SS1100: Intro to Programming for Space Applications

Final Project:
Programming Spacecraft Systems

1 Assignment

1.1 Instructions

Work in **groups of four** for this project.
Complete the steps on the following pages.
Please keep track of the total time you spend working on the project.

1.2 Submission

Turn in the requested screenshots, code snippets, and written report as part of your group's GitHub page for the project.

2 Procedure

2.1 Preparation

For this Final Project, you will be employing your programming skills to solve a variety of problems around a spacecraft. This will be arranged across a selection of subsystems, each with a specific task that requires a programmatic solution. Working as a group, develop code and responses to the tasks in the following sections. The goal of this project is to provide a final opportunity to experience working with code, collaborating on a coding project, and seeing how programming can be used to interact with space-related functionalities.

2.2 Requirements

1. Complete all of the tasks listed in each section, paying attention to the **Evaluation** subsections.
2. Program everything in the Python language.
3. Collaborate as a group and submit your project using GitHub.

2.3 Rubric and Standards

Table 1: Grading Rubric

Topic	√+		√		√-
Reaction Control Subsystem (RCS)	2.5	2.0	1.5	1.0	0.5
Thermal Control Subsystem (TCS)	2.5	2.0	1.5	1.0	0.5
Attitude Control Subsystem (ACS)	2.5	2.0	1.5	1.0	0.5
Command and Data Handling (C&DH)	2.5	2.0	1.5	1.0	0.5
Electrical Power Subsystem (EPS)	2.5	2.0	1.5	1.0	0.5
Remote Sensing Payload	2.5	2.0	1.5	1.0	0.5
GitHub Page	2.5	2.0	1.5	1.0	0.5
Written Responses	2.5	2.0	1.5	1.0	0.5
Total XX/20					

Table 2: Standards for Check System

Check Plus	**Check**	**Check Minus**
Outstanding code, additional tasks completed, comments and writeup above standard	Error-free code, primary tasks completed to standard, results populated in Github	Errors in code, missing tasks, results not as expected

2.4 Generative AI Use

You are allowed to use Generative AI (e.g., ChatGPT / Gemini / Copilot) for the coding portions of this project. Any use of GenAI needs to be disclosed, which you can do in-line. For example:

```python
def calculate_area(length, width):
    """This function was adapted from suggestions generated by Gemini"""
    return length * width
```

or

```python
# The following code segment is based off of the prompt in ChatGPT to
# "Give me the Python code the print the numbers from 1 through 10"
i = 0
while i < 10:
    print(i)
    i += 1
```

2.5 GitHub Repository

For this project, you will utilize GitHub to facilitate collaboration. One group member will copy the starting GitHub repository and invite the other members to join as collaborators. All of your code, visualizations, and data outputs will need to be uploaded to this repository, then turned in as part of your deliverables for the project.

Setup Instructions (NOTE: ONLY ONE GROUP MEMBER WILL DO THESE STEPS)

1. **Sign in to GitHub and Create a New Repository**
   - Go to GitHub online and log in to your account
   - Navigate to https://github.com/neal-macdonald/nps_ss1100_fall24.git
   - Find and click the `fork` button in the upper right corner of the screen.
2. **Repository Details**
   - Enter a **Name** for your repository
   - Optionally, add a **Description**
   - Leave "Copy the master branch only" selected, and hit **Create fork**
3. **Invite Collaborators**
   - Go to your newly-created repository page that you just forked (made a copy of)
   - Click on the `Settings` tab then select `Collaborators`
   - Click on `Add people`, enter your partners' usernames, then `Add to repository`
   - Also add user `neal-macdonald` as a collaborator

For more information about how to put GitHub to use, check out the following:

- Setting up GitHub Desktop: https://docs.github.com/en/desktop/overview/getting-started-with-github-desktop
- Copying your repository to your local computer: https://docs.github.com/en/desktop/adding-and-cloning-repositories/cloning-a-repository-from-github-to-github-desktop
- Video on using Github: https://youtu.be/iv8rSLsi1xo

2.6 Spacecraft Details

Your spacecraft has some basic characteristics to note. Some of these details will be needed for use in the tasks for each subsystem.

