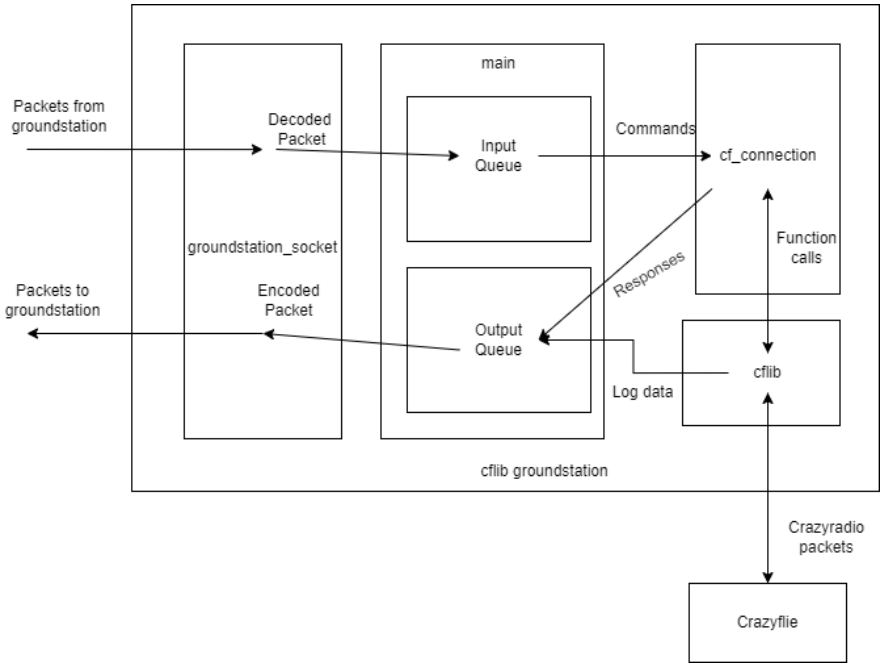
https://git.ece.iastate.edu/danc/MicroCART/-/wikis/MicroCART-Packet-Structure

Another major change in our second design is to replace the current GUI with the Pyrcocart GUI, and link it directly to the socket communication with the Backend. It is more responsive than the current GUI, and the code is much easier to read and modify, making it easier for future teams to pick up the project. We also determined it would be faster and more clear if the Pycrocart GUI communicated on the backend socket directly, so it can parse the incoming packets itself without the additional translation step done by the CLI.

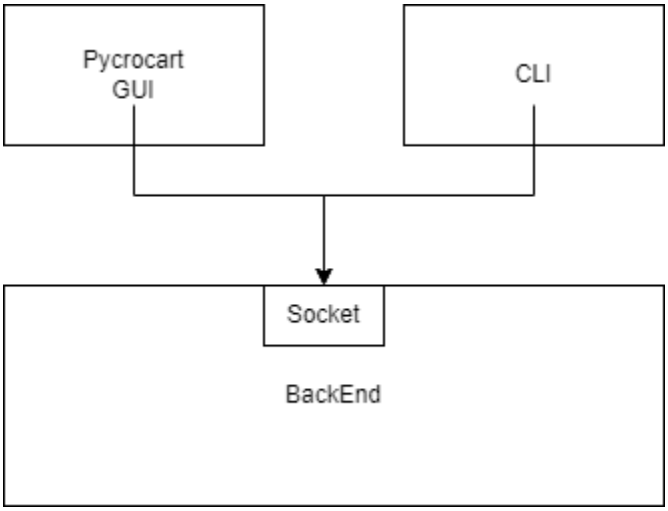
We have further expanded on our FlyPi design to include the camera, which will send its output to a program running on the FlyPi. That program will correctly format this data to be sent to the Groundstation, where it will be handled to display the output. This program will not interfere with the shared memory used by the Raspberry Pi to handle communication between the drone controls and the program receiving commands.

