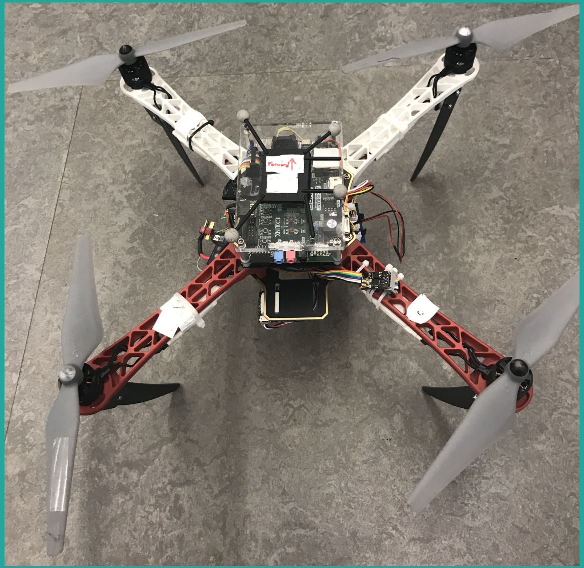
MicroCART

(Microprocessor Controlled Aerial Robotics Team)



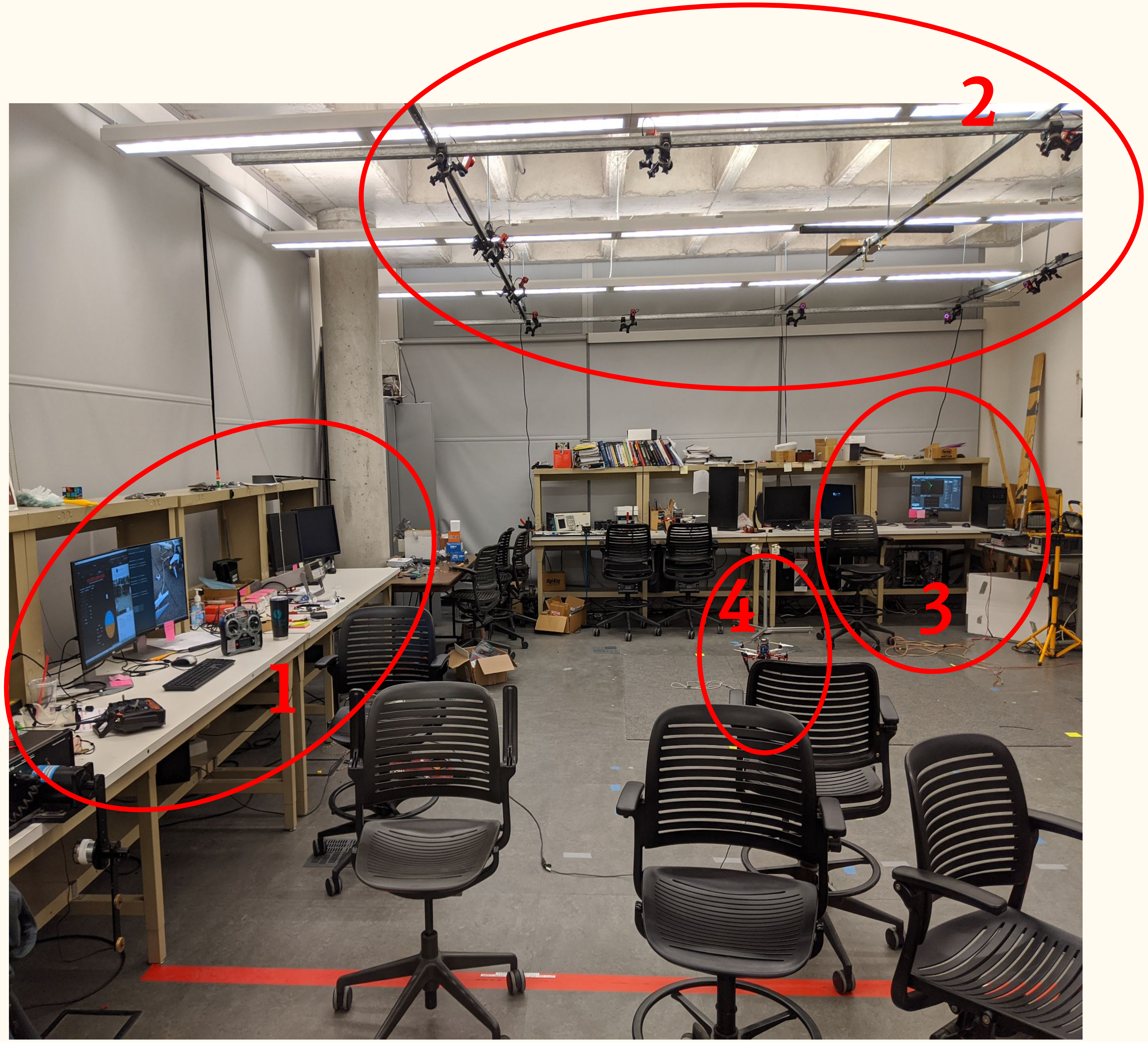
Evan Blough - Technical Team Lead, Embedded Software Lead
Kynara Fernandes - Ground Control Station Lead
Joe Gamble - Embedded Hardware Lead
Jacob Brown - Physical Hardware Lead
Aaron Szeto - Controls Lead
Shubham Sharma - Crazy Fly Lead
Client and Adviser - Dr. Phillip Jones

