MicroCART

Final Design Document

Team Number: 43
Client: Dr. Phillip Jones
Advisers: Dr. Phillip Jones
Team Members/Roles
Brandon Cortez - Test Stand Lead
Reid Schneyer - Test Stand Sub-team
Colton Glick - Firmware Lead, Git Manager
Ellissa Peterson - Team Member
Ryan Hunt - System Architect
Carter Irlmeier - Web Manager, Lighthouse Sub-team
Zachary Eisele - Groundstation Lead, Co-System Architect
Team Email: sdmay22-43@iastate.edu
Team Website: https://sdmay22-43.sd.ece.iastate.edu
Revised: 4/28/2022

Table of contents

Executive Summary

Development Standards & Practices Used

- CI/CD pipeline in git
- 3D printing guidelines
  - IEEE P3030
  - https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7308141
- Bluetooth/Radio communication standards
  - https://standards.ieee.org/standard/802_15_1-2002.html
- Crazy Real Time Protocol (CRTP)
  - Packet protocol used by Crazyflie
  - https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/functional-areas/crtp/

Summary of Requirements

- Test stand must record and transmit movement data to a host computer
- GUI must display all relevant data
- Drone firmware must be modular so that control logic can be removed and substituted
- Develop software to stabilize, and communicate with the mini-quadcopter.
- Develop testing rigs to allow the team and users (CPRE 488 students) to interact with the mini-quadcopter. For example, for tuning control algorithms to stabilize the mini-quadcopter
- Laboratory document instructions need to be clearly written to guide CPRE 488 students while working with the mini-quadcopter
- Demonstration of the mini-quadcopter's autonomous capabilities should be significant and show the team's technical abilities

Applicable Courses from Iowa State University Curriculum

- COM S 309, 319
- CPRE 288, 458, 488
- EE 333

New Skills/Knowledge acquired that was not taught in courses

- 3D Modeling Software
- Software Architecture of the Crazyflie Drone
- Operation of camera rig tracking system
- PID Controllers
- Socket Communication Protocols
- OpenVR API
- QT for GUI development
- Multi-Threaded, responsive graphical user interface

List of figures/tables/symbols/definitions

1. Team

1.1. Team Members

Brandon Cortez - Test Stand Lead
Reid Schneyer - Test Stand Sub-team
Colton Glick - Firmware Lead, Git Manager
Ellissa Peterson - Team Member
Ryan Hunt - System Architect
Carter Irlmeier - Web Manager, Lighthouse Sub-team
Zachary Eisele - Groundstation Lead, Co-System Architect

1.2. Required Skill Sets

- 3D Design & Printing
- GitLab code/issue management
- Software Architect
- Firmware Development
- GUI Frameworks
- PCB Design & Electronics Prototyping

1.3. Skill Sets covered by the Team

- 3D Design & Printing - Brandon, Reid
- GitLab code/issue management - Colton, Ellissa
- Software Architect - Colton, Zach
- Firmware Development - Ryan, Zach
- GUI Frameworks - Ellissa, Carter
- PCB Design & Electronics Prototyping - Reid, Brandon

2. Introduction

2.1. Problem Statement

CPRE 488 students need a functional drone system where they can write and test their own control logic for in-class labs to learn more about advanced embedded systems. We developed a platform to facilitate experimentation for these students. Additionally, we created impressive quadcopter demonstrations to show off to potential Iowa State students and industry contacts.

2.2. Requirements & Constraints

- Test stand must record and transmit rotational position data to a host computer
- GUI must display and graph drone position information
- Drone firmware must be modular so that control logic can be removed and substituted by student's code
- Must develop software to stabilize, and communicate with the mini-quadcopter.

- Must develop testing rigs to allow the team and users (CPRE 488 students) to interact with the mini-quadcopter. For example, tuning control algorithms to stabilize the mini-quadcopter

2.3. Engineering Standards

- Must be compatible with existing Crazyflie standards and systems
  - Low-level C on the quadcopter, and Python for client application
- CI/CD pipeline in git
- 3D printing guidelines
  - IEEE P3030
  - https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7308141
- Bluetooth/Radio communication standards
  - https://standards.ieee.org/standard/802_15_1-2002.html
- Crazy Real Time Protocol (CRTP)
  - Packet protocol used by Crazyflie
  - https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/functional-areas/crtp/

2.4. Intended Users and Uses

- Students taking CPRE 488 in the Spring 2022 semester will benefit from our project as the lab uses quadcopters as a teaching tool.
- Iowa State University is also a potential beneficiary, a demonstration of the drones will attract potential students and corporate representatives

3. Project Plan

3.1. Project Management/Tracking Procedures

Our team used a waterfall+agile management style. The nature of our project was such that we could immediately begin adding features to the Crazyflie, allowing an agile approach to work well. We planned to track our progress in the project through a GitLab kanban board. Tasks/issues would be created on the board as we began development, which would then be assigned to different members of the team. Additionally, milestones with due dates were created to make sure we kept a good pace as the project progressed.

3.2. Task Decomposition

Tasks Completed:

1. Investigate Crazyflie firmware architecture
   a. Learned how to modify and flash a new firmware to the Crazyflie
   b. What is the current architecture structure?
   c. Can the control code be easily modified?
   d. How easy is the control code to understand for new users?
2. Modified Crazyflie firmware to be as modular as possible

    a. Abstract the control code to a standardized interface to allow other control algorithms to be easily implemented through an adapter architecture
    b. Rewrite the existing control code to utilize the new interface
3. Write a basic PID control loop to maintain a stable hover, using the new interface
4. Develop ground station software to communicate with and control Crazyflie
    a. Start with a command-line interface on Linux
    b. Build a GUI/frontend once the backend is mostly working
5. Develop test stand hardware
    a. Determine what electronics will be used to record & communicate data
    b. Integrate chosen electronics into test stand for data collection
    c. Design and print test stand model to mount Crazyflie
6. Develop test stand software to measure and log rotation of the Crazyflie while held in test stand
    a. Should collect and record all desired data from the Crazyflie in real-time
    b. Should communicate with the ground station to allow for easy saving of log data
7. Write lab instructions and documentation for interfacing and using the modified Crazyflie
    a. Basic quick start guide
    b. Detailed proposed lab activities
8. Develop an demonstration that will be performed by the Crazyflie
    a. Research possible routes to take for demonstration
    b. Implement chosen route

3.3. Project Proposed Milestones, Metrics, and Evaluation Criteria

Project Milestones;

1. Working test stand prototype
    a. 3D printed and assembled
    b. Electronics selected and assembled
    c. Firmware is written and reporting back
2. Custom firmware running on Crazyflie
    a. First modified firmware running on Crazyflie
    b. Control software abstracted with adapter interface
    c. Existing control code running through adapter
3. Custom ground station CLI
    a. Control commands can be sent to Crazyflie and an acknowledgment is sent back
4. Ground station GUI based on CLI
    a. Basic GUI that sends pre-configured commands through the CLI

   b. More advanced GUI that displays flight data and allows for gamepad controls
5. Crazyflie flight demonstration
   a. Interacting with lighthouse system
   b. Running basic autonomous control script
   c. Complex autonomous drone demonstration

3.4. Project Timeline/Schedule

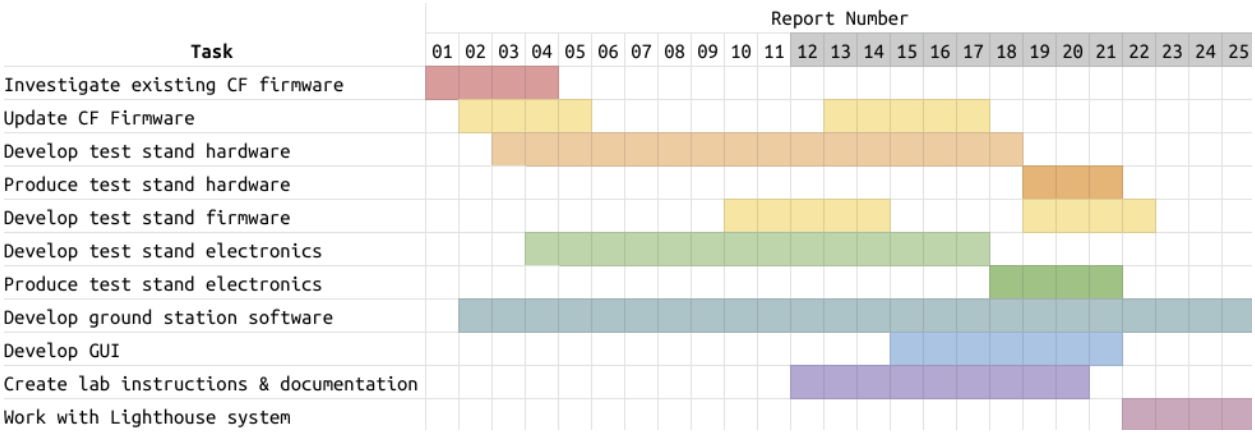


Table 1. Project plan Gantt chart of milestones and deliverables.

3.5. Risks And Risk Management/Mitigation

Risks:

1. Firmware is much harder to adapt to an adapter architecture, takes longer than expected, 60%
   a. Mitigation plan: Begin researching and modifying the Crazyflie firmware as soon as possible. May need to push back tasks that are dependent on this, the different control algorithms, and lab instructions.
2. Can't get voltage divider ratio to work for test stand electronics, 60%
   a. Use a 5v logic level microcontroller to read test stand data
3. CLI cannot communicate with Crazyflie, 30%
4. GUI takes too long to create, 40%

3.6. Personnel Effort Requirement

Task	Time Estimate (Hours)	Explanation
Investigate existing firmware	35	Depends on firmware complexity
Create control firmware	50	Depends on firmware complexity
Develop ground station software	200+	Create CLI & GUI for communicating with Crazyflie
Develop test stand hardware	32	Design & build test stand
Develop test stand software	30	Write firmware for test stand electronics
Write lab instructions	25	Create assignment for CPRE 488 lab
Create autonomous demonstration	40	Become familiar with system and implement

Table 2. Table of Personnel Effort Requirements by milestone

3.7. Other Resource Requirements

- Access to several Crazyflie drones to test and develop on
- Development computers running Linux to build and test the system on
- SIC access for 3D printing components of the test stand
- RC controllers
- Test Stand Control board components (see Bill Of Materials in appendix)

4. Design

4.1. Design Context

4.1.1. Broader Context

Our project has a fairly narrow context, all things considered. We're designing for Iowa State University students interested in learning more about complex embedded systems, specifically those taking CPRE 488 in the spring 2022 semester. The project addresses the societal need of needing skilled embedded programmers as more and more devices are created and manufactured.

Area	**Example**
Public health, safety, and welfare	The drone could cause minor injury if it collides with an individual
Global, cultural, and social	The drone will be used to teach engineering students more about advanced embedded systems and possibly inspire others to take interest in computer engineering
Environmental	The batteries the drones use are not great for the environment and can catch fire
Economic	An impressive drone demo could attract potential corporate or private donors, and potential future students

Table 3. Table of Broader Design Context

4.1.2. User Needs

CPRE 488 needs a functional drone system where they can write their own control logic for in-class labs to learn more about advanced embedded systems as well as a way to test and tune this control algorithm for table flight of the drone.

4.1.3. Prior Work/Solutions

We followed previous work that had been done for years, most recently by the 2020 MicroCART team. This is advantageous in that we have code and repositories (https://git.ece.iastate.edu/danc/MicroCART/-/tree/2020-team-final-state) to look at, but we definitely are operating a different project. We dealt with a much smaller version and different kind of drone than what has been used previously, which definitely differentiated our work from others.

4.1.4. Technical Complexity

The design consists of multiple components/subsystems that each utilize distinct scientific, mathematical, or engineering principles. These components/subsystems are:

- Groundstation
  - includes computer software skills
- Firmware flashed to the Crazyflie
  - includes computer engineering skills
  - Will utilize several popular software engineering principles such as Separation of Concerns, Modularity, Abstraction, and Incremental Development
- Test Stand
  - involves 3D modeling,
  - electrical engineering
  - embedded systems

4.2. Design Exploration

4.2.1. Design Decisions

- Wifi integration decision
- What exactly to do for the demonstration
- GUI layout
- What sensors will be embedded in the test stand

4.2.2. Ideation

We identified potential options for sensors needed through client and group requirement discussions and brainstorming:

- Sensors
  - Rotation
  - Vibration
  - Voltage
  - Gyroscope
  - IR

4.2.3. Decision-Making and Trade-Off

Our process for creating these pros and cons was through discussion with the client and group members to make sure we covered all areas of concern.

