Texas State's First Satellite Mission

An Independent Study in Pleiades Maia's Software Development

Author: Taylor Gilg
Advisor: Dr. Mylene Queiroz de Farias

April 2025

CS4395.251

The Pleiades Five Mission

- Pleiades 5 is an open-source, collaborative project between six universities that aims to construct, communicate with, and control small, Earth-orbiting satellites with more accessible and cost-effective embedded measuring devices.
- The long-term goal is to combine extensive documentation, software tools, and small satellite engineering materials into an educational kit that will enables high schools and colleges without established space labs to gain hands-on experience with space mission development, satellite development, and a finished, space-ready satellite in a matter of months.

TXST TEXAS STATE

Cal Poly Pomona

NORTHEASTERN UNIVERSITY 18 98 LVX VERITAS VIRTVS

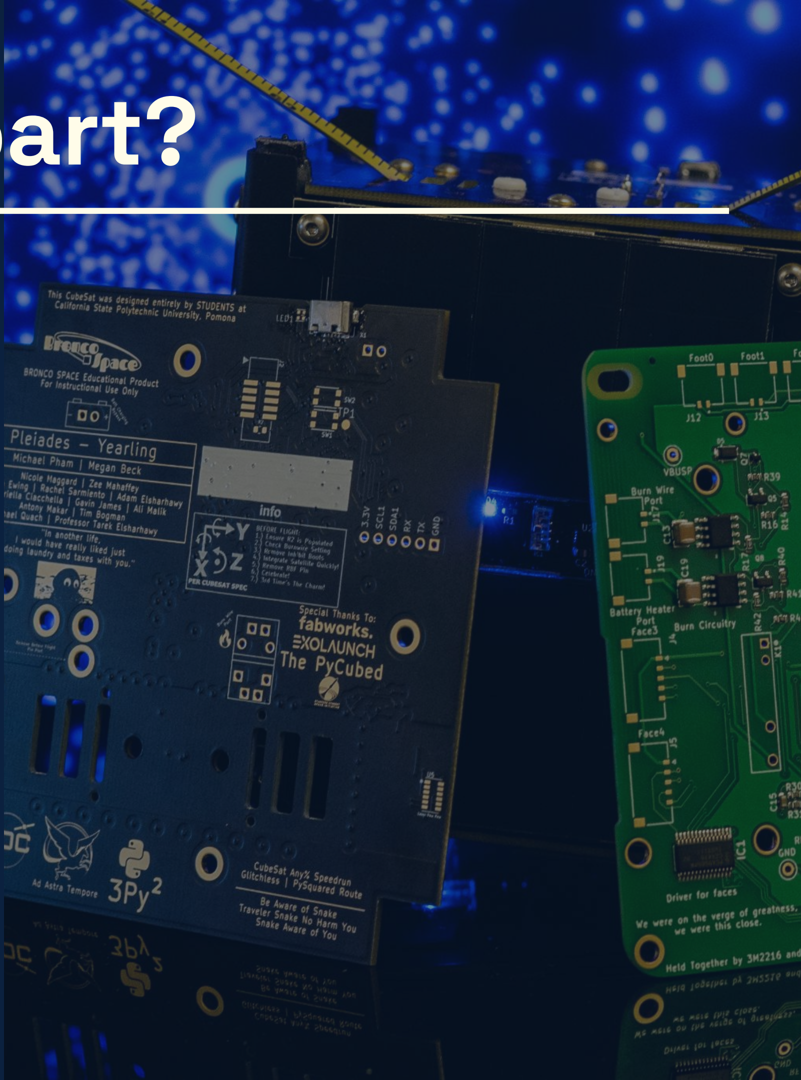
MT. SAC Mt. San Antonio College

UC SANTA CRUZ

COLUMBIA UNIVERSITY

What Set Pleiades 5 Apart?

- The collaborative and educational nature of the mission has informed the choice to make the design not only straightforward and easy to work with, but as modular and customizable as possible.
- The payload, various measuring devices and sensors, will be able to be easily switched out and replaced.
- This makes missions with the kit endlessly customizable to different research questions and able to claim valuable novelty, something really important when it comes to landing grants, research opportunities, and further experiences in the field and industry.


