Hyperloop Software Team 2020 **April 15th Design**

Software System Design by James Parker and Jacob Levy

Cornell Hyperloop's pod contains one embedded ODROID-C2 computer running the Ubuntu operating system, four Arduino Uno microcontroller boards each housing an ATmega328P microcontroller, and one Arduino Mega microcontroller board housing an ATmega2560 microcontroller. The four Arduino Unos are contained in the Fiducial Detector, Brake/Low-Speed module, and Front and Rear analog modules. The Arduino Mega was utilized in the Flight Control module due to the processing demands of its operationally critical task. These five modules communicate with the pod computer using USB connections carrying 100*kbaud* serial data, and the pod computer communicates with the pilot laptop over WiFi with an ESP8266 WiFi module. The ODROID utilizes information from the network state, Arduinos, and pilot computer to autonomously determine and control the state and operation of the pod.

Definition of modules:

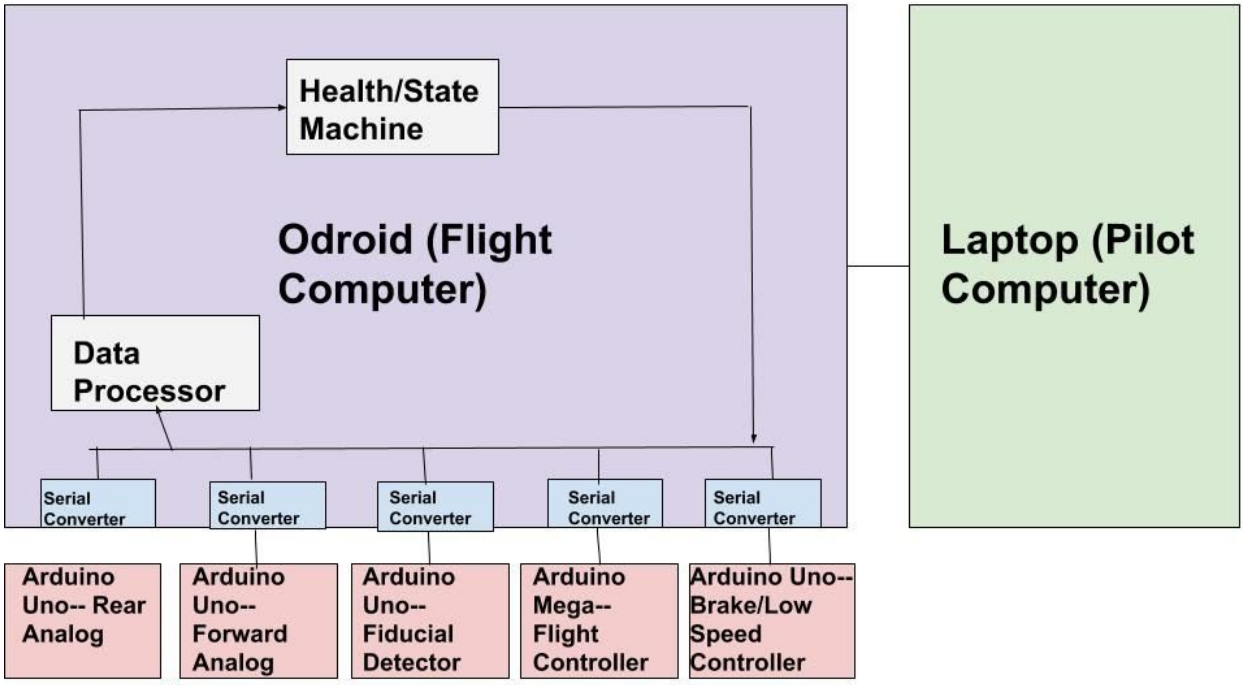
- The Fiducial Detector (FD) utilizes an optical sensor to detect the occurrence of fiducial markings along the tunnel. These marks are 4" wide and occur at 100 *ft* intervals. Assuming a relative pod speed of 110 *m/sec*, the marks will be measurable for 400 μs. Therefore, the optical sensor must have a minimum sampling rate of 5 *kHz*.

- The Brake/Low-Speed Module manages all actuation of the pod. State information from the hydraulic system controlling the brakes and the motor controllers associated with the low-speed system are sent to the ODROID for analysis. Brake caliper positions are controlled by an inner-loop controller in this module.

