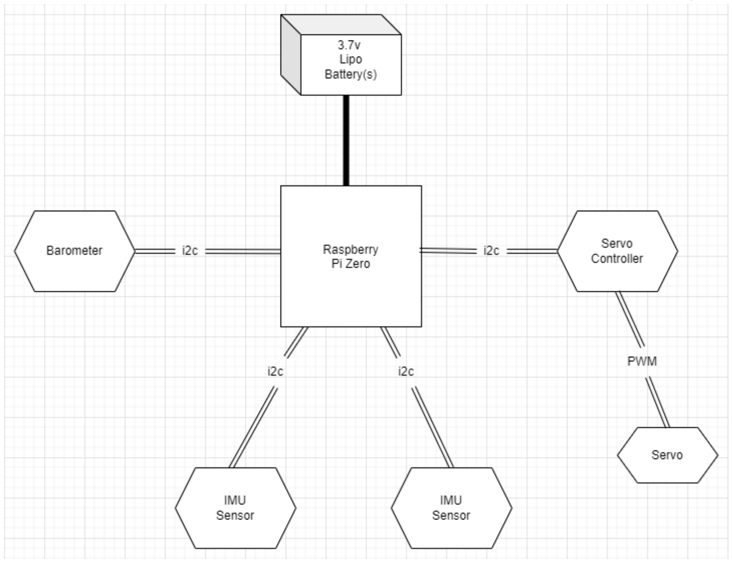
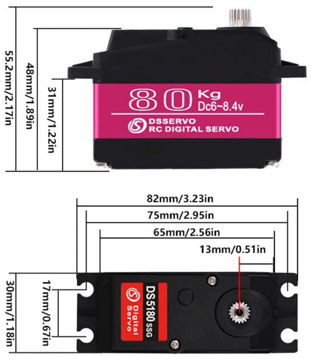
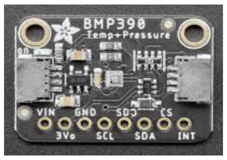
1.Software Approach

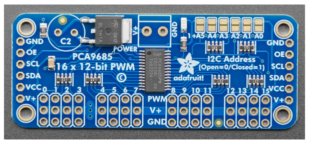
7.1 Sensors and Hardware

For the electrical implementation of the air brakes system, two Adafruit 9 DOF BNO055 IMU sensors will be used to get live velocity readings, and an Adafruit BMP390 barometer will be used for live altitude readings. The central control/processor will be a Raspberry Pi Zero, and to actuate the servo controlling the flaps, a PCA9685 Servo Driver will be used for more accurate and efficient PWM signaling. The system will be powered by a 3.7 Volt LiPo battery, and a Boost Converter may be used to up the power input to 5 Volts for the Raspberry Pi, servo controller, and servo logic. Initial testing of the hardware will be done on a breadboard, but the final electronics module to be used in the rocket will be a fully integrated PCB custom designed, printed, and soldered with all of the aforementioned components, aside from the servo. The Raspberry Pi Zero was chosen as the microcontroller of the system for its computing power ideal for the iterative RK4 predictions and PID calculations and its smaller size compared to other Raspberry Pi boards, making it easier to integrate with the rest of the electronics in the module. As the microcontroller of the system is a Raspberry Pi, the software of the control system will be written in Python 3.11.









7.2 State Machine

A state machine will be implemented in software that differentiates four states of the rocket's trajectory as they pertain to the air brakes system: burnout, active, full stop, and post-apogee. In the burnout and post-apogee states, the air brakes' flaps will be fully retracted and the servo actuation will be defaulted to zero percent, as no additional drag force is needed at these points in the rocket's trajectory. In the full stop state, they will be completely extended; at this point the target apogee has already been passed and a maximum amount of correction is needed to make this state as short as possible. While that full extension state mitigates error retroactively once the target is already passed, the real purpose of the air brakes system is to prevent passing the target apogee in the first place. So, the most critical state in the system is the active state, where the level of flap actuation will depend on predictions and calculated error correction in a PID control loop. This state will hopefully force the actual apogee to occur right around the target.

7.3 Control System

In the active state, the system will be controlled by a feedback control loop that implements a tuned PID error correction algorithm, the 4th-Order Runge Kutta (RK4) method to make predictions of the rocket's current projected apogee, and a Kalman filter to validate the input sensor data from the IMU sensors and barometer. In the code this will manifest as a main program implementing the PID loop, and two independent functions for predictions and filtering. The prediction function will directly take input data from the sensors, call the filtering function, and implement the RK4

method with the rocket's drag equation; the changing variables being current velocity, altitude, and surface area dependent on the air brake flaps. It will output a single altitude prediction per iteration of the PID loop.
A PID loop was chosen for this control system because

In the main PID algorithm, the error for each iteration will be calculated as *e* = |target apogee - current projected apogee|. The PID loop will correct this error each iteration by calculating a net corrective gain summing the current or proportional error, the integral or accumulative error since the first time step, and the derivative or rate of change of the error between the current and previous time steps. Including the approximated integral of the error controls steady state error, or the difference between the actual and target values as the system has reached an unchanging state. The derivative of the error limits overcorrection caused by the integral term, using the error's rate of change to predict future error values and adjust each iteration's net gain accordingly. The net gain or sum of these terms will be directly translated to the actuation of the air brake flaps, expanding or contracting them by a small margin each iteration, as the time steps of the PID loop will be about 0.1s. Initial *Kp*, *Ki*, and *Kd* coefficients will be initialized before the main loop, and scale the components of the net gain before it is applied to the actuation of the flaps. The values of these coefficients need to be tuned uniquely for the system and will be determined through testing.
Net gain calculation inside loop:

```
error = projected_apogee - target_apogee
integral += error * dt
derivative = (error - previous_error) / dt
PID_output = Kp * error + Ki * integral + Kd * derivative
flap_position = clamp(flap_position + PID_output, 0, 100)
```

