



**FACHHOCHSCHULE  
WIENER NEUSTADT**

Austrian Network for Higher Education

University of Applied Sciences

# CLIMB operation planning

Senior team project

---

Submitted by:

Aruzhan Koptleu

Giacomo Scomparin

Hamza Shehadeh

Kyuil Han

Supervisor:

Dr. Carsten Scharlemann

Wiener Neustadt, 30.01.2024

# Table of contents

<i>List of Abbreviations.....</i>	<i>III</i>
<i>List of figures.....</i>	<i>IV</i>
<i>List of tables.....</i>	<i>VI</i>
<b>1. Introduction .....</b>	<b>7</b>
<b>2. Operation planning scheme .....</b>	<b>8</b>
<b>3. Schedule GUI.....</b>	<b>10</b>
<b>3.1 STK 1 .....</b>	<b>10</b>
<b>3.2 MATLAB GUI .....</b>	<b>15</b>
<b>3.3 STK 2 .....</b>	<b>17</b>
<b>4. Tools.....</b>	<b>19</b>
<b>4.1 Torque tool .....</b>	<b>19</b>
Attitude mode control .....	19
4.1.1 Environmental disturbances.....	22
4.1.2 Torque due to the shift of the center of mass and misalignment of the thrust vector.....	25
4.1.3	
<b>4.2 Power tool .....</b>	<b>31</b>
4.2.1 Power tool overview.....	31
4.2.2 Input data .....	31
4.2.3 Battery models .....	32
4.2.4 Methods .....	34
4.2.5 Results and graphical output.....	37
4.2.6	
4.3.1 Future Power tool improvements.....	39
4.3.2	
4.3.3 <b>Propulsion tool .....</b>	<b>40</b>
Creating Imaginary Received Data.....	41
Data Handling.....	44
Grafana Visualization .....	47
<b>5. Graphical display - Grafana.....</b>	<b>50</b>
<b>6. Installation guide.....</b>	<b>52</b>
<b>6.1 Local PC planning tools installation steps .....</b>	<b>52</b>
<b>6.2 Local Grafana installation and setup.....</b>	<b>53</b>
<b>7. Conclusion .....</b>	<b>54</b>

<b>8.</b>	<b><i>References</i></b> .....	<b>55</b>
<b>9.</b>	<b><i>Appendix</i></b> .....	<b>56</b>
<b>9.1</b>	<b>Appendix 1:</b> .....	<b>56</b>

## List of Abbreviations

ADCS: *Attitude Determination and Control System*

CG: *Center of Gravity*

CSV: *Comma Separated Values*

HPOP: *High Precision Orbit Propagator*

JTP: *Junior Team Project*

LOS: *Line Of Sight*

SOC: *State of Charge*

STP: *Senior Team Project*

UHF: *Ultra High Frequency*

## List of figures

Figure 1: Operational planning scheme.....	9
Figure 2: Solar panels' location on simplified CLIMB model.....	10
Figure 3: Simplified CLIMB model with axes.....	11
Figure 4: Exemplary "Initial State" segment.....	12
Figure 5: Satellite parameters, which influence torque tool and maneuvers.....	12
Figure 6: Propagation segment.....	13
Figure 7: Thrust segment.....	14
Figure 8: Engine model used for thrust maneuvers.....	14
Figure 9: Layout of the MATLAB GUI.....	15
Figure 10: MATLAB GUI flowchart .....	16
Figure 11: MATLAB GUI tabs expanded.....	16
Figure 12: Removed burns in Astrogator .....	17
Figure 13: Example attitude profile.....	18
Figure 14: Torque types on CLIMB .....	19
Figure 15: Snip from Schedule GUI code .....	21
Figure 16: Exemplary attitude schedule with custom profiles.....	22
Figure 17: VNC (Velocity Normal Co-Normal) reference system visualization [1] .....	22
Figure 18: Fixed axis description in VNC reference system and visualization.....	23
Figure 19: Absolut angular momentum using the 'Two body' propagator.....	23
Figure 20: Absolute angular momentum using 'HPOP (High Precision Orbit Propagator).' .....	24
Figure 21: Absolute angular momentum using 'HPOP (High Precision Orbit Propagator) at low altitude.....	24
Figure 22: Yaw, pitch and roll over time from STK2.....	25
Figure 23: Yaw, pitch and roll over time after removing -180 to +180 limit.....	26
Figure 24: Angular momentum due to mode switch and environmental disturbances.....	27
Figure 25: Cumulative angular momentum due to center of mass shift in pitch and roll. ..	28
Figure 26: Cumulative angular momentum due to vector misalignment added to pitch. ...	28
Figure 27: Total angular momentum.....	29
Figure 28: Total angular momentum.....	29
Figure 29: Voltage vs capacity graph for a single cell .....	33
Figure 30: Capacity vs cycles graph for a single cell .....	33

