About Me

I am a senior Computer Engineering undergraduate student at the University of Wisconsin-Madison.

My current passion is working on embedded electronics and software for vehicle systems. Designing such systems from scratch has revealed to me some of the technical and operational complexities in building a physical product.

About Me

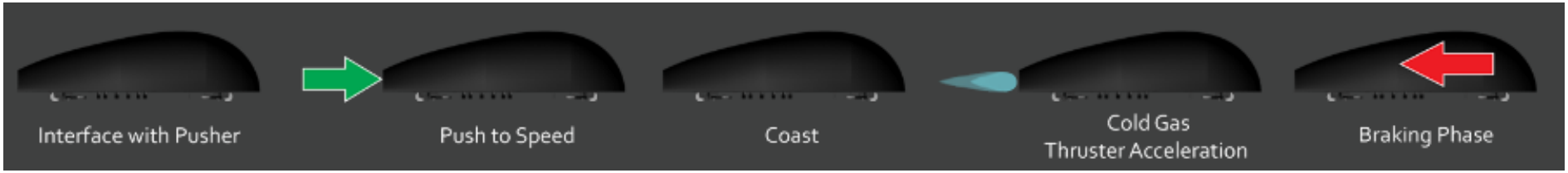
Iteratively deriving more efficient workflows and processes is a skill I have invested time developing and has helped me participate in a multitude of projects and work effectively with different technology stacks.

I have a strong appreciation for "test driven" development and generally thinking through problems as thoroughly as possible before getting deep into the work because it tends to result in the most uninterrupted time spent doing the enjoyable, "engineering" parts.

About Me

My home electronics lab when it was first set up

Hyperloop II: Electrical System



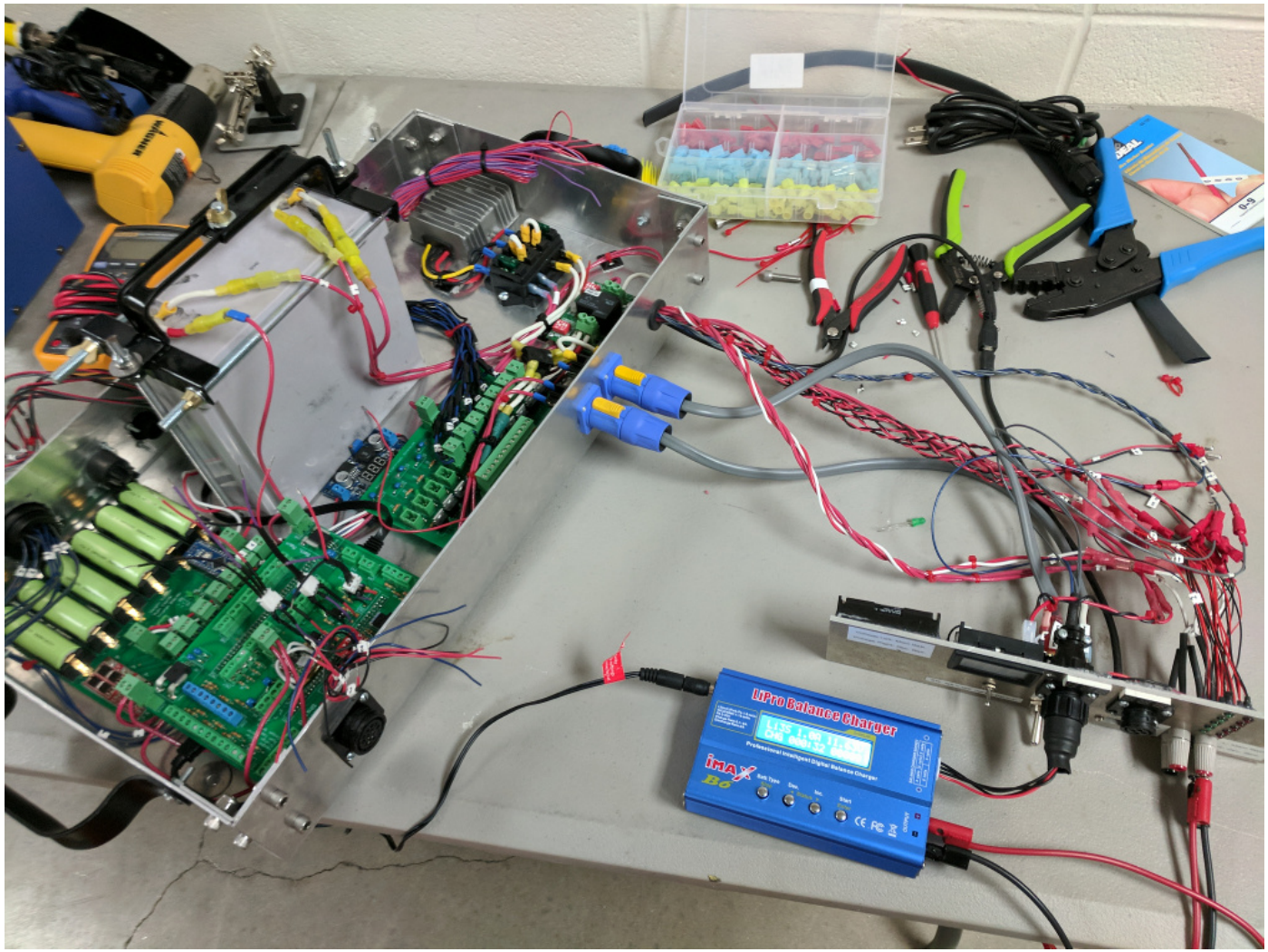
The image above captures the concept of operation of the second Badgerloop pod. It's a vehicle that must travel about one mile across a rail inside a vacuum.

this particular vehicle used two COPVs filled to 3000 PSI feeding a single "converging-diverging" nozzle.