7.4 RK4

The Runge Kutta methods are a means of solving ordinary differential equations, and they calculate the next value of a function by taking an average of slopes at different points across the current time step. They can be used to predict the value of a function at a future time step by performing this process iteratively, incrementing the time value each iteration. The 4th Order Runge Kutta Method or RK4 method evaluates the derivative of the given function at four different points within the current time step, giving more information about the behavior of the curve than the lower order Runge Kutta methods and making it more accurate. The final output value in RK4 is calculated by taking a weighted average of these four slope estimates, each scaled by a specific coefficient in the RK4 formula.

In this ABS system, the RK4 method will be used to predict the instantaneous projected apogee of the rocket within the PID control loop, essentially predicting how the rocket's altitude changes over time and finding its altitude value when its velocity hits zero. To apply RK4 and get output predictions of altitude and velocity, those values need to be treated as functions of time that depend on the forces acting on them, which will be assumed to be gravity and drag from air resistance. The ordinary differential equations modeling the altitude (h) and velocity (v) of the rocket are:

$$dh/dt = v$$
$$dv/dt = (-Fd - Fg) / m$$

Where Fd is the force of drag on the rocket, given in section ** aero, Fg is the force of gravity (9.81m/s)m, and m is mass. RK4 will estimate the altitude and velocity values over time by taking four intermediate calculations (derivatives) for each time step. So for a time step Δt, which will be around 0.1-0.5s, the state vector of velocity and altitude is:
$y = (h, v)$ and the derivatives of the state vector are:

$$dy/dt = (dh/dt, dv/dt) = (v, (-Fd - Fg)/m)$$

Applying the RK4 equations and updating the state vector, done each time step/iteration:

$$k1 = f(t, y) = (v, (-Fd - Fg)/m)$$
$$k2 = f(t + \Delta t/2, y + (k1 \cdot \Delta t)/2)$$
$$k3 = f(t + \Delta t/2, y + (k2 \cdot \Delta t)/2)$$
$$k4 = f(t + \Delta t, y + k3 \cdot \Delta t)$$

y_next = y + Δt/6 (k1 + k2 + k3 + k4)

This y_next value contains the prediction of altitude and velocity for one time step, so to get the predicted apogee value, this process will be iterated in a loop until the velocity value hits zero:

```
while v  > 0:
        h, v = rk4_step(t, h, v, flap_position)
        t += dt
```

At the end of this loop, *h* will contain the apogee prediction.

7.5 Kalman Filter

A Kalman filter combines measurements from a system's sensors with predictions from the system's mathematical model to produce an estimate that is more accurate than using either of those alone. This will be essential in ensuring the integrity of the velocity and altitude data we are getting from the BNO055 and BMP390 sensors, as at the speed the rocket is moving noise is expected to occur. The mathematical model equation uses the raw and noisy measurements to estimate the state of those variables with greater accuracy, and the equations can be broken down into two steps: prediction and correction. The prediction step predicts the current state of the system, i.e. the altitude and velocity values, based on the previous state and the equation of motion for the rocket. It updates the error covariance matrix, which represents how uncertain the filter is about its prediction. In the correction step, the input of new measurements from the sensors correct the predicted state, and the error covariance matrix is adjusted based on the difference between the predicted state and the actual measurement.

1. At the first measurement, initialize system state estimates of velocity and altitude, and the system state error covariance matrix P_1:
   $v_1,\ h_1,\ P_1$

2. At the second and for all subsequent measurements, reinitialize these values:
   $v_2,\ h_2,\ P_2$

3. For each measurement, predict the system state and system error, where A is the state transition matrix assuming velocity directly impacts change in altitude and velocity evolves based on drag and gravity,, and Q represents process noise or uncertainty for the system model as it fluctuates in its accuracy; the system's actual accelerations and decelerations contribute to this error:

   $A=((1, \Delta t), (0, 1))$
   $x_p = Ax_{k-1}$
   $P_p = AP_{k-1}A^t + Q$

4. Compute the Kalman Gain, where H is a state-to-measurement matrix to convert the system state estimate from the state space to the measurement space.

   $K = P_P H^T (HP_P H^T + R)^{-1}$

5. The Kalman Filter uses the Kalman Gain to estimate the system state and error covariance matrix for the time of the input measurement. After the Kalman Gain is computed, it is used to weight the measurement appropriately in two computations. Estimate new system state and system state error covariance matrix, where z_k is a measurement vector of the sensor data containing both altitude and velocity.

   $x_k = x_p + K(z_k - Hx_p)$

$$P_k = P_P - KHP_P$$

At the end of these steps, the state vector x_k holds v_k, the updated velocity estimate, and h_k, the updated altitude estimate, and these are the values that will be returned by the filtering function. The P_k error covariance matrix will not be directly used in the control loop, but will be internally used by the Kalman filter to weigh future measurements and predictions.