Figure 31: screenshot of battery status and models parameters Excel sheet .....	34
Figure 32: Simplified Power tool code structure and main functions .....	35
Figure 33: Screenshot from "CLIMB Power Budget V2" Excel sheet .....	36
Figure 34: Battery output Grafana visualization .....	37
Figure 35: Battery power output Grafana visualization (shorter time-frame).....	37
Figure 36: Battery SOC Grafana plot through entire scenario .....	38
Figure 37: Battery capacity Grafana plot through entire scenario.....	38
Figure 38: Battery SOC visualization with lower initial SOC .....	38
Figure 39: Screenshot of the planning dashboard .....	39
Figure 40: Examples of Declared Parameters .....	41
Figure 41: Pandas Dataframe and Date_Range .....	42
Figure 42: Noise Variables 'var1' and 'var2' .....	42
Figure 43: Non-Range Parameters .....	43
Figure 44: Importing the Thruster Data as 'rawData' .....	44
Figure 45: Column Headers and Table Shape .....	44
Figure 46: Importing a Tab Separated File (Backup).....	45
Figure 47: Data Calculation.....	45
Figure 48: Index Searching and Recording .....	46
Figure 49: For-loop Value Addition.....	46
Figure 50: Exportation.....	47
Figure 51: Propulsion Tool Grafana Main Page.....	47
Figure 52: Main Page Rows Closed .....	48
Figure 53: Creating a gauge plot .....	48
Figure 54: Propulsion Tool Grafana Main Page.....	49
Figure 55: Some of Grafana's data sources possibilities .....	50
Figure 56: Two different approaches for telemetry visualization using Grafana.....	51
Figure 57: Two different approaches for telemetry visualization using Grafana.....	52

## List of tables

Table 1: Inertia matrix of CLIMB .....	11
Table 2: Attitude modes' characteristics .....	20
Table 3: Center of mass shift in CLIMB .....	27
Table 4: Torque tool inputs: Torque_tool_data.xlsx .....	30
Table 5: Power tool input data.....	31

# 1. Introduction

This Senior Team Project (STP) deals with the short- and long-term operation of the CLIMB CubeSat. The complex operation of this satellite involves various operational modes, including propulsion maneuvers, S-band communication windows and momentum wheel de-saturation. To use each of these modes to its best extent and to climb as high and fast as possible, the operation of each mode and maneuver needs to be planned well in advance.

In the course of the recent Junior Team Project (JTP23 – Orbital assessment of CLIMB), an initial STK scenario was formulated to investigate the impact of different orbits and attitudes on the power and torque budget. Through this scenario, one can now evaluate the solar power generation during an orbit, analyze the influence of alignment maneuvers on the budget, and track the total spacecraft torque fluctuations throughout an orbit.

This STP aims to build upon the previously established understanding and the existing STK scenario. However, this project aims to investigate more realistic scenarios. It seeks to understand and model the long-term effects of external torques on the spacecraft torque budget, optimize alignment procedures, and simulate the behavior of the spacecraft over the next 72 hours. In collaboration with an ongoing master's thesis, the objective is to develop the processes and tools essential for planning spacecraft operations and visualize the simulated and received data. To be precise, this STP intends to establish tools for operation planning and status visualization. The outcomes of this effort include:

- The automatic generation of an STK simulation based on the current data received from the satellite.
- The establishment of a graphical tool to allow the operator to decide the upcoming maneuvers of the spacecraft and to simulate its behavior.
- The automatic creation of output files detailing new orbit parameters as well as the new data for power, torque and propulsion, which can subsequently serve as input files for the visualization and further processing.
- The creation of tools for analyzing and visualizing power and torque budgets, as well as thruster conditions.