This CubeSat was designed entirely by STUDENTS at California State Polytechnic University, Pomona
Bronco Space
BRONCO SPACE Educational Product For Instructional Use Only
Pleiades – Yearling
Michael Pham | Megan Beck
info
Special Thanks To: fabworks. EXOLAUNCH The PyCubed
Ad Astra Tempore
3Py2
CubeSat Any% Speedrun Glitchless | PySquared Route
Be Aware of Snake Traveler Snake No Harm You Snake Aware of You
Burn Wire Port
Battery Heater Port
Burn Circuitry
Driver for faces
We were on the verge of greatness, we were this close.
Held Together by 3M2216 and

What Set Pleiades 5 Apart?

NASA Selects New Round of Candidates for CubeSat Missions to Station

- With the help of the Pleiades project and similar collaborations of its kind, the construction cost of small satellite CubeSats has been reduced from approximately $50,000 to around $3,000 currently on the Pleiades project, although the goal is to go even lower to $1000.
- Despite construction not being the largest cost involved in a satellite mission– that would be the launch provider and environmental testing – Pleiades 5's launches and preliminary tests will be covered by NASA's CubeSat Launch Initiative (CSLI)

Pleiades Maia CubeSat

- Pleiades Maia's satellite is a 10×10×10cm cube called a CubeSat.
- For its size, it is referred to as a 1U, larger satellites being multiple U large.
- Inside the aluminum frame sits a flight controller board, battery board, and payload of sensors and measurement devices.
- The satellite's antenna attaches to the external flight board and each side is made up of solar panel boards, one side also including a camera.

Intercepting with the ISS

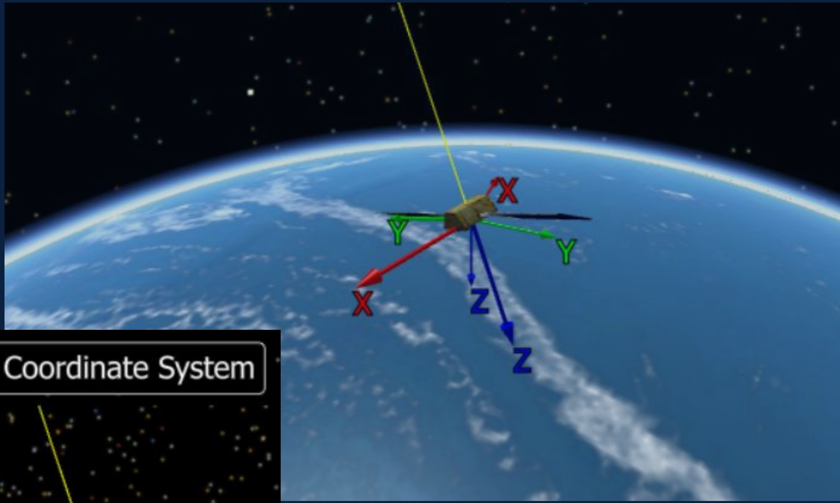
- Pleiades Maia's launch will have Texas State's satellite intercept with the International Space Station (ISS).
- Here, the astronauts on board will eject the craft from a specialized launcher built into the space station to deliver the satellite to its final destination in low earth orbit.

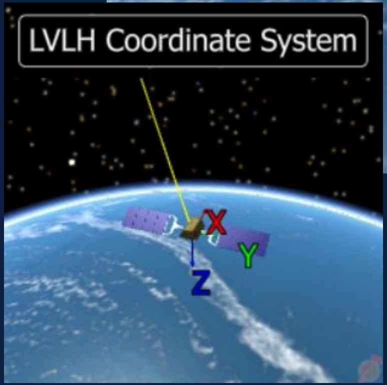
VOYAGER

Pleiades Maia: Attitude Control On A Budget

- Pleiades Maia's primary goal is to control satellite orientation with easily accessible, cost effective sensors.
- Traditional methods rely on either:
  - Star/earth imaging + complex algorithms
  - Satellite communication networks