- Affordability (keeping in mind project budget),
- Availability (what is in stock and what the lab already has),
- What fits our design
- What works with our firmware

4.3. Proposed Design

- We have a working test stand that lets students measure roll, pitch, yaw and the rates for each and sends that information to the ground station
- We have a ground station that allows us to communicate with the drone to receive information from its sensors and send setpoints or parameters.
- We have a GUI for the ground station which allows students to graph the data from the sensors or from their own firmware.
- We have created our own PID controller for the Crazyflie and made a stripped down version for the students to write their own control algorithms
- We have written the lab for the students where we introduce them to our system then they tune their own PID values and write the firmware which includes the PID equations
- We have an autonomous demonstration working as well as a way to control the drone using a VIVE controller

4.3.1. Design Visual and Description

Test Stand Model:

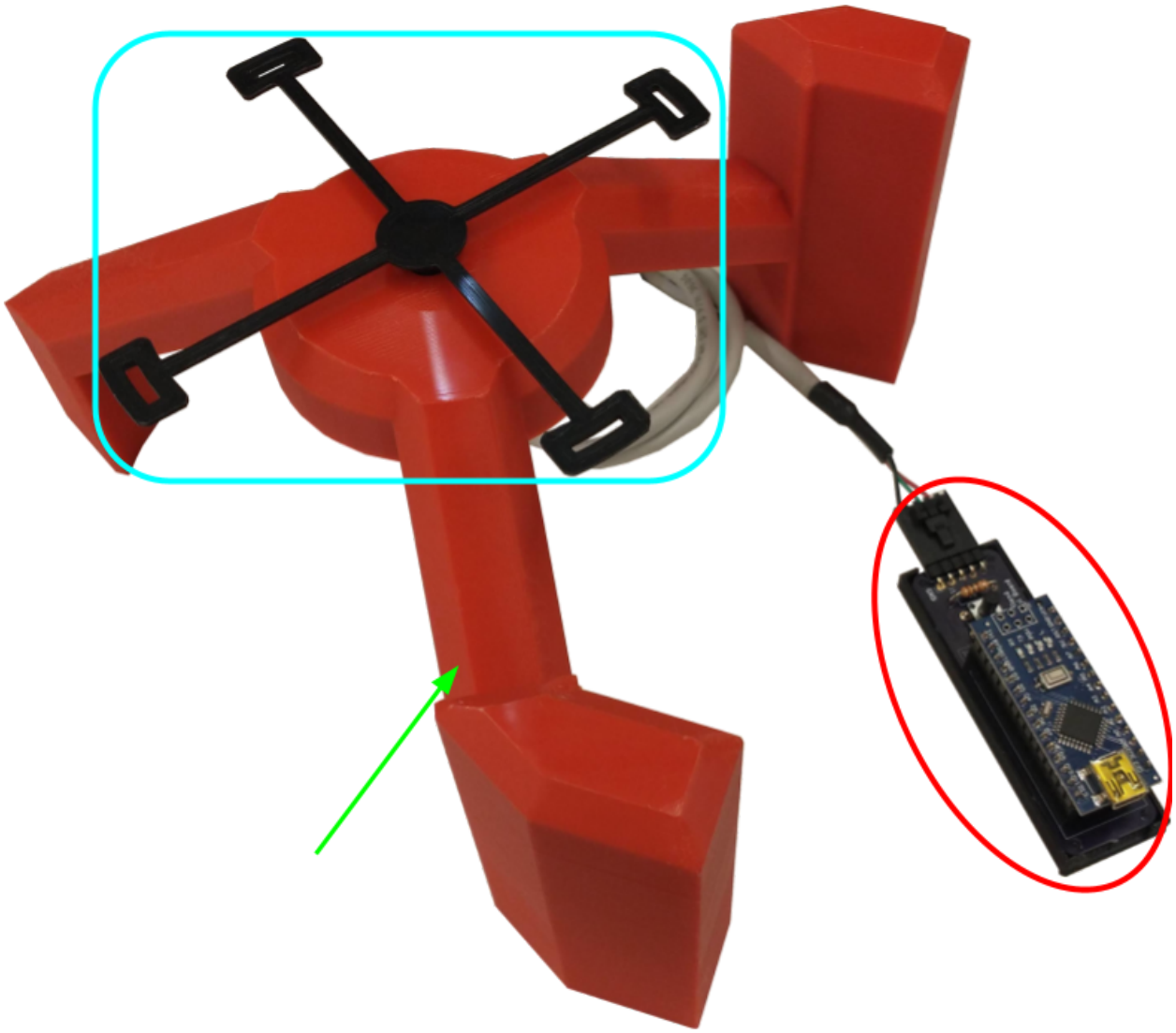


Figure 1. Crazyflie test stand and control board

The test stand assembly consists of 3 major components. Circled in red, the control board provides +5V and ground to the MA3 absolute rotary encoder mounted on the underside of the test stand. It reads the analog voltage value provided by the encoder, then translates that value into a rotational position, which is sent over USB serial to the ground station computer. The 6mm pushbutton on the control board serves a dual role. When held, it toggles the control board from reporting the rotational position to reporting the rotational rate, or vice versa. When in position mode, a short press of the button “zeros” the reading output, similar to the tare function on a digital scale.

The blue rounded rectangle shows the test stand quadcopter mount. This component has four holes that the plastic quadcopter legs fit into, and it holds the drone to the rotary encoder, preventing unintended liftoff. This component is fitted to the shaft of the MA3 absolute rotary encoder, and rotates the encoder shaft as the drone rotates.

The third major component of the test stand assembly is the test stand, indicated by the green arrow. The test stand houses the body of the MA3 absolute rotary encoder and allows the quadcopter to produce thrust and execute rotational maneuvers to test for yaw rate and position. The test stand is also designed to allow it to be rotated 90° and set on its side to allow the quadcopter to be tested on a different axis for pitch and roll values using a side-mounted quadcopter mount.

Figure 2. Crazyflie 2.0 mini quadcopter

The Crazyflie is a COTS open source development platform quadcopter. This is what the students tested their control algorithms on. We modified the firmware of the Crazyflie so that the PID controller was broken into modular parts and then stripped down for the students to fill out. We also created our own PID controllers from the modularized code to prove that it could be done.

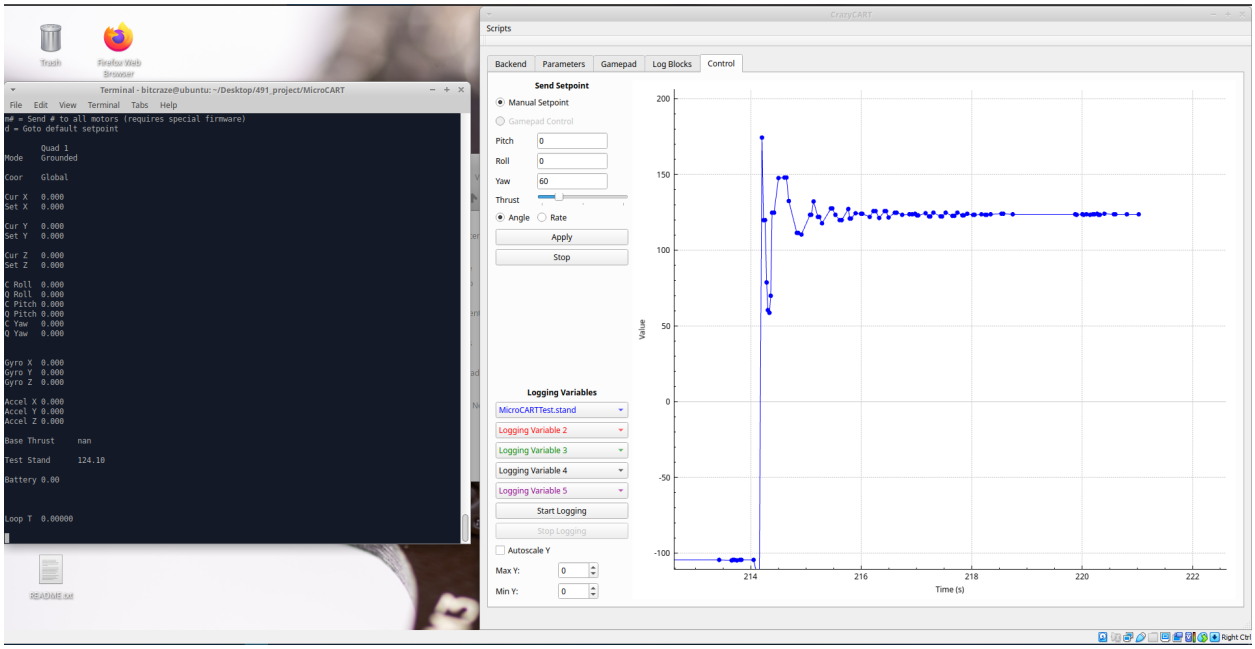


Figure 3. Groundstation and CrazyCart GUI

The groundstation backend was initially created by a MicroCart team a few years ago as a way to communicate with the larger drones they had via CLI commands, an adapter was made by a more recent team which utilizes the ground station but communicates with the Crazyflie. We made additions and updated outdated code to support the functionality that we wanted. We then created a GUI, that works with the backend, that would allow for easier means of communication with the drone and display the data that was being communicated.

4.3.2. Functionality

Our design lets students create and test their own control logic for the drone and gather data from the test stand as well as to show off when prospective students visit.

Our design fits the functional and non-functional requirements. Of course there could be improvements such as a few more features that would make students' lives easier when writing control algorithms and a few bugs that are documented. As this is a continuous project there will be ever changing and adding of requirements to make the best lab possible. As for this year's lab all of the requirements were met.

4.3.3. Areas of Concern and Development

Since we have completed our project and almost all of the students were able to complete the lab we met the needs and requirements there. Since

this is an ongoing project and there will be a team improving upon what we have created this year, the needs have shifted into needing documentation to integrate future teams into what we have done easily.

Our immediate plans are to document every part of our project in gitlab by giving instructions of how they were developed, how to continue development, and how to use what we have created. There are also a few students who have indicated that they want to work on the project next semester so we are in the process of showing them what we have made and giving them hands-on experience beforehand.

4.4. Technology Considerations

In working on our senior design project, we had to determine which type of protocol we would use to communicate between the ground station and the Crazyflie drone. The ground station that Bitcraze provides uses a radio antenna, but Professor Jones wanted us to look into using either Bluetooth or WiFi to communicate in our ground station software. While the Crazyflie does have an existing Bluetooth IC, we would have to use a separate board for WiFi communication, such as the ESP8266. Professor Jones' concern was that Bluetooth might have more latency than going with the Wifi option, since we could use socket communication over Wifi. We ended up sticking with radio communication as that was already implemented on the ground station and Crazyflie drone side.

4.5. Design Analysis

Yes, our proposed design worked as the students were able to complete the lab with all of the functionality that was necessary. We did get a lot of feedback from the students with what features could be added/improved upon. We have a list of these which we discussed with ourselves and with our client on which were the most important then implemented from there.

4.6. Design Plan

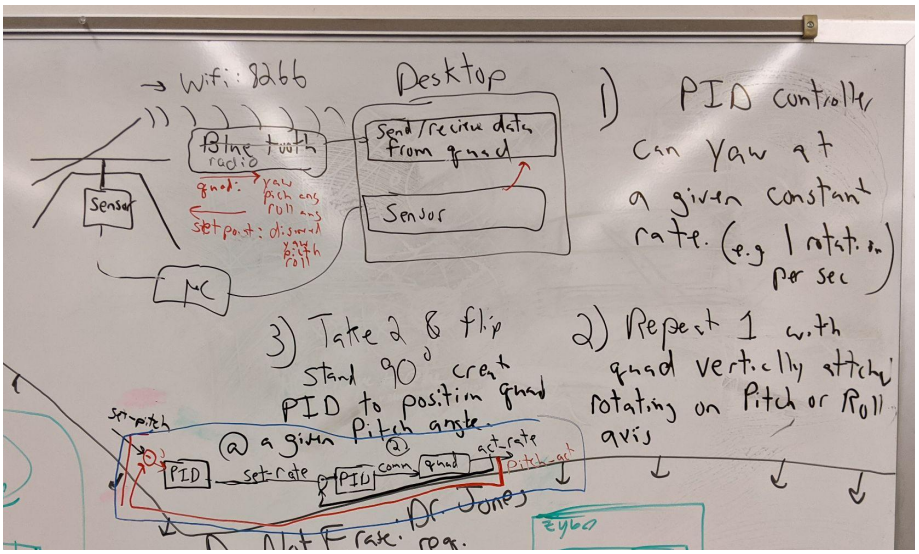


Figure 4. Module Interface Diagram

Our design plan is divided into three teams, the ground station, the test stand, and the firmware. I will first go over the individual requirements for each team then the requirements of all of the teams together. The plan for the ground station is to start with the previous year's MicroCART ground station as there was an adapter made for the Crazyflie last year, then improve upon it with any information that would be beneficial for students to use in the 488 lab. For the test stand the design plan is to first create a test stand similar to one created from a previous team then see what could be improved upon with it, make a new design with the previous flaws in mind. This will be continued till the design fits all of the needs. The next step would be adding sensors onto the stand so that students can get measurements from the Crazyflie quadcopter and create hardware that will display the sensor data. The design plan for firmware is to become familiar with the current firmware and figure out how the PID controls work, then implement our own PID controller so that it could then be stripped down into a template for 488 students. The plan for making these parts work together will be first setting up communication between the Crazyflie and ground station with data being sent and read both ways. The second part of communication would be the groundstation getting data from the test stand sensors and displaying that data. To begin, this data link will be through the Crazyflie radio. In future iterations we plan to switch to bluetooth or wifi or radio communication to standardize the system more.

5. Testing

5.1. Unit Testing

Quadcopter software:

- Any units inside of the quadcopter coding can be tested, this would include a variety of functions. An example would be a flying algorithm. This can be tested by making a testbench for the function and feeding it numbers that we would know the outcome for, then comparing to

expected output. This will eventually also be able to be tested on our own testing station where we can verify the actual response of the quadcopter from these equations. We also used the built in unit tests of the Crazyflie which test basic logging parameters verification as well as compilation error checking..

Test Stand:

- The unit testing for this part of the project would pertain to making sure we are getting correct data from the sensor. This can be done by measuring the values vs expected values of what the sensor is measuring.

GUI ground station:

- The unit testing for the GUI will be for it to display the proper information, this may include some calculations which would also be a part of the unit test. These would be tested by writing a testbench and making sure the values match the expected

5.2. Integration Testing

The integration in our design was between the GUI ground control and the test stand and between the quadcopter and the GUI ground control. We tested the interfacing with the GUI ground control and test stand by sending data from the test stand to the GUI and making sure it gets what was actually sent. We tested the communication between the quadcopter and the GUI by sending a command with the GUI and checking if the quadcopter responds accordingly.

5.3. System Testing

The system we are testing in our design is from the base station to the GUI then to the quadcopter. This was tested by first sending a command from the GUI to the quadcopter where then the test stand sensors recorded data from the movements of the quadcopter then sent that data to the GUI.

5.4. Regression Testing

When creating new additions we used version control in conjunction with testing to ensure that all bugs and issues are resolved before the master branch is updated.

5.5. Acceptance Testing

We demonstrated that our design met the requirements to our client by giving in-person demonstrations of our project progression. We involved our client, as well as the TAs, in this by guiding them through how our solution works first by individually showing them each part then how they all work together.

