Startup Tasks

* Calibrate Sensors
  + Altimeter – use pressure reading at location
  + Accelerometers/IMU – zero out
    - On power up not on launch rail
  + Filter?
    - Min and max threshold values
  + Set origin points/axis direction/reference frame relative to magnetometer
    - Determine magnetometer offset from coordinate system
* Start onboard timer
  + Need to record when ignition occurs to be used in subsequent calculations
* Check status of ports
  + Check if output pins are HIGH, if so set them LOW

Main Tasks

* Calculate Position
  + Equations
    - Velocity
      * Assume Initial Velocity in x, y, and z are all 0
      * Vn = Voa + Vi
      * For each axis, loop through and use the accelerometer data to calculate the new velocity. Store the old velocity to be used as Vi in the next iteration.
        + Magnitude Calculation
      * Store components of velocity in a vector?
    - Position
      * Assume start at origin of x, y, z coordinate system
      * Xn = Xot + Xi (or use altimeter value)
    - Orientation
      * Assume that the angles are all 0 at origin
      * Thetan = Theta­I + Theta\*t
        + [Alternative Calculation](https://stackoverflow.com/questions/15490990/obtaining-orientation-using-gyroscope-and-accelerometer)
      * Direction:
        + Magnetometer: What kind of data is sent out?
* Check Apogee Conditions
  + Apogee Detection
    - Orientation: Either angle must be 180 degrees
    - Acceleration: Only in down direction/perpendicular to ground
    - Velocity: Velocity is zero or velocity vector horizontal/parallel to ground
    - If either condition met, set apogee flag
  + Drogue
    - Apogee flag must be set
    - x seconds after/before(?) Apogee
  + Main
    - Apogee flag must be set
    - At/below xxxx ft
* Tilt Protection
  + If orientation is greater than

**------------------------------------ draft draft draft ------------------------------------------------------**

**Functions are defined outside the loop body, or within in another .h file within the same directory**

/\*

Loop structure to behave “like” the main method

\*/

Loop(){

If (turnOn()){

do{

IsOkay();

InFlight();

}while( apogeeReached() || deployFlag);

} else {

transmitMessage( “System Check Failed”);

}

// code to reach here at end of flight

} **//end of loop**

// method make sure everything is working properly

String isOkay(){

// declared twice, once before inFlight() is called and second within inFlight()

// string returned is error message

}

// method running in flight

void inFlight(){

/\* objectives

do{

}while( // determine what should the values should be)

// writeToFile has a delimiter or icon designating when values outside of “fight time” are being recorded in the csv file . Code

}

boolean apogeeReached(float[] imuValues) {

// internal logic used to determine whether apogee has been reached or not

}

void writeToFile(float[] imuValues, int mode){

// writes values to micro SD in “packet format”

**All values will be within one row. New line, new set of data at time (t)**

|1. Sensor data (IMU, Altimeter, Accelerometer) if value is not present at (t) a “-” will take its place)

|2. Error codes (TBD)|

1. Create .csv / .txt file within setup();
2. Open csv / .txt
3. Write values in the aforementioned format to the file
4. Close file   
   }

boolean turnOn(float[] imuValues){

// method to determine if inFlight is to be called

/\*

logic for determining wether it is in flight or not

1. Sound (db over a given value indicates the rocket motor has been ignited)
2. Z axis acceleration
3. Go signal from groundstation & returning an T/F value after running the is Ok method //**my fav option**

\*/

}

void transmitMessage(String message){

// calls Arduino method to

}

// Idk how to have the onboard timer at all whilst I have other operations running