Design Visual and Description

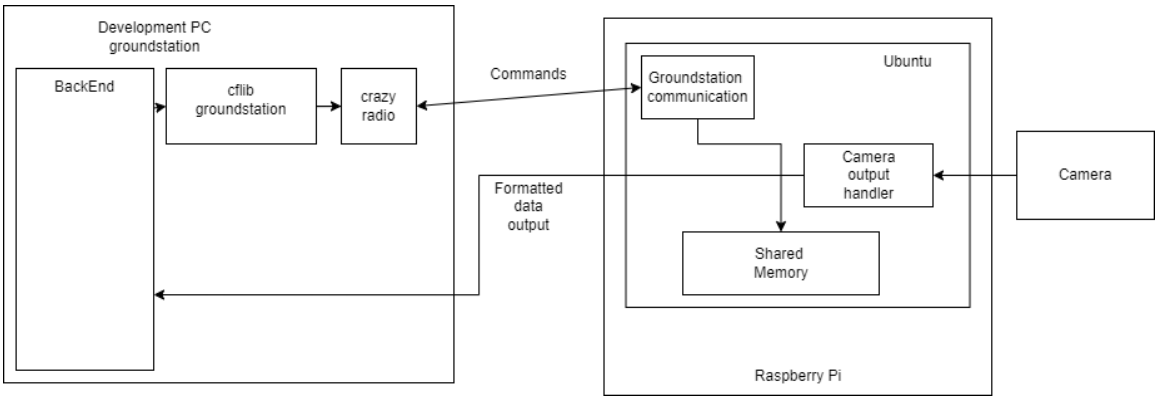
The full layout of the new cflib Groundstation is shown below. Each box on the diagram is a separate python class which contains some part of the functionality (communication with Groundstation, communication with Crazyflie, etc). The arrows represent the data flowing between the classes, which will be handled as internal data structures and function arguments. The essential strategy is to have the main class have two queues which will handle incoming and outgoing packets between the Groundstation and the Crazyflie. This design also removes the need for any file handling as all logs and parameters will be handled by packets rather than using on-system files.



Also, since the Pycrocart GUI is built in python, it is more suitable for directly receiving packets on the Backend socket rather than needing to use the CLI to receive messages. This should speed up the process since another step is removed and more packet types will be able to be handled by the Pycrocart GUI directly. A diagram below shows the new interaction between the frontend components and the Backend.



Another diagram of the FlyPi is also shown below, adding to the diagram above by showing how the camera will interact with the remainder of the system. It shows how the Ubuntu cores will handle the camera output and how it will send that data back to the backend directly.



4.8 Technology Considerations

MP-4 Lab:

- C++ GUI/Framework
  - Pros: Mostly developed, communicates with front end
  - Cons: Less responsive, will mostly likely need to run on Virtual Box Linux
- Python GUI/Cflib
  - Pros: Faster, uses open source software, more portable
  - Cons: Requires more rework of our framework, some parts of the architecture may still require Virtual Box Linux

FlyPi:

- Usage of FreeRTOS

    - Pros: Lightweight, low barrier of entry, single application layer (locality)
    - Cons: Single application layer (More complex overhead), lower OS flexibility
- Usage of Ubuntu
    - Pros: Open source, secure, minimal OS overhead
    - Cons: Restrictive hardware forwarding support, adaptation might require extra work

4.9 Design Analysis

Our first design was mostly complete and would have met the requirements, but as we began to work with each component more in detail we learned much more about the existing infrastructure and the likely cause of the speed issues and crashes that were occurring. Thus, our second design was much more mature, and although it didn't conflict with our original design, it included more details that captured our new ideas on how to improve the infrastructure. As we continue to work on adding and modifying new components, no doubt our architecture will evolve and become more detailed as we encounter new challenges and bugs which will need to rethink our design to handle.

Due to our completion of the cflib Groundstation rewrite, we have been able to quantitatively analyze that part of our design to determine whether our design was successful in reducing crashes and increasing responsiveness for the end-user using the GUI. The table below shows approximated averages for the amount of time to perform a specific action using both of the Groundstations, timed by hand. There were overall no crashes, and the times for most events were faster or equivalent to times before the rewrite.