5.6. Results

The results of our testing showed that each part of the design works as stated in the requirements. The ultimate test of the students completing the lab was a success as almost every group was able to complete the lab. We also received

feedback of what students thought about the lab, which was relatively successful, this proves the usefulness of our project.

6. Implementation

6.1. Crazyflie Firmware

The Crazyflie provided an excellent existing base to work from. The code is published as an open source project by Bitcraze and is available on Github (see references at the end of the report). The firmware is written in low level C and allows for direct control over the Crazyflie's microcontroller. However, the firmware is also written in a modular manner that allowed our team to modify the control algorithms for the CPRE 488 students. Further work was done by our team to simplify the structure around the stabilization module to allow students to focus on the PID control algorithms. We simplified the stabilization by renaming variables and removing logic where it was no longer necessary. Additional work was done to implement firmware flashing over USB by automatically placing the quad into DFU mode when a special USB packet was sent to it. This feature was not fully utilized in the CPRE 488 lab but was contributed to the official Crazyflie open source project.

6.2. Test Stand Firmware

The test stand firmware is fairly simple, and is written using the Arduino IDE. It reads an analog value from the pin connected to the MA3 absolute rotary encoder, then scales the 0-1023 analog value to a -180 to 180 value corresponding to the degree position of the encoder. When the button on the PCB is pressed, it "zeros" the reading. When the button is long pressed, the firmware switches into rotation rate mode, which gets a moving average of the rate of the encoder rotation. Regardless of mode, the firmware then sends a single double precision value over serial every 100 milliseconds.

6.3. Ground station

The ground station consists of three parts: the MicroCART ground station, Crazyflie adapter, and the Crazyflie ground station. The MicroCART ground station was made by student's from previous years and was originally meant to communicate with custom quadcopter's they had made. This ground station needed to continue to be used so that in the future it could support multiple different types of quadcopters at the same time. In this project the main utility of the MicroCART ground station was command processing. The MicroCART ground station then sent commands to the adapter. The adapter was in charge of translating packets from the MicroCART configuration to the Crazyflie configuration. The Crazyflie ground station was the part of the system that communicated with the quadcopter itself. It would receive command packets from the adapter and send them to the Crazyflie, and receive logging and parameter information from the Crazyflie and store it for later use.

7. Professionalism

7.1. Project Specific Professional Responsibility Areas

Professional Responsibility	Project Application
Work Competence	**Why/why not?** - Our project will be used by future students thus it needs to be of high enough quality that it can be comprehended in the future to aid in education. **Performance(Medium):** We are continuously working to better our code but our documentation could be improved to help with future comprehension.
Financial Responsibility	**Why/why not?** - We are not buying or selling any products of significant value. Performance(N/A):
Communication Honesty	**Why/why not?** - We've had to communicate our progress to our contact on a weekly basis. **Performance(High):** We write up weekly status reports, as well as communicate frequently in discord and through email with our contact.
Health, Safety, Well-Being	**Why/why not?** - We do have potentially harmful equipment being used in our workroom. **Performance(High):** Safety protocols are in place and used during development and testing.
Property Ownership	**Why/why not?** - We use university-provided resources and technology in our development laboratory. **Performance(High):** We have been respectful of the lab & equipment, especially the sensitive drone electrical components.
Sustainability	**Why/why not?** -

	Sustainability is not a large part of our project as we use primarily small amounts of standard-grade plastic components. **Performance(N/A):** The scope of our project uses limited supplies and creates minimal waste.
Social Responsibility	**Why/why not? -** Our project does not have a high social responsibility because it is going to be used primarily only by future students. **Performance(High):** Although we do not have a high social responsibility, we are developing our project with the safety and education of the user in mind.

Table 4. Table of project-specific professional responsibilities

7.2. Most Applicable Professional Responsibility Area

Professional Responsibility:
Communication Honesty

Description, Demonstration, and Impacts:
For our project communication honesty consists of reporting our work to our client truthfully, transparently, and without deception. We demonstrated this responsibility by providing detailed reports to our client on a weekly basis that communicated our team's progress, what we achieved, what we are spending our time and resources on, and what our plans for the future are. Communicating clearly and honestly in this manner has allowed us to receive critical feedback from our client on our performance in order to improve our team's performance as the project progressed.

8. Closing Material

8.1. Conclusion

Our solution successfully fulfilled our client's product requirements. Our system allowed CPRE 488 students to write and test the controls of the Crazyflie drone from a ground station, while sensors in the test stand monitor the movement and behavior of the drone.
Some aspects of the project that we would have done differently include,

- Having a more in depth discussion with client about project requirements early on
- Allocating less time to research and more time to development
- Starting development on the ground station GUI earlier

- Using a web based GUI to improve performance and usability

8.2. Design Evolution since 491

At the end of last semester we had the following completed:

- Prototype of the test stand
- Prototyped control board on a breadboard
- Able to send rate setpoints via CLI from ground station
- Created a copy of existing controller as student controller in drone firmware

At the end of this semester we have completed:

- Test stand that allows for measurement of attitude and attitude rate
- Control board PCB that translate analog value to rotational position and rate
- Ability to get/set parameters, receive log data, send angle, rate and position setpoints from the ground station via CLI
- GUI that connects to the ground station and can perform the CLI commands as well as graph the logging data
- Created out own version of the firmware that was later stripped down and used as a template for students to write their own control logic
- Created multiple demonstrations with the drones ranging from autonomous flight to flight controlled by vive controller

8.3. References

US Digital, “MA3 Miniature Absolute Magnetic Shaft Encoder”, MA3 datasheet, Jun. 2021 [Accessed 06-Dec-2021].

“Bitcraze,” *GitHub*. [Online]. Available: https://github.com/bitcraze. [Accessed: 06-Dec-2021].

“Distributed Autonomous Networked Control Lab / microcart,” *GitLab*. [Online]. Available: https://git.ece.iastate.edu/danc/MicroCART. [Accessed: 06-Dec-2021].

“Start here,” *Bitcraze*. [Online]. Available: https://www.bitcraze.io/documentation/start/. [Accessed: 06-Dec-2021].

“CRTP - Communication with the Crazyflie,” *Bitcraze*. [Online]. Available: https://www.bitcraze.io/documentation/repository/crazyflie-firmware/2021.06/functional-areas/crtp/. [Accessed: 06-Dec-2021].

9. Appendices

9.1.1. Operation Manual

You can find documentation for our project in our gitlab wiki here:

https://git.ece.iastate.edu/danc/MicroCART/-/wikis/home
We have also included the instructional lab document located at the end of this report.

9.1.2. Alternate Versions

- **WiFi communication**: During planning, our client initially wanted the Crazyflie to communicate over WiFi using TCP or UDP sockets. This would have standardized the communication protocol and allow the Crazyflie quadcopters to be operated without the need of the Crazyradio. This would require an additional board to allow for WiFi communication and was theoretically possible. However, we underestimated the amount of work that would be required in implementing WiFi communication and did not have enough time to include it. It was dropped in favor of more critical features for the CprE 488 lab.
- **Bluetooth communication**: Similarly to WiFi communication, our client wanted to use bluetooth communication to control the Crazyflie quadcopter. The Crazyflie already has the capability to communicate over bluetooth with a smartphone, However, again we underestimated the complexity of the system. The communication within the Crazyflie firmware was far more complex than initially anticipated. Thus we had to drop this in favor of more critical features for the CprE 488 lab. Our final design utilized the already tried and tested Crazyradio for communication.
- **OptiTrack System**: Coover 3050 contains a 12 camera, ceiling mounted OptiTrack system. This is an optical motion capture system that uses IR markers to detect objects within its range. The system in 3050 is outdated, as is the software that they use. “Tracking Tools”, from 2013, is what is available, while “Motive” is the currently supported application for tracking. This application is upwards of $1000 just for the license, while the alternative Lighthouse Positioning System by Bitcraze is around $500 for a full setup. Because Bitcraze makes the Crazyflies as well, the lighthouse system is extremely well integrated and documented. This newer system is also portable, meaning it can be moved and demonstrated almost anywhere, unlike our OptiTrack system.

9.1.3. Other Considerations

Test Stand Control Board Bill of Materials

Item	Quantity per board	Total Quantity
Arduino Nano	1	12
100 ohm resistor	1	12
6mm push-button	1	12
5 pin 2.54mm 90 deg header pin	1	12
15 socket 2.54mm header pin socket	2	24
Test Stand Control Board PCB	1	12
Miniature absolute magnetic shaft encoder	1	12

9.1.4. Code

The code for our project can be found on our Gitlab repository from the link below:
https://git.ece.iastate.edu/danc/MicroCART/-/tags A new tag will be created for the final state of our repository called `2022-team-final-state`, if this tag does not exist yet, the current state of the master branch is our project code.
The code consists of the three main sections, the Crazyflie firmware, the ground station software, and the test stand software. They are located in /crazyflie_software/crazyflie-firmware-2021.06, /groundStation/, and /test_stand_firmware respectively.

CPRE 488

MP-4

UAV Control
Document Version 1.1

Table of Contents

Introduction

A MAN HAS FALLEN INTO THE RIVER IN LEGO CITY!
Start the new rescue quadcopter!
HEY!
Program and tune the quadcopter, and off to the rescue!
Prepare the lifeline, lower the stretcher, and make the rescue!
The Crazyflie collection from MicroCART!

Say hello to your Crazyflie drone! The goal at the end of this lab is to be able to smoothly control the Crazyflie with the algorithms that you will write and test. First, we will give you a brief overview of the Crazyflie system, as well as how to get set up with development for this lab.

Basic Crazyflie operation

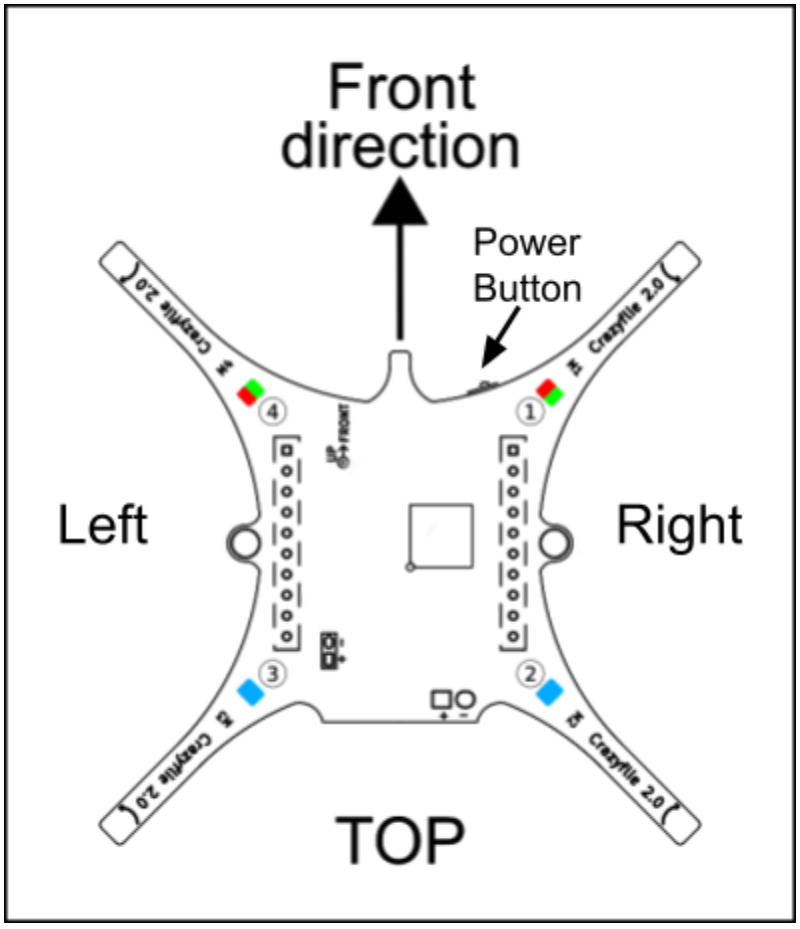


Figure 1. Crazyflie top down diagram

There is only one physical button on the Crazyflie, the power button. To start the Crazyflie:

1. Plug in a charged battery
2. Press the power button located near the front right arm of the drone

3. Wait for all four props to do a short spin and the startup tone to play.
4. Place the Crazyflie on a **flat surface to allow its sensors to calibrate**. This is indicated by the flashing red LED. If it flashes quickly, then the sensors have been calibrated properly and the drone is ready to fly. If the LED flashes slowly, then the sensors have not been calibrated yet. If some hardware is damaged on the Crazyflie it may fail to pass its self check on startup. This is indicated by the red led flashing quickly 5 times. In this case the hardware may be inoperable. If this is the case, notify a TA or the instructor.

Crazyflie LED Codes

One of the main ways the Crazyflie communicates its status is with the four LEDs mounted to the surface.

LED code	Meaning
2 solid BLUE	All normal, indicates the back of the Crazyflie
2 Slow flashing BLUE (1 hz)	Crazyflie is in bootloader mode and is ready to be flashed by radio
1 Fast flashing BLUE (2 hz)	Crazyflie is in DFU mode and is ready to be flashed by USB
Back left BLUE flashing	Charging while plugged into USB. Percentage of time LED is on indicates battery level.
1 slow flashing RED (0.5 hz)	Crazyflie is on but sensors are not calibrated. Place on a flat surface and keep still to calibrate
1 fast flashing RED (2 hz)	Sensors are calibrated and ready to fly
5 short RED pulses followed by a gap	Self test failed, hardware may be damaged, notify a TA or the instructor
1 solid RED	Low battery
5 short GREEN pulses	Self test passed, all normal

Crazyflie System Overview

This is the complete Crazyflie control system. You will only be modifying a small portion of it, but it will be helpful to understand the full scope of the system you are interfacing with.

The control process starts with the state estimator module receiving sensor data and using it to calculate the drone's current attitude (its rotation, i.e. roll, pitch, and yaw). The state estimator then sends the calculated attitude to the state controller module, which also receives a setpoint from the commander module (in our case, this is user input specifying the desired attitude or

attitude rate and thrust). The state controller module contains a cascading PID controller that uses the inputs from the state estimator to calculate the actuation force needed. That is then sent to the power distribution module where the actuation force is converted to motor power then the loop starts over again.