## 2. Operation planning scheme

To outline the overall architecture of the project and the interdependencies of all its sub-parts and tools, a high-level scheme for the operation planning is created. It discusses how each piece works with the others, its function, its needed inputs and outputs and how it integrates into the general workflow. Figure 1 visualizes this scheme in the form of a diagram.

Starting the planning cycle with the data received from SatNOGS (Satellite Networked Open Ground Stations). It is the open-source network of satellite ground stations that will be used to communicate with CLIMB. The sent data comprises the spacecraft's TLE (Two Line Elements), including its attitude and orbit parameters at the time of sending, as well as necessary data on its subsystems. The crucial subsystems for this project are the battery module, ADCS (Attitude Determination and Control System), and the thruster. Optional data, such as magnetometer readings or system temperatures, may also be included during a downlink transmission but are not taken into consideration in this project.

This data is then fed into STK by a MATLAB script, automatically creating a scenario, which extends 72 hours from the timestamp of the received data (henceforth called STK1). The STK1 scenario records all the times for possible communication windows for the S-band antenna (which needs a direct line of sight between spacecraft and ground station) as well as the perigee-passes of the spacecraft. These timestamps at perigee mark possible time windows for thrust maneuvers. It should be noted that STK1 assumes a burn (thrust maneuver) at every perigee, thereby creating a slight difference between the timestamps of the predicted flight path and the one chose by the operator later on. This option was chosen, because CLIMB anticipates performing a burn during most of its perigee passes. The time difference is minimal; however, if one wishes to reduce it further, additional iterations on the STK2 scenario have to be done.

The next step is for the operator to decide on the maneuvers to perform by using the Schedule GUI discussed in the 3<sup>rd</sup> chapter. They can decide at which perigee-passes the thruster should fire and during which communication windows the spacecraft should face the ground station. The operator can easily disable all burns in case of an unexpected behavior of the satellite. They cannot however activate an automatic de-saturation mode since there is not enough known about its properties.

This additional user input is then used to create another STK scenario (STK2). It performs only the maneuvers directed by the operator and more precisely models the attitude of the spacecraft. Additionally, STK2 checks for overlaps between the selected S-band communication windows and the thrust maneuvers, including the slew period before and after both. The necessary results of this scenario are then automatically exported in the form of .csv (Comma Separated Values) files to serve as input for the three subsystem tools as well as directly into the visualization tool.

The tools are described in more detail in chapter 0. Essentially, they combine the data from SatNOGS and STK2 and model the behavior of each subsystem based on external information provided by the manufacturer of said system or otherwise gathered or measured.

Lastly, this information is displayed in several Grafana dashboards to allow the operator to decide whether the planned course of action is feasible and to provide an overview of the gathered data and its predicted values in the future. If they approve the simulated plan, it can be uplinked to the satellite by the ground station. If the outcome is not satisfactory, the user has to change their inputs in the Schedule GUI and rerun the simulation until an acceptable result is found.