	Time between logging event and plot (s)	# of restarts	Time to set a parameter (s)	Time to get a parameter (s)	Time to set param from JSON file (s)	Set parameter completion rate
Original Infrastructure (12 cores)	4.51	3	0	2.06	19.73	100%
With cflib groundstation (12 cores)	4.78	0	0	0.746	19.56	100%

5 Testing

5.1 Unit Testing

MP-4:

Since the infrastructure has several distinct software components, unit testing can be performed on each software module individually to ensure it is operating as intended. The front-end GUIs (Pycrocart and current) will need to be tested manually by providing user inputs to ensure that both the correct changes to the GUI are made and the correct packet is sent as displayed in the terminal. This process could be streamlined with a spreadsheet of possible test cases for a tester to manually enter.

The backend software modules (backend and cflib Groundstation) will also be tested individually, however due to the complex nature of the interfaces between them, it may be necessary to test these components using other units in the system and then checking the terminal output (see more in integration testing below).

FlyPi:

Because of the partition of the cores, we can test the FlyPi's components separately: the FreeRTOS core can be tested manually by ensuring that the drone flies as intended when it is sent commands from the Groundstation. The testing of the camera will depend more on which camera is selected to attach to the FlyPi, and by extension the testing of the Ubuntu Linux cores and the Groundstation code needed to handle the incoming data will depend greatly on which camera and which protocols are chosen.

5.2 Interface Testing

MP-4:

Interface testing will be more important to a thorough testing suite than unit testing, due to the level of complexity in communication between the different components. Since no modifications will be made to the out-of-box Crazyradio firmware to communicate with the Crazyflie, that interface will not need to be tested (beyond its relevance to other tests). The rest of the components communicate via UNIX sockets. The most efficient way to test the interface between the components would be to run them in isolation, then insert print statements when each process sends and receives a message and ensure that messages are correctly getting passed between.

FlyPi:

As explained above, interface testing for the FlyPi will mainly involve testing the interface between the Groundstation and network communication program on the drone, and the interface between the Ubuntu cores and the Groundstation which will relay the camera data. Manual testing of the first interface will involve sending commands to the FlyPi drone and verifying that the drone behaves as anticipated. Testing the second interface will involve visually inspecting the camera output that is received by the Groundstation to ensure it is correct.

5.3 Integration Testing

MP-4:

Integration and interface testing will be performed in a similar manner: running the necessary components along with the component-under-test and ensuring that the same output is generated when certain inputs are entered. Running the entire system (front-end, backend, Groundstation, and Crazyflie) will be a good test to ensure that the component-under-test is integrated correctly into the system at large. Ensuring that the drone behaves as expected on the test stands will be another important part of integration tests where the Crazyflie is involved.

5.4 System Testing

MP-4:

The most complete way to test if our implemented changes fulfill the requirements for MP-4 is to follow the lab directions for MP-4 to complete the lab as a student would, and ensure that the process is easier, faster, and more well-documented than before. This can be done in portions as we make specific changes to the infrastructure, but should also be done from start to finish once all of our changes have been made.

FlyPi:

Since there is no specific use case for the functionality of the FlyPi, system testing will mainly involve using the GUI to control the drone by setting setpoints and thrust. Future design work will create a more formalized system for handling camera output, in which system testing can be formulated.

5.5 Regression Testing

MP-4:

Much like for systems testing, regression testing for MP-4 would involve ensuring that a student can complete MP-4 without any major technical issues, with fewer bugs and crashing than was present before. Any component added or changed must not inhibit the completion of MP-4 in any way, and any additions to the lab or for the FlyPi system must not interfere with the MP-4 infrastructure. Specifically, students must be able to control the drone via a smooth GUI, view logging output of the drone, and flash firmware to the drone as needed.

Using Command Line Interface commands as shown below will form the basis of regression testing, since all changes to the system must conform to the same packet structure. Thus, testing a CLI command, changing a component, and then running the same CLI command again and testing that the same result is received will be the main form of regression testing.