I was responsible for the embedded electronics and web-based user interface that would be used to view pod telemetry while it's inside the tube and send manual commands if/when necessary.

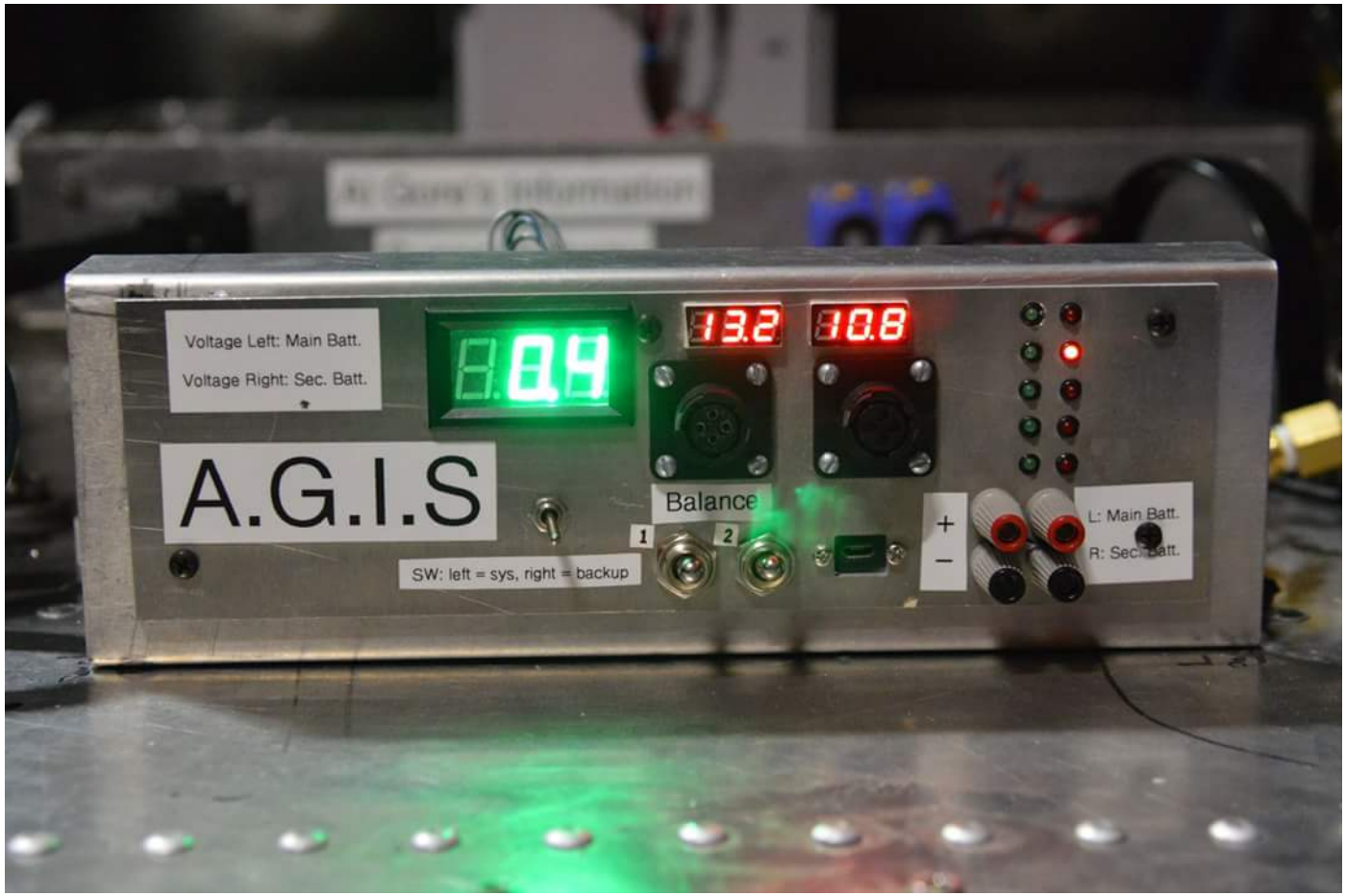
Hyperloop II: Electrical System

Testing charging via the control panel



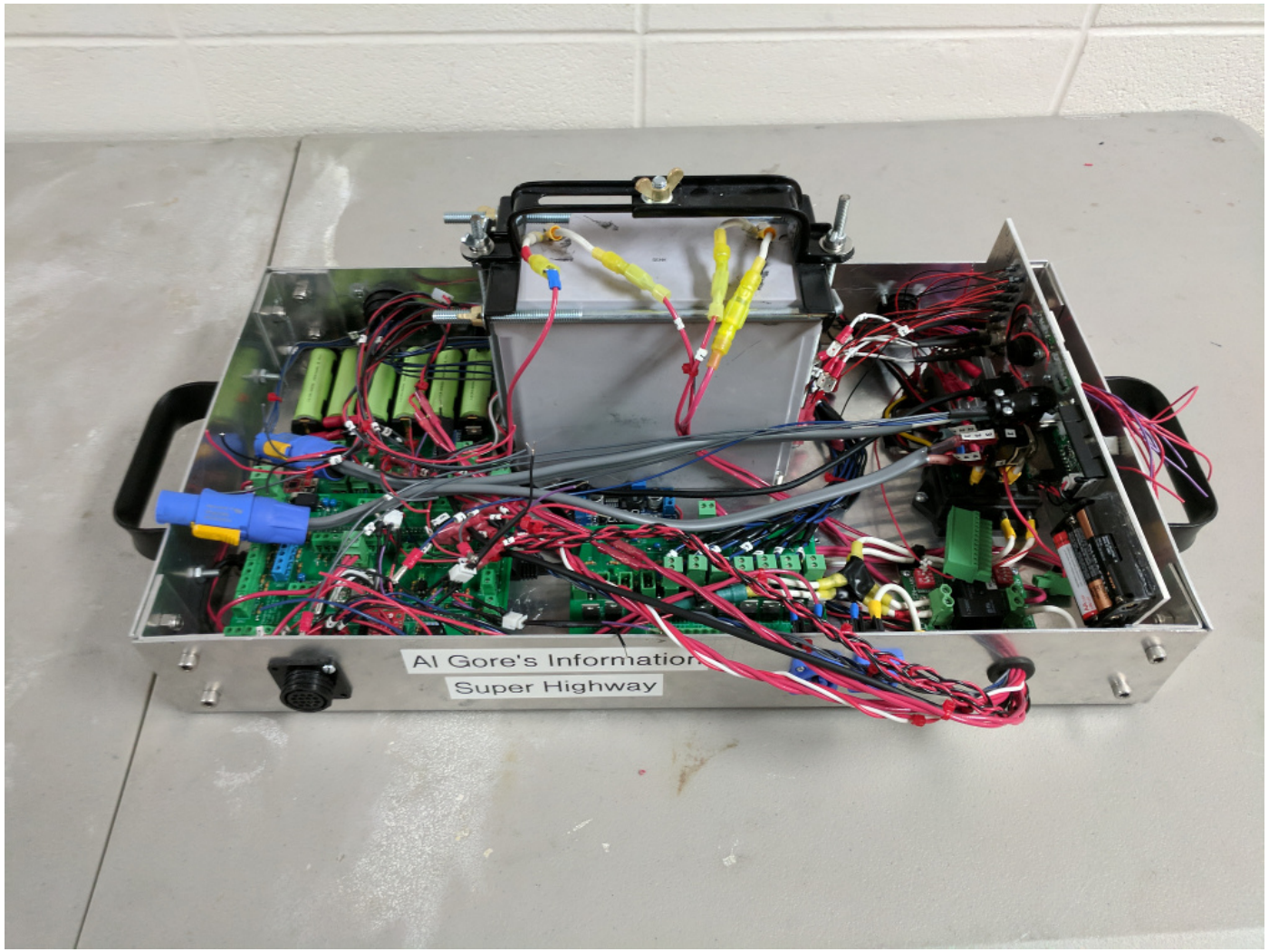
Hyperloop II: Electrical System

Control panel for easy access to charging, programming and status LEDs