Table 3: Satellite Characteristics

Characteristic	**Value**
Dimensions	3 m x 2 m x 1 m
Mass	500 kg
Solar Panel Area	20 m^2
Moment of Inertia	1500 kgm
Power Generation	100 W
Altitude	600 km
Design Life	5 years
Equilibrium Temperature	30° C
Data Storage Capacity	200 GB
Propulsion System	Chemical
Attitude Control System	Reaction Wheels
TT&C Station Frequency	1.5 GHz
Weight Distribution	40% payload, 60% structure
Communication Protocol	TCP/IP
Sensor Suite	Multispectral Imager

3 Propulsion / Reaction Control Subsystem (RCS)

3.1 Background

Introduction The Reaction Control Subsystem (RCS) is crucial for maneuvering a spacecraft in space, enabling precise changes in orientation and trajectory. It comprises small thrusters strategically positioned around the spacecraft, allowing for accurate control of its attitude. By firing these thrusters in specific sequences, the RCS can make necessary adjustments to maintain stability and align with mission objectives. This subsystem is particularly vital during critical phases such as orbital insertion, maneuvering, and docking operations, ensuring the spacecraft remains on course.

Mathematics Spacecraft thrusters function by applying force to the craft, converting either chemical or electrical energy into motion. This process can be represented by two key formulas: one for calculating thrust and another for determining the change in velocity. Both formulas stem from the fundamental equation of **Force equals Mass times Acceleration**, and they are presented here as separate equations to highlight their relationship to thrust.

$$\mathbf{T} = \dot{\boldsymbol{m}} \times \boldsymbol{v}_{\mathbf{e}}$$

where $\mathbf{T}$ is Thrust (Newtons)
$\dot{\boldsymbol{m}}$ is mass flow rate (kg/s)
$\boldsymbol{v}_{\mathbf{e}}$ is effective exhaust velocity (m/s)

$$\Delta \boldsymbol{v} = \frac{\mathbf{T} \times \Delta \boldsymbol{t}}{\boldsymbol{m}_{\mathbf{0}}}$$

where $\Delta \boldsymbol{v}$ is change in velocity (m/s)
$\Delta \boldsymbol{t}$ is time ellapsed (s)
$\boldsymbol{m}_{\mathbf{0}}$ is spacecraft mass (kg)

3.2 Configuration

The RCS for your spacecraft is comprised of three thrusters, all of the same size and type. They have maximum limits for thrust and propellant flow rate, which if exceeded could damage them and cause mission failure. The following tables describe these limits and the arragement of the thrusters on the spacecraft.

Table 4: RCS System Limits

Parameter	**Max Limit**
Thrust	100 Newtons
Flow Rate	0.05 kg/s
Exhaust Velocity	2000 m/s

Table 5: Thruster Orientations

Thruster	**Orientation**
Thruster 1	Forward (along the x-axis)
Thruster 2	Lateral (along the y-axis)
Thruster 3	Vertical (along the z-axis)

3.3 Tasks

Getting Started In the `RCS` folder of your repository, you will find a script named `rcs_script.py`. Your task is to complete the functions in this script.

Malfunction Detection Create a function that checks input thrust values and angles for each thruster against the set limits. If a thruster is outside of these limits, the function should print a message describing:

- The name of the malfunctioning thruster
- The parameter (exhaust velocity or flow rate) exeeding the limits
- The value of how far outside of tolerance it is

Velocity Change Calculation Create a function that performs calculations to determine the change in velocity resulting from a maneuver event. Your function must take as inputs the:

- Mass Flow Rate (kg/s)
- Effective Exhaust Velocity (m/s)
- Time Ellapsed (s)

As the output, your function should return the change in velocity in meters per second (m/s)

3.4 Evaluation

Test these inputs against your code. First, check if any of the values are out of tolerance, then attempt to calculate the change in velocity for the following inputs:

1. Firing a thruster for 5 seconds, with $\dot{m} = 0.02$ and $v_e = 1000$
2. Firing a thruster for 3 seconds, with $\dot{m} = 0.06$ and $v_e = 1000$
3. Firing a thruster for 10 seconds, with $\dot{m} = 0.05$ and $v_e = 2000$

For a Check on this Task: Provide your code for Malfunction Detection and Velocity Change Calculation, then the results for the three thrusting events above.

