# Implementation of Parallel Multi-Core Algorithm for Airbrake Deployment on Experimental Rockets

**Oscar H. Baños-Mancilla**

Facultad de Ingeniería
Universidad Nacional Autónoma de México, Mexico City
oscar.banos@ingenieria.unam.edu

**Keywords:** Multi-core, FreeRTOS, Airbrake, Experimental Rocketry.

## Abstract

This works covers the implementation of a parallel multi-core algorithm for computing the ideal altitude to deploy an airbrake of an experimental rocket. The mathematical model and analitic equations are not subject of study, however they are presented as context to the reader. The implemetation was developed using Pico-SDK and FreeRTOS for the RP2040.

## 1 Introduction

In experimental rocketry, an airbrake is a mechanical device designed to increase aerodynamic drag during flight, primarily to control the rocket's velocity and improve trajectory accuracy. It is commonly used to limit the apogee by deploying flaps or plates into the airflow, thereby slowing the rocket during ascent. This is especially useful in precision-targeted missions, such as those in rocketry competitions where a specific altitude must be reached. Airbrakes are typically controlled by an onboard flight computer that uses real-time sensor data to adjust deployment dynamically via actuators like servos or linear motors.

## 2 Mathematical Model

The simplest way to simulate the effects of the airbrake is through a one-degree-of-freedom simulation, where the rocket follows a perfectly vertical flight path[1].

$$\begin{cases} \dfrac{\mathrm{d}^2h}{\mathrm{dt}^2} = \dfrac{T(t) - D(\frac{\mathrm{d}h}{\mathrm{dt}}, C_d, A, \rho) - W(m)}{m(t)} \\ \dfrac{\mathrm{d}m}{\mathrm{dt}} = -\dot{m}(t) \end{cases}$$

Where:

$T \rightarrow$ Thrust produced by the motor
$W \rightarrow$ Rocket's weight
$D \rightarrow$ Drag force
$C_D \rightarrow$ Rocket's drag coefficient
$A \rightarrow$ Rocket's cross section area
$\rho \rightarrow$ Atmosphere's density
$m \rightarrow$ Rocket's mass
$\dot{m} \rightarrow$ Mass flow of the motor
$h \rightarrow$ Rocket's altitude
$t \rightarrow$ Time

An estimate of the rocket's flight trajectory can be obtained through the numerical solution of the system of ordinary differential equations presented. One of the most widely used techniques for this purpose is the Euler method, a first-order numerical integration scheme for solving ordinary differential equations. This approach is commonly employed in the simulation of one-degree-of-freedom models, as the presence of the drag force introduces nonlinearity into the system. Furthermore, the propulsion force is typically not described by an analytical function; instead, it is derived from experimental data obtained during a static fire test.

Despite being a nonlinear model, the free-flight model (i.e., in the absence of propulsion) admits an analytical solution under the condition that the velocity remains positive, corresponding to the rocket's ascent phase. The moment in time at which free flight starts is commonly known as M.E.C.O. (Main Engine Cut Off). The following expression[2] determinines the apogee given fixed parameters of$\rho$, $C_D$ and $A$ at current values of $h_0$ and $v_0$.

$$h_{max} = \frac{m}{\rho C_D A} \cdot \ln\left(1 + \frac{v_0^2}{V_t^2}\right) + h_0 \tag{1}$$

Considering

$$V_t = \sqrt{\frac{2mg}{\rho C_D A}}$$

Where:

$V_t \rightarrow$ rocket's terminal velocity
$v_0 \rightarrow$ initial velocity at free flight
$h_0 \rightarrow$ initial height at free flight

## 3 Apogee Optimization

For an airbrake to fulfill its purpose, it is necessary that the rocket is initially capable of naturally exceeding the target apogee. This ensures that the airbrake can induce a reduction

in the maximum altitude, minimizing the error between actual and target apogees. Hence, previous to the implementation of the algorithm it is necesarry to simulate two possible apogees, one without deploying the airbrake, and one with the airbrake deployed since the beginning of free flight. The range of altitude values between this two simulated apogees determines our posible target apogees.