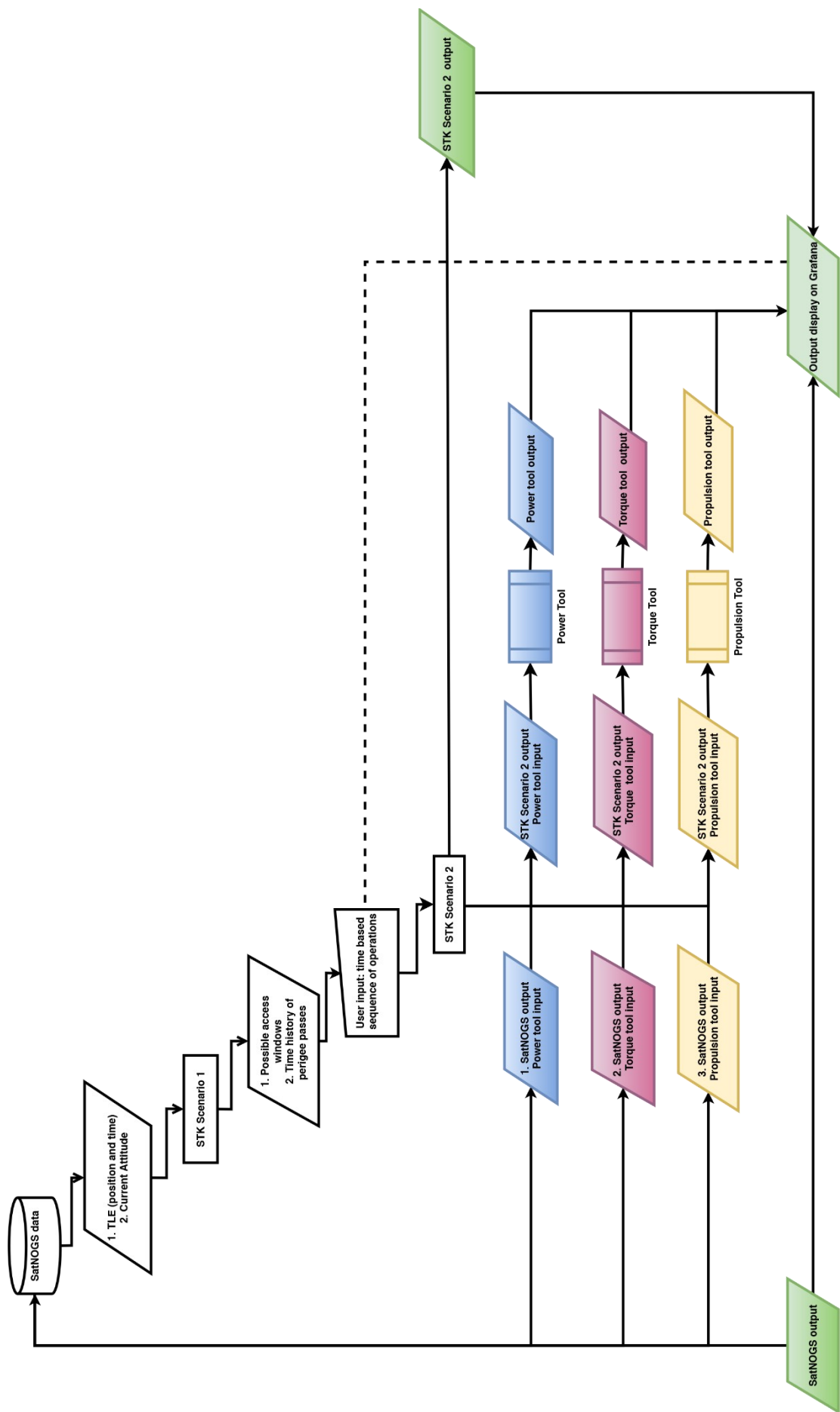


Figure 1: Operational planning scheme

### 3. Schedule GUI

#### 3.1 STK 1

This first STK scenario was created manually and is accessed and modified when opening the GUI to adapt to the most up-to-date parameters of the spacecraft.

The scenario features the ground station in the FH in Wiener Neustadt, equipped with the antenna that allows for a LOS (line of sight) communication with the satellite in a cone half angle of  $85^\circ$  with either the S-band antenna or the UHF (Ultra High Frequency) antenna.

A simplified CLIMB model was created with solar panels and some on the back as shown in Figure 2.