For a Check Plus on this Task: Improve your Velocity Change Calculation function so that it can take into account the *direction* of the thrust. Your formula will have to take as inputs three separate Mass Flow Rates, Effective Exhaust Velocities, and Time Ellapsed - one for each Thruster, as each can be controlled independently. As an output, you will have to return the change in velocity in each of the three axes: X, Y, and Z. Evaluate your formula against the following data:

- Firing Thruster 1 for 15 seconds, with $\dot{m} = 0.04$ and $v_e = 2000$
- Firing Thruster 2 for 4 seconds, with $\dot{m} = 0.03$ and $v_e = 2000$
- Firing Thruster 3 for 3 seconds, with $\dot{m} = 0.01$ and $v_e = 2000$

4 Thermal Control Subsystem (TCS)

4.1 Background

Introduction The Thermal Control Subsystem (TCS) is essential for maintaining the spacecraft's temperature within safe operational limits. It regulates heat generated by onboard systems and external environmental factors, ensuring that all components function optimally. By employing various thermal management techniques, such as insulation, radiators, and heaters, the TCS can effectively dissipate excess heat or provide necessary warmth. This subsystem is particularly critical during phases of the mission where temperature fluctuations can impact system performance.

Mathematics The thermal management of a spacecraft can be modeled using heat transfer equations. The key formulas involve calculating heat transfer rates and temperature changes, which are derived from the principles of thermodynamics. The TCS operates based on three primary heat transfer mechanisms:

- **Conduction:** Heat transfer through solid materials.
- **Convection:** Heat transfer through fluids (liquids or gases).
- **Radiation:** Heat transfer through electromagnetic waves.

The heat transfer can be modeled using the following equation:

$$Q = mc\Delta T \tag{1}$$

where Q is the heat energy (in joules), m is the mass (in kilograms), c is the specific heat capacity (in J/kg°C), and ΔT is the change in temperature (in °C).

4.2 Configuration

The TCS for your spacecraft consists of multiple components, including radiators, heaters, and thermal insulation. Above all, the goal is to maintain all of the systems within a set range of temperature values.

4.3 Tasks

Temperature Control Create a function that takes as an input a current temperature and target temperture, then calculates the signed magnitude of the difference. To simulate a realisic temperature control system, the temperature can only be changed by a scaled amount at a time. Therefore, you will need to calculate 25% of the temperature difference to determine how much the current temperature can be changed at once. The output of your function will be the input current temperature, modified by this scaled value, to produce the new current temperature. To summarize:

1. Receive two floats as inputs, representing current temperature and desired temperature.
2. Calculate the difference between the two input values.
3. Modify the current temperature, changing it by no more than 25% of the total difference between the current and target temperatures.
4. Print how much your TCS is changing the temperature by.
5. Return, as a float, the new current temperature.

4.4 Evaluation

For a Check on this Task: Provide your code and a screenshot of some test values of your own choosing used to test its functionality.

For a Check Plus on this Task: Open the script `thermal_control.py` located in the `TCS` folder. This script is designed to slowly change the input temperature, mimicing the changing temperature of a spacecraft. Observe that there is a portion that is set aside for you to edit. Your task is to modify your above function to work in this space, reacting to the changing temperature and attempting to guide it to the target temperature. The script will run for a set number of cycles, printing out the current temperature after each time step. The goal is to get the current temperature to stablize around the target temperature value.

5 Attitude Control Subsystem (ACS)

5.1 Background

Introduction The Attitude Control Subsystem (ACS) is vital for maintaining and controlling the orientation of a spacecraft in space. It utilizes a combination of sensors, actuators, and control algorithms to ensure that the spacecraft can achieve and maintain its desired attitude. By adjusting the orientation, the ACS enables the spacecraft to align with communication targets, optimize solar panel exposure, and stabilize during maneuvers.

5.2 Configuration

The ACS for your spacecraft consists of multiple components, including reaction wheels, Inertial Measurement Units (IMUs), and magnetometers. Specifically, it has a three-axis stabilization system, allowing the ADC to correct the orientation of the satellite in X, Y, and Z axes. The system can determine its current orientation using its sensor suite, then send commands to the reaction wheels on how to change the attitude to the desired directions.

5.3 Tasks

Calculating Rotation Create a function that performs calculations to determine the change in angle resulting from a maneuver event. Your function must take as inputs a two-item list containing:

- Current orientation (X, Y, and Z)
- Desired orientation (X, Y, and Z)

For example, a valid input could be `input_1 = [(30,60,90),(0,10,15)]`. For outputs, your function should produce a three-item tuple showing the signed magnitude of the rotation required in each axis.