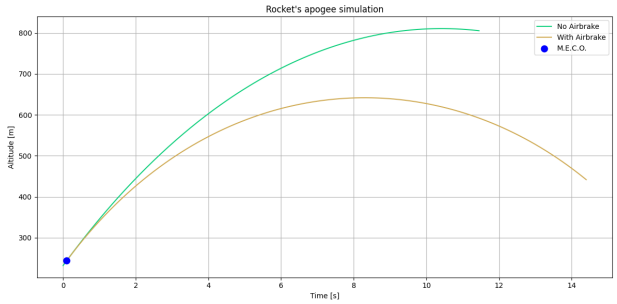


Figure 1. Minimum and Maximum Apogees

### 3.1 Mission Objective

Given the simulated apogees, we are free to determine a target apogee between $\pm 650m$ and $\pm 800m$. For the present work, an apogee of $700m$ is chosen.

A simulation using (1) was used to determine the exact altitude at which the airbrake should be deployed. This value was subsequently used to measure the error of our implementation relative to the simulation.

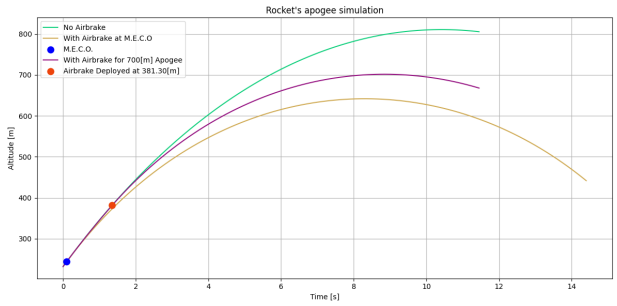


Figure 2. Airbrake for 700m apogee

## 4 Implementation

An experimental rocket mission involves the launch and successful recovery of the vehicle, in which an onboard flight computer plays a very important role, since it is in charge of deploying the main parachute and in this specific case, it is responsible for activating the airbrake at the optimal moment.

Based on this, the flight computer should be at least be capable of:

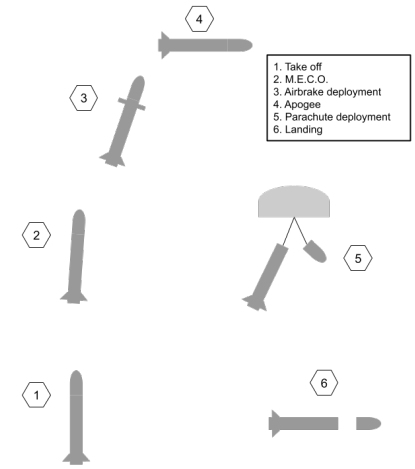


Figure 3. Mission Stages

- Reading sensors for estimating altitude, total acceleration and velocity
- Calculating the optimal moment for activating the airbrake
- Deploying the main parachute shortly after apogee is reached

### 4.1 General Overview

To meet these requirements, it was decided to use an RP2040 due to its multicore technology and FreeRTOS support.

Core 0 - runs FreeRTOS and is in charge of 3 concurrent tasks: reading sensor data via I2C bus, deploying main parachute for succesful recovery and determining the start of the free-flight phase.

Core 1 - only after the free-flight phase has started, recieves the sensor data via a pipeline between both cores and starts computing (1) to determine the exact altitude for activating the airbrake. When such altitude is reached, it activates it.

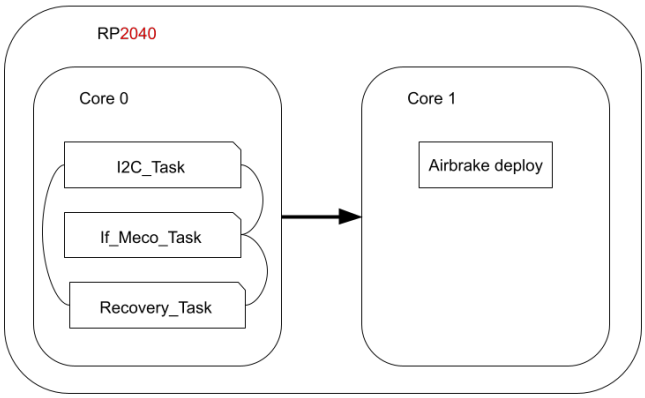


Figure 4. Multi-Core Software Design

To validate de system, a Hardware-In-The-Loop simulation

was executed, with a set of experimental data previously sampled at a frequency of 2.4[Hz].