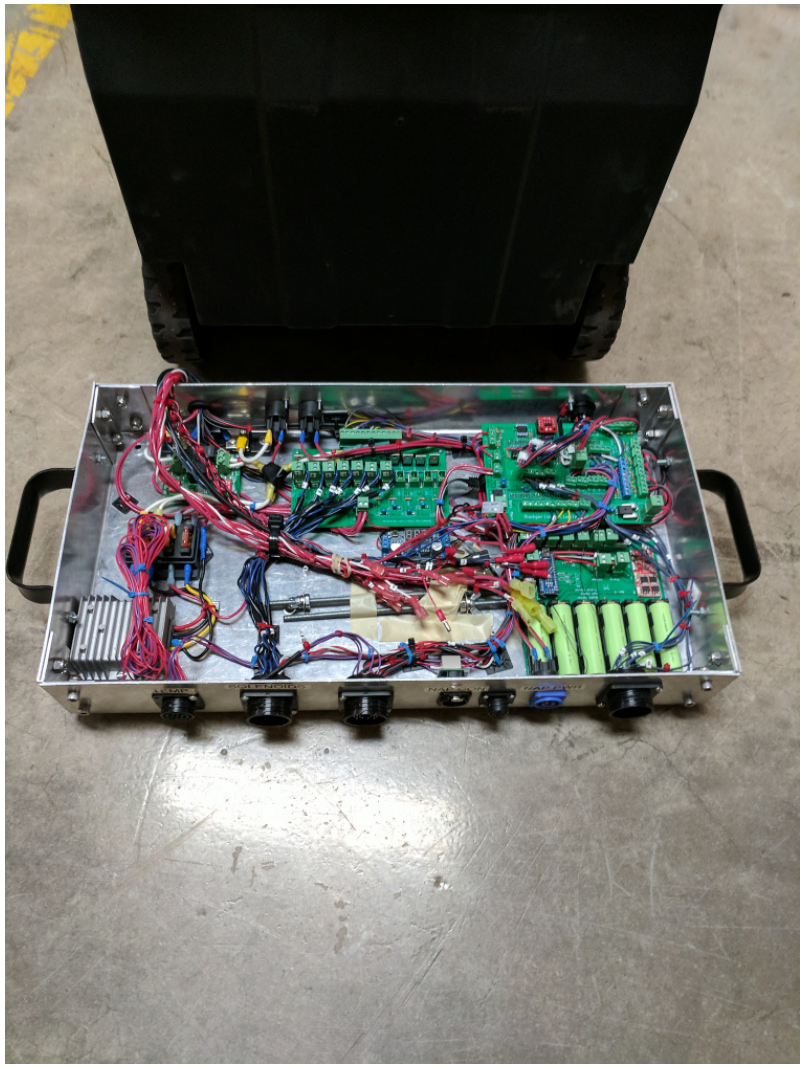
Hyperloop II: Electrical System

Pod's entire electrical system in one image



Hyperloop II: Electrical System

Battery and some harnessing removed for transport



Hyperloop II: Dashboard

The following slides include screenshots of the web-based dashboard I created to monitor and control our pod for the second hyperloop competition (Summer 2017).