Rotating a Satellite You will now write a script that rotates a satellite to a desired orientation. In the `ADC` folder, you will find two scripts: `rotate_me.py` and `your_adc_script.py`.

First, run `rotate_me.py` by itself - it should produce a file called `current_state.txt` in the same folder. This new file contains three random values for the current X, Y, and Z orientation of the satellite.

Now, open the script `your_adc_script.py`. This has two lines premade for you:

- The first line importing the `rotate_me` script
- The last line calling the main function from `rotate_me` with a tuple of values

Your job is to create code in `your_adc_script.py` that finds the current state of `current_state`, then calculates the corrections needed to get to a target orientation (below in the Evaluation section). Replace the last line in this script with your output tuple, which will then trigger the `"rotate_me"` script to run and update `"current_state.txt"`. Be aware - the `rotate_me.py` script adds a random error to the corrections, so you'll have to run your script more than once before you get to the target orientation!

5.4 Evaluation

For a Check on this Task: Put your code together, then use the following tuples as the target values. Stop once the values held in `current_state.txt` are within 0.1 of the target value in all three axes.

- (100,200,300)
- (0,0,0)
- (3,30,300)

Hint: to start over, just delete the `"current_state.txt"` file, then re-run `"rotate_me.py"` - it will repopulate that text file with a new starting orientation.

For a Check Plus on this Task: Incorporate your Calculating Rotation script into `"your_adc_script.py"` - don't just copy and paste it in - it should be imported as a module just like the `rotate_me` script is.

6 Command and Data Handling (C&DH)

6.1 Background

The Command and Data Handling (C&DH) subsystem is responsible for processing commands received from ground control and routing them to the appropriate spacecraft subsystems. It ensures that commands are executed correctly and that data is managed efficiently. This subsystem is crucial for maintaining the spacecraft's operational integrity and ensuring that all systems function as intended.

6.2 Configuration

Your spacecraft's C&DH system utilizes a formatted messaging schema to route data to the correct subsystems. It utilizes the following structure:

- Block 1: Subsystem Code (3-digit abbreviation)
- Block 2: Instruction / Command Code
- Block 3: Parameter value (context-dependent numeric values)

with each block being separated by a ":" character. See Table 7 in the appendix for the three-digit codes and commands listing. As examples, the command to fire the X-axis thruster for 10 seconds would be formatted as: `RCS:CMD01:10`, while setting the target temperature for the thermal control to 30°C would be `TCS:CMD04:30`.

6.3 Tasks

Command Parsing and Routing Write a script that can parse incoming commands based on the formatted schema. The script should take as input a string containing the message, verify the contents, and output at tuple containing:

- The full name of the destination subsystem/payload device
- The text description of the command (e.g., START_DATA_ACQUISITION)
- The parameter value

Your code should include basic error checking, such as ensuring a proper three-digit code and command has been passed, as well as ensuring that the parameter value passed is of a numeric type.

6.4 Evaluation

Test these inputs against your code:

- `EPS:CMD01:0`
- `ACS:CMD04:-1`
- `RCS:INVALID:0`

For a Check on this Task: Provide your code for Command Parsing and Routing, along with the results for the commands listed above.

For a Check Plus on this Task: Enhance your script to include a value checking function.

1. Open the `command_dict.py` script located in the `C&DH` folder, and note that this dictionary now contains ranges and sets of acceptable values for each command
2. Add code to your function that checks if the values inputted as part of the commands are in tolerance
3. Re-run the three Evaluation inputs above
4. Submit your updated code as a new file in your repository

7 Electrical Power Subsystem (EPS)

7.1 Background

Introduction The Electrical Power Subsystem (EPS) is vital for providing and managing electrical power throughout the spacecraft. It ensures that all onboard systems receive the necessary energy to operate effectively, from communication systems to scientific instruments. The EPS typically includes solar panels, batteries, and power distribution units, which work together to generate, store, and distribute electricity. This subsystem is particularly critical during phases of the mission where power demands fluctuate, such as during solar eclipses or high-energy maneuvers. A well-designed EPS is essential for maintaining the spacecraft's functionality and extending its operational lifespan.