FlyPi:

Since the processing abilities of the FlyPi are being overhauled completely, the only pre-existing functionality that needs to be considered is the ability of the raspberry pi to control the drone's motors. The best way to test this would simply be to ensure that the drone flies when

commands are sent. When future changes are made to the FlyPi, having a static packet structure and thorough documentation will prevent our proposed functionality from breaking.

5.6 Acceptance Testing

MP-4:

Client approval will be the first stage of acceptance testing. This will be done by demoing our changes and showing that MP-4 still functions as before with our changes. The second stage will be in the actual delivery of MP-4, where students can provide feedback on their experience, and comparing with last year's results we can determine if there was an improvement in the quality of experience.

FlyPi:

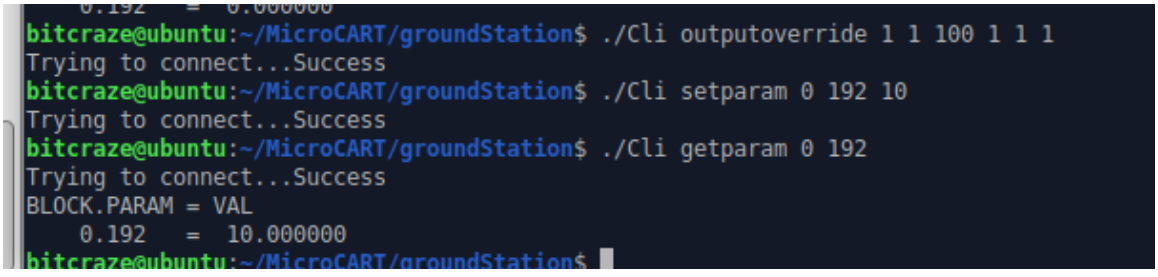
Much like with MP-4, the acceptance of the FlyPi will depend primarily on the client, and whether the FlyPi fulfills its functional requirements. In the future, the FlyPi will need to perform an interesting demonstration, and its ability to perform this demonstration will be a large component of acceptance testing.

5.7 Results

The acceptance and system testing for our changes will involve completing parts of MP-4, to ensure the components are functional and the process is smooth for CPRE 488 students. The lab document, including all of the steps to complete the lab, is linked below:

http://class.ece.iastate.edu/cpre488/labs/MP-4.pdf

The main form of internal testing will be using the Command-line interface to send certain packets, and then looking at the terminal output to see if the drone has been updated correctly (as shown below, set param sets a parameter to 10 and then get param retrieves that value stored on the drone). This will be used to determine that the system is functioning, since the CLI sends packets through all of the components to the drone.



The form linked below will be provided to students when they complete the lab, and its results will form the basis for our acceptance testing, since it will be real-world feedback on whether we achieved our goals and were successful in meeting the functional requirements for MP4.

https://docs.google.com/forms/d/e/1FAIpQLSfGyFpQvUNnkW7a7cJh7Ma-vDpCveiDsoQIRF8wEzlFCLm_Rg/viewform?usp=sf_link

6 Implementation

Our implementation plan is as described in section 3.4 “Project Schedule” above.

7 Professionalism

This discussion is with respect to the paper titled “Contextualizing Professionalism in Capstone Projects Using the IDEALS Professional Responsibility Assessment”, *International Journal of Engineering Education* Vol. 28, No. 2, pp. 416–424, 2012

7.1 Areas of Responsibility

Area of Responsibility	**Definition**	**NSPE Canon**	**SE Code of Ethics**
Work Competence	Perform work of high quality, integrity, timeliness, and professional competence.	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	Deliver products and services of realizable value and at 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, without deception, and understandable to 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	Minimize risks to safety, health, and well-being of stakeholders.	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 property, ideas, and information of clients and others.	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	Protect environment and natural resources locally and globally		2.02. 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	Produce products and services that benefit society and communities	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.

7.2 Project Specific Professional Responsibility Areas

Work Competence (Medium): Work competence is important for our project since proficiency in the necessary skills to complete the project will ensure that next year's team inherits a robust, well-documented, and well-executed project that will speed up their familiarity with the project.

Since we possess most of the necessary skills listed above, but are weak on a few (such as PCB design and bare metal CPU work), we have a “Medium” rank in this area.