    <u>Both very resource intensive!</u>
- We plan to use:
  - Gyroscope – rotational speed
  - Magnetometer – magnetic field direction (craft and earth)
  - Light sensors – sun position
  - Magnetorquers – apply torque using Earth's magnetic field





Team Contributions

Infrastructure & Tooling

CI pipelines & Pre-commit hooks
(automated testing & formatting)

PR Template

Make tool integration

Intellisense

Architecture & Code Quality

Refactored structure for modularity & maintainability

Introduced ADRs, type hinting, dependency injection

Config to replace hardcoded values

Testing & Documentation

Introduced PyTesting

Config checker

Developed structured logger

Began MkDocs-based documentation site

Team Contributions

Embedded Work & Firmware

Reintroduced SD card support & asyncio

Refactored NVM bit flags

Upstreamed firmware to CircuitPython

State of Health Reporting

Streamlined firmware onboarding

Ground Systems & Communication

Began ground station software development

Gained amateur radio certifications

Refactoring radio test

Outreach & Team Culture

Weekly demos, onboarding, & technical interviews

PyTexas conference presentation

Designed & preparing mission patches

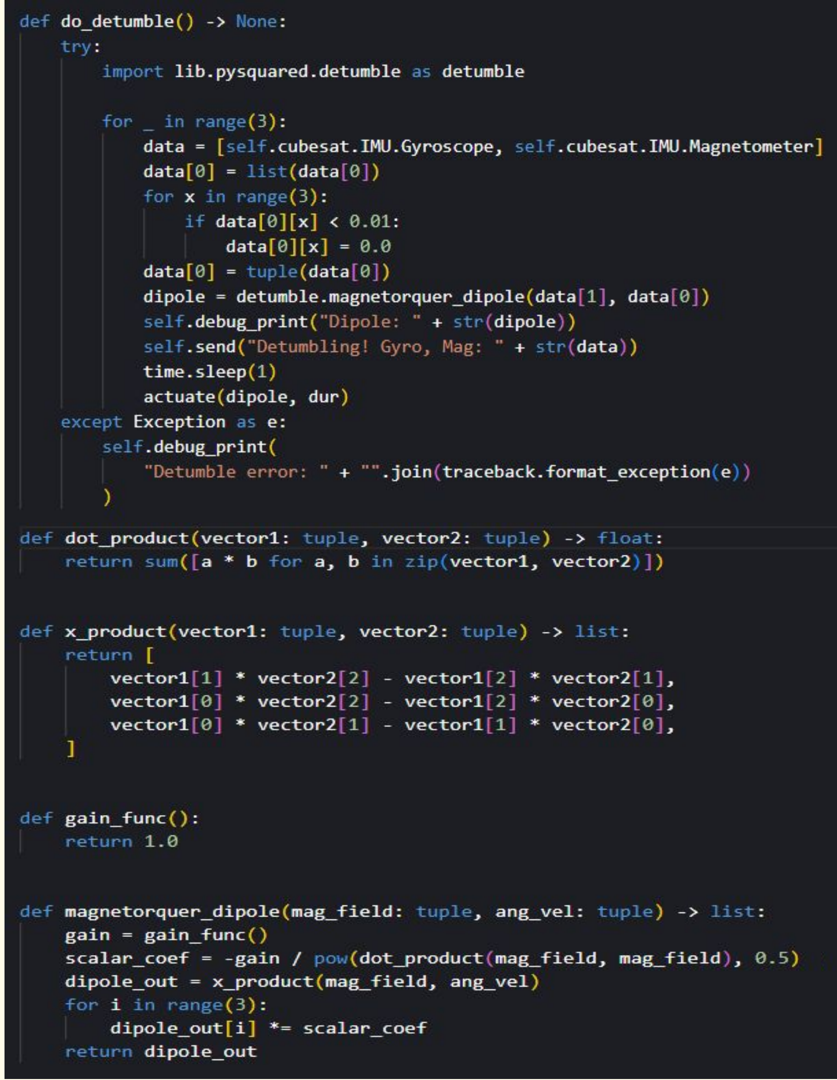
Introducing Pytests to PROVES kit

Focused on detumbling algorithm: stabilizing satellite orientation post-launch

- Reads gyroscope & magnetometer data
- Calculates torque via magnetorquer_dipole() to guide attitude control