MicroCART
(Microprocessor Controlled Aerial Robotics Team)

Project Vision

- MicroCART aims to develop a drone platform to support graduate research and flight demonstrations.
- Future senior design teams, graduate students, and prospective ECPE students would benefit from the development of our platform
- The following features have been added into the existing MicroCART system
  - Built and flew a second quadcopter drone
  - Integrated an adapter into our software to control the Crazyflie drones
  - Built a tuning stand to tune controls for Crazyflies
  - Added additional widgets to the existing Graphical User Interface
  - Integrated an adapter that controls vehicles with MAVLink protocol

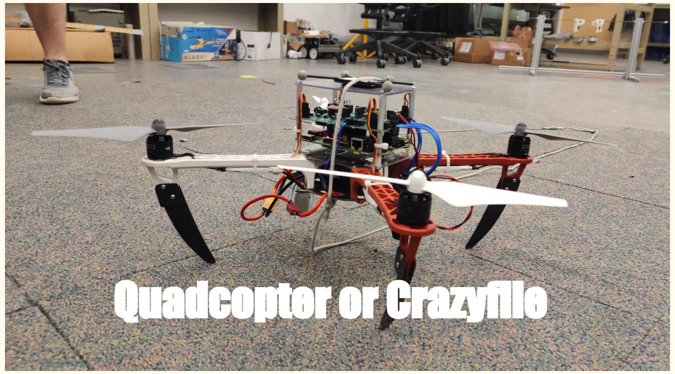
Our Workspace



Drone position data

Log data

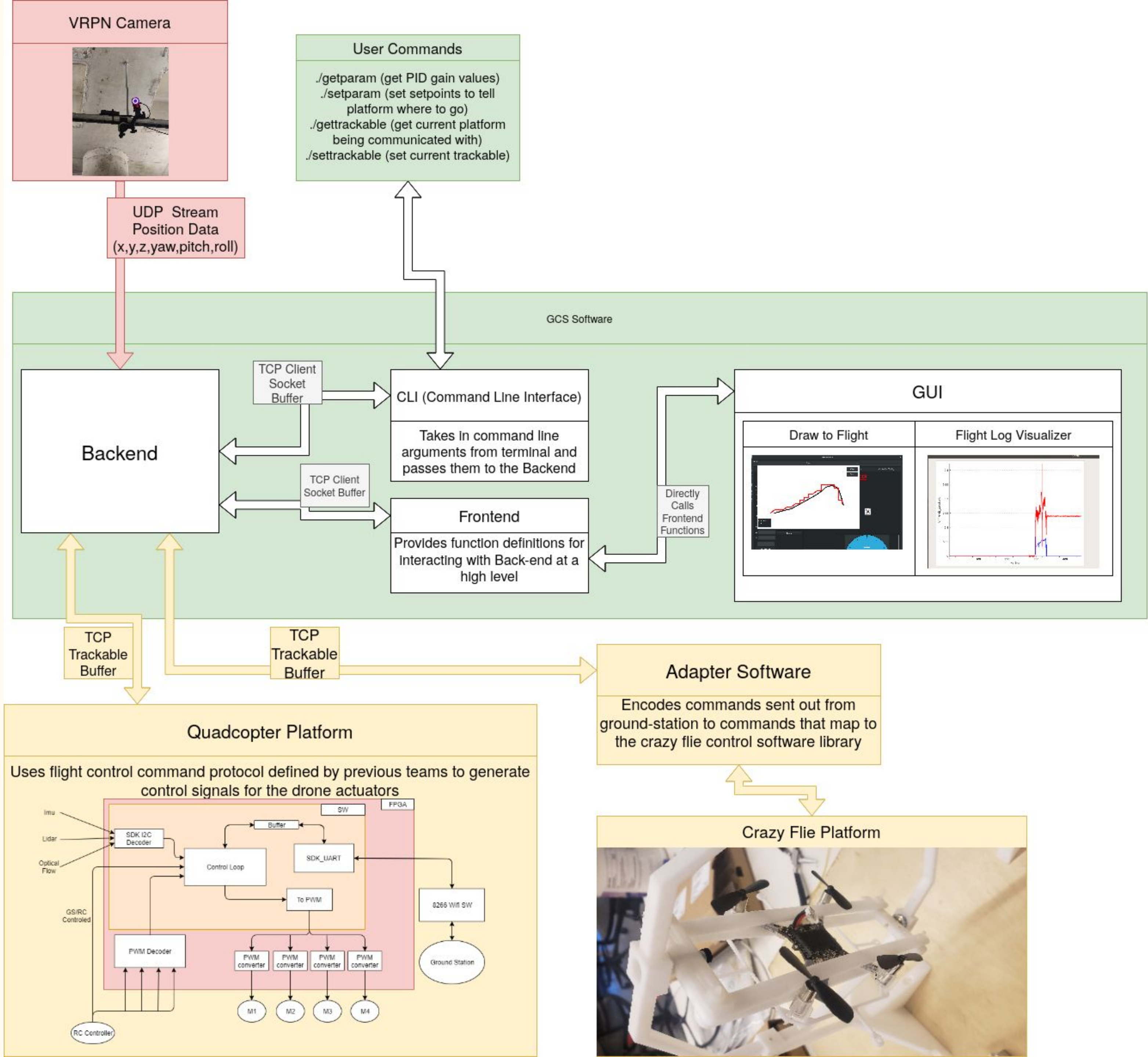
User input



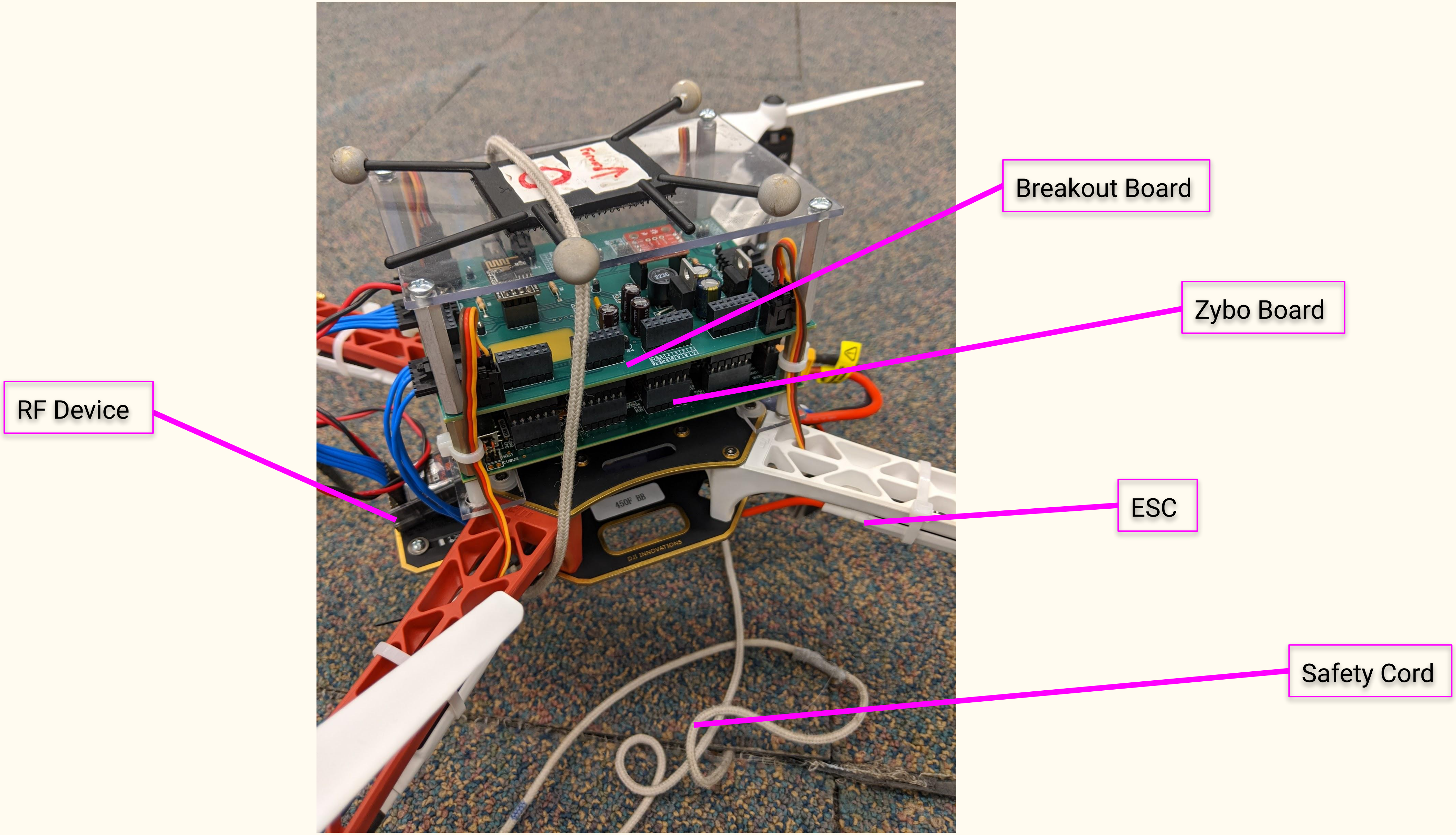
Failsafe Manual Input

System Design

Detailed Design



Quadcopter Hardware



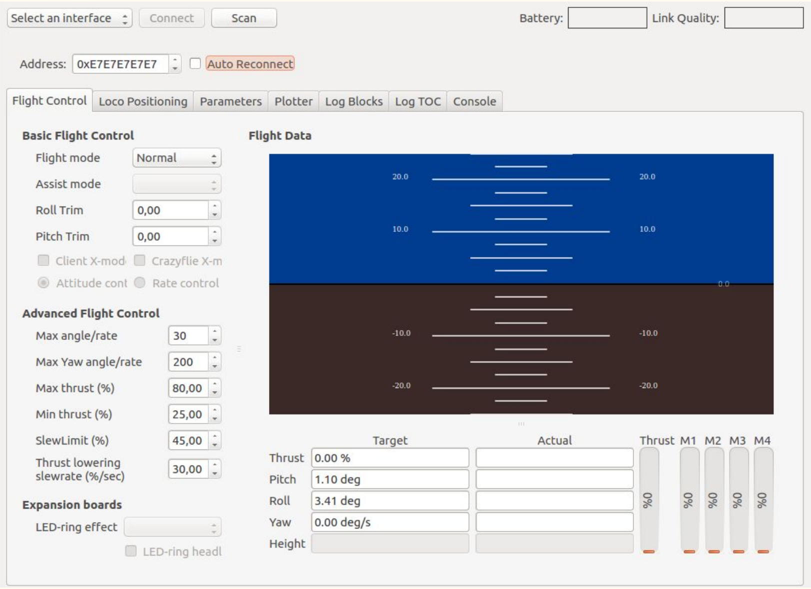
Tuning Stand

- Designed using Autodesk Inventor
- 3-D printed
- Encoder interface using microcontrollers
