

# MicroCART

FINAL REPORT

**SDMay24-32**

Dr. Phillip Jones

Trevor Friedl, Justin Kenny, Steven Frana,  
Will Maahs, Clayton Kramper, Travis Massner  
sdmay24-32@iastate.edu

<https://sdmay24-32.sd.ece.iastate.edu>

*Revised April Of 2024*

## Table of Contents

<b>1 Definitions</b>	<b>3</b>
1.1 Glossary of Terms	3
<b>2 Introduction</b>	<b>4</b>
2.1 Problem Statement	4
2.2 Intended Users and Uses	4
2.3 Skill required covered by the team	4
2.4 Project management style used by team	5
<b>3 Revised Design</b>	<b>6</b>
3.1 Requirements & Constraints	6
3.2 Engineering Standards	6
3.3 Security Concerns and Countermeasures	6
3.4 Design Evolution	7
<b>4 Implementation</b>	<b>9</b>
4.1 Detailed Design	9
4.2 Description of Functionality	11
4.3 Notes on Implementation	11
<b>5 Testing</b>	<b>12</b>
5.1 Process	12
5.2 Results	13
<b>6 Broader Context</b>	<b>15</b>
<b>7 Conclusion</b>	<b>16</b>
7.1 Review of Progress	16
7.2 Value of Design	17
7.3 Future Steps	17
<b>Appendix 1: Operation Manual</b>	<b>19</b>
<b>Appendix 2 Initial Design(s)</b>	<b>25</b>
<b>Appendix 3 Other Considerations</b>	<b>27</b>
<b>Appendix 4 Code</b>	<b>28</b>

# 1 Definitions

## 1.1 GLOSSARY OF TERMS

**Crazyflie** - The small drone used for the Embedded Systems Design (CPRE 488) “MP-4” Lab. It is manufactured by Bitcraze, and has open-source Embedded Systems Designware, which can be easily written.

**FlyPi** - Larger custom quadcopter built by previous MicroCART teams. Composed of four brushless motors, a previous MicroCART team-produced PCB controller, and a Raspberry Pi Zero 2. 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** - Also known as the “Crazyflie-lib.” Open-source Python library created by Bitcraze used for communicating with the Crazyflies using client-side software.

**cflib Groundstation** - New software module that communicates with the Crazyflies using cflib and replaces the Crazyflie Adapter and Crazyflie Groundstation.

**Crazyradio** - Bitcraze-manufactured USB radio stick that is designed to send packets to and from the Crazyflies from a host PC.

**FreeRTOS** - Free real-time operating system, which is 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.

**CPRE 488** - Also known as “Embedded Systems Design”. A class offered at Iowa State University currently taught by professor Dr. Phillip Jones. Teaches computer engineering students advanced topics in embedded systems such as FPGA programming and control systems.

**MP-4** - Standing for “Machine Problem-4”, one of the five labs taught in CPRE 488, emphasizing applications of control systems for open-source quadcopters using Crazyflie technology.

## 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 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 learning control systems theory easier and remove any frustrations with the current infrastructure.

Prospective ISU Students/Alumni - The FlyPi will be an exciting 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.

### 2.3 SKILL REQUIRED COVERED BY THE TEAM

- Embedded systems design knowledge
  - Microcontrollers
  - Raspberry Pi environment
- Understanding of software and hardware architecture
- Knowledge of Oracle VirtualBox and Virtual Machines
- Data Networking
- Software and Programming Languages
  - Python
  - C
  - C++
  - Qt Creator
- Linux Environment Experience
- Basic knowledge of Control Systems

## 2.4 PROJECT MANAGEMENT STYLE USED BY TEAM

For this project, the MicroCART team has passed down a repository of developed and documented code each year for the use of continuing progress of the project's infrastructure. Throughout our project, we adopted both a waterfall and an agile methodology to aid in developing our project to ensure a seamless experience with testing and keeping priorities on a timeline. Using Gantt charts allowed us to visualize the estimated time breakdown required for each task.

We handled specific tasks and issues that needed to be completed by creating git issues, all of which were posted on our git repository to keep track of the progress of specific tasks and estimated timelines for their completion. This allowed us to work towards completing simpler tasks as we worked towards larger goals, allowing us to keep track of more significant project deadlines throughout the semester. Through git, we also created forked branches for each task to allow team members to create pull requests after their tasks. This allowed other members to review the changes made towards the completion of a problem and allowed for milestones to be tested before integrating back into the primary system.

## 3 Revised Design

### 3.1 REQUIREMENTS & CONSTRAINTS

#### MP-4 Lab:

##### Functional:

- The GUI does not crash or freeze and communicates with the backend.
- Combine the Crazyflie Groundstation and Crazyflie Adapter using cflib.

##### Non-Functional:

- Improve video and wiki documentation for the lab and upload them to the MicroCART YouTube channel.
  - A 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 the control board

#### 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 online and is available for access for students and researchers.

### 3.2 ENGINEERING STANDARDS

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

### 3.3 SECURITY CONCERNS AND COUNTERMEASURES

Since the Crazyflie platform created by Bitcraze is open-source, there are no major security concerns with open access to the MicroCART git repository.

### 3.4 DESIGN EVOLUTION

#### Groundstation Architecture:

Upon the start of the project design phase, we were left with the use of a final design from last year's MicroCART iteration for the Groundstation system used for the infrastructure implemented for CPRE 488's MP-4 lab. Appendix 2, Figure 15, details the high-level structure of the Groundstation design responsible for interfacing all peripherals throughout the project. Below in Figure 1, we feature the necessary changes to change our system from using the Crazyflie Adapter to our newly created cflib adapter.