- The Front and Rear Analog modules communicate with a range of sensors (thermistor, potentiometer, accelerometer, etc.) across the pod's suspension, lateral control components, and power supplies.

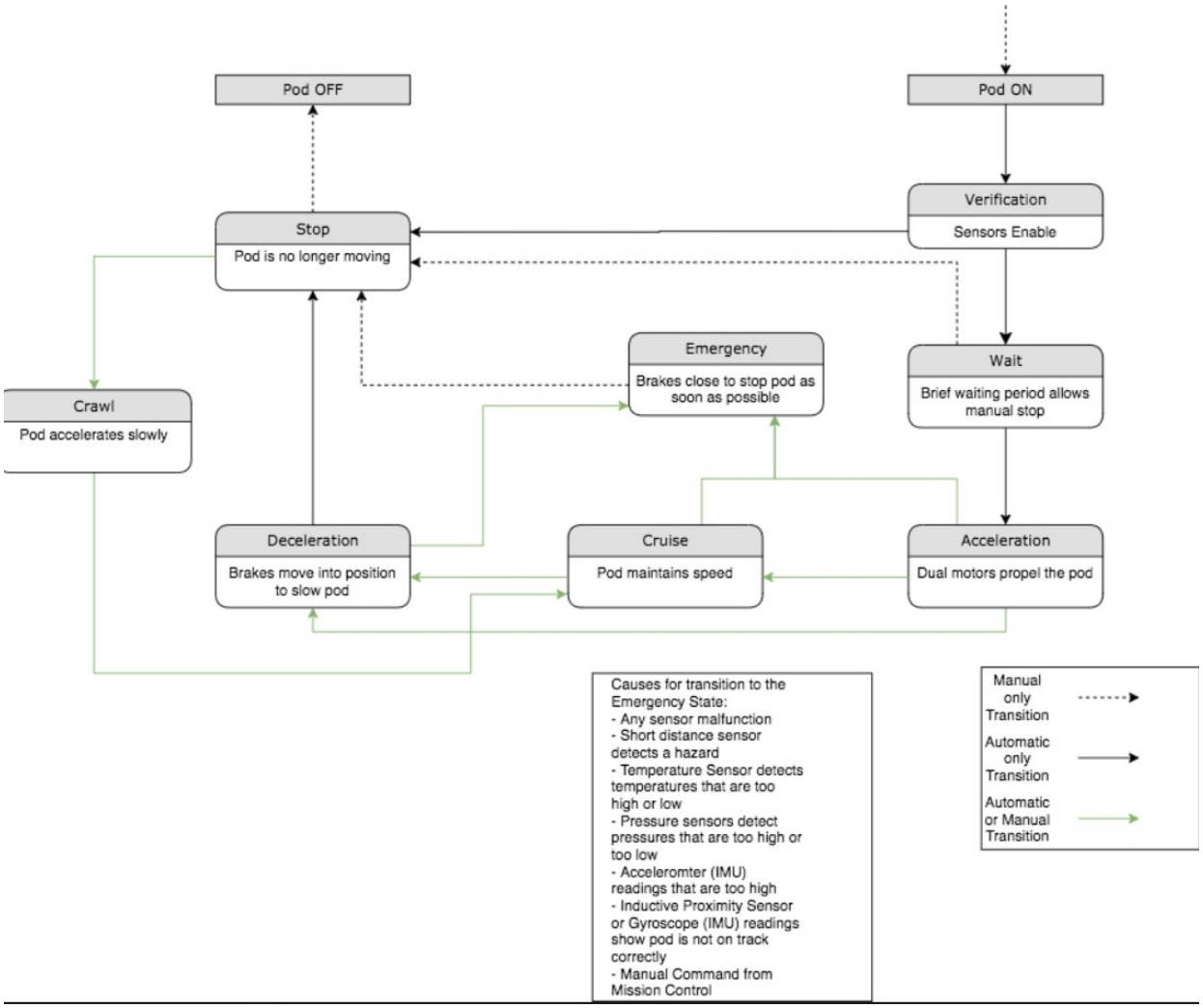
- The Flight Control Module communicates with the IMU and the Fiducial Detector to determine the current physical state of the pod. The output of the IMU is used to estimate the pod's velocity, orientation, and position. These estimates are utilized in conjunction with information from the FD to filter fiducial marker detections in an Extended Kalman Filter. The resulting estimates are then passed through a trajectory-tracking microcontroller to generate brake commands for the pod.