- construction is light enough to allow free movement of crazyflie
- Ultra-stable duck-tape mounting system

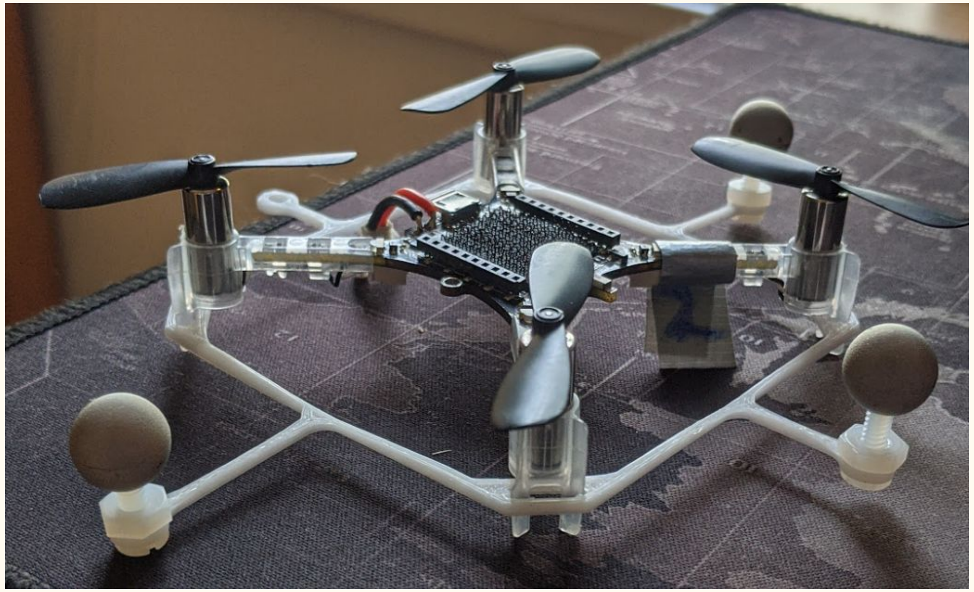
Crazyflie Firmware & Adapter

- Fixed issues with the previous adapter
- All Crazyflies have been updated to version 2020.02 firmware for better P2P communication support
- Tweaked the adapter to work with the new firmware

- Implement adapter in the Crazyflie VM for more versatile development
- Setup port forwarding for passthrough communication for camera system through the VM