### 4.2 I2C Task

This data set was uploaded into a second RP2040 serving as an I2C slave. The master askes for an 8 byte data frame, containing altitude and total acceleration of the rocket. This is achieved with a Task using FreeRTOS repeating every $\frac{1}{f_s}[s]$ to simulate the sampling frequency.

Two shared variables were used for saving altitude and total acceleration. A mutex was implemented every time this values were updated.

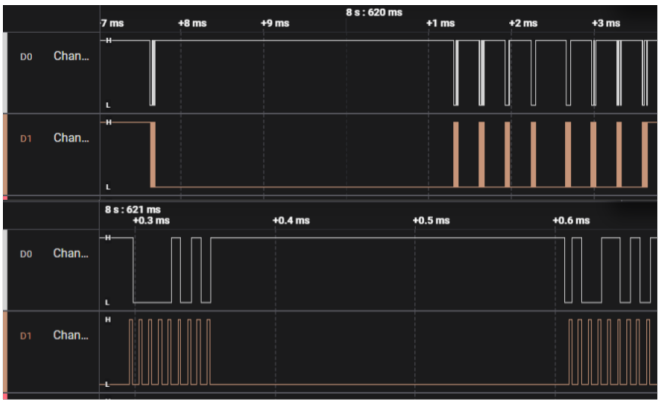


Figure 5. I2C Communication

### 4.3 Recovery Task

Constantly updates the rocket state based on the meassurements read by I2C bus.

1. On Ground
2. Ascending
3. Descending

If current altitude is less than previous altitude the rocket is descending, then, main parachute is deployed via a GPIO signal.

### 4.4 Detecting M.E.C.O. Task

As previously stablished, analitic solution presented in 1 is valid only during free-fight phase, which starts inmediatly after M.E.C.O. hence the importance of determining this stage.

Thanks to the physics of vertical motion, we know when the fuel has run out, there is no longer a positive thrust force along the vertical axis. As a result, there is no propulsion-induced acceleration. Although the rocket continues ascending due momentum previously acquired, the only forces acting on the vehicle are drag and gravitational acceleration, both of which are negative along the vertical axis. Therefore, it is possible to detect M.E.C.O. when total acceleration starts decreasing.

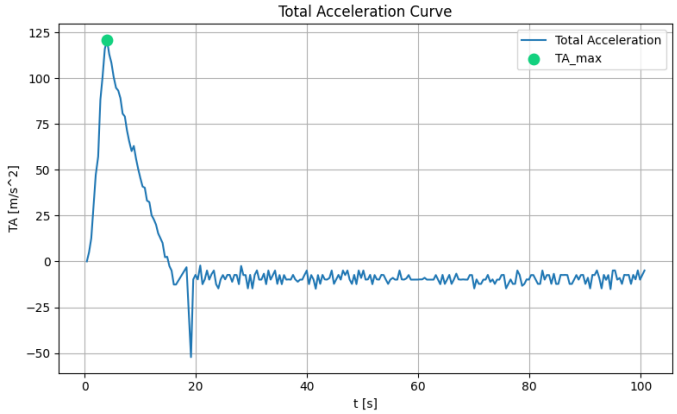


Figure 6. Total Acceleration

Based on this, a simple algorithm was implemented within a FreeRTOS Task that compares currenr total acceleration and previous total acceleration. Only after M.E.C.O. is detected, values for current altitude, acceleration and velocity are passed to Core 1 using the FIFO implemented by multicore library included in the Pico SDK.

```
if(meco){
    mutex_enter_blocking(&mutex);
    multicore_fifo_push_blocking(*(uint32_t *)&svHeight);
    multicore_fifo_push_blocking(*(uint32_t *)&svVelocity);
    mutex_exit(&mutex);
}
```