Diagram by Jacob Levy:



Note: Connections between Arduino and Odroid and Odroid and Pilot Computer are currently being researched by fellow software team members, so the mechanism of those connections is left blank

Updated state diagram by David Wolfers



The Pros and Cons of Using Arduinos and the Odroid by Nithish Kalpat:

Arduino

Pros

- No thermal issues
- Arduino library, tools, and community have made our work easier
- More possibilities than other microcontrollers
- All sensors/detectors function with Arduino
- Works well with thermistor temperature sensors
- Simple/easy to learn

Cons

- Processing power
- Lack of working memory

ODROID

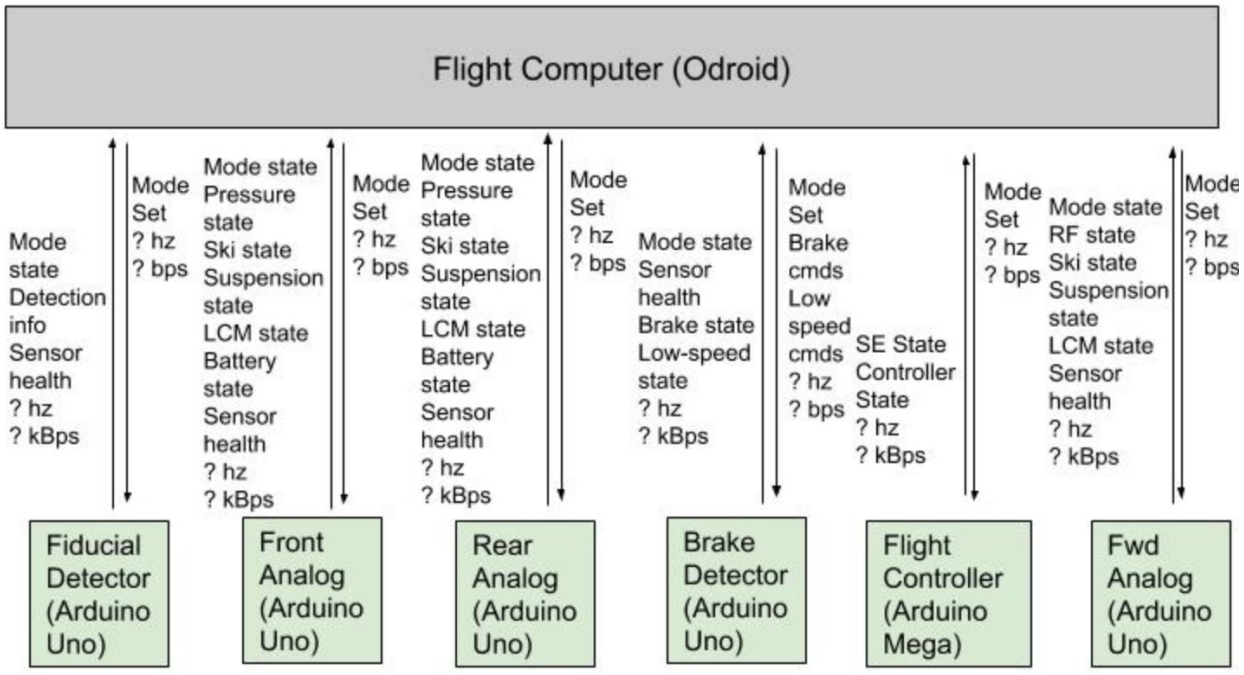
Pros

- Most powerful single-board computer at its cost
- More RAM and processing power for our needs
- Relatively small, so will fit well in pod
- Versatile, many operating systems run on it
- Linux interface is easy to use and simple
- USB ports

Cons

- Needs separate Wifi module to connect to internet
- Needs different types of cords to get power
- Some thermal issues during use

Diagram by Adeline:



Research on State estimation and diagram shown below by Christopher Yuan

State Estimation

When running on the track, our pod system is programmed to follow a certain sequence of states, as detailed on our state diagram. Successful performance of the pod demands an accurate, realtime estimate of the current state of the pod when running, as to know when to run certain commands and transition between states. The main obstacle in accurate state estimation is the time delay between sending out commands to various pod systems, and receiving sensory data on pod performance.

Kalman Filters:

Diagram shown down below.