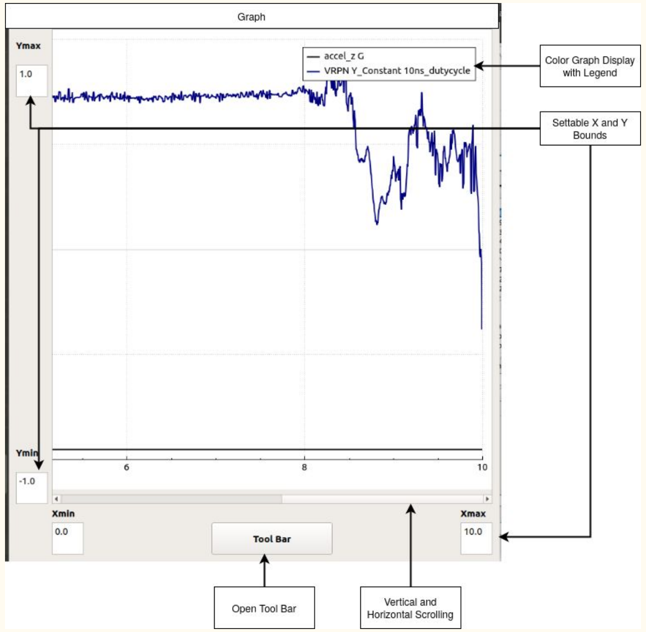
Crazyflie client running on the new VM

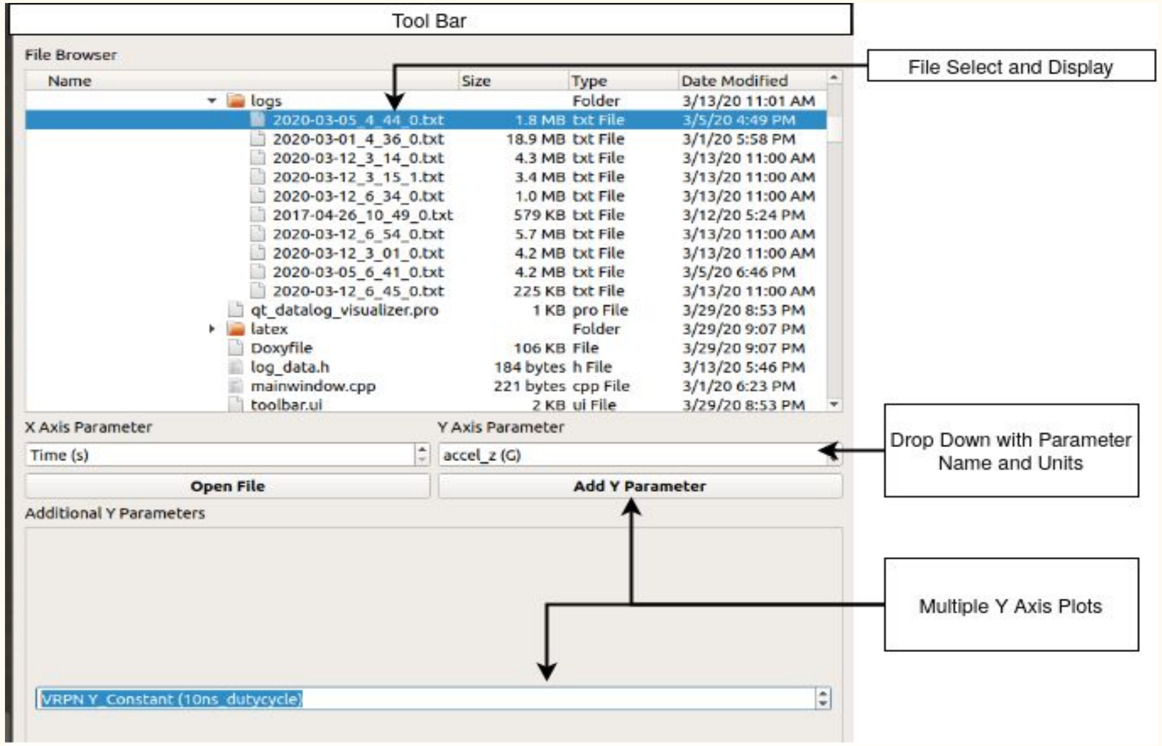


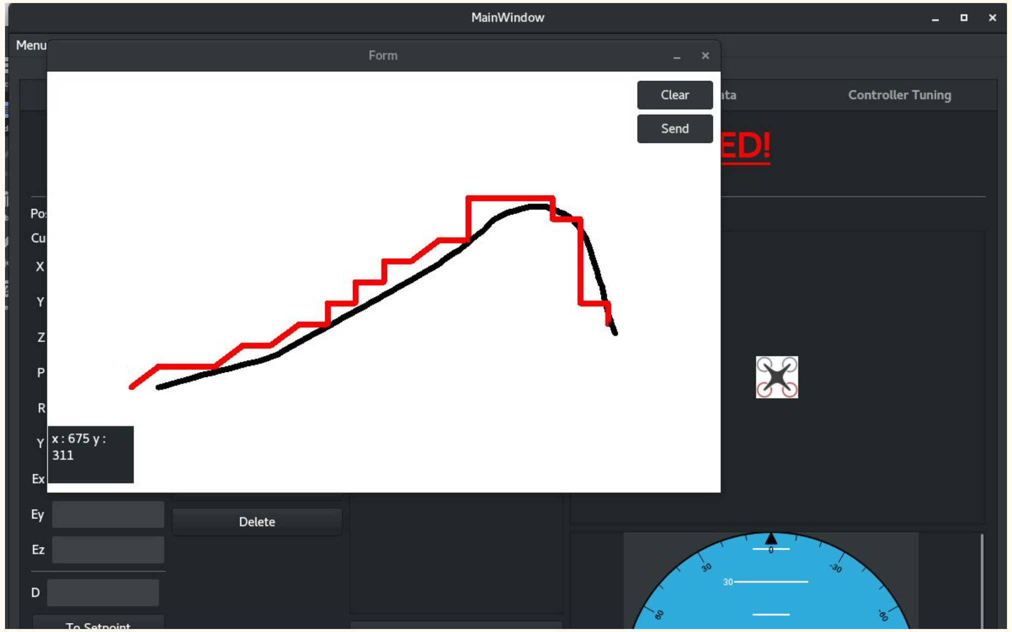
Crazyflie set up with the IR trackers

New GUI Features

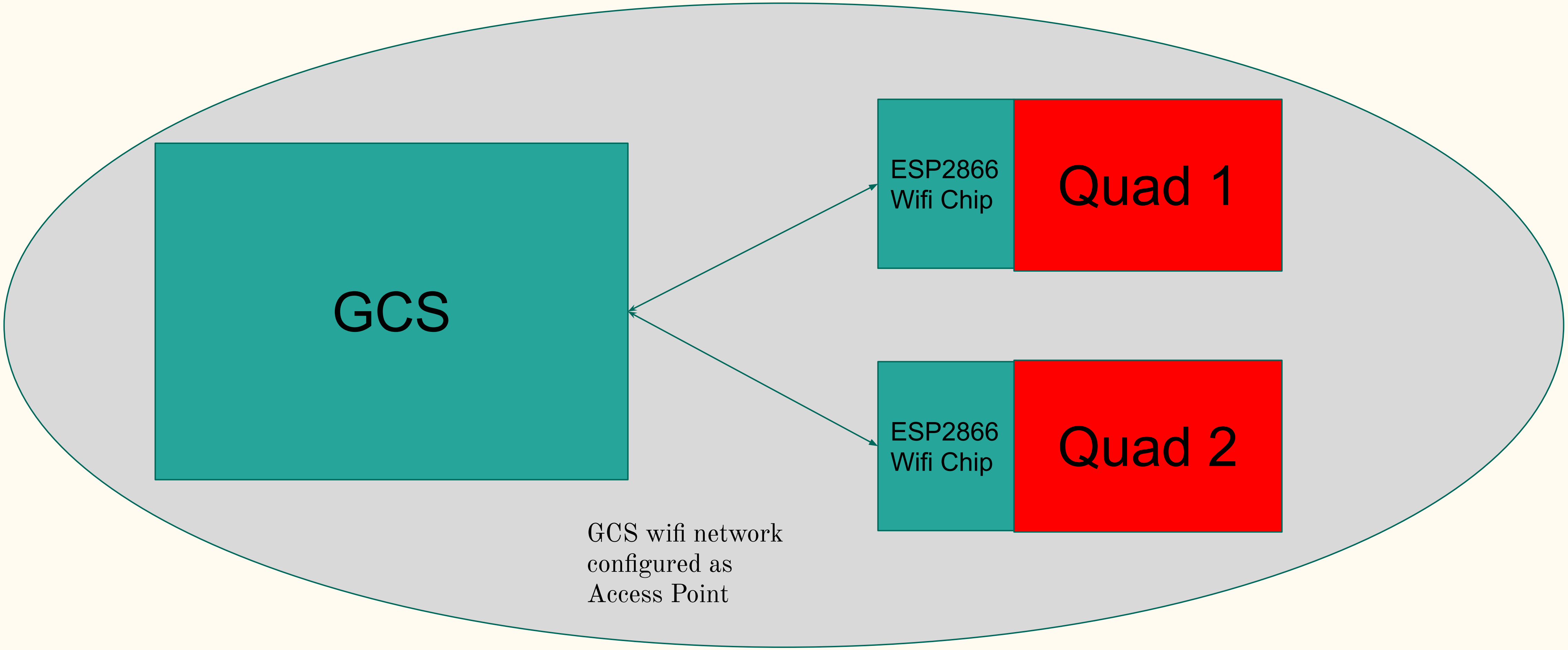
- PID Adjuster
- Draw Flight Path Model
- Log Data Visualizer







Multiple Quadcopters



MAVLink Adapter

```
File  Edit  View  Search  Terminal  Help
rmasl@rmasl-Latitude-E5440:~$ ./getparam 0 0
```

MAVLINK — MICRO AIR VEHICLE COMMUNICATION PROTOCOL

python

ARDUPILOT

MicroCART CLI Interface

The CLI interface gets commands like getnodes, setparam, and getparam from command line to set configuration for the autonomous vehicle

TCP Client Local ⟷ TCP Server Local

MicroCART Backend

The Backend processes the client requests and formats their requests to our messaging protocol shown below. The messages will get sent to the mavlink_adapter socket specified in the trackables_t struct in config.c

TCP Client Local ⟷ TCP Server Local

MAVProxy/MAVlink Adapter

The adapter retrieves the formatted messages and remaps them to messages with similar functionalities in the MAVLink library. Not every type of command is able to be mapped to a MAVProxy equivalent because the MAVLink library is several thousand lines of source.

The functionalities supported by the adapter are getting and setting waypoints/setpoints, arming the vehicle, setting the navigation mode, and setting and recieving the values of the MAVPARAMs stored on the vehicle.

The microcart commands supported by this adapter are getnodes, getparam, and setparam.

TCP Server Local or IPv4 ⟷ TCP Client Local or IPv4

MAVProxy

Mavproxy is an opensource software that can be run on any machine with Python 3. An example would be an RPi. Mavproxy will foward the command messages recieved from the adapter socket to the mavlink platform

Arbitrary communication type

This is dependent on the system configuration. MAVproxy supports connections to devices through TCP/UDP, USB/UART, and COM ports.