You will be implementing the State Controller's cascading PID in part 2 of this lab.

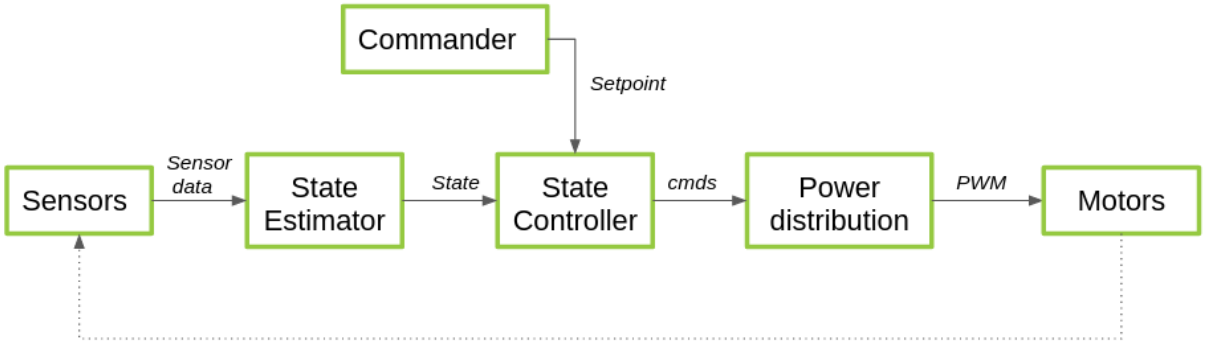


Figure 2. Crazyflie control diagram

The Crazyflie runs off of a cascaded PID system where the output of the first PID controller is then used as an input for a second PID controller. This layout can be seen in figure 3, the output from the attitude PID controller, the desired attitude rate, becomes the input of the attitude rate PID controller. In part 2, you will be implementing the attitude and attitude rate PID controllers for roll, pitch, and yaw.

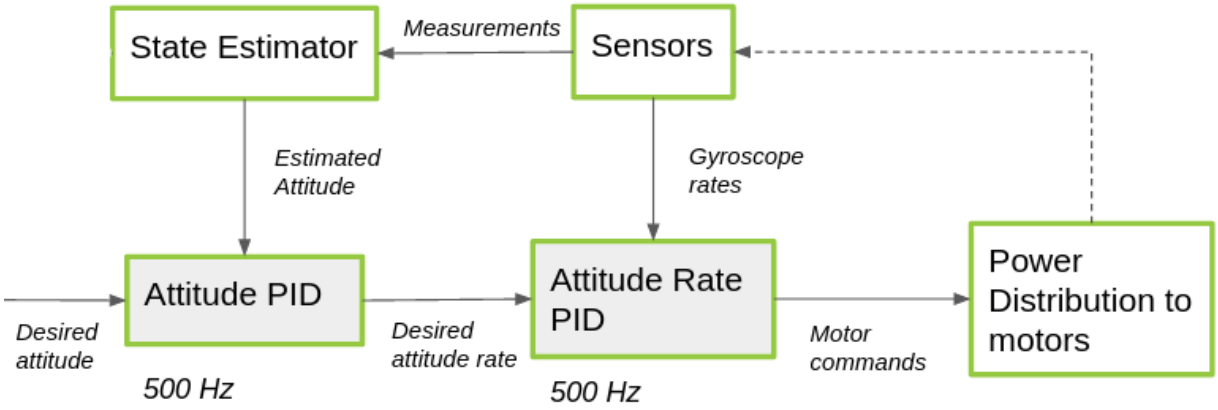


Figure 3. Cascading PID diagram

Virtual Machine Details

The virtual machine has been configured to have the necessary utilities to develop the Crazyflie firmware. Below are detailed instructions for completing different tasks within the virtual machine. Login username is **bitcraze** and password is **crazyflie**

Importing the Virtual Machine Instance

New users will not be able to see the VM instance until it is imported.

1. Open Virtual box application
2. Go to machine → add
3. Navigate to the VM installation, C:\cpre488\mp-4\
4. Open the .vbox file
5. The virtual machine should now be available

Folder Sharing with the Virtual Machine

1. Go to VM settings, machine → settings then shared folders
2. On the far right click the blue folder with a green cross to add a new shared folder
3. Set the folder path to a folder on the windows system to share with the VM
4. Select Auto-mount
5. For the mount point enter '/home/bitcraze/shared-folder' or some other specified mount point
6. Select make permanent
    a. This option is only available while the vm is running
7. Select ok on both dialog boxes to complete the shared folder setup
8. You should now be able to navigate to /home/bitcraze/shared-folder in the VM and see the files in your shared folder

Exporting Code From the Virtual Machine

Due to the read only VM image, **you must export your work from the VM before shutting down**. The VM will be reset on reboot and all changes will be reverted.

There are two options to maintain and export your changes from the virtual machine. The first is to use a local git repository stored on your x drive or a usb flash drive. This minimizes network usage. The second option is to use a standard GitHub or GitLab repository. This option is best for working with other people in your group at the same time. If you do use the second option we ask you make your repo private.

With a bare git repository on the host machine

1. Create a bare repository on your x drive or a removable media device
    a. 'git init --bare my_repo_name.git'
    b. This will create a **folder** called my_repo_name.git with no working tree.
2. Share this folder with the VM and mount it at **'/home/bitcraze/transfer-repo.git'**, see details on sharing a folder with the VM
3. Once the bare repository is accessible from within the VM, the Microcart repository should already be setup to use '`/home/bitcraze/transfer-repo.git`' as a remote, check this with '`git remote -v`'
4. Commit and push your changes to your shared folder
    a. If this fails try setting the remote url again with '`git remote set-url origin <absolute path to shared folder>`'

5. Your changes will now be on the bare repo in your shared folder, these changes can now be reapplied later
    a. This can be verified by navigating to '`/home/bitcraze/transfer-repo.git`' and running `git log Lab_Part_2`

With normal GitHub/GitLab repository

1. Note: the first commit may take a while as all the git history must be uploaded
2. Create a new **private** blank repo on github or gitlab, obtain the url or ssh address to the repo
3. In the Lab_Part_* folder run 'git remote set-url origin <address of remote repo>'
    a. Note this set-url command must be run each time the VM is rebooted due to the immutable hard drive setup
4. Commit and push your changes as normal

Importing Work Into the Virtual Machine

With a bare git repository on the host machine

1. Ensure the bare repository is accessible within the VM, see details on sharing a folder with the VM, mount it at **'/home/bitcraze/transfer-repo.git'**
2. The Microcart repo should already be setup to use **'/home/bitcraze/transfer-repo.git'** as a remote
3. From within the Lab_Part_* folder run 'git pull'

With normal GitHub/GitLab repository

1. Set the remote repository with 'git remote set-url origin <address of remote repo>'
2. From within the Lab_Part_* folder run 'git pull'

Final export from the virtual machine

Final export will copy all files that have been modified since the original state of the Microcart repo.

1. Ensure folder sharing and your git repository are setup correctly
2. Commit all changes within the git repository to be exported
3. From the **root of the Lab_Part_2** folder, Run '`cp -pv --parents $(git diff Lab_Part_2_tag --name-only) <DESTINATION-DIRECTORY>`'
    a. The destination should be a shared folder **other than your transfer repo** so the files are accessible from the host machine
    b. This will copy all modified files since the Lab_Part_2_tag into the destination directory

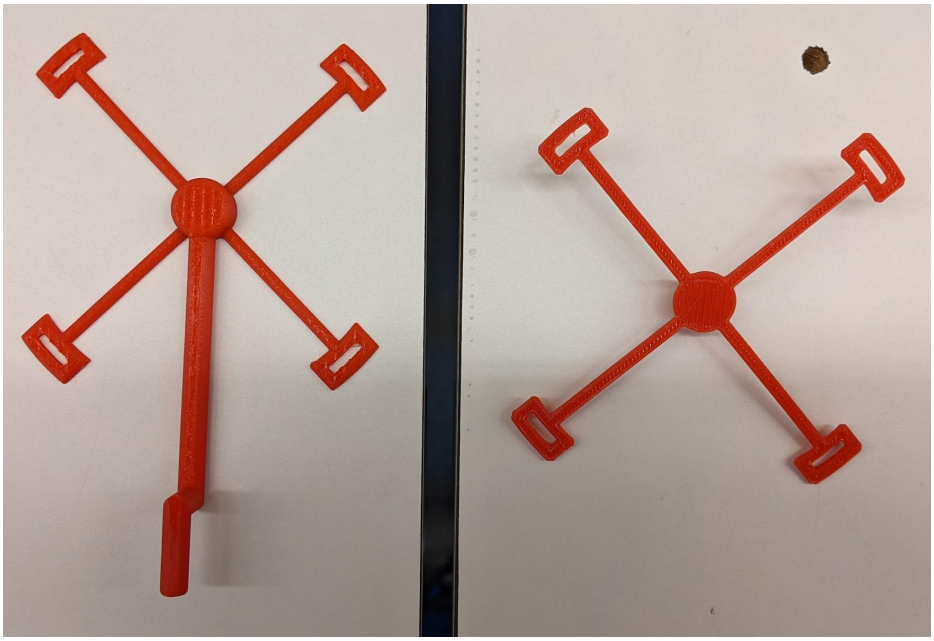
Test Stand Details

Test Stand Components

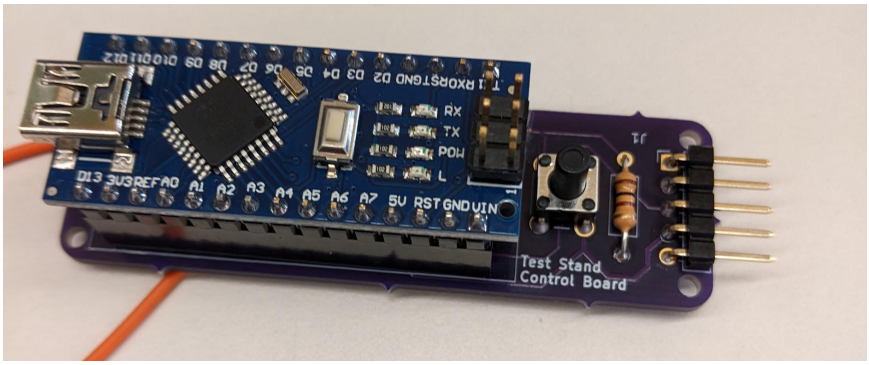
The test stand for MP-4 is a bit more complex than the ones used in MP-1, and consists of three major components, plus a couple wires to connect them. The test stand base holds the rotary

encoder used to measure position, and can be used in two different configurations depending on the drone orientation needed.

The test stand mount attaches to the encoder shaft, and holds the Crazyflie drone in place through friction. There are two different mounts, which can hold the drone in either a horizontal or vertical position.



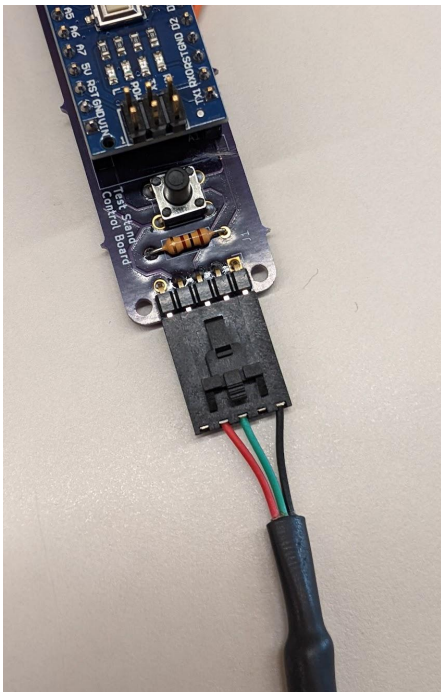
Finally, the test stand control board reads the rotary encoder value, and transmits either positional or rotation rate data to the computer.



Basic Setup

1. Start the Crazyflie drone on a flat surface before you attach it to the desired mount
2. Attach the drone to the mount by inserting each pair of clear plastic legs to a corresponding slot in the mount. If you're using the vertical drone mount, make sure that the drone is mounted appropriately to measure either pitch or roll
3. Once the drone is mounted, insert the shaft of the mount into the hole on the top of the test stand base, making sure it fits snugly and doesn't rub against the sides.

4. Plug the small three pin end of the gray cable into the rotary encoder. While you are able to only insert the plug in the correct orientation, please do not force the plug if you are met with resistance. Both the cable and the rotary encoder are fairly expensive, and we only have so many replacements
5. Plug the other end of the gray cable into the 5 pin connector on the test stand control board. Make sure to pay attention to the orientation of the connector, since this end of the cable can be plugged in backwards, although doing so shouldn't harm anything. Make sure that the black wire on the cable connects to the connector pin with the square solder pad.



6. Finally, connect the Arduino Nano to your PC using a mini USB cable.

Using the Test Stand

The test stand has two modes, where it reports either positional data or the rotation rate (in deg/sec). The LED labeled "L" on the Arduino Nano is used as a mode indicator, and is **on when the control board is in position mode, and off when in rate mode**. You can switch between modes by **pressing and holding the black pushbutton mounted to the control board PCB**. Additionally, the black pushbutton is used in positional mode to zero the reported reading (sort of like "tare" on a digital scale). **Short pressing the button while in positional mode will reset the reading to treat the drone's current position as zero**. While you will be able to see the reported data in the GUI, you can read the data reported by the controller by connecting to its COM port with PuTTY (or similar) at a baud rate of 9600.

Ground Station, Graphical User Interface (GUI)

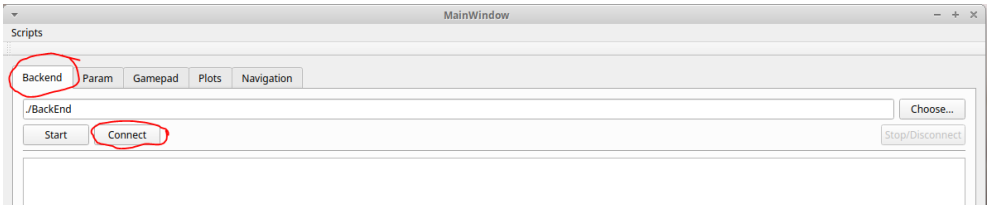
The ground station software is what you will mainly be using to communicate with the Crazyflie. It has been pre installed and set up on the virtual machine for this lab.