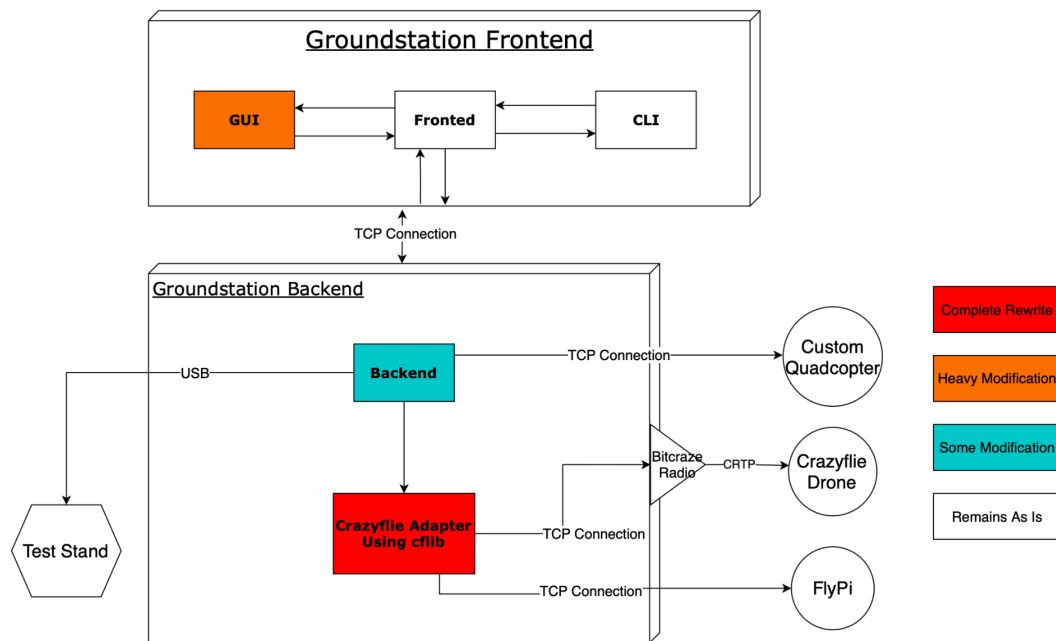


Figure 1: Updated Groundstation architecture with noted modifications

Slotting our new cflib adapter into the infrastructure requires changing three main components of our Groundstation design: the backend adapter, our new Crazyflie Adapter (using cflib), and the GUI presented on the Groundstation Frontend. Changes to the GUI are necessary as the changes in the backend infrastructure will need to handle the data differences and any other bugs caught along the way. The backend subsystem will need changes allowing for new connections to the cflib-based adapter to ensure that the backend encapsulates that data similarly to the old adapter. And finally, with importing the cflib technology, the adapter endpoints required a total rework for the design changes to be implemented functionally.

## FlyPi:

As described in Figure 2, the Crazyradio is responsible for sending CTRP Packets, a standard developed by Bitcraze for packet structure formatting into the network communication program intended to transmit packets over on the Raspberry Pi Zero 2W. After packets were sent and received, data would be sent into the shared memory space and parsed between the Ubuntu and Baremetal cores in order to control the flight and other necessary calculations for the FlyPi quadcopter.

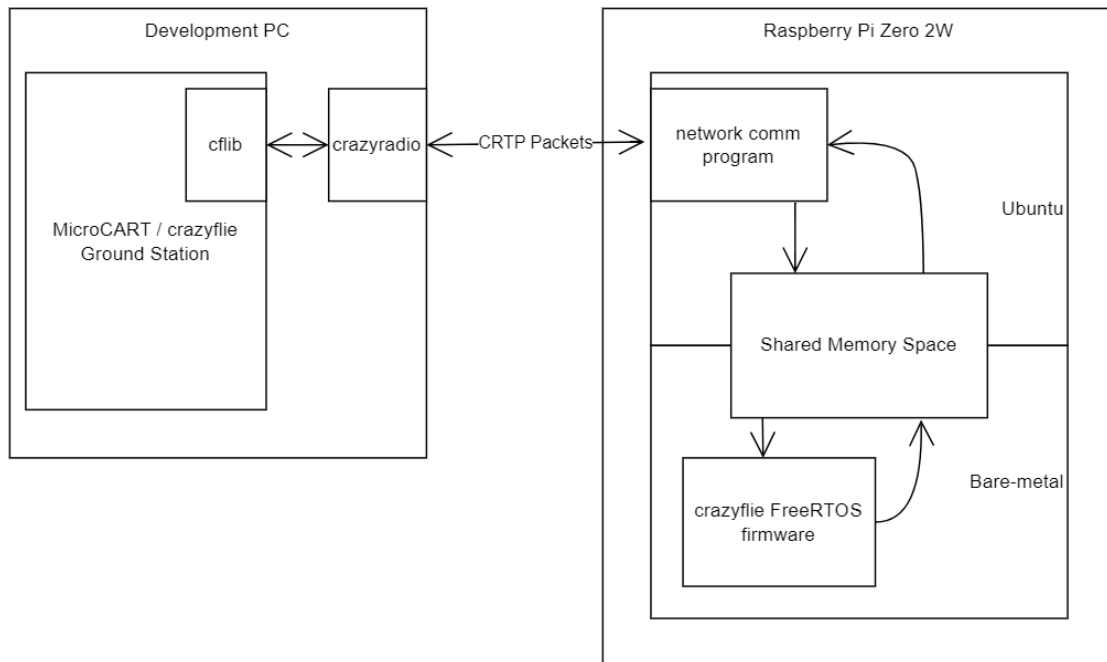


Figure 2: Initial FlyPi Design

The FlyPi was initially intended to use the crazy radio packets to transmit data between the Groundstation and quadcopter. We modified this design to use a TCP connection between the Groundstation backend and quadcopter. We then deploy the adapter on the Ubuntu side of the drone. This allows us to take advantage of a more robust connection, use less hardware, and parse the packets into a format suitable for the FreeRTOS firmware. These modifications are shown in Figure 4.



## 4 Implementation

### 4.1 DETAILED DESIGN

#### MP-4/Infrastructure

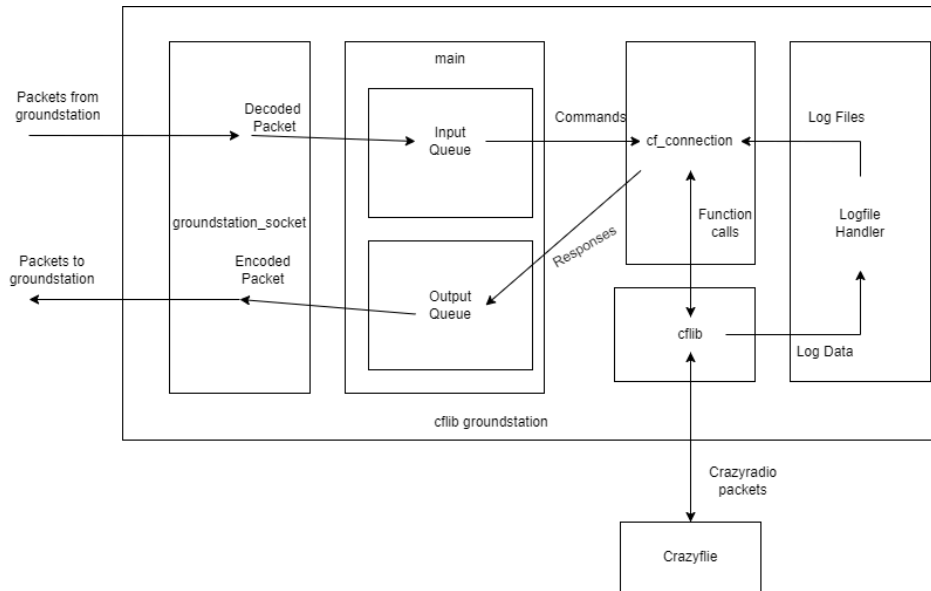


Figure 3: Detailed Crazyflie adapter block diagram

A detailed diagram of the internal structure of the new Crazyflie adapter using cflib is shown in Figure 3, showing the individual classes and the data flow between them. The `groundstation_socket` class manages the TCP connection to the Backend and is responsible for encoding and decoding packets as they leave and enter the system. The `main` class holds two queues, one for input and one for output, which organizes incoming and outgoing packets between the drone and the Backend. When an incoming packet is pulled from the queue, it is parsed to determine which function from `cf_connection` to call. The `cf_connection` function manipulates the data accordingly and calls the appropriate cflib function to produce the desired effect on the drone. The results from the cflib function are packed and placed into the output queue and are then sent back to the Backend. There is a particular case for logging data sent to the `LogfileHandler` class. The `LogfileHandle` class writes logging data to local files on the system, and the path to this file is sent back to the Backend.