Mathematics The EPS operates based on several key principles of electrical engineering. The fundamental equations governing power generation and consumption are:

Power Calculation

$$P = V \cdot I$$

where P is power in Watts (W)
V is voltage in Volts (V)
I is current in Amperes (A)

Energy Storage

$$E = P \cdot t$$

where E is energy in Joules (J)
P is power in Watts (W)
t is time in seconds (s)

These equations highlight the relationship between power generation, consumption, and energy storage, which are critical for the effective operation of the EPS.

7.2 Configuration

The EPS for your spacecraft consists of multiple components, including solar panels, batteries, and power distribution units. Each component has specific limits for voltage, current, and power output, which, if exceeded, could lead to system failures. The following tables describe these limits and the arrangement of the components on the spacecraft.

Table 6: Solar Panel limits

Parameter	Max Limit
Voltage	28 V
Current	10 A
Power Output	280 W

7.3 Tasks

Getting Started In the `EPS` folder of your repository, you will find a script named `eps_script.py`. Your task is to complete the functions in this script.

Available Power Create a function that performs calculations to determine the instantaneous incoming power from the solar panels. It will also need to check for inputs that exceed the solar panels' maximum limits as specified above, along with code reducing that parameter to the max level in order to prevent damage to the system. Your function must take as inputs the:

- Voltage (V)

- Current (A)

As the output, your function should return the Power in Watts (W).

Battery Charging Create a second function that performs calculations to determine the total energy available for battery charging. It will incorporate the output of your previous function as well as a time value, specifically:

- Power Delivered (W) - *output from your other function*
- Time Elapsed (s)

As the output, your function should return the total energy in Joules (J).

7.4 Evaluation

Test these inputs against your code:

1. Voltage of 25 V, Amperage of 10 A, running for $t = 3600$ seconds
2. Voltage of 30 V, Amperage of 8 A, running for $t = 1800$ seconds
3. Voltage of 15 V, Amperage of 12 A, running for $t = 7200$ seconds

For a Check on this Task: Provide your code for Available Power and Battery Charging, as well as the results of three inputs above.

For a Check Plus on this Task: Enhance your functions to account for fluctuating inputs over time. Your formula will have to take as inputs a list of voltages and amperages over corresponding time intervals, with each interval's values held in a tuple. As an output, you will have to return the total energy calculated over the entire period. Evaluate your formula against the following data in the format: (`voltage`, `amperage`, `duration`)

- [(22,7,300), (40,7,60), (25,10,200), (10,4,600)]
- [(0,7,300), (30,10,60), (28,10,200), (10,10,10)]

8 Remote Sensing Payload

8.1 Background

Introduction The primary payload on your satellite is a remote sensing instrument. You will help gather, process, and prepare the data it collects for downlinking to the ground. The raw data that comes from the sensor is very different than what you are likely used to seeing presented as part of a satellite image, so first we will discuss some basic principles.

Radiance vs. Reflectance Radiance is the amount of light that is emitted, reflected, or transmitted by a surface per unit area and per unit solid angle. It is measured in watts per square meter per steradian ($W/m^2/sr$). Reflectance, on the other hand, is the ratio of the reflected radiation from a surface to the incident radiation on that surface. It is a dimensionless quantity, often expressed as a percentage.

Bit Depth and Data Types Recall that there are a variety of different primitive data formats and bit depths that can be used to hold data in a digital format. Bit Depth refers to the number of ones and zeros required to hold a piece of data, with depths such as 8-, 16-, 32-, and 64-bit being common values. Data Types include integers, floating point values, strings, and booleans, with integers and floats being most commonly used in remote sensing data files. Raw data is often collected as floating point values, then quantized and converted into integers of a certain bit depth in order to optimize file sizes for storage and transmission.

Mathematics The relationship between radiance (L) and reflectance (R) can be expressed as:

$$R = \frac{L}{L_{incident}} \tag{2}$$

where $L_{incident}$ is the incoming radiance. This formula demonstrates that reflectance is a function of how much outgoing energy was received by the sensor (L) compared to how much energy originally arrived at the surface $L_{incident}$. This requires knowledge about the total incoming EM radiation arriving at the surface, which for a passive, optical remote sensing system can be derived with information about the Sun:

$$R = \frac{L \cdot \pi}{E_{solar} \cdot \cos(\theta)} \tag{3}$$

where E_{solar} is the solar irradiance and θ is the solar zenith angle.