The embedded system is one STM32 Nucleo 144 F767ZI microcontroller development board. We wrote all of the firmware from-scratch and used LwIP to implement the network stack.

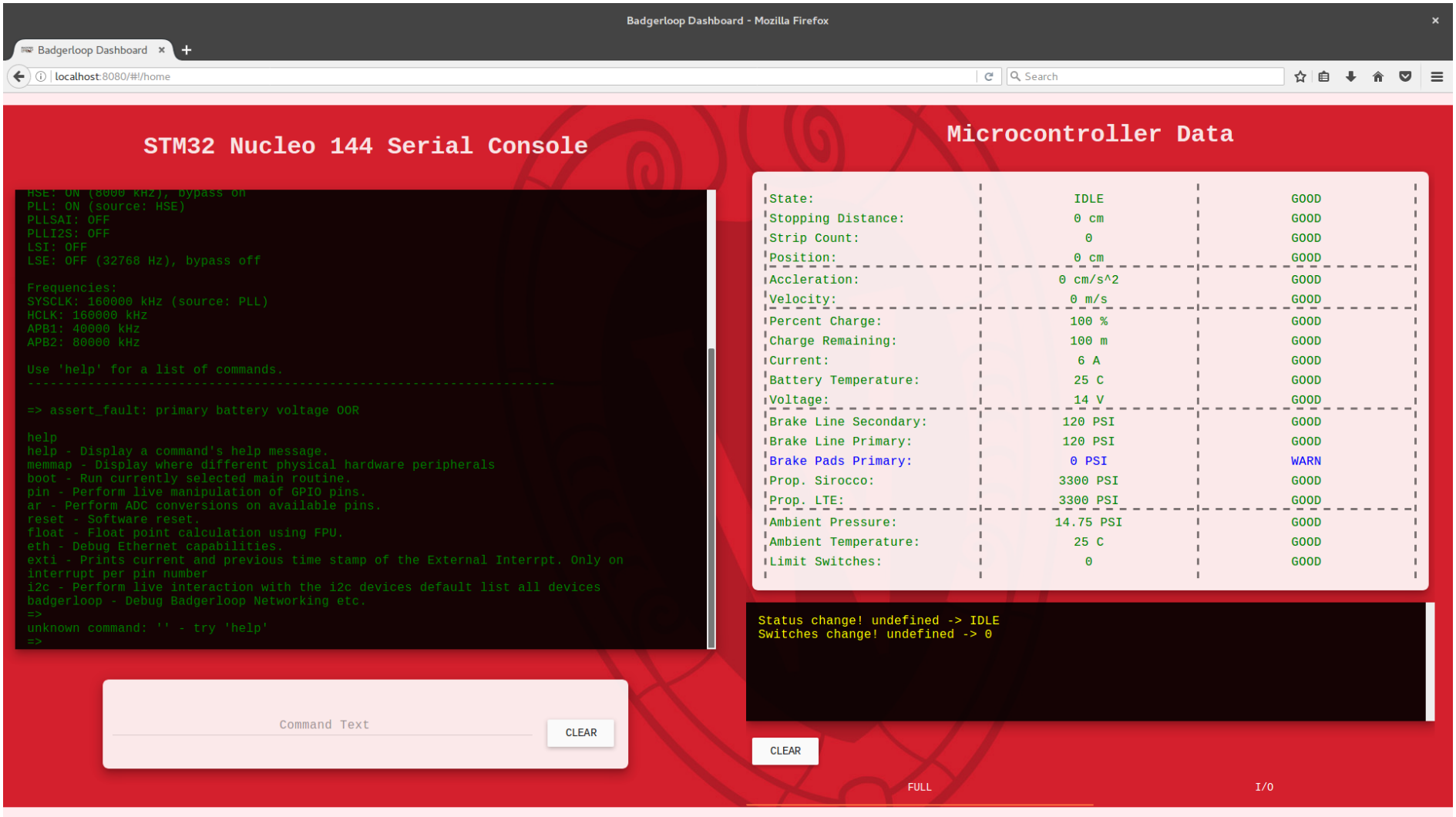
The "terminal" shown in the following slides is "redirected `stdout`" from the microcontroller, not an SSH session or userspace Linux program.

Bootstrapping the microcontroller's serial terminal to a web-based UI is the key feature of this UI.

The dashboard source repository is here.