Description

From [Wikipedia](https://en.wikipedia.org/wiki/Kalman_filter): "In statistics and control theory, Kalman filtering, also known as linear quadratic estimation (LQE), is an algorithm that uses a series of measurements observed over time, containing statistical noise and other inaccuracies, and produces estimates of unknown variables that tend to be more accurate than those based on a single measurement alone, by estimating a joint probability distribution over the variables for each timeframe. The filter is named after Rudolf E. Kálmán, one of the primary developers of its theory.

The Kalman filter has numerous applications in technology. A common application is for guidance, navigation, and control of vehicles, particularly aircraft, spacecraft and dynamically positioned ships. Furthermore, the Kalman filter is a widely applied concept in time series analysis used in fields such as signal processing and econometrics. Kalman filters also are one of the main topics in the field of robotic motion planning and control, and they are sometimes included in trajectory optimization. The Kalman filter also works for modeling the central nervous system's control of movement. Due to the time delay between issuing motor commands and receiving sensory feedback, use of the Kalman filter supports a realistic model for making estimates of the current state of the motor system and issuing updated commands."

Notes

It seems that Kalman filters are the most suitable model to filter and process the sensory feedback from the pod to estimate what state our pod is in, integrating data from multiple sources. They have also been used in previous years. MIT's 2017 FDR mentions on pg. 96 about their use of an EKF (Extended Kalman Filter).

Kalman filters are able to account for any deviations in our sensor data, namely sensor drift and inaccuracies across sensors.

Below is a diagram from a previous year's PDB of the state diagram:

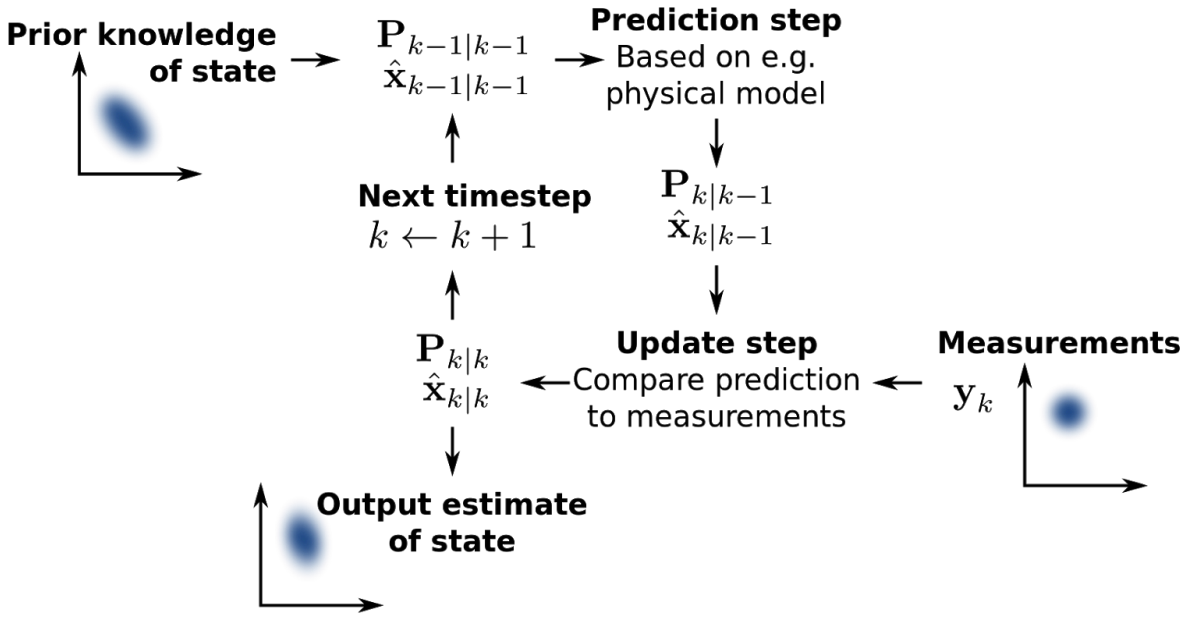
Sensor Data
Refer to the [Specs](https://github.com/cornellhyperloop/software/tree/master/Specs) folder for more information about the modules.

With data from the modules (fiducial detector, brake/low-speed module, front and rear analog modules, and our flight control module), we should be able to estimate the position, velocity and acceleration parameters of our pod that would be enough to allow us to estimate the pod state. In reference to MIT's 2017 FDR, they also only used these parameters for state estimation.