## FlyPi

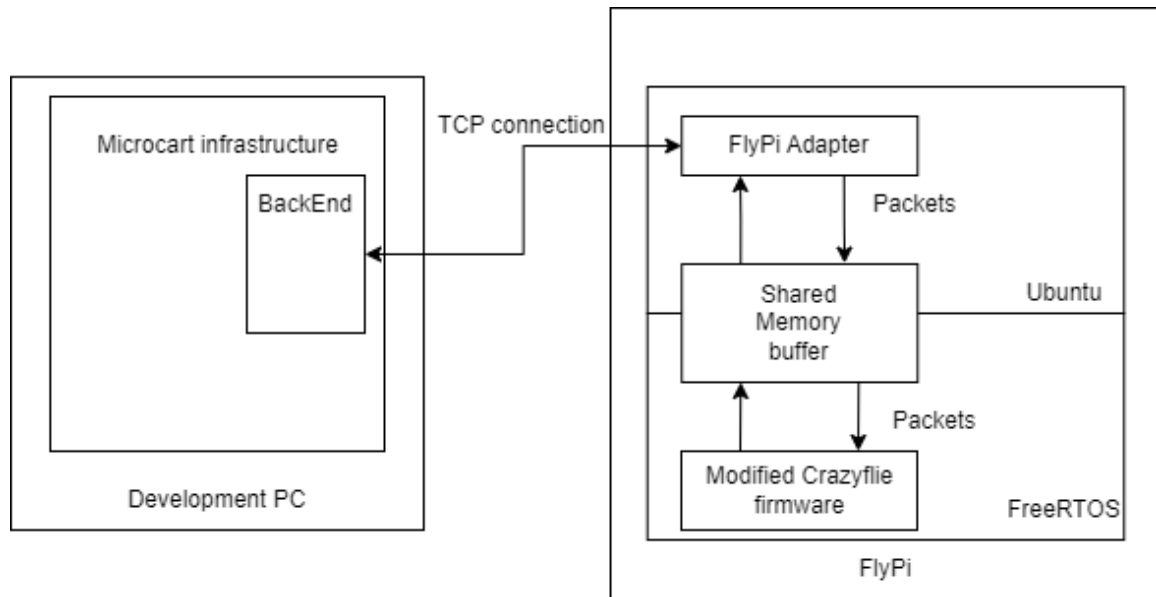


Figure 4: Detailed FlyPi block diagram

The design in Figure 4 fulfills the requirements for the FlyPi since it includes partitioning the Raspberry Pi CPU. Since the FlyPi will be using a modified version of the Crazyflie firmware, keeping the packets as intact as possible throughout the process is a high priority for this design. To communicate with the FlyPi, the FlyPi adapter running on the Pi Zero's Linux cores accepts a TCP connection from the BackEnd to handle packet flow. This FlyPi adapter is a modified version of the Crazyflie adapter. Still, instead of using cflib to transmit packets over the radio, it places packets into shared memory using the Python 'mmap' command. The modified Crazyflie firmware running on the FreeRTOS cores can then read any command packets from the shared memory buffer and place data to be sent back to the BackEnd in the shared memory buffer for the adapter to read.

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 files was partially responsible for the slow speeds and crashing of the GUI. To rectify this issue, we designed a new packet structure 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>

## 4.2 DESCRIPTION OF FUNCTIONALITY

The functionality of the MP4 Infrastructure design beyond what is described above is as follows: when using the infrastructure, a shell script in the repository can start all components and connect them in succession. After a drone is started up and finished its startup sequence, the user can specify a radio number when running the shell script that will open the Crazyflie adapter and attempt to connect it to the drone. After this is successful, the BackEnd is started, and the socket connection between the BackEnd and Crazyflie adapter is opened. Then, the GUI is started, connecting to the BackEnd via socket and sending command messages. The user can then use the GUI to send the desired command messages through the infrastructure to the drone. Each part of the infrastructure and also be run individually for greater control and testing purposes.

Our designs for the MP4 Infrastructure and FlyPi are intended to provide a high level of decoupling between the various systems comprising the entire MicroCART project. The Frontend options give the user greater flexibility in controlling the drone and displaying their data. The Backend allows for connecting other peripherals to the system, such as the lighthouses or camera trackers. Having separate adapters for each quadcopter type allows the same Frontend and BackEnd tools to be used with different quadcopters or even the future additions of other technology.

Concerning the FlyPi design, our most significant task concerned our need to partition the cores into running different instances of operating systems, for example, one of the cores will need to run FreeRTOS while the other three cores will be required to run Ubuntu. The functionality of the design is intended to be as described in Figure 4. Appendix 1 contains detailed information about how the cores are partitioned.

We can consider our current design implementation to be satisfactory to our current design's previously agreed functional and non-functional requirements, as the functionality of the designs improves the quality of the MP4 system and partitions the FlyPi cores according to the requirements. Our big focus was to shorten any critical paths from our initial design to support minimized overhead while delivering an accurate and persistent connection to different systems throughout the Frontend, Backend, Crazyflie, and FlyPi.

## 4.3 NOTES ON IMPLEMENTATION

Although we could finish the Crazyflie adapter with cflib and the partitioning of the FlyPi cores, there were some tasks we didn't possess the time or resources to complete this semester. We could not fully move away from the file-based logging system as described in our revised packet structure, but we have left detailed notes for next year's team to pick up where we left off. Likewise, the FlyPi adapter currently contains a basic shared memory implementation as proof of concept. Still, it doesn't fully implement a ring-style shared memory buffer, as the application will demand when fully completed. Due to hardware differences in cross-compilation, we never finished modifying the Crazyflie firmware to run on the Raspberry Pi architecture. Likewise, we left detailed instructions for next year's team to pick up on FlyPi development quickly.

## 5 Testing

### 5.1 Process

Due to the nature of our project and the fact that our project's infrastructure used many different parts of the software at once, we found that manually testing the changes we made worked the best and gave us a good picture of whether our changes were functioning correctly. To test the different aspects of our project, we implemented a few methods of manual testing.