To connect to a Crazyflie and open the GUI, make sure the Crazyflie is powered on and the crazyradio is available in the vm, then run the command **crazycart <radio channel of Crazyflie>**. If everything connects successfully, the GUI will open. Below are some details on how to perform tasks in the GUI.

Connect to the backend

Connects the GUI to the backend, you will need to do this every time you start the GUI

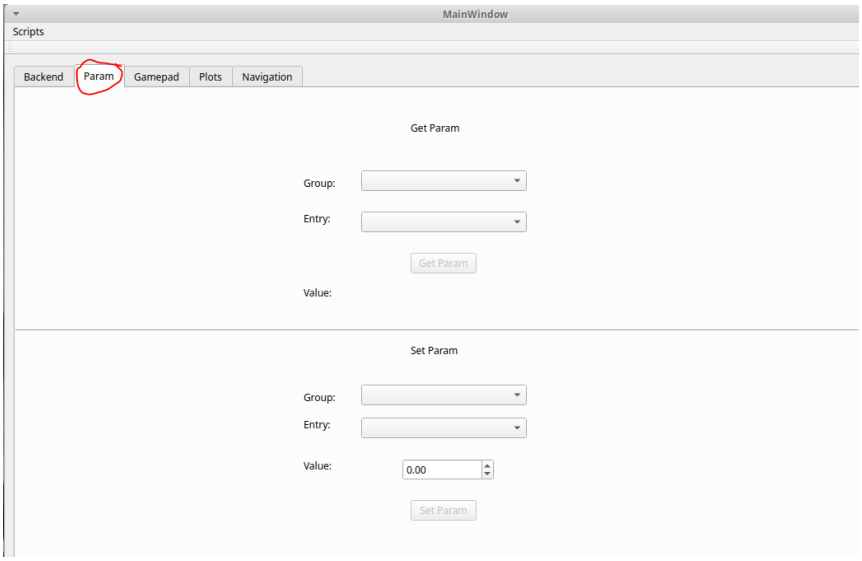
1. Navigate to the backend tab
2. Make sure the text box says ./BackEnd
3. Click connect



Get/Set Parameters

Lets you view and set parameters of the Crazyflie

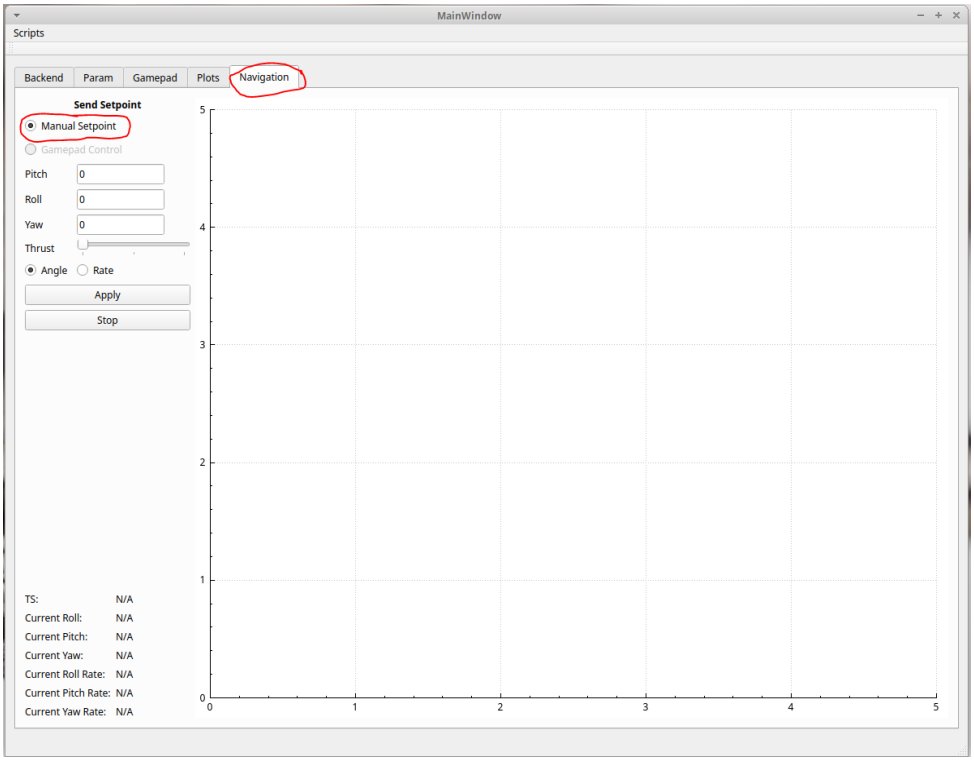
1. After connecting to the backend, navigate to the param tab
2. The top half lets you view the parameter values of the Crazyflie while the bottom half lets you set those same parameters
3. You first have to select the group that the parameter is part of then you can select a specific parameter in that group



Sending setpoints

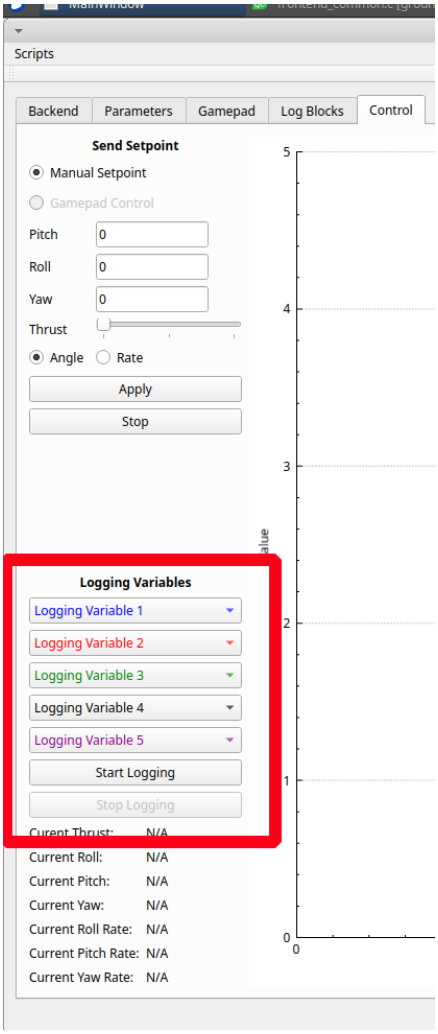
Sends setpoints to the drone

1. Navigate to the control tab
2. Make sure manual setpoint is selected
3. You can enter desired pitch/roll/yaw and slide the thrust in the boxes below
4. You then chose if you want to send a rate or angle setpoint then click apply to send it
5. Clicking stop will send a stop setpoint (all inputs 0) to the drone



Graphing Log Variables

1. See Adding New Logging Variables if needed
2. With the crazyflie connected, navigate to the Control tab
3. In the side bar you can specify up to 5 variables from **active** logging blocks to plot on the right

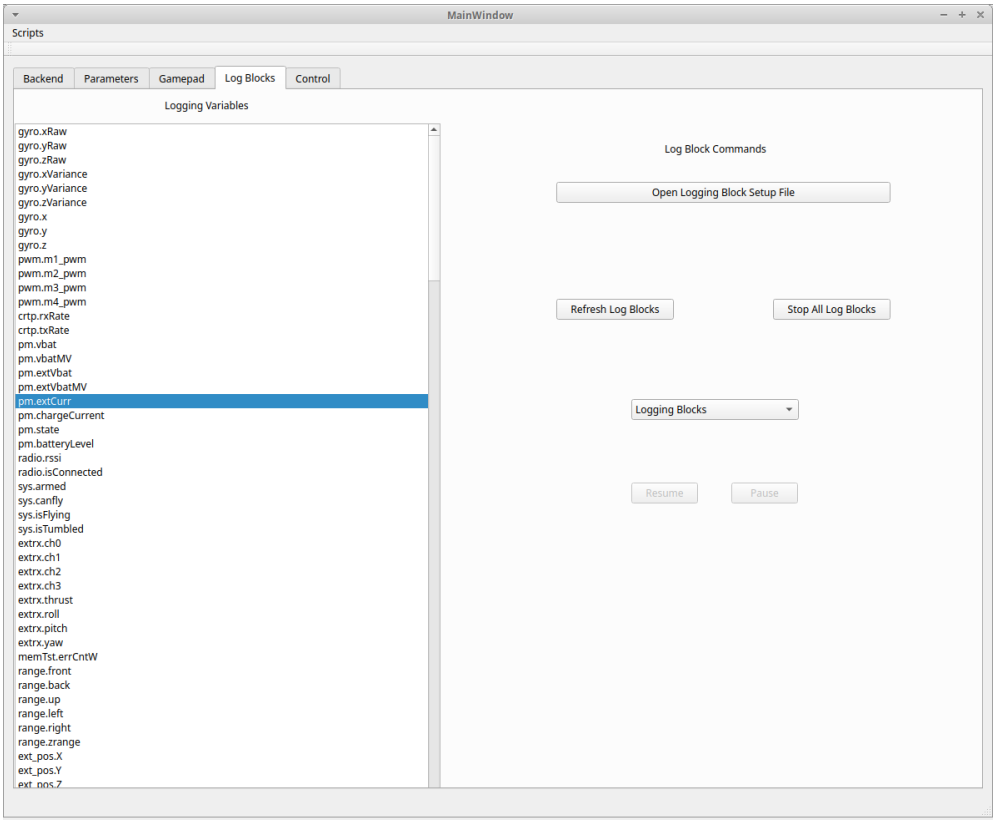


4. Once variables are selected click start logging to capturing data
5. Press the stop logging button when you are done capturing data.
6. The data will be displayed in close to real time. After plotting has stopped you can zoom in and move around the graph.
    a. If nothing is plotted after clicking start logging, refreshing the log block can help, see Adding New Logging Variables

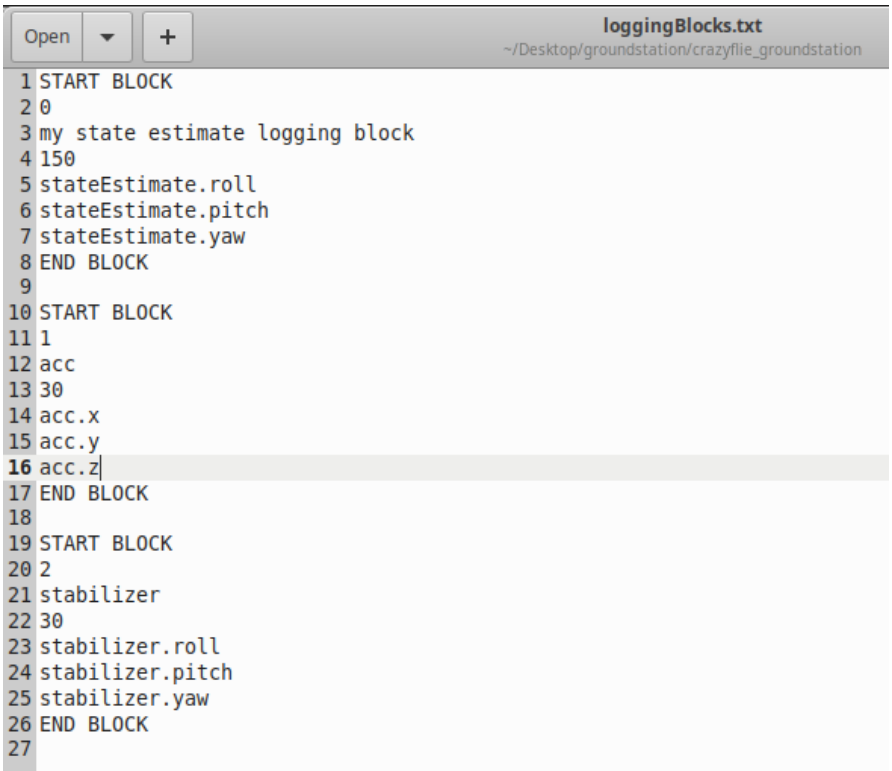
Adding New Logging Variables

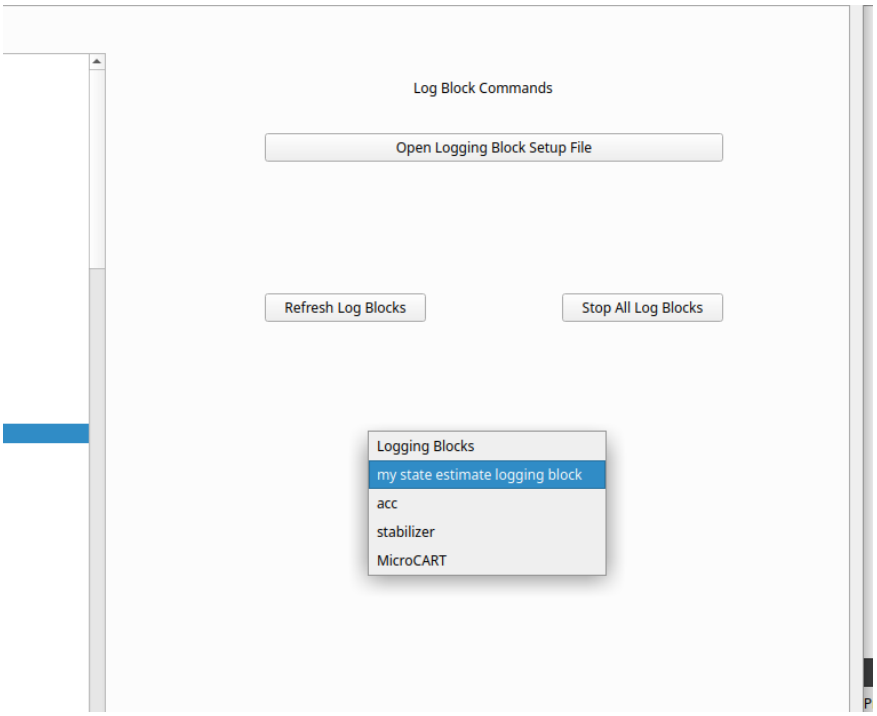
The crazyflie uses what are called “Log Blocks” to define what variables are logged and sent to the ground station. A log block specifies what values to send and at what rate. Log blocks can be paused and resumed after initial setup. Note the crazyflie communication has a limited bandwidth, **only enable around 10 logging variables at a time.**