Additional Contributions:

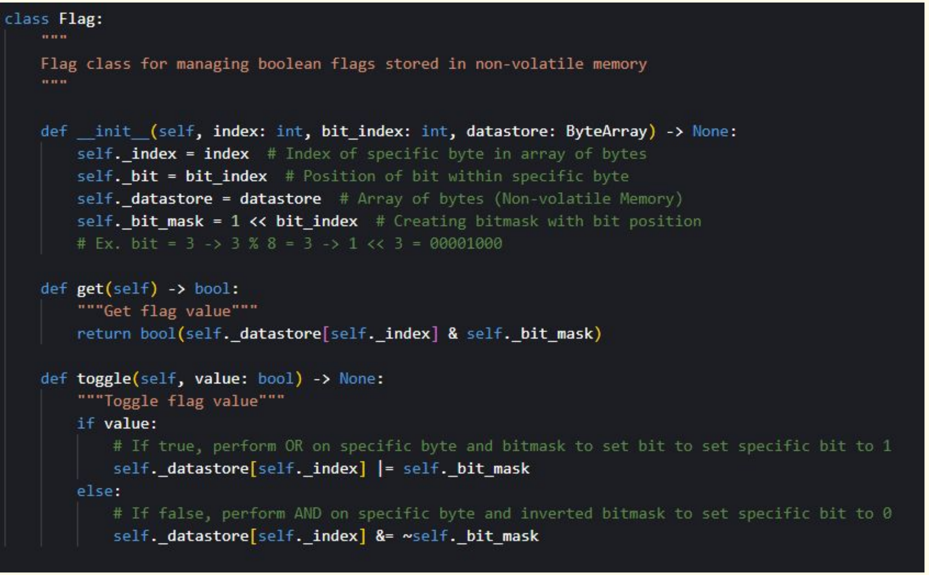
- Automated tests via Make tools (make test)
- Simulated realistic & edge-case sensor values
- Helped serve as a resource for other team members with test writing



```python
def do_detumble() -> None:
    try:
        import lib.pysquared.detumble as detumble

        for _ in range(3):
            data = [self.cubesat.IMU.Gyroscope, self.cubesat.IMU.Magnetometer]
            data[0] = list(data[0])
            for x in range(3):
                if data[0][x] < 0.01:
                    data[0][x] = 0.0
            data[0] = tuple(data[0])
            dipole = detumble.magnetorquer_dipole(data[1], data[0])
            self.debug_print("Dipole: " + str(dipole))
            self.send("Detumbling! Gyro, Mag: " + str(data))
            time.sleep(1)
            actuate(dipole, dur)
    except Exception as e:
        self.debug_print(
            "Detumble error: " + "".join(traceback.format_exception(e))
        )
```

```python
def dot_product(vector1: tuple, vector2: tuple) -> float:
    return sum([a * b for a, b in zip(vector1, vector2)])


def x_product(vector1: tuple, vector2: tuple) -> list:
    return [
        vector1[1] * vector2[2] - vector1[2] * vector2[1],
        vector1[0] * vector2[2] - vector1[2] * vector2[0],
        vector1[0] * vector2[1] - vector1[1] * vector2[0],
    ]


def gain_func():
    return 1.0


def magnetorquer_dipole(mag_field: tuple, ang_vel: tuple) -> list:
    gain = gain_func()
    scalar_coef = -gain / pow(dot_product(mag_field, mag_field), 0.5)
    dipole_out = x_product(mag_field, ang_vel)
    for i in range(3):
        dipole_out[i] *= scalar_coef
    return dipole_out
```