To test the changes to the Crazyflie adapter and cflib, we acted as a student and manually went through the MP4 lab. We made sure to test every aspect of the changes that we made and made sure that the MP4 lab experience was as expected. Through the MicroCART GUI, there is a tab that allows the user to set parameters for the logging of the crazyflie drone variables. We tested this functionality to test whether the `getparam` and `setparam` worked as expected and whether it would be completed in a timely manner. We did this multiple times to get a good understanding of the quality of the changes.

When getting the MP4 lab ready for students to use, we thoroughly tested each of the crazyflie drones that we had in order to determine whether they were suitable to be used by the students for the MP4 lab. To do this, we would test various functionalities of the MP4 lab that the drones should have been able to successfully perform. When testing we would also test from the perspective of multiple different user models. For example, we would test the GUI like we were a user who didn't know anything about the framework or infrastructure and this was their first time using the GUI, or a user who had some knowledge of controls but no example with this specific framework. Doing this ensured that we tested as many edge cases as we could that may result in unexpected behavior.

From these tests, we were able to determine what needed to be repaired on the drones. The first test we ran on the drones was to test whether the drone would turn on. If the drone was unable to turn on, this usually indicated that there was a problem with the power cord connection or some other hardware problem with the drone. Next, we would test whether the drone was able to be flashed with the firmware through the crazyradio. If the drone was unable to be flashed with the up-to-date firmware, this usually indicated that there was a connectivity issue either with the crazyradio or crazyflie. After this, we would test whether we were able to open the lab GUI. Since the GUI needed a connection from the crazyflie drone to open properly, the GUI not opening (or opening in a faulty state) usually indicated that there was a connectivity issue with the crazyflie. Finally, if the drone was on, with the up-to-date firmware, and the GUI opened, we would test whether the crazyflie was able to receive thrust commands from the GUI. If the drone thrust after receiving these commands, this would indicate that there were connectivity issues or that the drone propellers were faulty. If the drone passed all of these tests, we determined that the drone was able to be used by the MP4 lab. If the drone did not pass the tests, we would alleviate the associated issue.

It is also worth noting that since our infrastructure is used in the MP4 lab, many students in 488 were testing our infrastructure by doing the lab as well. This gave us an excellent opportunity to gather feedback from a class of users. Comparing the feedback given by last year's students to this year's students was an effective way to test our modifications to the framework. We also were able to incorporate some of the minor changes into the MP4 lab infrastructure, and left notes for next year's team with the feedback we receive and how it should be addressed.

## 5.2 RESULTS

We received The following results from our testing of the MP4 lab with our new Crazyflie adapter. In Figure 5, you can see the different metrics that we tested, such as the time between logging events received, the number of times the GUI had to be restarted, the time it took to get and set a parameter, and various other metrics. As you can see in Figure 5, the results from our testing showed that our new Crazyflie adapter significantly improved the time it took to get a parameter from 2.06 seconds down to 0.746 seconds. From the results, we also noticed that the number of times that we had to restart the GUI dropped from three times to zero times. Among the other metrics, our new Crazyflie adapter seemed to perform similarly.

	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%

*Figure 5: Crazyflie adapter testing results*

Figure 6 shows our current, up-to-date spreadsheet of the status of the 12 crazyflies. Many of these drones have undergone various repairs throughout the year, so this spreadsheet has been updated frequently with their status. As you can see, many of the drones are in a healthy state. They passed the tests we ran on them and can thrust, showing they are MP4 lab-ready. There are a few drones, drones four and eight, that passed all of our tests. However, these drones have a loose power cable that often shuts off the drone if mismanaged. These drones will need their power cable to be resoldered onto the drone to ensure they are ready for next year's MP4 lab. Another drone that is having power cable problems is drone three. Drone three's power cable is completely disconnected from the drone so that will need to be resoldered onto the drone as well. Finally, drones five and nine have an issue with them that we call "the blue light of death." This refers to when the drone is in a broken state, indicated by only one of the four lights on the drone illuminating. We have faced this problem many times throughout the year, and many times, we were unable to find a consistent solution to this problem. Usually, the solution was one of several different remedies, such as reflashing the drone via USB or setting the drone to a bootloader mode; however, there still seemed to be several times when our solutions didn't work.

MP4 Drones	Channel	Status	Turns on	Flashable	GUI Opened	Thrusts
1	10	Healthy	x	x	x	x
2	20	Healthy	x	x	x	x
3	30	Power Cabel needs resoldered				
4	40	Power cable is very loose	x	x	x	x
5	50	Blue light of death	x	x		
6	60	Healthy	x	x	x	x
7	70	Healthy	x	x	x	x
8	80	Power cable is very loose	x	x	x	x
9	90	Blue light of death; power button doesn't seem to work	x	x		
10	100	Healthy	x	x	x	x
11	110	Healthy	x	x	x	x
12	120	Healthy	x	x	x	x

Figure 6: MP4 lab drone testing results

## 6 Broader 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.

## 7 Conclusion

### 7.1 REVIEW OF PROGRESS

#### **MP-4 Lab:**

For the MP4 lab, we simplified the current infrastructure by combining the Crazyflie groundstation and adapter using cflib in Python. This allows for better response times when retrieving or setting parameters for the drone. This also results in fewer GUI crashes, and better response times. We successfully slotted this new component into the MP4 infrastructure and manually tested it to ensure it was valid. No major bugs arose from this component during the duration of the MP4 lab. After experiencing some other issues with the VM and the Crazyflie hardware, we are leaving detailed documentation of what issues we faced and what solutions we found to ensure next year's team can create an even more valuable MP4 experience.

We had originally planned to also slot the python-based Pycrocart GUI into the MP4 infrastructure. Despite making significant progress towards this goal, we were unable to finish this before the MP-4 lab began. When it became clear to our team that we would not be able to complete the development of the new GUI before the MP-4 lab, our team decided to shift more of our attention to the development of the FlyPi. In conjunction with this, after working with infrastructure more closely and learning its inefficiencies, we designed a new packet structure that would eliminate the use of reading and writing to files for plotting logging data. We were unable to implement this feature either, but we have created documentation on the progress towards these two goals.

#### **Fly Pi:**

After modifying some open-source tutorials, we have successfully partitioned the cores for AMP (Asymmetric Multi-Processing) on the Raspberry Pi. One core runs FreeRTOS, while the other three cores run Ubuntu Linux. This setup can be run on a Raspberry Pi 3, 4, and Zero 2. We have been able to successfully run several simple baremetal applications which we created, such as the LED blink example shown in Appendix 1. We can successfully communicate bytes between these cores using our own baremetal implementation of shared memory for the Raspberry Pi 3 and Zero 2, while using a shared memory library for the Raspberry Pi 4.