1. Navigate to the **Log Blocks** tab



2. The list on the left shows all available logging variables from the Crazyflie firmware
3. To view and modify what logging variables are sent, click the “Open Logging Block Setup File”



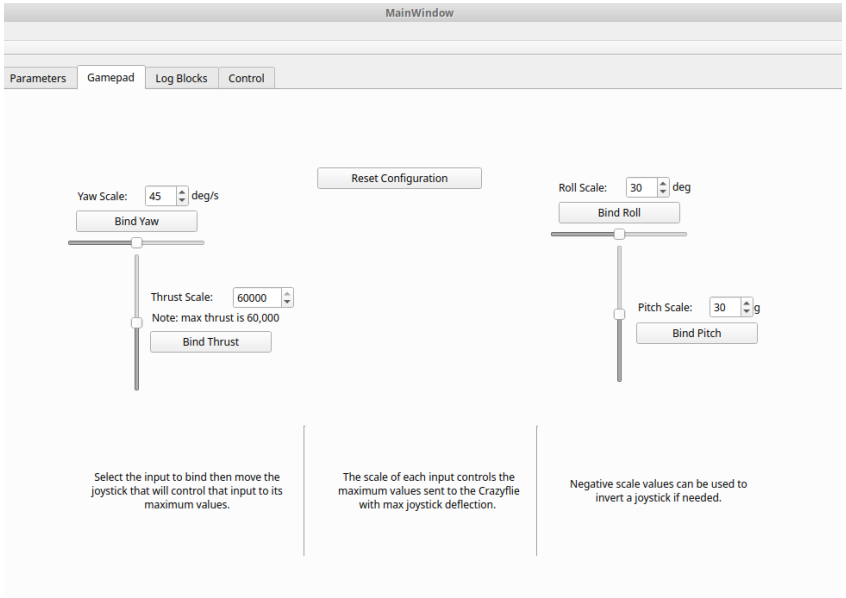
b. The logging blocks have a defined format that must be followed
    i. A block definition starts with `START BLOCK`
    ii. The next line is the **logging block integer id**, it must be **unique** to other logging blocks defined
    iii. The next line is the **name** of the logging block, this can be any string you'd like
    iv. The next line is an integer defining the **rate** of the logging block, a higher number pushes more data to the ground station. **Note, too high of a rate can overload the graphing visualization and cause the GUI to slow down.** Safe values are **below 100**.
    v. The following lines define the **variables** that are in the logging block, these are defined by the firmware and can be viewed in the large list mentioned above. Any number of variables can be in the log block and they do not have to be from the same logging group.
    vi. A block definition ends with `END BLOCK`
    vii. Any lines that are outside of a block definition are ignored
7. Save and close the logging block definition file
8. Click **Refresh Log Blocks** to send the new definition file to the crazyflie
9. The drop down on the right should now show your new logging block and will be active
    a. 

10. If needed, you can increase available bandwidth during logging by pausing all other log blocks, selecting your new one from the drop down, and resuming it.
    a. The stop all log blocks removes the log blocks from the Crazyflie. You will need to pause each log block one by one

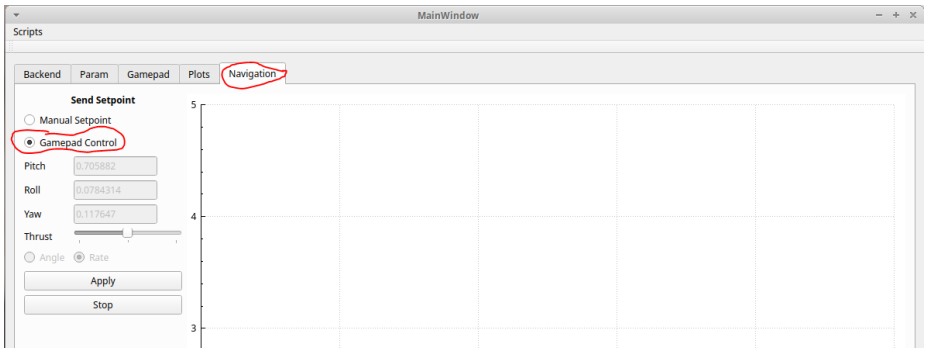
Gamepad Control

Control Crazyflie with gamepad

1. Be sure to pass the usb gamepad to the VM by clicking the USB symbol in the bottom right of the window and selecting the gamepad
2. Navigate to the Gamepad tab
3. The provided gamepad should be configured already, if not you can click the configure tab then move the joystick to the max and min value to calibrate it



4. Navigate to the control tab and select the gamepad control button after you have plugged in your controller. The ground station will immediately start sending setpoints to the crazyflie.



Command Line Interface (CLI)

The CLI is the base of the communication with the Crazyflie drone. It can optionally be used for basic tasks. You shouldn't need to interact with the CLI during the course of the lab, but it can be useful for debugging if something goes wrong. It can be opened by adding a **"nogui"** flag to the end of the crazyCART script. `crazycart <radio channel> nogui`

Run commands with ./Cli <command> <parameters>...

Further usage details can be found by appending `--help` to the end of a command. The following commands are currently implemented.

Command	**Description**
./Cli outputoverride <enable> <Time> <Throttle> <Pitch> <Roll> <Yaw>	output override will send a setpoint that lasts a set amount of time with the specified throttle, roll pitch and yaw. With enable set to 1 it will send the setpoint as a rate and 2 will send the setpoint as an angle
./Cli getparam <block_id|'block_name'> <param_id|'param_name'>	Get param will get the value of specified param. Note only the param id is used for this command. The param id is found in the logging TOC, use `getlogfile 1` command to find this file
./Cli setparam <block_id|'block_name'> <param_id|'param_name'> <value>	Set param will set the value of the specified param
./Cli getlogfile <id>	Get log file will get a certain log specified by the id of 0: data log 1: param id 2: logging toc
./Cli logblockcommand <id>	The log block command performs specific tasks on log files for the specified id of 0: delete all log blocks 1: refresh all log blocks 2: load log blocks 3: delete log block 4: resumelog block 5: pause log block

Part 1: PID Tuning

In the first part of this lab you will be tuning PID values of the default controller through experimentation. This will be done through the ground station GUI which will display information from the test stand.

You will be provided a semi-working controller that you will need to flash onto the Crazyflie. This file can be found in the **Lab_Part_1** folder on the desktop. From there follow the directions in the Flashing Crazyflie section. This controller works except that all PID constants have been set to 0, you will view and set these constants from the ground station GUI. **Important note: The PID constants will be reset to 0 when the Crazyflie reboots!** Be sure to copy down your current values as you work to avoid data loss.

A note when tuning the roll and pitch axes, be sure to **set the e_stop parameter under the sys group to 0** during testing. By default the Crazyflie will kill motor power if it detects it is tumbling (these tests trigger a false positive). The Crazyflie must be rebooted if this happens.

Getting Set Up for Part 1

Basic Setup

1. Setup the virtual machine environment
    a. Import virtual machine
2. Open virtual box VM
    a. Login username is **bitcraze** and password is **crazyflie**
3. Plug in the Crazyradio into the usb port
4. In the bottom right corner of the VM there is a USB icon, make sure the Bitcraze Crazyradio is selected under this menu
    a. This can be finicky, may have to physically reconnect the radio a couple of times for it to connect successfully to the VM
5. Turn on drone by pressing the button located on the front of the drone
6. Get the radio channel for your drone from the provided spreadsheet
    a. Crazyflie Status Sheet
7. Flash the drone with the **pre-compiled firmware** that has the no PID values set from the Lab_Part_1 folder. See the Flashing the Crazyflie section for details
8. Place the drone on a level surface (not the test stand) and allow it to calibrate
    a. See LED codes for more details
9. Connect to the crazyflie by running `crazycart <radio channel>`
10. Connect the GUI to the drone via the backend tab
11. To make sure you are connected to the drone send a low thrust setpoint with no roll, pitch, or yaw setpoint, the motors should spin up
12. Once ready proceed to Part 1: PID Tuning

Flashing the Crazyflie

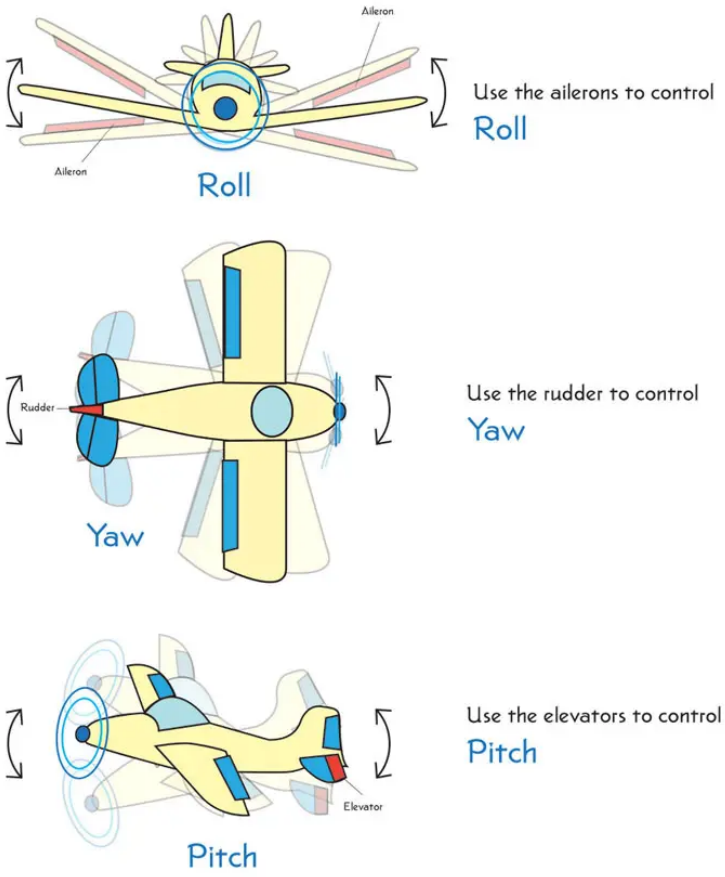
1. Navigate to the Crazyflie firmware folder, ex: '~/Desktop/Lab_Part_1/crazyflie_software/crazyflie-firmware-2021.06/'
2. Plug in the Crazyradio into the usb port

3. In the bottom right corner of the VM there is a USB icon, make sure the Bitcraze Crazyradio is selected under this menu
    a. This can be finicky, may have to physically reconnect the radio a couple of times for it to connect successfully to the VM
4. Create the file '**crazyflie_software/crazyflie-firmware-2021.06/tools/make/config.mk**' if it doesn't already exist and open with VS code
5. Add `CLOAD_CMDS = -w radio://0/<radio_channel>/2M` to the file and replace `<radio_channel>` with your Crazyflie's radio channel
    a. Your radio channel can be found in this spreadsheet Crazyflie Status Sheet
    b. This is a one time process and should not have to be done for any subsequent flashes unless you change which drone you are using or what folder you are flashing from
6. Make sure the Crazyflie is powered on and running
7. For Part 2 of the lab, compile the Crazyflie firmware with `make CONTROLLER="Student"` from the root of the Crazyflie firmware
8. Then run `make cload` to begin flashing the compiled firmware to the Crazyflie specified earlier.

Attitude Rate Control

We will begin by tuning the attitude rate controller. This controls the rate of rotation of the yaw pitch and roll.

Tip: When tuning your rate controller, allow for a "looser" control. IE the percent overshoot and settling time can be a bit larger than typically desired. This allows us to tune the attitude controller "tighter" later. It is very difficult to get both controllers tight and is not recommended.



https://highsierrapilots.club/tahoe-minden-reno-discovery-flight/roll-pitch-yaw-diagram/

1.1 Yaw Rate

First you will learn how to measure yaw rate with the test stand. Put the test stand so that it is standing with the three legs on the ground with the attachment that will hold the drone parallel to the ground, as figure 4 shows.

Figure 4. Yaw rate test stand setup

Now, connect to the Crazyflie and open the groundstation GUI. We will be graphing the test stand rotation rate and the yaw attitude rate setpoint. **Details on how to use the test stand and how to setup plotting can be found above**.

You can now send **yaw rate setpoints and thrust setpoints** via the ground station which will tell the Crazyflie to rotate at a certain speed, the test stand sensor will then measure the actual rate and display that on the GUI. However, with no PID constants set, the Crazyflie will not respond to setpoints. Your task is to change these constants through the GUI by setting parameters.

Relevant logging variables:

- MicroCART.Test_stand
    - This is the test stand data
- ctrlStdnt.yawRate
    - This is the yaw rate setpoint
- Optional
    - ctrlStdnt.r_yaw
        - This is the crazyflie's on-board sensor for yaw rate measurements

Be sure to write down the PID values you find while tuning. They will be used in the second half of the lab and are required for submission. Also, the PID values are reset when the Crazyflie reboots, so be sure to write them down frequently!

Relevant parameters:

```
  Group: s_pid_rate
- yaw_kp
- yaw_ki
- Yaw_kd
```

If the ground station becomes unresponsive or stops sending setpoints to the crazyflie, stop it by pressing ctrl + c in the terminal you launched it and restart it. The PID constants are stored in the crazyflie's memory so they should be unchanged.

Your goal for this part of the lab is to demonstrate that you can send a yaw rate setpoint to the Crazyflie and then verify through the GUI ground station that the Crazyflie follows that setpoint closely.

1.2 Pitch Rate

Now change the mount attachment so that the Crazyflie will be held vertically and the **left or right side** of the drone is facing the table, as shown in figure 5.