Bit Flag Refactor: NVM Storage Optimization



```python
class Flag:
    """
    Flag class for managing boolean flags stored in non-volatile memory
    """

    def __init__(self, index: int, bit_index: int, datastore: ByteArray) -> None:
        self._index = index  # Index of specific byte in array of bytes
        self._bit = bit_index  # Position of bit within specific byte
        self._datastore = datastore  # Array of bytes (Non-volatile Memory)
        self._bit_mask = 1 << bit_index  # Creating bitmask with bit position
        # Ex. bit = 3 -> 3 % 8 = 3 -> 1 << 3 = 00001000

    def get(self) -> bool:
        """Get flag value"""
        return bool(self._datastore[self._index] & self._bit_mask)

    def toggle(self, value: bool) -> None:
        """Toggle flag value"""
        if value:
            # If true, perform OR on specific byte and bitmask to set bit to set specific bit to 1
            self._datastore[self._index] |= self._bit_mask
        else:
            # If false, perform AND on specific byte and inverted bitmask to set specific bit to 0
            self._datastore[self._index] &= ~self._bit_mask
```

Reworked how critical system flags are stored in non-volatile memory (NVM):

Implemented a Flag class to manage single-bit values in a byte array

- .get() → reads a specific bit (status check)
- .toggle() → sets/clears a bit (update flag)

Why it Matters:

- Tracks critical events: burn wire success, brownout, soft reboots, etc.
- Enables data persistence across satellite reboots

Validated with PyTest:

- Verified flag behavior in normal and edge-case conditions
- Ensures long-term reliability and maintainability for flight software

Reintegrating SD support

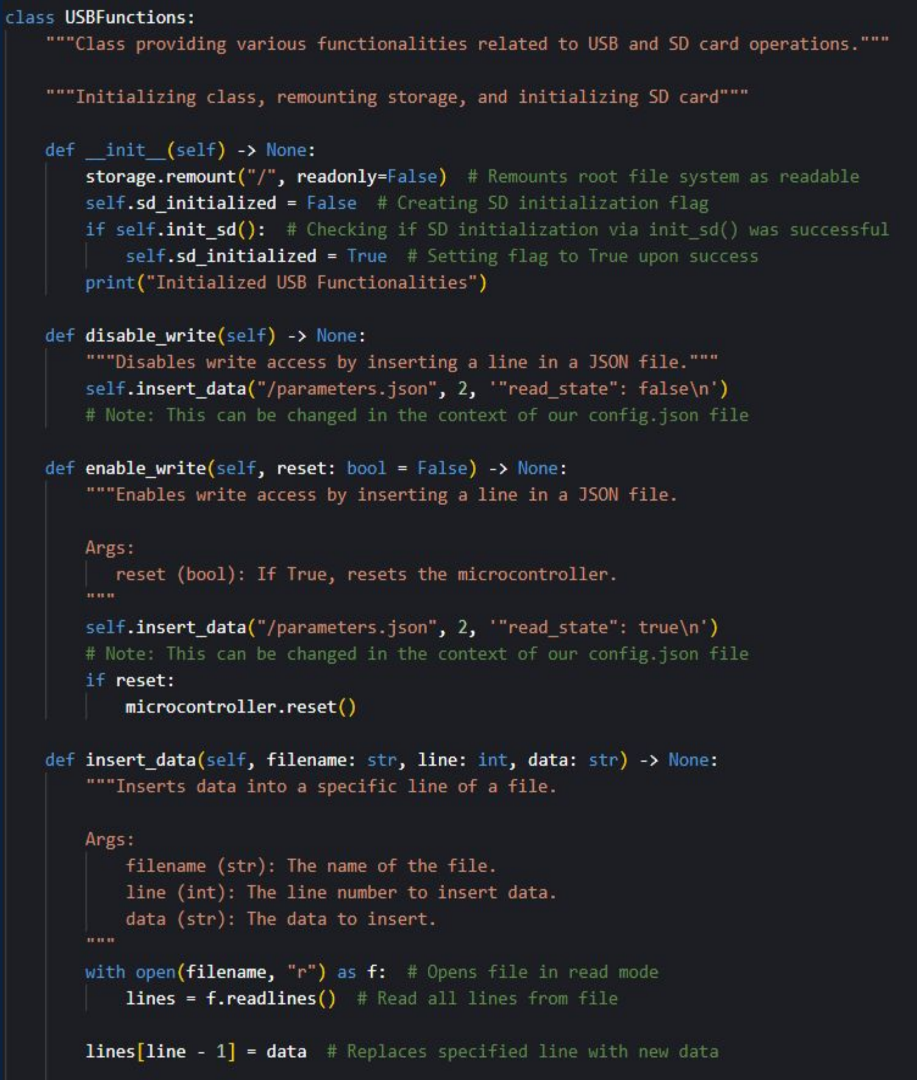
- ➔ Type hinting & exception handling
- ➔ Code documentation & structure improvements