We have made progress refactoring the CrazyFlie firmware in order to port it over to the Raspberry Pi. In its current state it is compilable, but requires some further changes in order to run it on the version of FreeRTOS we would like to use for the Raspberry Pi's. Due to time constraints, we will be unable to fully cross-compile the firmware by the end of the semester, but we are creating detailed steps on our process, showing the errors we received, and describing how thought these issues might be fixed.



## 7.2 VALUE OF DESIGN

### MP-4 Lab:

The MP-4 lab is an extremely valuable exercise for understanding PID controllers. CPRE 488 students are able to adjust the PID constants for the controllers used by the drones. This allows students to gain practical experience in tuning PID controllers. The MP-4 infrastructure also enables students to visualize their changes to the controller by graphing setpoints and sensor values, allowing students to gain a deeper understanding of the effect each constant has on the behavior of the controller. Our particular design aims to provide a more enriching experience by removing many of the difficulties students face when working with the lab infrastructure so they can focus more on learning the control systems concepts.

### FlyPi:

The FlyPi drone offers researchers the potential to experiment with a larger vehicle with expanded capabilities. This drone gives users the ability to use a larger drone to experiment with and test their control algorithms. The FlyPi offers an increased payload capacity, which could support the addition of multiple useful and interesting features, such as a camera or some sort of carrying mechanism.

The FlyPi also gives users access to a significantly higher amount of computation power. The microprocessor on the Raspberry Pi Zero 2 has four cores, as opposed to the crazyflie drone's microprocessor, which only contains a single core. This allows us to run all of the drone's firmware on one core of the Raspberry Pi Zero 2 using the FreeRTOS operating system. Meanwhile, the other three cores are utilizing the Ubuntu Linux Operating System which can be used however the users/developers best see fit. This computation power enables support for any number of additional features a user/developer may have.

## 7.3 FUTURE STEPS

### MP-4 Lab:

The biggest flaw in the infrastructure is the GUI application. More specifically the logging functionality of the GUI. We have observed that setpoint packets from the GUI to the drone are frequently dropped. This results in the drone getting the wrong setpoint, which can make tuning a drone difficult. Additionally, we have noticed that the GUI frequently experiences a high latency when plotting live data it is receiving from the crazyflie drone's onboard sensors. Future teams should consider rewriting the GUI application. Adding a packet for plotting instead of reading/writing from a file may improve the system's overall responsiveness. Ideally, there should be two separate packets: one being used for logging large amounts of data that does not need to be represented as a graph in real-time, and the other being used to relay a smaller amount of data that is intended to be graphed in real-time. Implementing the new packet structure we proposed would be an efficient way of achieving this goal.

### FlyPi:

In order to get the FlyPi in the air using the Raspberry Pi Zero 2, our team estimates that the largest task will be to rewrite the crazyflie firmware such that it can be run on the FlyPi. The most challenging part of rewriting the firmware will be the IMU firmware. The first step in this process is to find all of the firmware code that pertains to hardware specific to the crazyflie drones, and rewrite it for the hardware specific to the FlyPi. This will not be a trivial task. This will require an intimate understanding of the IMU and peripheral board designed by last year's team, which our team did not have the time to fully grasp. Finishing the firmware rewrite will also involve ensuring the version of FreeRTOS used is compatible with the Raspberry Pi and making any changes to the drone firmware to integrate a potentially different version of FreeRTOS.

Another future step for the FlyPi will be creating a more robust shared memory buffer between the cores. Our team executed a simple version of a shared memory buffer as proof of concept, but we didn't have the time to complete a version suitable for the needs of the system. The system will eventually need two ring-style buffers, one each for the partitions to read and write from. More code will have to be written on both the baremetal C and python-based adapter applications to manage and use these shared memory buffers correctly.

## Appendix 1: Operation Manual

### MP4 Infrastructure Operation Manual

The steps to operate the MP4 infrastructure are shown below. This is a simplistic version of what the system is capable of for illustrative purposes. For more detailed instructions, theory being operation, and best practice, consult the MP4 Lab document given to students:

<https://class.ece.iastate.edu/cpre488/labs/MP-4.pdf>

1. After opening the VM, attach a battery to the Crazyflie and take note of which number it has on the bottom. Multiply this number by ten to get the radio number, which will be important in later steps.
2. Connect a Crazyradio to the development PC and select it in the VM's menu under the path Devices > USB > Bitcraze AB Crazyradio.
3. If using the shell script to start the infrastructure, open a terminal and use the “crazycart [radio number]” command to launch all necessary components. After some time and terminal output, the GUI should be active on the screen, as shown in Figure 7. If this step is followed, skip to step 7

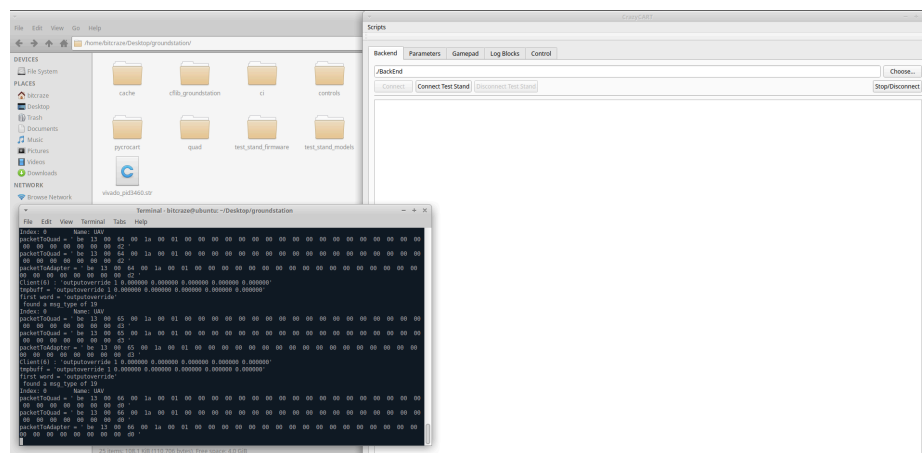


Figure 7: GUI operating with terminal output shown

4. If starting the components manually, open a terminal in the “cflib\_groundstation” folder on the repository and enter the command: ‘python3 main.py [radio number]’. The output will be displayed as below.

```
bitcraze@ubuntu:~/Desktop/groundstation/cflib_groundstation$ python3 main.py 70
['main.py', '70']
starting cflib groundstation
Opened groundstation socket.
Server is listening for incoming connections...
Connect quad
```

Figure 8: Output from starting crazyflie adapter with cflib

5. Open a terminal in the “groundStation” folder on the repository. First run ‘make’, then run ‘./BackEnd’. This will begin the Backend with the output below:

```

bitcraze@ubuntu:~/Desktop/groundstation/groundStation$ ./BackEnd
backendSocket = 3
Starting client socket: UAV
Trying to connect...Success
Ignoring VRPN information...

```

Figure 9: Output from starting BackEnd

- Open another terminal in the 'groundStation' folder and use the command './GroundStation' to start the GUI. After the GUI appears, tuning can begin.