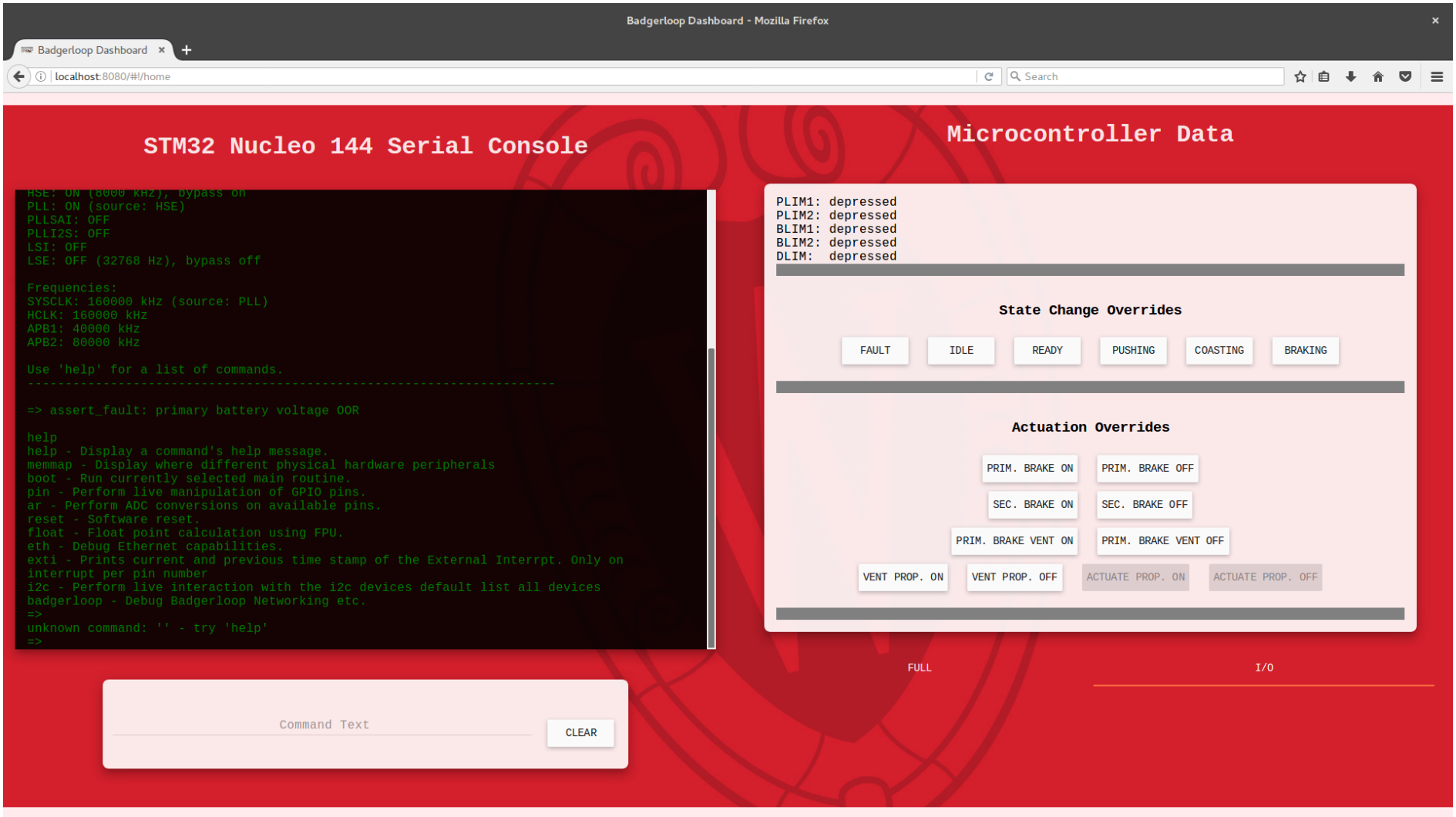
Hyperloop II: Dashboard

Console left (showing boot post and `help` command), data table right



Hyperloop II: Dashboard

Console left (showing post and `help` command), manual IO menu right



Hyperloop II: Dashboard

`memmap` output and `badgerloop` sub-commands

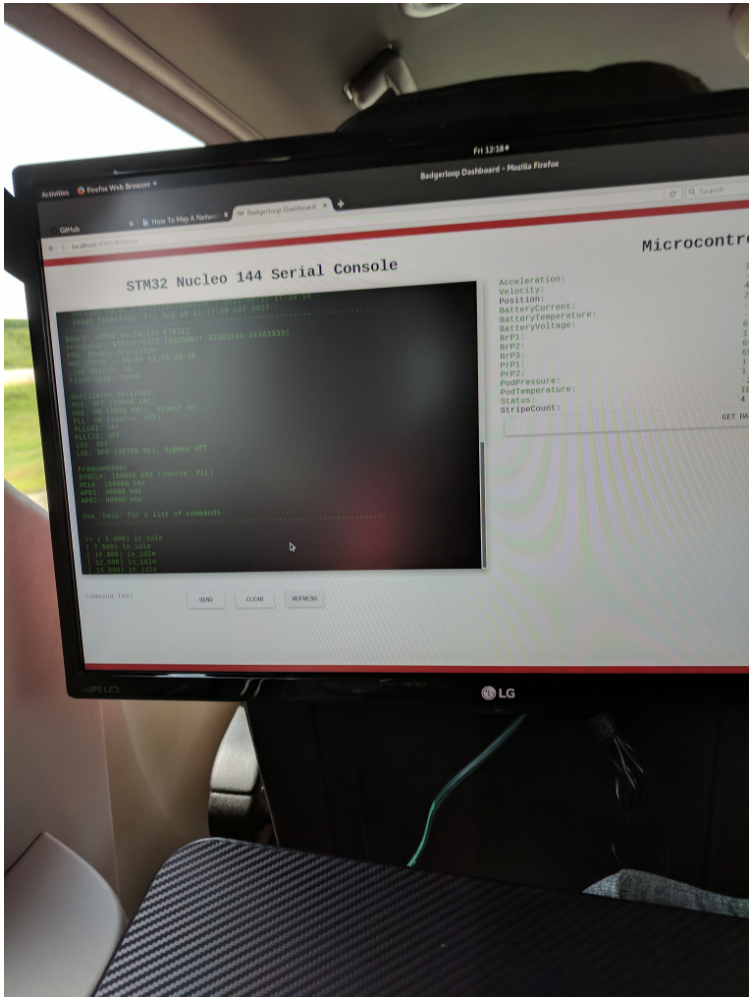
```
APB1 UART5 40005000 1023 bytes
APB1 UART4 40004c00 1023 bytes
APB1 USART3 40004800 1023 bytes
APB1 USART2 40004400 1023 bytes
APB1 SPDIFRX 40004000 1023 bytes
APB1 SPI3/I2S3 40003c00 1023 bytes
APB1 SPI2/I2S2 40003800 1023 bytes
APB1 CAN3 40003400 1023 bytes
APB1 IWDG 40003000 1023 bytes
APB1 WWDG 40002c00 1023 bytes
APB1 RTC & BKP rgstrs 40002800 1023 bytes
APB1 LPTIM1 40002400 1023 bytes
APB1 TIM14 40002000 1023 bytes
APB1 TIM13 40001c00 1023 bytes
APB1 TIM12 40001800 1023 bytes
APB1 TIM7 40001400 1023 bytes
APB1 TIM6 40001000 1023 bytes
APB1 TIM5 40000c00 1023 bytes
APB1 TIM4 40000800 1023 bytes
APB1 TIM3 40000400 1023 bytes
APB1 TIM2 40000000 1023 bytes

=> help badgerloop
Debug Badgerloop Networking etc.
Usage: badgerloop
DBTO - Don't brake timeout
MBTO - Must brake timeout
BCT - Braking count threshold
ACCEL - Accelerometer impulse cap
TEP - Target end position
CMPS - Centimeters per strip
override [ on | off ] - stop DAQ and override sensor data
fault - print current fault message
=>
```