Financial Responsibility (N/A): Since our project is contained wholly within the university and requires little funding or material beyond what it already possesses, this area does not apply to our project.

Communication Honesty (High): The importance of this in our project comes mostly from the communication with our advisor, and later with the students who will be completing MP4. Communication is important to resolve any outstanding issues we are having, ensure his approval of our work as we progress, and keep our progress in line with his larger visions for the project. Due to our weekly meetings and status reports that are delivered to him, we rank as “High” in this category.

Health, Safety, and Well-being (High): The only safety concern with our project is the minor concern of injury due to improper drone flight. Several methods of preventing this are in place, such as safety nets and tethers. For these countermeasures, we have the rank “High” for this group.

Property Ownership (High): Since our project is contained entirely within ISU and contains no particular material with a need to maintain a high-level of confidentiality, this area is not particularly relevant. But due to our need to keep all property within the confine of our lab space, we have included and given ourselves a ranking of “High.”

Sustainability (N/A): Our project has no environmental impacts, so this area does not apply to our project.

Social Responsibility (High): Our project doesn’t particularly interact in the short-term with the public at large, but considering the community of ISU students taking CPRE 488, then this is somewhat applicable as MP4 improvements will benefit that community. Since a major focus of our project is MP4 improvement, we have rated ourselves “High.”

7.3 Most Applicable Professional Responsibility Area

Communication Honesty is the most applicable responsibility area since most of the guidance, requirements, acceptance, and feedback for this project come from communication with our advisor. Since this is a technically complex project, communication between group members is also important.

8 Closing Material

8.1 Discussion

The final result of our project will be a quality and fully-functional MP4 lab for students, the main end to which all of our MP4 requirements are aimed towards. By gathering student feedback on our lab, we can also identify future improvements, and implement more changes after the lab is finished or minimally leave detailed instructions for next year's team. For the FlyPi, our project will end with the Raspberry Pi attached and controlling the drone successfully, and the camera attached and displaying output to another PC. If this is too ambitious of a proposal, then similarly we will leave detailed instructions for how next year's team can proceed and finish what we have started.

8.2 Conclusion

Our progress thus far has primarily been towards MP4 improvements, as we have finished the cflib Groundstation component except for some minor bug fixes and testing yet to be completed. We have begun adapting the Pycrocart GUI to be integrated with the wider infrastructure, but with our plans for a new packet structure being finalized that component will need to be extended further to handle those new packets. Our goal is to have a MP4 implementation prepared using the old packet structure, cflib Groundstation, and the Pycrocart GUI by the end of winter break, and then a MP4 implementation using the new packet structure by the time that the administration of MP4 lab begins after spring break. No significant work has been done on the FlyPi so far, but our goal is to have the Raspberry Pi Cores partitioned and mounted onto the drone by spring break, and then the camera attached and functional by the end of the semester. Our main plan of action is to focus on MP improvements for the remainder of this semester (or until the first implementation is finished), then divide the team to work on the FlyPi and the second MP4 implementation.

8.3 References

No outside references were used for this document.

8.4 Appendices

Microcart Git Repository: https://git.ece.iastate.edu/danc/MicroCART

8.4.1 Team Contract

Team Name sdmay24-32

Team Members:

1) Justin Kenny
2) Trevor Friedl
3) Steven Frana
4) Travis Massner
5) Will Maahs
6) Clay Kramper

Team Procedures

1. Day, time, and location (face-to-face virtual) for regular team meetings:
    a. Tuesday 1-2pm Advisor Meeting - Coover 3050
    b. Thursday 2-3:30pm Internal Team Meeting - Coover 3050
2. Preferred method of communication updates, reminders, issues, and scheduling (e.g., e-mail, phone, app, face-to-face):
    a. Discord
3. Decision-making policy (e.g., consensus, majority vote):
    a. Majority Vote
4. Procedures for record keeping (i.e., who will keep meeting minutes, how will minutes be shared/archived):
    a. Meeting notes will be kept in the google drive folder.
    b. Notes on GitLab issues.