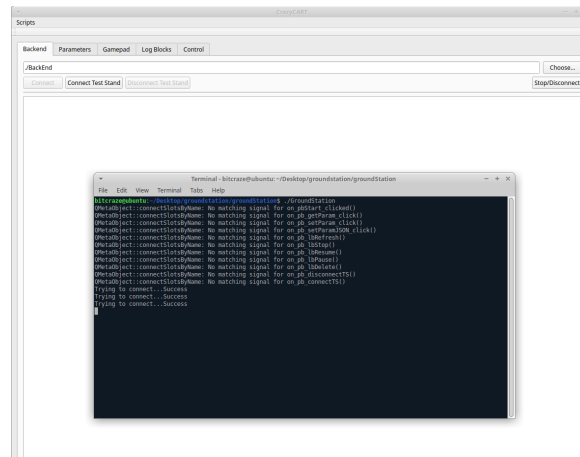


Figure 10: Launched GUI with terminal output

- To begin tuning, use the "Parameters" to alter parameters on the drone. For the MP4 lab, the "s\_pid\_attitude" and "s\_pid\_rate" groups contain all the desired entries. Once an entry in the dropdown has been selected, a value can either be set with the "Set Param" or read with the "Get Param" button, as shown in Figure 11 below.

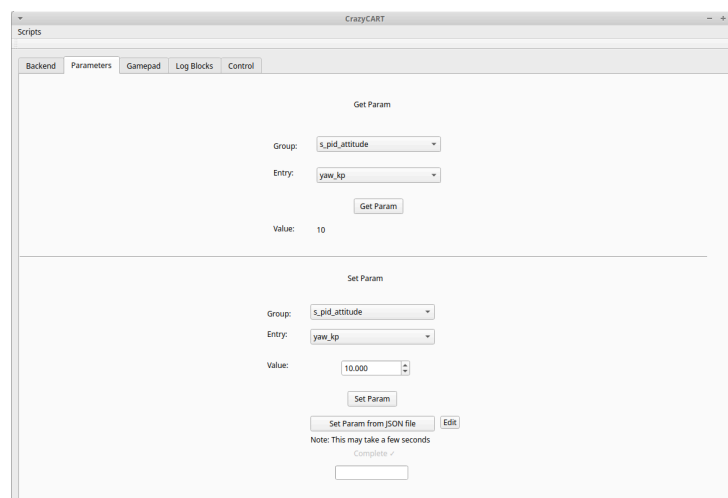
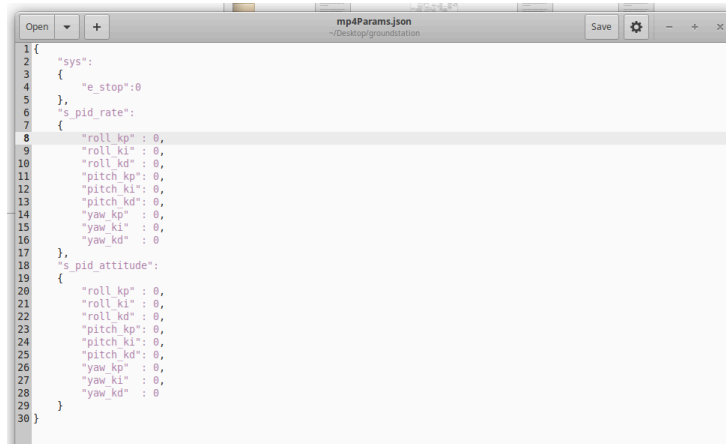


Figure 11: Getting and setting parameters

- a. Params can also be set in batches using a JSON file. To do this, click the “Edit” button to open the JSON file, edit the file as desired, then close it and click “Set Param from JSON file”, as shown in Figure 12 below.



```

1 {
2   "sys":
3   {
4     "e_stop":0
5   },
6   "s_pid_rate":
7   {
8     "roll_kp" : 0,
9     "roll_ki" : 0,
10    "roll_kd" : 0,
11    "pitch_kp": 0,
12    "pitch_ki": 0,
13    "pitch_kd": 0,
14    "yaw_kp"  : 0,
15    "yaw_ki"  : 0,
16    "yaw_kd"  : 0
17  },
18  "s_pid_attitude":
19  {
20    "roll_kp" : 0,
21    "roll_ki" : 0,
22    "roll_kd" : 0,
23    "pitch_kp": 0,
24    "pitch_ki": 0,
25    "pitch_kd": 0,
26    "yaw_kp"  : 0,
27    "yaw_ki"  : 0,
28    "yaw_kd"  : 0
29  }
30 }

```

Figure 12: Parameters JSON file

8. The “Control” tab is used to control the drone and view output. To send a setpoint to the drone, enter the desired setpoint in the top left, move the thrust slider to the desired location, and press “Apply”. Use the “Stop” button to halt the drone.

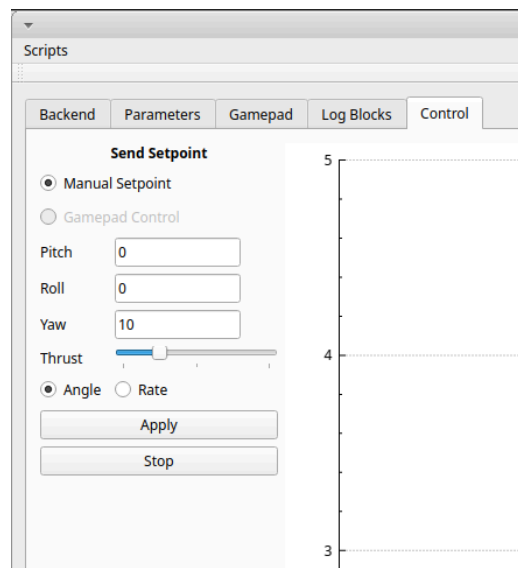


Figure 13: Sending a setpoint to the drone

9. To view the logging data from the drone, select logging parameters from the dropdowns in the bottom left and click “Start Logging”. Logging output should be shown in the logging panel, as in Figure 14 below.