MAVLink Platform

The Mavlink platform will act as a microcontroller unit for the vehicle platform. The MAVLink is in charge of doing sensor processing and actuation. These computations are handled by another opensource software called ArduPilot. Ardupilot can come preinstalled on some hardware modules. Ex: Pixhawk

Trackables Config

```
//Original Quadcopter
[0] = {
    "mavlink", //Name
    0, //Set to 0, the backend will modify on start
    //Wifi Variables
    QUAD_IP_DEFAULT,
    QUAD_PORT_DEFAULT,
    //Adapter Variables
    1,
    "",
    "/home/rmasl/Desktop/MicroCART/groundStation/adapters/mavlink/cmake-build-debug/mavlink_adapter.socket",
    NULL,
    //Local Communication Variables
```

Get Param Data format

data index	0 - 1	2 - 3
parameter	node ID	node parmID
bytes	2	2

Get Nodes Response Data Format

data index	0 - 2*N-1	2*N - 4*N-1	4*N - (< 4096)
parameter	Node IDs	Node type IDs	Node names
bytes	2*N	2*N	< 4096

Set Param Data Format

data index	0 - 1	2 - 3	4 - 7
parameter	node ID	node parmID	param val
bytes	2	2	4

Micro Cart General Message Format

Index	0	1	2	3	4	5
Message Parameter	Begin Character	Message Type	Message ID	Data Length	Data	Checksum
Bytes	1	2	2	2	var	1

Requirements

Functional Requirements

- Integrate a GCS adapter to control Crazyflies
- Design/Integrate a GCS adapter to control MAVLink platforms in simulation
- Build a second quadcopter that can receive control commands from the GCS
- Control multiple Crazyflies and quadcopters with the GCS

Non-functional Requirements

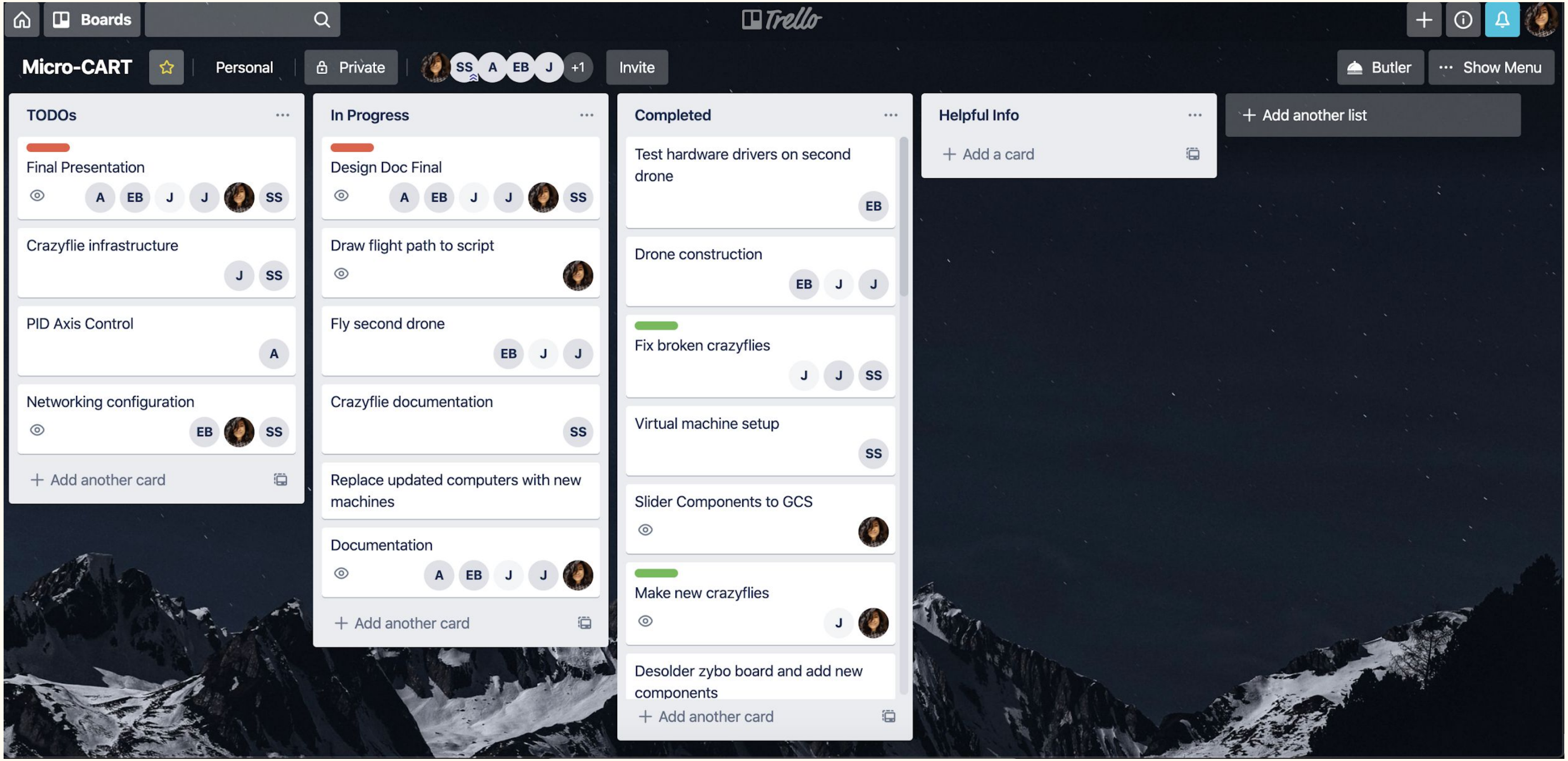
- Functional requirements must have substantial documentation for future teams
- Maintenance of current system response time / latency
- Maintaining existing flight accuracy for multiple vehicles
- GUI Feature that can display flight logs in the quadcopter’s format
- GUI Feature that allows a user to draw patterns that are converted to setpoints
- GUI Feature that allows a user to set PID gains for the quad’s control algorithm
- Build a tuning stand to tune flight controls of Crazyflie platform