Figure 7. Multicore FIFO

### 4.5 Airbrake Deploy Function

If data has been pushed into the FIFO, Core 1 starts computing (1) with the rocket parameters according to the airbrake. With a simple comparison, if the calculated altitud is already grater than the target apogee, Core 1 activates the airbrake via a GPIO signal.

```
while(true){
    uint32_t raw_h0 = multicore_fifo_pop_blocking();
    uint32_t raw_v0 = multicore_fifo_pop_blocking();

    float h0 = *(float *)&raw_h0;
    float v0 = *(float *)&raw_v0;

    h_max = (m / (rho * C_d_AB * A_AB)) * log(1 + (pow(v0,2)) / (pow(Vt_AB,2))) + h0;

    if(h_max > apogee && airbrake_state == NOT_DEPLOYED){
        gpio_put(AIR_BRAKE, 1);
        airbrake_state = DEPLOYED;
        printf("------------------------------------------------------------------\n");
        printf("Airbrake desplegado a: %.2f\n", h0);
        printf("------------------------------------------------------------------\n");
    }
}
```

Figure 8. Target Apogee Computation

## 5 Results

As previously mentioned, the dat aset available for the Hardware-In-The-Loop simulation was sampled at 2.4[Hz], extremely slow for evaluating multicore performance, hence, this dataset was modified with polynomial interpolation to simulate a higher sampling frequency.

### 5.1 Single-Core Implementation

To meassure performance of the system first it was tested as a single-core application, without Core 1, and implementing Airbrake Deploy Function as another task running on Core 0. Data sets simulating 40, 60 and 80 Hz were used. When a 100 Hz was used, meassurements were inconsistent, resulting in airbrake not being deployed.

- Simulated Airbrake Deployment Altitude: 381.30m

| Sampling Frequency | Airbrake Deployment Altitude | % Error |
|---|---|---|
| 40 Hz | 401.2 m | 5.27% |
| 60 Hz | 395.6 m | 3.59% |
| 80 Hz | 388.6 m | 1.9% |
| 100 Hz | not deployed | 100% |

Table 1. Single-Core Implementation Results

### 5.2 Multi-Core Implementation

This implemetation was tested starting with 100 Hz data set. It is to be noted, however, that decreasing error is not a consequence of parallel computing but of higher sampling frequency and thus, a better interpolation. Performance is to be meassured based on how fast and implementation can sample.

- Simulated Airbrake Deployment Altitude: 381.30m

| Sampling Frequency | Airbrake Deployment Altitude | % Error |
|---|---|---|
| 100 Hz | 386.17 m | 1.27% |
| 130 Hz | 384.22 m | 0.75% |
| 150 Hz | 382.95 m | 0.43% |
| 170 Hz | 382.30 m | 0.26% |
| 200 Hz | not deployed | 100% |

Table 2. Multi-Core Implementation Results

## 6 Conclusion

It is clear that with a multi-core implementation we can almost double up the sampling frequency without obtaining inconsistent behaivour of the system.

Thanks to both FreeRTOS and MultiCore the complexity of this project was the managing of shared variables, but, with mutexes correctly implemented all problems were solved.

However, funcions running in Core 1 were executed concurrently, so there is a big window for improvement when using FreeRTOS SMP. Also, the solution presented in this work does not include saving data into a memory nor telemetry capabilites, both equally important when working on experimental rocketry. Including concurrent tasks managing those two requirements would have considerably reduce the performance of the system, so real symmetric multi-processing would be necesarry in order to eliminate bottlenecks resulting on sending telemetry and data saving functions.

## Anex

- GitHub repo: https://github.com/harun2301/Proyecto-Final—PMN.git
- Evidence: https://youtu.be/CZTHzgFmAiE

## References


[1] G. P. Sutton, “Rocket Propulsion Elements,” *Wiley*, 2001.

[2] S. Polo, “Implementación de un Airbrake,” *Propulión UNAM*, 2023.