Considerations + Future Steps
We should research more into how EKFs differ from traditional Kalman Filters, and the suitability of both to our needs.

Our implementation of the Kalman Filter should include:
* measurements of response time between different points (issuing commands, receiving data, estimating state, reissuing updated commands)
* tests of overshooting time estimates
* tests of undershooting time estimates
* tests of false positives and false negatives

Links
* [MIT FDR](http://zerocm.github.io/zcm/)
* [Wikipedia](https://en.wikipedia.org/wiki/Kalman_filter)



User Interface Planning:
UI spec by Vianca Hurtado
Pages:

- Dashboard
    - Header
        - 7 columns split into:
            - Button options for Estop and Clear temetry
            - States for FSM state, BAC Mode; RADM Wheel state, RADM Clamp state; SpaceX Telemetry Status; Pod stopped; Battery 1 volts, amps, temperature; Battery 2 volts, amps, temperature
                - States should change colors -green within ideal parameters, and red when values exceed good levels
    - Left Center
        - Toggles between images of Cam Rear and Cam Front
    - Right Center
        - 13 tabs to alternate between that will show one of the following sets of information and options
            - Mode, Teleop, Low Speed, Cam Rear, Cam Front, Nav, Nav Plot, IMU X, Net, Health Test, Configs, Data Plots, Sim
                - These display all system diagnostics and sensor health
    - Footer
        - State Estimation which consists of an animation showing the pods current location on a like like plot

Inter module communications
Research done by Michael Guan

Communications
Communications protocols for passing information between the modules in the Hyperloop pod

LCM (Lightweight Communications and Marshalling)
Set of libraries and tools for passing messages and marshalling data, in systems where high-bandwidth and low latency is essential.

Benefits of LCM
* Simple C-like syntax that allows for sending packets of information
* Can be used with ZCM across Arduino serial lines as well as other embedded systems
* Little overhead and flexible with module types

LCM Logging

* The LCM package also supports logging functionality that records LCM traffic through a network
* Can be used to perform diagnostics after flight runs

Usage
* Relay information between Pilot Computer and Flight computer(Odroid C2) through LCM tunnel

Supported Platforms
* Platforms
  * GNU/Linux
  * OS X
  * Windows
  * Any POSIX-1.2001 system (e.g., Cygwin, Solaris, BSD, etc.)
* Languages
  * C
  * C++
  * C#
  * Go
  * Java
  * Lua
  * MATLAB
  * Python

Additional Packages
* libbot2
  * set of libraries, tools, and algorithms that are designed to facilitate robotics research
  * LCM utility programs including LCM tunnelling (lcm-utils)
  * Visualization of data with OpenGL and GTK2 (vis)
  * Process management tools for controlling many processes (procman)

Links
* [LCM downloads](https://github.com/lcm-proj/lcm/releases)
* [Website and documentation](http://lcm-proj.github.io)
* [libbot2] (https://github.com/libbot2/libbot2)

ZCM (Zero Communications and Marshalling)
Use to send information within Arduino serial lines. Mostly compatible with LCM

Supported Platforms
* Platforms
  * GNU/Linux
  * Web browsers supporting the Websocket API
  * Any C89 capable embedded system
* Languages

* C (89 and greater)
* C++
* Java
* MATLAB (Using Java)
* NodeJS and Client-side Javascript
* Python
* Julia (both v1.0.3 and v0.6.4)

Modules Definitions and Technical Reqs by Jack Stettner
Links
* [ZCM website](http://zerocm.github.io/zcm/)

Modules

Fiducial Detector

Inputs

- Optical Sensor
  - Data source: Arduino

Objective

- Detects optical markings within the tube at a high frequency to allow for accurate pod positioning

Technical Requirements

- TBD

Flight Controller

Inputs

- IMUs
  - Data source: Arduino
- Fiducial Detector

Objective

- Leverages sensor data to estimate pod position
- Dispatches controls to the pod to produce braking commands to ensure pod trajectory is intended

Technical Requirements

- TBD

Low-Speed Module

Inputs

- Braking Commands
- Analog sensor data

Objective

- Actuates everything on the pod including motors and brakes

Technical Requirements

- TBD

Front and Rear Analog

Inputs

- Variety of pod sensors (lateral distance, pod suspension, temperature of batteries)

Objective

- Provide data to the Low-Speed Module

Technical Requirements

- TBD