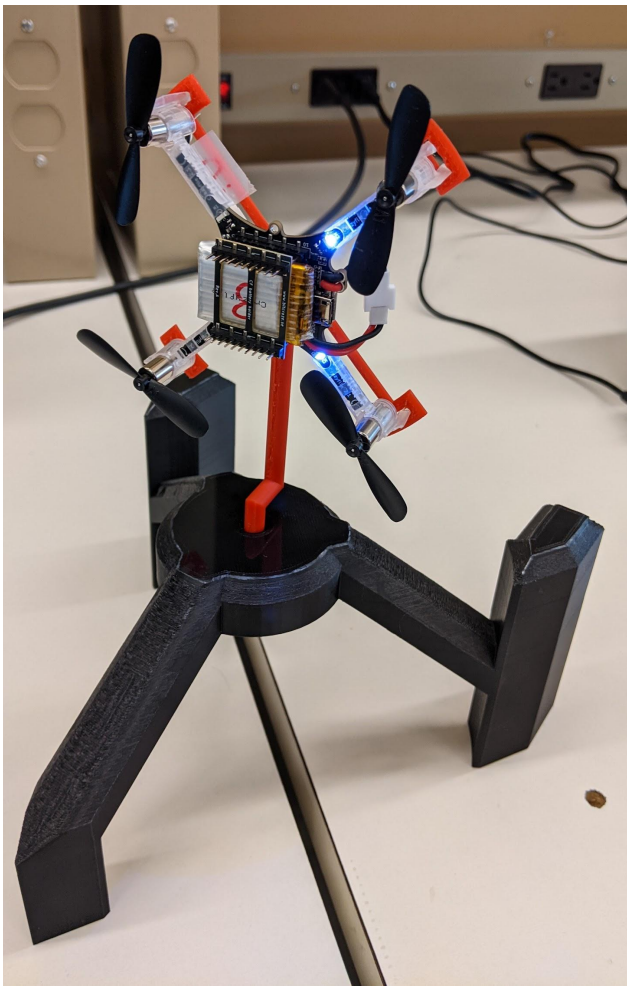


Figure 5. Pitch rate test stand setup

You will now be tuning pitch rate, this is how fast the Crazyflie tilts up or down. Repeat the process you did for tuning the yaw rate, but with the appropriate pitch rate parameters and logging values.

Remember to set the e_stop parameter under the sys group to 0

Relevant logging variables:

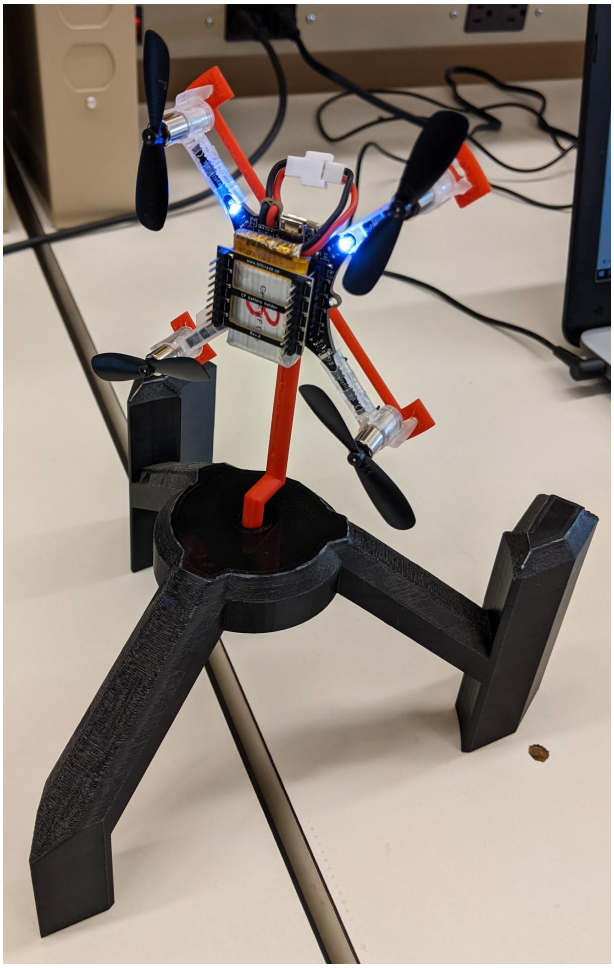
- MicroCART.Test_stand
    - This is test stand data
- ctrlStdnt.pitchRate
    - This is the pitch rate setpoint
- Optional
    - ctrlStdnt.r_pitch
        - This is the crazyflie's on-board sensor for pitch rate measurements

Relevant parameters:

```
  Group: s_pid_rate
- pitch_kp
- pitch_ki
- pitch_kd
```

1.3 Roll Rate

Change the orientation of the drone so that the **front or back** is facing the table.



You will now be tuning roll rate, which is how fast the Crazyflie tilts to the side. Repeat the process you did for tuning the yaw rate and pitch rate, but with the appropriate roll rate parameters and logging values.

Remember to set the e_stop parameter under the sys group to 0

Relevant logging variables:

- MicroCART.Test_stand
    - This is test stand data
- ctrlStdnt.rollRate
    - This is the roll rate setpoint
- Optional
    - ctrlStdnt.r_roll
        - This is the crazyflie's on-board sensor for roll rate measurements

Relevant parameters:

```
   Group: s_pid_rate
-  roll_kp
```

- roll_ki
- roll_kd

Attitude Position Control

We will now tune how the Crazyflie holds a specific attitude angle. Recall from figure 3 that the attitude PID controller provides the input to the attitude rate controller. Therefore, it is important that the rate controller works well before continuing.

1.4 Yaw

Change the test stand setup to how you measured yaw rate. You will now be tuning yaw, which is the angle that the Crazyflie is oriented. This is done similarly to yaw rate except you will be sending degrees rather than degrees per second setpoints. **Note**, there is a button built into the test stand for setting the 0 point of rotation, this can be used to approximately sync up the test stand angle measurement with the Crazyflie's built-in measurement.
In the end you will demonstrate that you can make the Crazyflie rotate to and hold a specific yaw angle, confirming it through the GUI ground station.

Relevant logging variables:

- MicroCART.Test_stand
    - This is test stand data
- ctrlStdnt.yaw
    - This is the yaw setpoint
- Optional
    - stateEstimate.yaw
        - This is the crazyflie's state estimator for yaw angle

Relevant parameters:

Group: s_pid_attitude
- yaw_kp
- yaw_ki
- yaw_kd

1.5 Pitch

For this part of the lab we will turn the test stand on its side and attach the other mount that will hold the Crazyflie parallel to the floor. Ensure the **left or right side** of the drone is facing the test stand, as shown in figure 6.

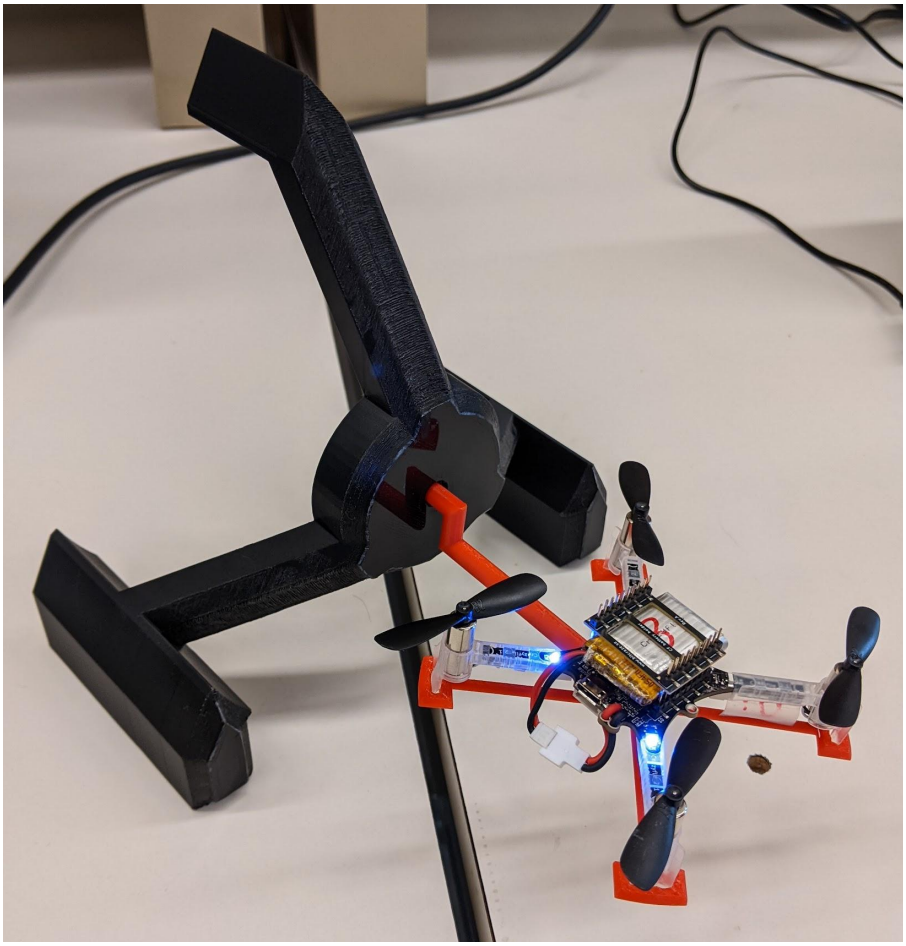


Figure 6. Pitch test stand setup

Repeat the process you did for tuning the yaw, but with the appropriate pitch parameters and logging values. You will demonstrate that you can make the Crazyflie rotate to and hold a specific pitch angle, confirming it through the GUI ground station.

Remember to set the e_stop parameter under the sys group to 0

Relevant logging variables:

- MicroCART.Test_stand
    - This is test stand data
- ctrlStdnt.pitch
    - This is the pitch setpoint
- Optional
    - stateEstimate.pitch
        - This is the crazyflie's state estimator for pitch angle

Relevant parameters:

```
Group: s_pid_attitude
- pitch_kp
```

- pitch_ki
- pitch_kd

1.6 Roll

To measure the roll you will turn the drone 90 degrees such that the **back or front** of the drone is facing the test stand, as figure 7 shows.

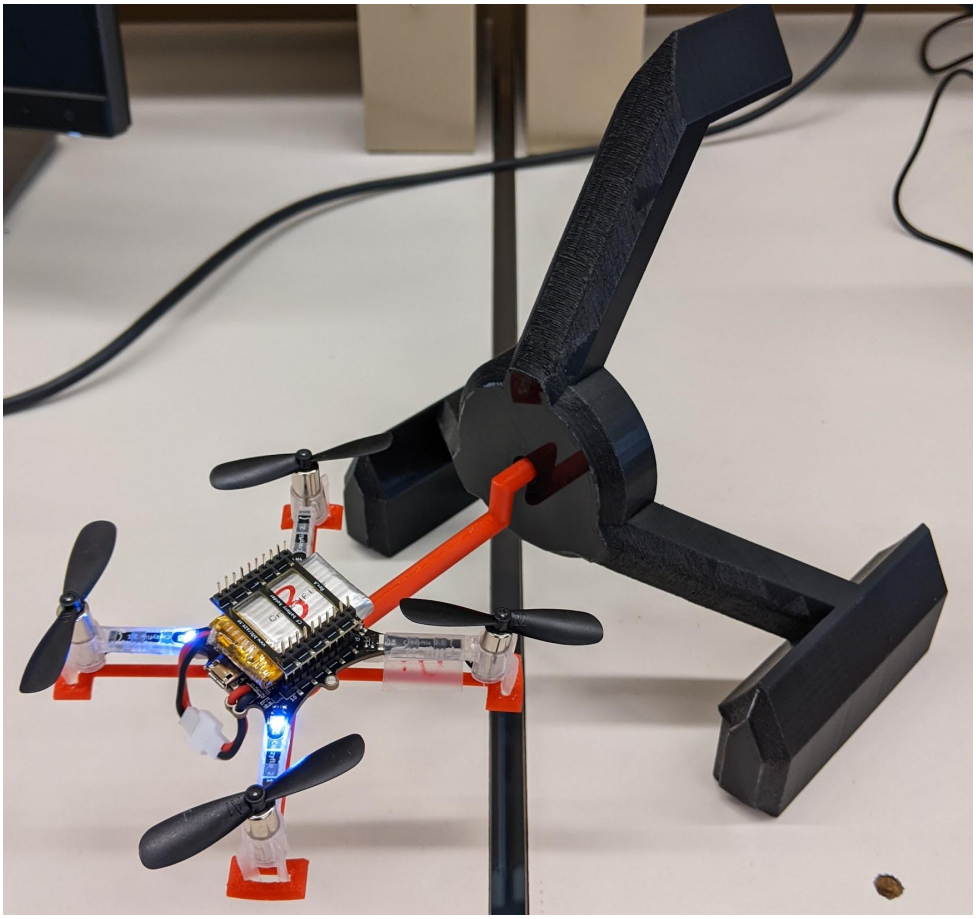


Figure 7. Roll test stand setup

Repeat the process you did for tuning the yaw and pitch, but with the appropriate roll parameters and logging values. You will demonstrate that you can make the Crazyflie rotate to and hold a specific roll angle, confirming it through the GUI ground station.

Remember to set the e_stop parameter under the sys group to 0

Relevant logging variables:

- MicroCART.Test_stand
    - This is test stand data
- ctrlStdnt.roll

        - This is the roll setpoint
    - Optional
        - stateEstimate.roll
            - This is the crazyflie's state estimator for roll angle

Relevant parameters:

```
Group: s_pid_attitude
- roll_kp
- roll_ki
- roll_kd
```

1.7 The Maiden Voyage

This final section of part 1 isn't required but it may be a good idea (and fun) to check how your PID values work in real flight!

At this point you should be able to fully control all axes of the drone through a gamepad. Before you take off for real, it's always a good idea to check that your inputs do what you think they do while **on the test stand**.

See Gamepad Control for details on setting up and using a gamepad with the ground station GUI. Gamepad control uses a **mixed setpoint** setup where **pitch and roll** are given as **absolute angles**, and **yaw** is given as a **rate.**

Once everything looks ok on the test stand, carefully try to take off and get a feel for how she handles. **Be aware of others in the lab!** And try not to crash it too hard.