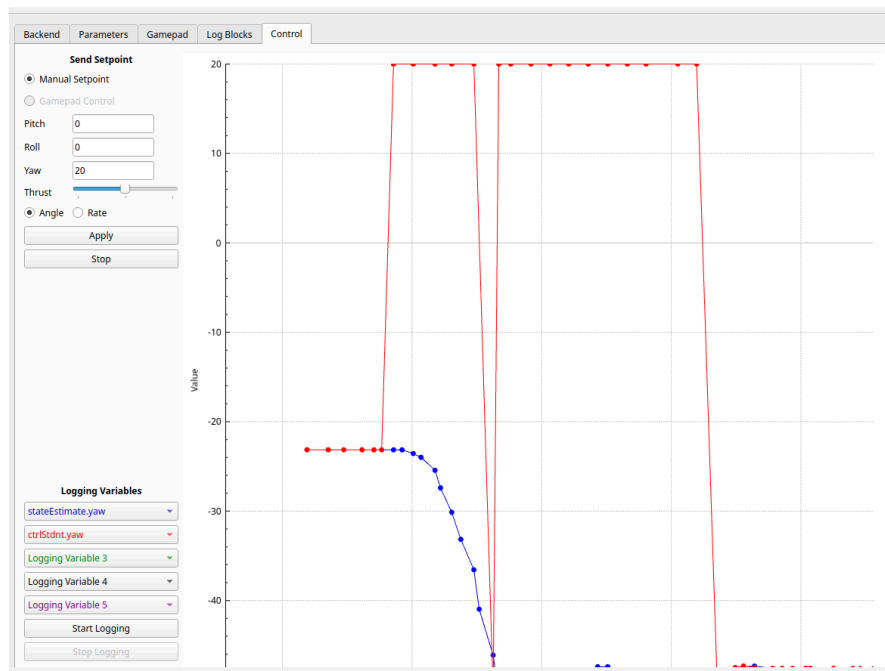


Figure 14: Logging data

## FlyPi/Raspberry Pi Zero 2 Operations Manual

Below are the instructions for compiling and running the LED blink example we have created for baremetal FreeRTOS on the Raspberry Pi Zero 2W. This once again is an illustrative example of a simple baremetal application as a proof of concept, since the shared memory and crazyflie firmware are unfinished and too complex to detail here.

1. To set up the hardware a TTL cable will need to be connected to the Pi so that UART can be used to connect to the u-boot console and type commands to the Pi. Raspberry Pi pinout diagrams are readily available online. The following pin connections should be made for this example;
  - a. GPIO 14: TTL TX cable (white).
  - b. GPIO 15: TTL RX cable (green).
  - c. Ground: TTL ground cable (black).
  - d. GPIO 17: LED high connection.
  - e. Ground: LED ground connection.
2. Once these pins have been connected, open the “uboot-compiler” folder in the repository and open a terminal. It is imperative that this terminal remains open for the remainder of this process. Enter the following command:  

```
'PATH=`pwd`/install-lnx/gcc-arm-9.2-2019.12-x86_64-aarch64-none-elf/bin:$PATH'
```
3. Remove the SD card from the Pi and attach it to a microSD card reader, then insert it into the development PC
4. Using the terminal from Step 2, run the following commands to cross-compile the baremetal application:

```
cd bm_led_blink/FreeRTOS/Demo/CORTEX_A72_64-bit_Raspberrypi4/uart
make CROSS=aarch64-none-elf-
sudo cp ./uart.elf /media/bitcraze/system-boot/
```

Figure 15: Terminal commands for cross-compilation

5. Eject the SD card from the development PC and insert it into the Raspberry Pi.
6. Connect the TTL cable to a computer with puTTY installed, and check the device manager to see which COM port the TTL cable is connected to. Replacing “COM3” with the correct COM port, enter the settings into the puTTY menu as shown in Figure 16 below.

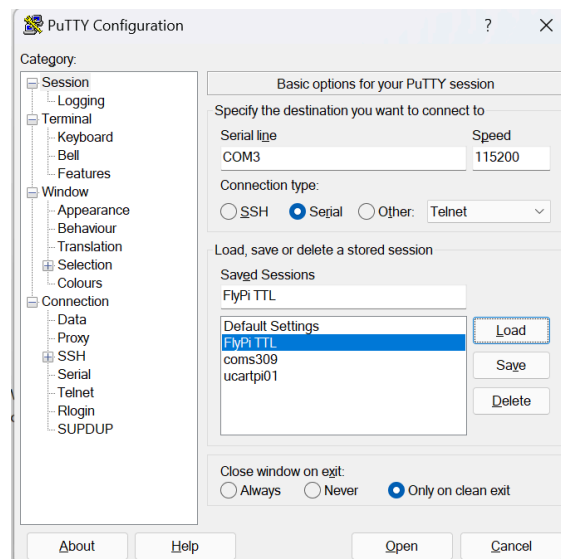


Figure 16: FlyPi TTL settings

7. Power on the Raspberry Pi, and press any key when the ‘Hit any key to stop autoboot’ message appears. Now, the u-boot console will be displayed in the PuTTY terminal.
8. Enter the commands below into the u-boot console.

```
setenv autostart yes
dcache off
fatload mmc 0:1 0x28000000 uart.elf
dcache flush
bootelf 0x28000000
```

Figure 17: U-boot commands

```
DRAM: 948 MiB
RPI 3 Model B (0xa02082)
Core: 83 devices, 13 uclasses, devicetree: board
MMC: mmc@7e202000: 0, mmcnr@7e300000: 1
Loading Environment from FAT... Unable to read "uboot.env" from mmc0:1...
In: serial,usbkbd
Out: serial,vidconsole
Err: serial,vidconsole
Net: No ethernet found.
starting USB...
Bus usb@7e980000: USB DWC2
scanning bus usb@7e980000 for devices... 3 USB Device(s) found
scanning usb for storage devices... 0 Storage Device(s) found
Hit any key to stop autoboot: 0
U-Boot> setenv autostart yes
U-Boot> dcache off
U-Boot> fatload mmc 0:1 0x28000000 uart.elf
282648 bytes read in 39 ms (6.9 MiB/s)
U-Boot> dcache flush
U-Boot> bootelf 0x28000000
## Starting application at 0x20001788 ...
## Application terminated, rc = 0x0
```

*Figure 18: U-boot command output*

9. After running these commands, the LED will turn on. Enter the following commands to boot Ubuntu Linux on the Pi then.

```
dcache on
run bootcmd
```

*Figure 19: U-boot linux commands*

10. After this, Linux will boot. The LED should remain on after Linux finishes, proving that the bare metal application still operates after Linux has booted.



## Appendix 2 Initial Design(s)

### Groundstation Infrastructure:

All initial designs given to us were implementations of previous designs described by the recent iteration of MicroCART from last year's senior design team. The initial design of the GroundStation features high-level connections from the GroundStation frontend out to the GroundStation backend and so forth to any peripherals using the infrastructure on experiment or lab PCs. Then, the GroundStation backend is responsible for interfacing all peripherals through the backend and the prepackaged Crazyflie Groundstation. In other words, the "Groundstation Backend" is our adaptation of the system accountable for interfacing with other quadcopters than just the Crazyflies.