SD Module Supports:

- SD initialization & file mounting (/sd)
- File operations: read, write, append, insert, delete
- Directory management & unique filename creation
- USB mode toggle via JSON config

Future Use:

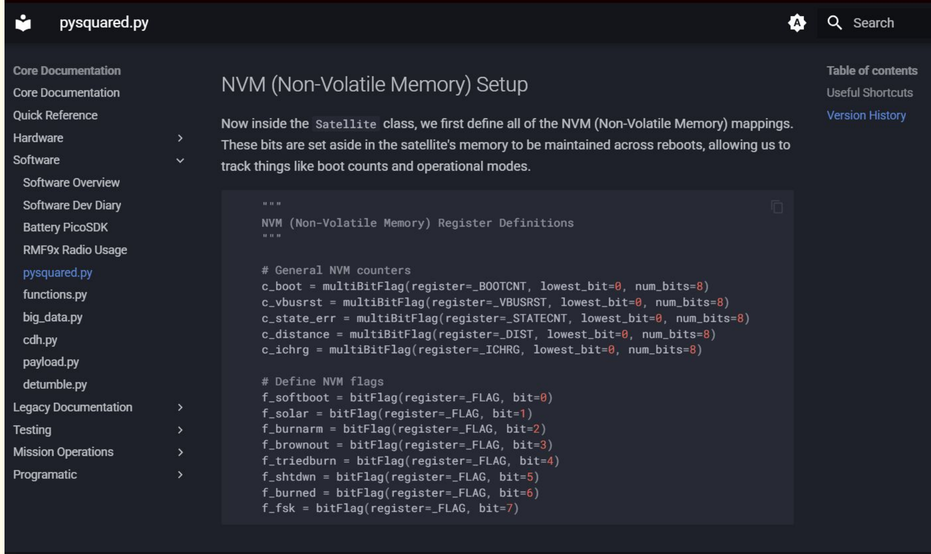
- Store system logs across reboots
- Enable data downlink to ground stations
- Support automated log cleanup to preserve memory space



```python
class USBFunctions:
    """Class providing various functionalities related to USB and SD card operations."""

    """Initializing class, remounting storage, and initializing SD card"""

    def __init__(self) -> None:
        storage.remount("/", readonly=False)  # Remounts root file system as readable
        self.sd_initialized = False  # Creating SD initialization flag
        if self.init_sd():  # Checking if SD initialization via init_sd() was successful
            self.sd_initialized = True  # Setting flag to True upon success
        print("Initialized USB Functionalities")

    def disable_write(self) -> None:
        """Disables write access by inserting a line in a JSON file."""
        self.insert_data("/parameters.json", 2, '"read_state": false\n')
        # Note: This can be changed in the context of our config.json file

    def enable_write(self, reset: bool = False) -> None:
        """Enables write access by inserting a line in a JSON file.

        Args:
            reset (bool): If True, resets the microcontroller.
        """
        self.insert_data("/parameters.json", 2, '"read_state": true\n')
        # Note: This can be changed in the context of our config.json file
        if reset:
            microcontroller.reset()

    def insert_data(self, filename: str, line: int, data: str) -> None:
        """Inserts data into a specific line of a file.

        Args:
            filename (str): The name of the file.
            line (int): The line number to insert data.
            data (str): The data to insert.
        """
        with open(filename, "r") as f:  # Opens file in read mode
            lines = f.readlines()  # Read all lines from file

        lines[line - 1] = data  # Replaces specified line with new data
```