Hyperloop II: Dashboard

`ar` (analog read) command output, raw 10-bit ADC

Hyperloop II: Dashboard

Development on the Dashboard on the road to the competition from Wisconsin



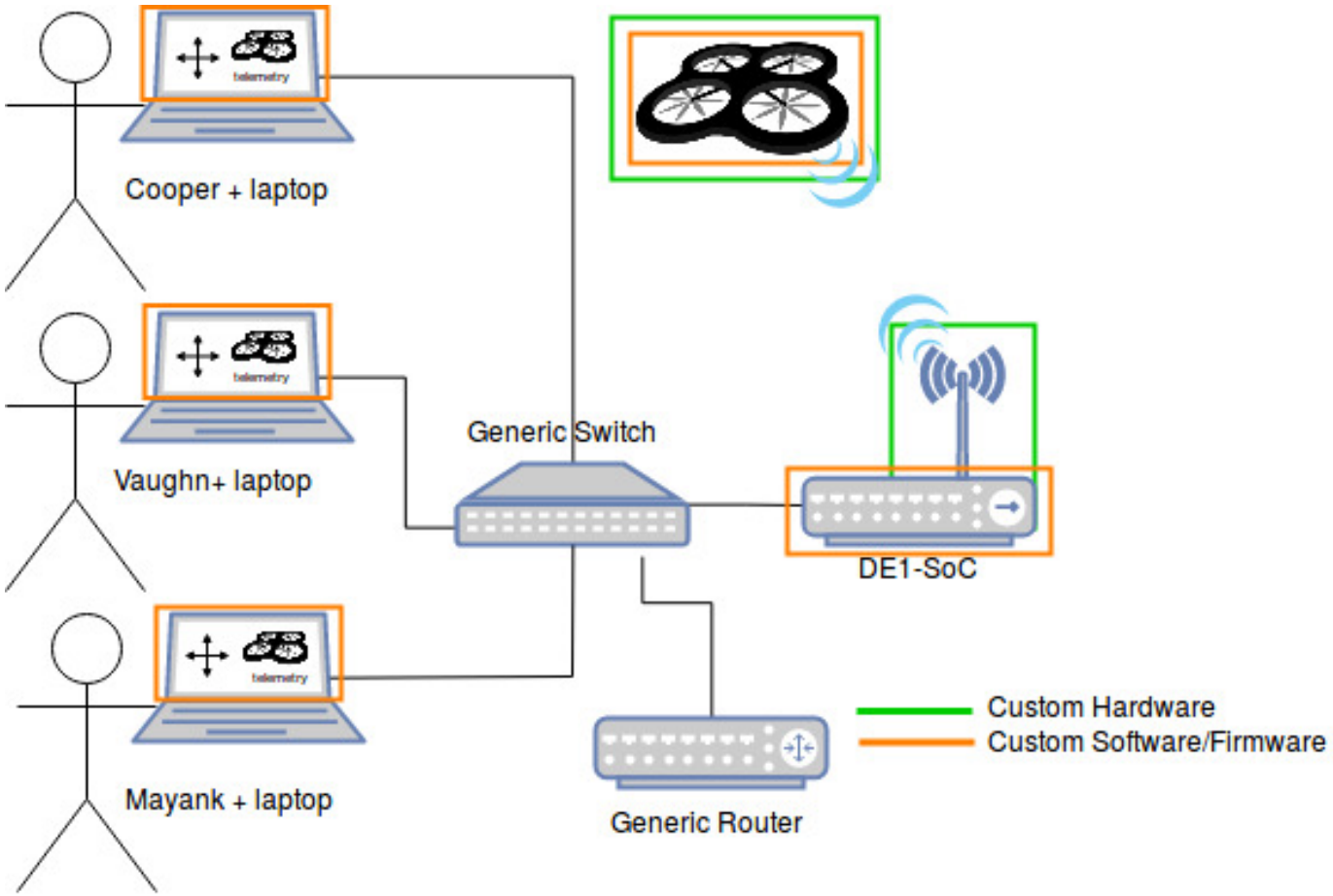
Senior Design: Overview

Planning to update this with progress, right now the finalized proposal and presentation slides are complete and we're working on keeping additional online documentation relevant.

The goal is to build a quadcopter that communicates with a ground station and can be controlled and monitored through a web-based user interface.

Design details are (currently) best captured by the proposal.