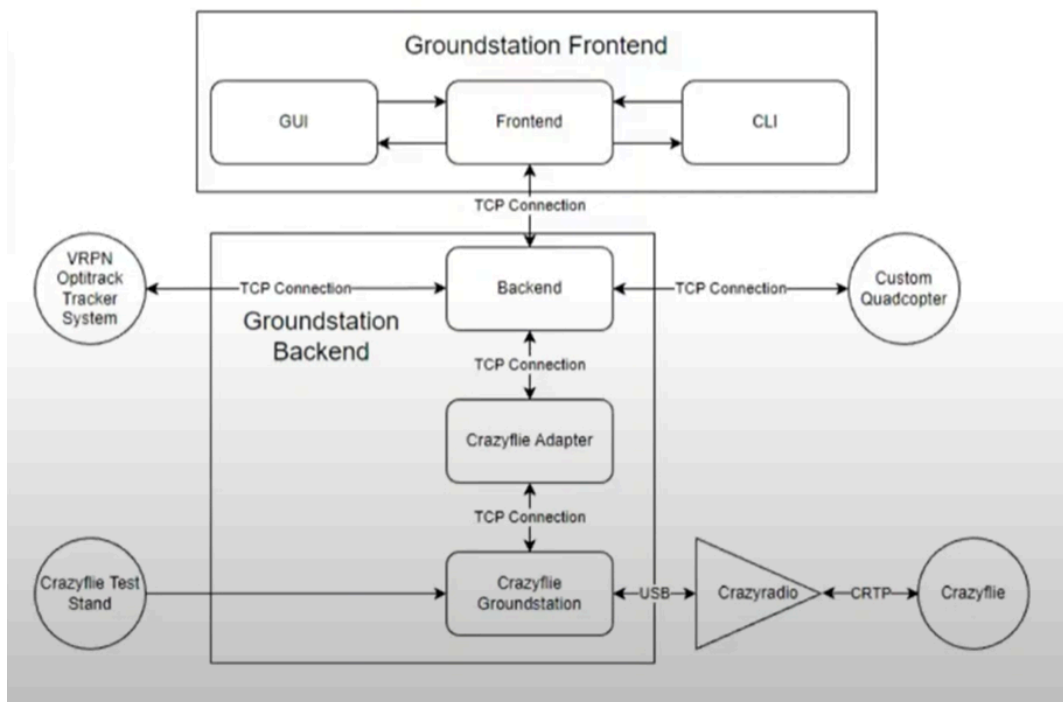


Figure 20: Initial design for Groundstation high-level architecture

**Groundstation Frontend:** Includes the necessary components for our GUI to interact with the data being passed in by the backend. The Groundstation Frontend displays this information within the GUI to show cleanly formatted data and raw packet data featured in the CLI. The frontend subsystem then directly interacts with the backend subsystem to handle the proper transfer of packets through a singular TCP connection.

**Groundstation Backend:** The backend connects a few different main components: our internal backend subsystem, an adapter (in the case of Figure 20, the Crazyflie Adapter), and the Crazyflie Groundstation. Our backend subsystem features multiple TCP protocol connections that allow packets to be sent to other interfaceable quadcopter systems and VRPN Optitrack tracking systems to have higher-level control of the drones in 3D space-presenting software. The backend also

features a TCP connection out to the Crazyflie Adapter, aiding in the formatting of packets to be sent from Crazyflie-specific hardware to other custom drones, as well as other peripherals such as our Crazyflie Test Stand and the Crazyradio responsible for wireless networking.

### Pycrocart GUI Initial Design:

Figure 21 below denotes the usage of the Pycrocart GUI in replacement of the standard C++ GUI currently used throughout the infrastructure. The packets sent by the Pycrocart GUI are entirely independent of the methodologies used to encapsulate the packets used throughout the current C++ GUI and format the Crazyradio packet using the CRTP protocol (Bitcraze-standard Crazyradio packets). A TCP socket becomes open between the Pycrocart process and the backend. It transfers packets between the GUI/CLI and the backend subsystem in the Groundstation infrastructure.

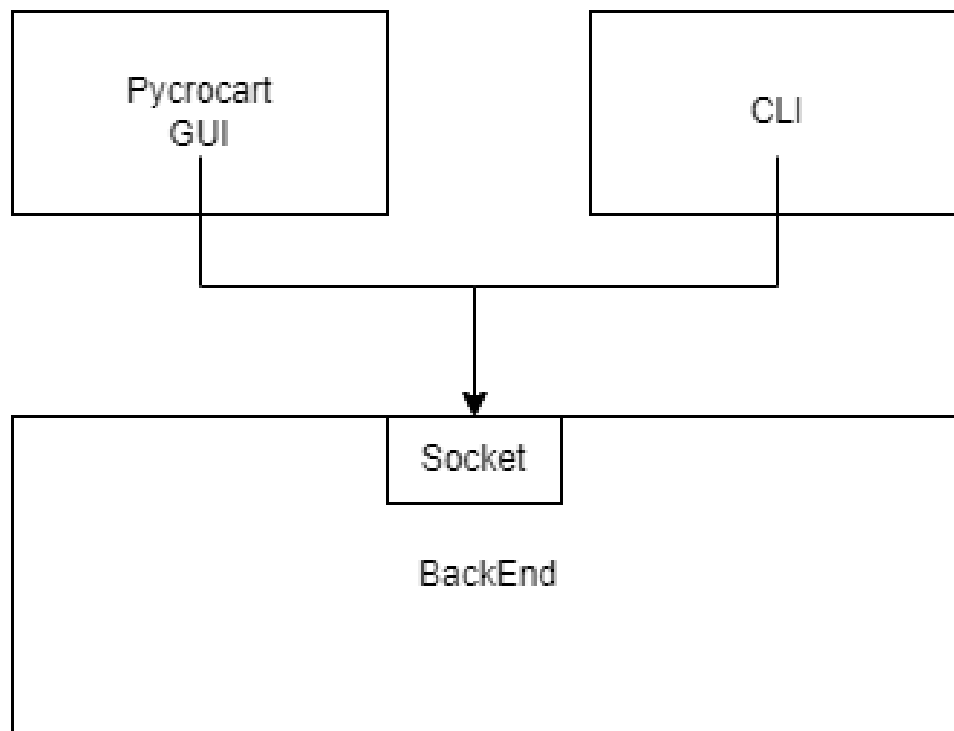


Figure 21: High-level structure of FlyPi slot into current MicroCART infrastructure

## Appendix 3 Other Considerations

As mentioned previously, MicroCART is an ongoing senior design project. As such, we credit all previous teams who have contributed to working towards our goal. We also appreciate the time, effort, and knowledge of Dr. Jones, who encouraged us to overcome obstacles and achieve our maximum potential as a team. One of the particular difficulties of this project is getting a new team acclimated with the infrastructure every year, and we appreciate his dedication to helping us learn and grow in the context of this project. Furthermore, we would like to thank several members of the previous MicroCART team for being continually available to answer questions and assist despite the duration of their studies coming to an end as an undergraduate student at Iowa State University.

## Appendix 4 Code

All past, present, and future code for the MicroCART team can be found on the MicroCART GitLab repository: <https://git.ece.iastate.edu/danc/MicroCART>