Next Tasks

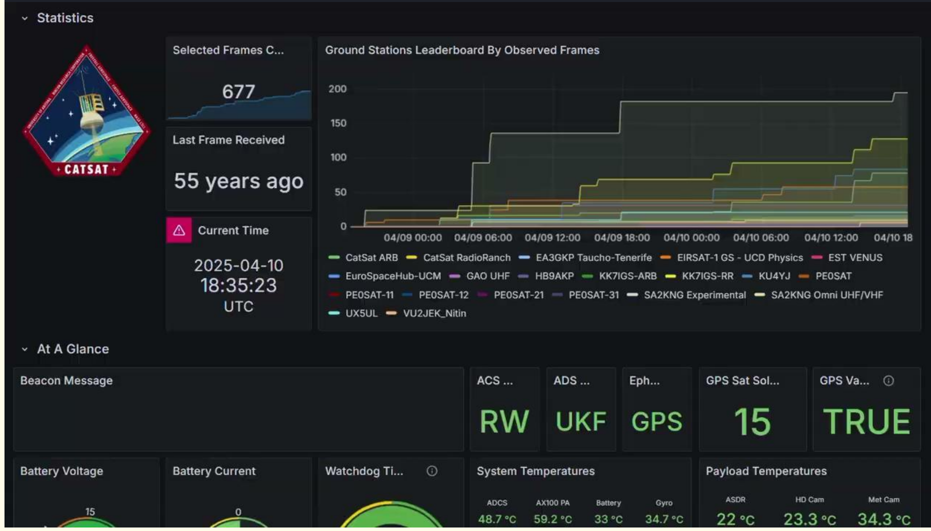
1. Create comprehensive user facing documentation
   MkDocs:
   - Static site generator, to build a browsable, locally launchable HTML website
   - Eventually host via Github Pages for automatic regeneration of documentation upon any changes to the code base
2. Integrate Grafana to ground station to display and monitor up to date satellite data
3. Finalizing preparations to send the Pleiades Maia mission patches I designed to production



NVM (Non-Volatile Memory) Setup

Now inside the `Satellite` class, we first define all of the NVM (Non-Volatile Memory) mappings. These bits are set aside in the satellite's memory to be maintained across reboots, allowing us to track things like boot counts and operational modes.

```
"""
NVM (Non-Volatile Memory) Register Definitions
"""

# General NVM counters
c_boot = multiBitFlag(register=_BOOTCNT, lowest_bit=0, num_bits=8)
c_vbusrst = multiBitFlag(register=_VBUSRST, lowest_bit=0, num_bits=8)
c_state_err = multiBitFlag(register=_STATECNT, lowest_bit=0, num_bits=8)
c_distance = multiBitFlag(register=_DIST, lowest_bit=0, num_bits=8)
c_ichrg = multiBitFlag(register=_ICHRG, lowest_bit=0, num_bits=8)

# Define NVM flags
f_softboot = bitFlag(register=_FLAG, bit=0)
f_solar = bitFlag(register=_FLAG, bit=1)
f_burnarm = bitFlag(register=_FLAG, bit=2)
f_brownout = bitFlag(register=_FLAG, bit=3)
f_triedburn = bitFlag(register=_FLAG, bit=4)
f_shtdwn = bitFlag(register=_FLAG, bit=5)
f_burned = bitFlag(register=_FLAG, bit=6)
f_fsk = bitFlag(register=_FLAG, bit=7)
```



Selected Frames C... 677

Last Frame Received: 55 years ago

Current Time: 2025-04-10 18:35:23 UTC

Beacon Message

ACS ... RW
ADS ... UKF
Eph... GPS
GPS Sat Sol... 15
GPS Va... TRUE

Battery Voltage 15
Battery Current 0
Watchdog Ti...

System Temperatures: ADCS 48.7 °C, AX100 PA 59.2 °C, Battery 33 °C, Gyro 34.7 °C

Payload Temperatures: ASDR 22 °C, HD Cam 23.3 °C, Met Cam 34.3 °C

Conclusion & What Comes Next

- The team has made extensive contributions, resulting in significant improvements to software infrastructure, testing reliability, and system-level functionality.

- The tasks I worked on this semester not only granted me technical experience, but also gave me valuable insight into how to communicate technical decisions clearly and contribute to a living codebase with mission-critical applications.

- With a launch scheduled for early 2026 and our lab anxious to start on a new 2U satellite mission of our own, our team is preparing to complete core development by the end of summer 2025.