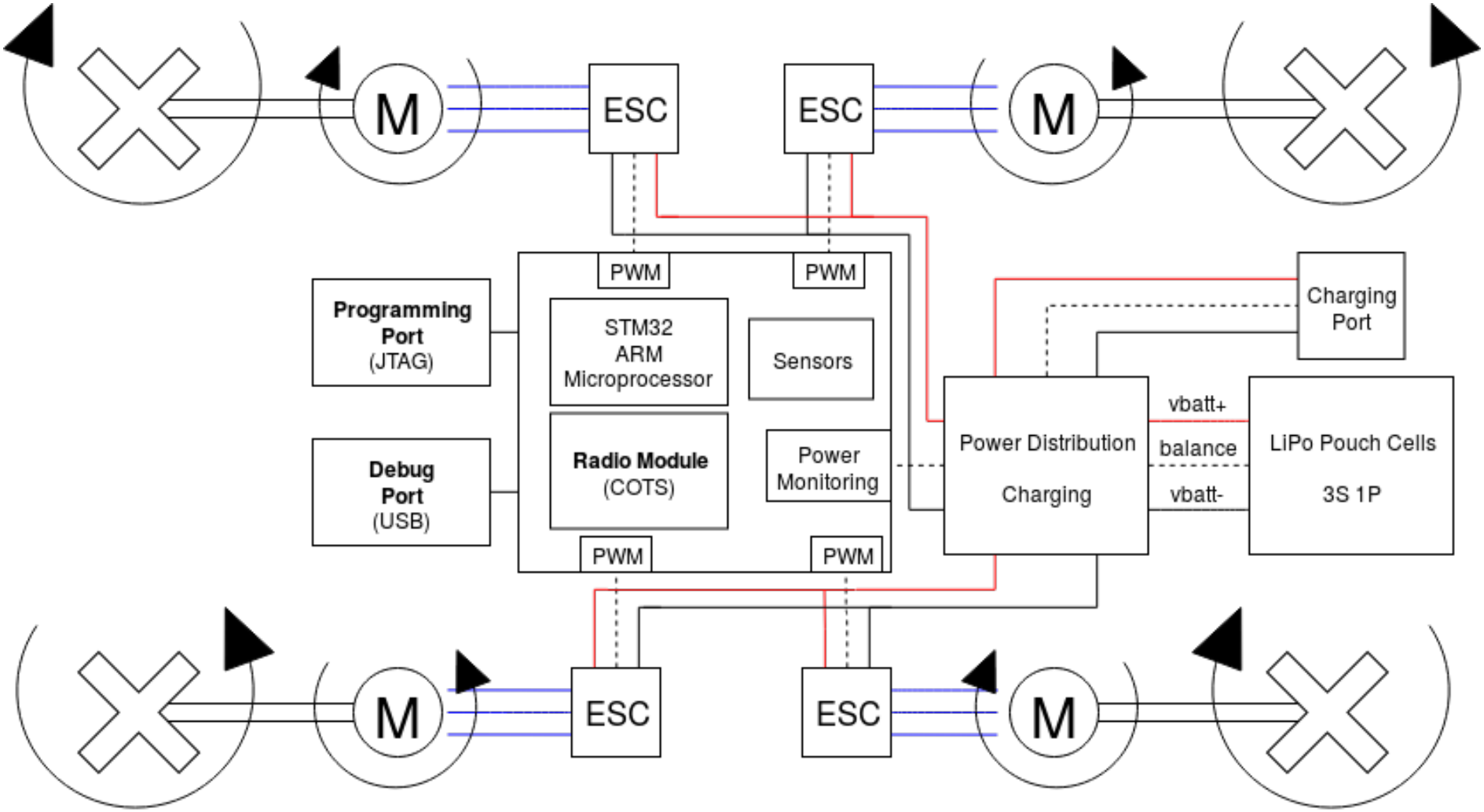
Senior Design: Concept of Operation



The essence of what we're working on

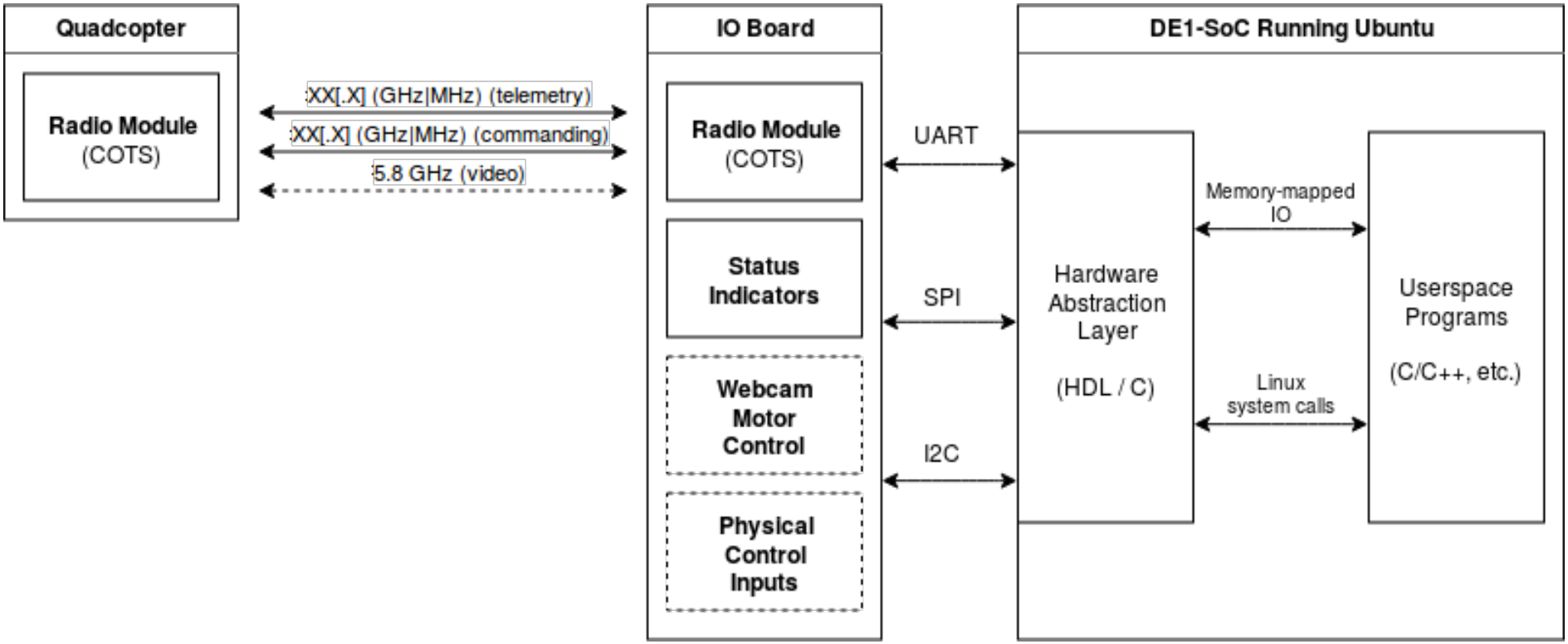
Senior Design: Block Diagram

Quadcopter physical architecture



Senior Design: Block Diagram

Ground station software architecture and components



Senior Design: Block Diagram

User interface software architecture

API Commands (A)

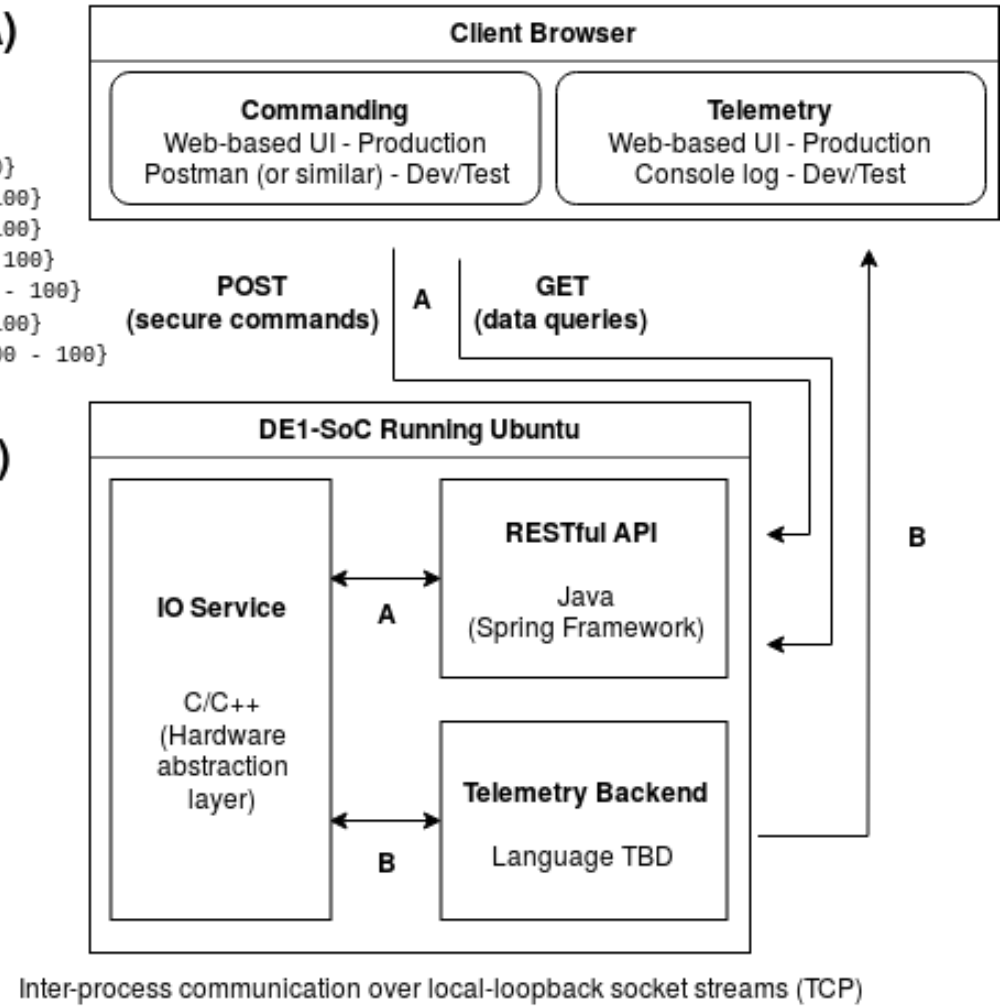
HTTP over TLS

```
https://host/move/up/{0 - 100}
https://host/move/down/{0 - 100}
https://host/move/left/{0 - 100}
https://host/move/right/{0 - 100}
https://host/move/forward/{0 - 100}
https://host/move/back/{0 - 100}
https://host/move/rotate/{-100 - 100}
...
```

Telemetry Data (B)

Secure WebSocket

```
telemetry_packet {
  timestamp: 1536646557,
  age: 15,
  type: "sensors",
  data: [
    temperature: 22,
    pressure: 101325,
    gyro: {
      rate_xy: -1,
      rate_xz:  2,
      rate_yz: -3
    }
    ...
  ]
}
```



Senior Design: Power Supply Module

12V 2A input creates 5V and 3.3V rails with "power good" LEDs, just implements the reference design for a switch-mode and linear regulator