In a calibrated remote sensing system, it is often possible to simplify this formula based on a multiplicative factor (k) and an additive factor (b). This approach allows for a more flexible transformation:

$$R = k \cdot L + b \tag{4}$$

8.2 Tasks and Evaluation

Getting Started In the `Payload` folder of your repository, you will find a script named `payload_script.py` and three CSV files: `red.csv`, `green.csv`, and `blue.csv`. Your task is to complete the functions in the Python script to process the data from the CSV files.

For a Check on this Task: Create a function that takes as inputs separate Red, Green, and Blue data from the remote sensing instrument. This will be in the form of a CSV file, one per band, each holding a field of radiance values representing the raw image data.

1. Load the raw R, G, and B band data into your IDE or script
2. Ensure that all bands are of the same dimensions
3. Combine the bands into a single RGB image
4. Visualize the results

Hint: If working this section in Python, utilize **numpy** for this, in particular `np.stack()`

Test your function on the image data in the CSV files provided in the Payload folder of the GitHub repository. Save a screenshot of your output to the GitHub repository.

For a Check Plus on this Task: Perform the following additional steps:

Convert Radiance to Reflectance Radiance values are often converted into reflectance before use, so you will next perform this conversion. Create a function that takes as an input the RGB image created in the previous task, as well as two additional values:

- $k = 0.8$ (multiplicative scaling factor)
- $b = 0.1$ (additive scaling factor)

Your function should return an image where each pixel is rescaled to the range [0, 1]

Rescale Reflectance to 8-bit Digital Numbers (DN) The last step in this preprocessing of the data is to convert the data into an integer format from the floating point value it is currently in. This is to save space and reduce the transmission time in downlinking the data to the ground. Create a function that:

1. Takes the reflectance image created in the last step
2. Scales each pixel to fit an 8-bit Digital Number (i.e., a value between [0, 255])
3. Converts the values from floats to integers

Hint: be sure to do the rescaling operation first - if you convert a decimal value between [0,1] to an integer, it will lose all of the information it holds.

Visualize and Save the Image Finally, create a function that saves your image to file. This function should:

1. Take the image from the previous step as the input, along with the file name and folder location.
2. Save it using an appropriate file format such as `.png` or `.jpeg`

9 Questions for Writeup

Answer the following questions, submitting them by editing the README file on the main page of your GitHub repository.

1. What was your experience in collaborating? Talk about what steps you used to ensure the inputs from group members worked - code testing, GitHub branch management, etc. - and how you divided up the workload for the project.

2. What was the most challenging section, and why? Feel free to provide more than one response if there is a difference of opinion in the group.

3. If you employed Generative AI tools, how did you do so? Discuss which tools you used, the prompts you utilized, how you ensured the results were valid, and how you integrated the code into your tasks.

4. What other resources did you use to find solutions? Online sites, books, references, etc.

5. In what way could this project be improved for future quarters?

10 Appendix

Table 7: Subsystem Commands

Subsystem	Code	Commands
Reaction Control Subsystem	RCS	CMD01 - THRUST_X CMD02 - THRUST_Y CMD03 - THRUST_Z CMD04 - SAFE_MODE
Thermal Control Subsystem	TCS	CMD01 - HEATER_ON CMD02 - HEATER_OFF CMD03 - VENT_OPEN_RADIATOR CMD04 - TEMP_SETPOINT
Attitude Control Subsystem	ACS	CMD01 - ROTATE_X CMD02 - ROTATE_Y CMD03 - ROTATE_Z CMD04 - SAFE_MODE
Command & Data Handling	CDH	CMD01 - TRANSMIT_HIGH CMD02 - TRANSMIT_LOW CMD03 - RECEIVE_MODE CMD04 - SAFE_MODE
Telemetry, Tracking, & Command	TTC	CMD01 - TRANSMIT_MODE CMD02 - RECEIVE_MODE CMD03 - TRACKING_MODE CMD04 - SAFE_MODE
Electrical Power Subsystem	EPS	CMD01 - BATTERY_CHARGE_MODE CMD02 - POWER_ON_MODULE CMD03 - POWER_OFF_MODULE CMD04 - VOLTAGE_SETPOINT
Payload System (and number)	PL1 / PL2	CMD01 - START_DATA_ACQUISITION CMD02 - STOP_DATA_ACQUISITION CMD03 - CALIBRATE_SENSOR CMD04 - SAFE_MODE