Part 2: Writing the Control Algorithm

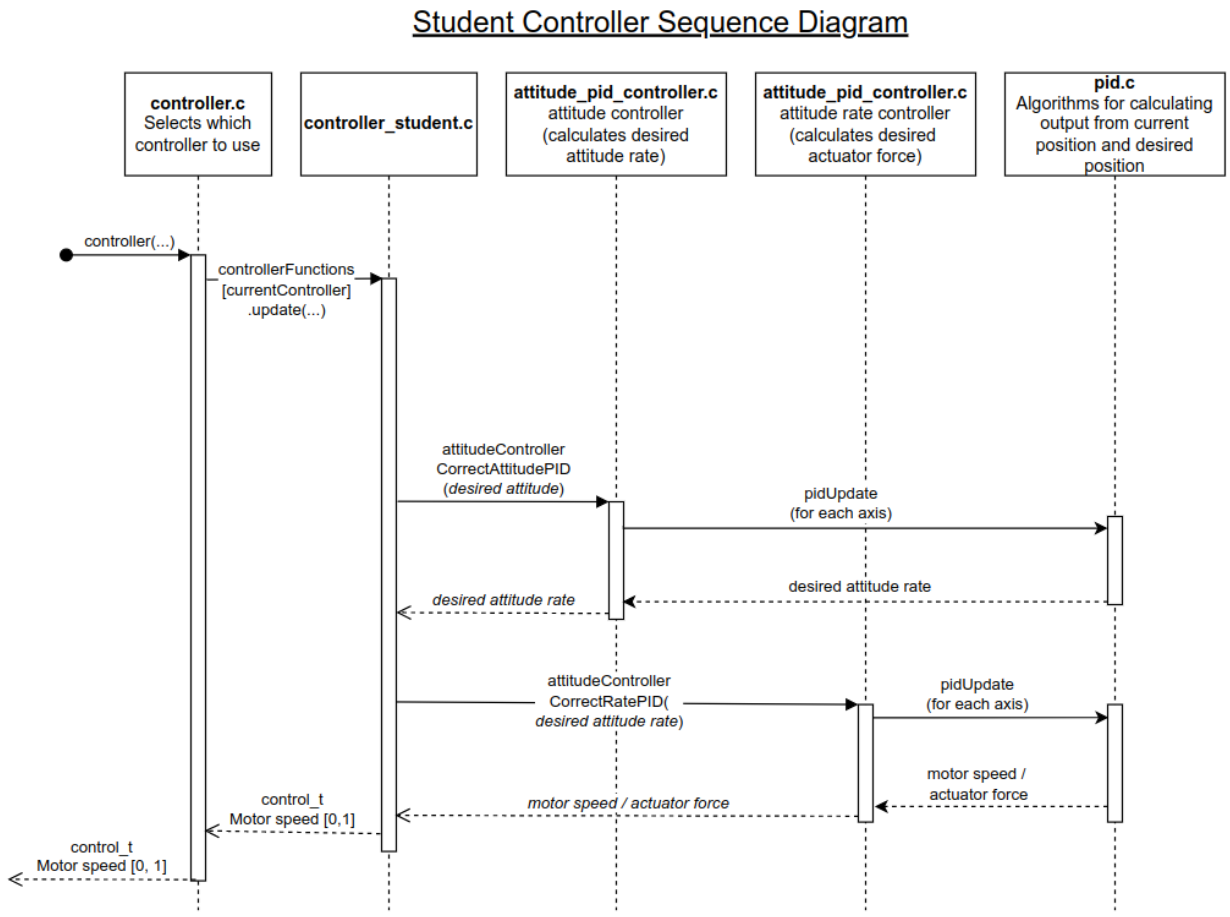
For this part of the lab you will be writing your own control algorithms. **Be sure to set up code exporting from the VM so you don't lose your work.**

All locations where you will need to write new code have been commented with **488 TODO**. You can use VS code's built-in search function to find all occurrences to make sure you haven't missed anything.

Control Layout

The high level student controller, defined in `controller_student.c`, manages setting up setpoints and forwarding them to the attitude controller, whose output gets fed into the attitude rate controller. Both the attitude and attitude rate controller are defined in `student_attitude_controller.c`. The attitude and attitude rate controller utilize the base pid algorithms defined in `student_pid.c`. The output from the attitude rate controller gets

passed back to the high level student controller where it then gets forwarded to other modules outside the scope of this lab.



Understanding the Code

As a part of writing your own algorithms, it is important to understand the data structures used in the firmware. Additionally, it can be useful to specify what logging information to send to the ground station. Below are details on both subjects which will help in the development of your algorithms. Additionally there are some further details on compiling the Crazyflie firmware below.

Logging Instructions

1. At the bottom of the controller_student.c and student_attitude_controller.c you will see a list of log commands. It should look like this in structure but with many more LOG_ADD commands and a couple of groups.

```
LOG_GROUP_START(pid_attitude)
```

```
/**
 * @brief Proportional output roll
 */
LOG_ADD(LOG_FLOAT, roll_outP, <insert address here>)
LOG_GROUP_STOP(pid_attitude)
```

2. This is how information is communicated to the ground station such as current yaw, pitch, roll and other information
3. In the <insert address here> spot you will put a pointer to the address of the **global** variable you want to log (ex: you have named your output pitch variable `outPitch`, you would put `&outPitch` in the field and it would look like the following)

```
LOG_ADD(LOG_FLOAT, roll_outP, &outPitch)
```

4. Now you know how to send information to the GUI ground station :)
5. The parameter macros below the logging are how the GUI sets constants in the firmware, like you did with the PID constants in part 1.

Data Structures

There are four main structs that you will have to be familiar with that the state controller uses to communicate with other modules, all of them are defined in the `/src/modules/interface/stabilizer_types.h` file.
A small but important struct is the attitude struct, this contains roll, pitch, and yaw values as floats as well as a timestamp. Depending on the control context (viewable in the `setpoint.mode` struct) the values in this struct can represent degrees, or degrees per second for each axis.

```
typedef struct attitude_s {
 uint32_t timestamp;  // Timestamp when the data was computed

 float roll;
 float pitch;
 float yaw;
} attitude_t;
```

Figure 8. Attitude struct

The first main struct is the state struct: it contains the Crazyflie's current attitude as well as other parameters that will not be of any use to you. Note that the current attitude rate is not available in the state struct. For this value you will need to read directly from the gyroscope described in the `sensorData` struct.

```
typedef struct state_s {
 attitude_t attitude;       // deg (legacy CF2 body
                            // coordinate system,
                            // where pitch is inverted)
 quaternion_t attitudeQuaternion;
 point_t position;          // m
 velocity_t velocity;       // m/s
 acc_t acc;                 // Gs (but acc.z without
                            // considering gravity)
} state_t;
```

Figure 9. State struct

The next major struct that you will interact with is the setpoint struct. This struct contains the setpoint information provided by the commander module which received its information from the ground station setpoints or a human input device. Like the other structs, it contains many fields of information you do not have to worry about. You need only be concerned with the fields for **attitude, attitude rate, and thrust**. The others like velocity and acceleration are not important because you will only be implementing an attitude controller, not a position or velocity controller.

Additionally, the mode struct controls how the data in the setpoint struct is interpreted. This information is set automatically based on the control method set by the ground station. For this lab, we are concerned with only two or three control modes, attitude control, attitude rate control, and mixed attitude control.

Attitude control
If:

- `setpoint.mode.x y` and `z` are set to `modeDisable`
- AND
- `setpoint.mode. roll pitch` and `yaw` are set to `modeAbs`

Then, the controller will use the values given by `setpoint.attitude`.

Attitude rate control
If:

- `setpoint.mode.x y` and `z` are set to `modeDisable`
- AND
- `setpoint.mode. roll pitch` and `yaw` are set to `modeVelocity`

The controller will use the values given by `setpoint.attitudeRate`. This mode ignores most of the control algorithm and only uses the attitude rate controller to stabilize and control

the quad. This mode is useful if you want to hold a steady rotation speed of the quad body, this is what some other quads call “acro mode”.

Mixed Attitude control

If

- `setpoint.mode.x y` and `z` are set to `modeDisable`
- AND
- `setpoint.mode. roll` and `pitch` are set to `modeAbs` and `setpoint.mode.yaw` is set to `modeVelocity`

This allows the roll and pitch to be specified as a rate and the yaw to be an angle. This is the method used when a gamepad controller is used.

```
typedef struct setpoint_s {
 uint32_t timestamp;

 attitude_t attitude;      // deg
 attitude_t attitudeRate;  // deg/s
 quaternion_t attitudeQuaternion;
 float thrust;             //0 - 60,000
 point_t position;         // m
 velocity_t velocity;      // m/s
 acc_t acceleration;       // m/s^2
 bool velocity_body;       // true if velocity
                           //is given in body frame;
                           // false if velocity is
                           // given in world frame

 struct {
   stab_mode_t x;
   stab_mode_t y;
   stab_mode_t z;
   stab_mode_t roll;
   stab_mode_t pitch;
   stab_mode_t yaw;
   stab_mode_t quat;
 } mode;
} setpoint_t;
```

Figure 10. Setpoint struct

The next struct is the control struct. This is the output of your control algorithm and represents the force to apply to the drone's body. It is sent to the power distribution module which converts it into motor commands. Note that these are 16 bit int values, so a conversion must take place from a float.

```
typedef struct control_s {
 int16_t roll;
 int16_t pitch;
 int16_t yaw;
 float thrust;
} control_t;
```

Figure 11. Control struct

The last struct that you will interface with is the sensorData struct. This contains raw data from several sensors, including the gyroscope. You will need to access information contained in this struct for some of your rate PID calculations.

```
typedef struct sensorData_s {
 Axis3f acc;               // Gs
 Axis3f gyro;              // deg/s
 Axis3f mag;               // gauss
 baro_t baro;
#ifdef LOG_SEC_IMU
 Axis3f accSec;            // Gs
 Axis3f gyroSec;           // deg/s
#endif
 uint64_t interruptTimestamp;
} sensorData_t;
```

Figure 12. Sensor data struct

Compiling The Crazyflie Firmware

The crazyflie firmware can be compiled from the firmware root folder, `/Lab_Part_2/crazyflie_software/crazyflie-firmware-2021.06/` by running `make CONTROLLER="Student"`. After successful compilation, the binary files will be placed in the root of the firmware folder and can then be flashed to the crazyflie by following the instructions above.

`make unit` can also be used to run unit tests on the firmware, the unit tests are defined in `/Lab_Part_2/crazyflie_software/crazyflie-firmware-2021.06/test/`. You are free to add additional unit tests, however it is not required.

Writing the Code

Now that you understand the structure of the firmware, it's time to start writing your own algorithms. You may want to review Flashing the Crazyflie, when you're ready to compile run `make CONTROLLER="Student"` from the root of the Crazyflie firmware.

Below is the suggested order of additions to make to the firmware.

2.1 General PID

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

The first thing that you will write is a general PID function and struct. The PID struct that we provide you will be empty and you will decide on what should be included in it, that is defined in **student_pid.h**. You are encouraged to make as many helper functions that you would like in your **student_pid.c** file to help with roll, pitch and yaw calculations.

The first thing I would recommend writing is the **PidObject** struct in `student_pid.h`. This struct is used to hold the data that is used for all other PID calculations so it is required to write many of the other functions.

Next, write the basic getters and setters for the PidObject in `student_pid.c`.

Now we can actually write the PID algorithm in the `studentPidUpdate` function.

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

2.2 Attitude Rate Controller

The attitude rate controller's main functions are in `student_attitude_controller.c`, this is where you should begin working.

2.3 Attitude Controller

The attitude controller's main functions are also in `student_attitude_controller.c`.

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

2.4 Student Controller, Bringing it all together

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

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

Final Check

Before you go flying your Crazyflie for real, it's a good idea to verify everything works as intended on the test stand. Attach the drone to the test stand and briefly check that all axes respond how you expect. For this step you can use manual setpoints or a gamepad connected to the ground station, see here for details on using a gamepad with the ground station.

If all looks good, take her for a spin and see how she handles! **Be careful of others** in the lab and try not to crash it too hard!

What to submit

- Kp, Ki, and Kd constants for yaw rate, pitch rate, roll rate, yaw, pitch, and roll
- All documents that were edited in the firmware to complete part 2, see final export for details

If you're hungry for further challenges, take a look at the **extra credit** section of this lab.

Extra Credit

1. Give feedback on this lab
    a. This is a new lab developed by the MicroCART team as a senior design project. We would appreciate some feedback on what you enjoyed about the lab and what can be improved.
    b. Google Form Link
2. Test Stand Data Visualization
    a. As in MP-1, extra credit may be awarded to teams that make a creative visualization for the test stand position or rate data.
3. Manual flight obstacle course
    a. If teams can demonstrate their stable control algorithm by manually flying through an obstacle course extra points can be awarded. Film your obstacle runs and submit.
4. Autonomous flight
    a. The Crazyflie has many autonomous capabilities that we have not even touched in this lab. Extra credit will be awarded if you can write a short script to takeoff, fly forward, and land without using a controller.

b. You could write a short script that uses the ground station CLI to send commands over and over or you could use the crazyflie python library to send high level commands to the Crazyflie.
    i. An additional circuit board, the flow deck, is required to use the crazyflie python library. If you wish to pursue this, talk to a TA and they can acquire a flow deck from the MicroCART team (microcart-f2021@iastate.edu).
    ii. Details on writing python scripts for the crazyflie can be found here

Document Version Changelog

- Version 1.1
    - Changed optional logging variables in lab part one from gyro.x, y, z to ctrlStdnt.r_roll, r_pitch, r_yaw
    - Changed git instructions for vm to use “Lab_Part_* folder” instead of “Microcart folder”
    - Updated student controller sequence diagram to have desired attitude **rate** being passed to the attitude rate controller
    - Removed “Test Stand Control Board Rate mode” extra credit & replaced it with test stand visualization extra credit