Constraints

- We had to adapt to an existing system.
  - Code, frameworks
  - Workspace and computers
  - Parts
- Crazyflie adaptation to current GCS
  - Communication mode
  - Communication format
  - Communication latency
- Tuning stand that can hold crazyflie platform based on physical dimensions

Conceptual Design Approach - Agile

- One week sprints
- Scrum styled stand-up
- Meeting with stakeholder
- Grooming/Planning for next sprint
- Trello agile board



Trello Agile Board

Tasks

- Drone Assembly
  - Board Assembly (Complete)
  - Chassis Assembly (Complete)
  - PCB Testing (Complete)
  - Interconnect assembly and integration (Complete)
- Quadcopter Development
  - Extend networking interface for swarm flight (Complete)
  - Support for control algorithm swapping (Not Complete)
  - Addition of Linux to second Core (Not Complete)
- GUI Development
  - PID Slider, Draw to Flight, Datalog Visualizer
- Crazyflie development
  - Research into existing Crazyflie Adapter (Complete)
  - Update Crazyflie firmware to the latest version (Complete)
  - Set up a development environment for the Crazyflie (Complete)
  - Tuning table created for testing Crazyflie axis rotation (Complete)
  - Embedded system design for control system feedback on turning table (Not Complete)

Risks

- Equipment Risks
  - Aged development machines
  - Battery overcharge
- Knowledge Area Risks
  - Team inexperience with certain development fields
  - Networking configuration of swarm flight
- Cost Risks
  - Dependent on hardware modules
  - Sensors, Lidar, IMU, etc.
  - Buying these components impedes progress

Mitigation of Risks

- Upgrading development platforms to support future teams and speed up development
- Performed extensive research during fall break to determine ideal networking configuration
- Frequently made bills of materials to get materials before we actually needed them
- Workplace safety measures like goggles, tether cord for the drone while flying, and a kill switch on the RC controller

Engineering Standards and Design Practices

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

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

2. IEEE Code of Ethics provided by the IEEE organization

Prototype Implementation

Prototype

- New drone built from available parts to match existing drone
- Previous year’s team progress handed down to this year’s team
- Zybo-board shield
- Multiple drone flight tested using crazyflies rather than quadcopter
    - Tuning table designed for these tests

Prototype Implementation

- Previous teams have built prototypes
- Final design plans from last year implemented to continue progress
- Prototyping mostly involves models and simulations
- 3-D printed turntable

Project Plan

Project Proposed Milestones

1. Design and 3-D print a tuning table
2. Set up a Crazyflie development environment
3. Having multiple autonomous Crazyflie controlled with our GCS software
4. Build a secondary Quadcopter drone
5. Connect two Quadcopters to Ground Station at the same time
6. Fly two Quadcopters at the same time
7. Mavlink Adapter handles getparam, setparam, and getnodes commands
8. Mavlink Adapter can change vehicle parameters and initiate autonomous navigation
9. Implement new helpful GUI features

Project Timeline - First semester

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

Project Timeline - Second Semester

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

Project Milestones Results

1. Tuning table was designed and printed
2. Created Crazyflie development environment
3. Set up multiple working Crazyflies but can only fly one at a time
4. Built second fully functional drone
5. Connecting 2 drones at the same time: still work in progress
6. Only able to fly 1 MicroCART drone at a time
7. Mavlink adapter supports control for any autonomous vehicle with the Mavlink protocol
8. Mavlink adapter can set vehicle parameters and initiate autonomous navigation
9. Implemented PID sliders, Datalog visualiser, and Draw to Flight

Testing

Component/Unit testing

- Board testing for short/opens and voltage supplies
- Code compilation and debugging for VHDL systems level and on onboard computation
- Ground station functionality w/o networking
- Packet protocol data alignment

Interface/integration testing

- Signal sent/received
- Changing control variables for multiple fliers
- Communication delays are satisfactory
- Testing GUI improvements

System level/Acceptance testing

- Drone flight
- MAVLink Adapter
    - Supports Missions, get/set params, and get nodes
- Tuning Stand
- Datalog visualizer

Conclusion

Schedule Progress and Future Plans

- We have made significant progress towards the completion of the senior design project
- Summary of our accomplishments
  - Built second working drone
  - Crazyflie flight
  - Designed and built 3-D printed tuning table
  - Made various GUI features
  - Communicated with multiple drones at once
  - Crazyflie swarm capabilities
- Goals for future teams
  - Get drone swarm working
  - Adapt GUI for multidrone flight
  - Implement control algorithm swapping
  - Install Linux in second drone core

Thank you

- Dr. Jones - Professor Advisor
- Matt Cauwels, James Talbert - Graduate Advisors

No Thank You

- Coronavirus