Participation Expectations

1. Expected individual attendance, punctuality, and participation at all team meetings:
    a. Valid reasons must be presented for missing a meeting with proper notice.
2. Expected level of responsibility for fulfilling team assignments, timelines, and deadlines:
    a. Keep track of timeline using Git issues.
    b. Valid reasons and notice provided if internal deadlines can not be met.
    c. External deadlines will always need to be met.
3. Expected level of communication with other team members:
    a. Respond within one day on weekdays.
4. Expected level of commitment to team decisions and tasks:
    a. Each member completes his fair share of work.

Leadership

1. Leadership roles for each team member (e.g., team organization, client interaction, individual component design, testing, etc.):
    a. Technical Leads: Steven and Clayton
    b. Project Manager: Trevor
    c. Scribe: Justin
    d. Team Organization Lead: Will
    e. UI/UX Designer: Travis
2. Strategies for supporting and guiding the work of all team members:
    a. Stay in communication if issues arise.
    b. Share resources and demonstrate new features so all members can use them
    c. Store things like instructions for use and passcodes is a robust location

3. Strategies for recognizing the contributions of all team members:
    a. Look at git commit and issues distribution.

Collaboration and Inclusion

1. Describe the skills, expertise, and unique perspectives each team member brings to the team.
    a. Trevor: Skills in computer hardware, embedded systems, and several EE concepts (signals and systems, PCB design, analog/digital data acquisition, etc.) Lots of experience with Java, C/C++, RHEL, and VHDL. I've held lots of experience with being on different technical teams and understanding the workflow from an engineering perspective.
    b. Justin: Python, Java, C, VHDL, embedded systems. I have some experience working on full-stack cross-disciplinary teams. Also some experience with GUI design.
    c. Clay: Experience in embedded systems development in petalinux using Python, VHDL, C/C++. I have experience developing aviation software concerning audio and navigation.
    d. Steve: Python, C/C++, VHDL. I have experience developing tools for aviation software, hardware design for FPGA's, real-time programming, Linux driver development, and embedded systems projects.
    e. Will: Experience as a Controls Engineer in the development of large automated systems. Java, C/C++, Python, VHDL. Developing UIs using Android Studio.
    f. Travis: Python, C, Java, HTML, JavaScript, embedded systems, Android Studio, Linux navigation, cybersecurity/networking.
2. Strategies for encouraging and support contributions and ideas from all team members:
    a. Keeping meeting notes.
    b. Standup-esque meeting on thursdays
    c. Periodic demos to show what team members have been working on.
3. Procedures for identifying and resolving collaboration or inclusion issues (e.g., how will a team member inform the team that the team environment is obstructing their opportunity or ability to contribute?)
    a. Open communication with the entire group.

Goal-Setting, Planning, and Execution

1. Team goals for this semester:
    a. Improve infrastructure for CPR E 488 lab.
    b. Stabilize last year's GUI.
    c. Create a product to demo to prospective ISU students.
    d. Continue work on larger quadcopter.
    e. Get an A in the class.
2. Strategies for planning and assigning individual and team work:

    a. Use Git issue to track work
    b. Project/Team members would be in charge of assigning/picking up tasks as needed.
3. Strategies for keeping on task:
    a. Check-up in weekly meetings

Consequences for Not Adhering to Team Contract

1. How will you handle infractions of any of the obligations of this team contract?
    a. Bring it up at the team meeting or reach out to the person individually.
2. What will your team do if the infractions continue?
    a. Let Dr. Jones know.

****

a) *I participated in formulating the standards, roles, and procedures as stated in this contract.*
b) *I understand that I am obligated to abide by these terms and conditions.*
c) *I understand that if I do not abide by these terms and conditions, I will suffer the consequences as stated in this contract.*

1) Justin Kenny DATE 9/7/2023
2) Trevor Friedl DATE 9/7/2023
3) Steven Frana DATE 9/7/2023
4) Clay Kramper DATE 9/7/2023
5) Will Maahs DATE 09/07/2023
6) Travis Massner DATE 9/8/2023