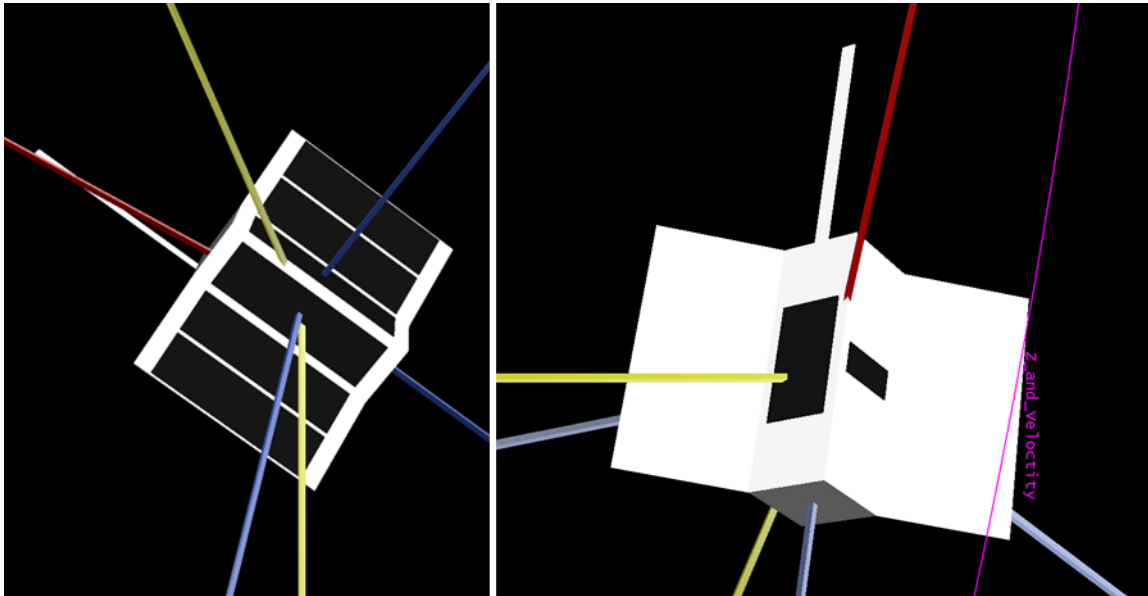


Figure 2: Solar panels' location on simplified CLIMB model

It represents the rough outline of the real model and is then loaded to correctly estimate the drag force experienced by the spacecraft, as can be seen in Figure 3.

The body axes form a right-hand system, with the z-axis opposite to the velocity vector and the x- and y-axes pointing towards the satellite faces covered in solar cells.

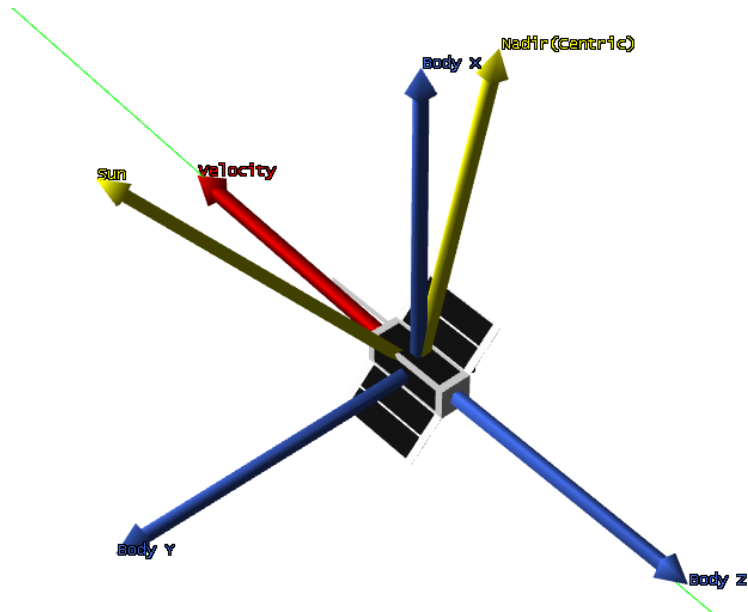


Figure 3: Simplified CLIMB model with axes

The dry mass of CLIMB is given at 4,5 kg according to the newest 3D model. The inertia matrix, which is particularly important for the torque tool, looks as follows:

Table 1: Inertia matrix of CLIMB

	<b>x</b>	<b>y</b>	<b>z</b>
<b>x</b>	0,1432 kg·m <sup>2</sup>	-0,0039 kg·m <sup>2</sup>	-0,0062 kg·m <sup>2</sup>
<b>y</b>	-0,0039 kg·m <sup>2</sup>	0,1461 kg·m <sup>2</sup>	-0,0054 kg·m <sup>2</sup>
<b>z</b>	-0,0062 kg·m <sup>2</sup>	-0,0054 kg·m <sup>2</sup>	0,0235 kg·m <sup>2</sup>

The orbit propagation is handled through the “Astrogator” propagator in STK, where the satellite can be programmed to do various maneuvers. STK1 starts with an “Initial State” segment. Here, the latest telemetry is automatically inserted to provide a starting point for the further simulation. Moreover, the initial spacecraft parameters such as the dry mass, drag coefficient or solar radiation pressure coefficient are defined. Figure 4 shows such an exemplary initial orbit:

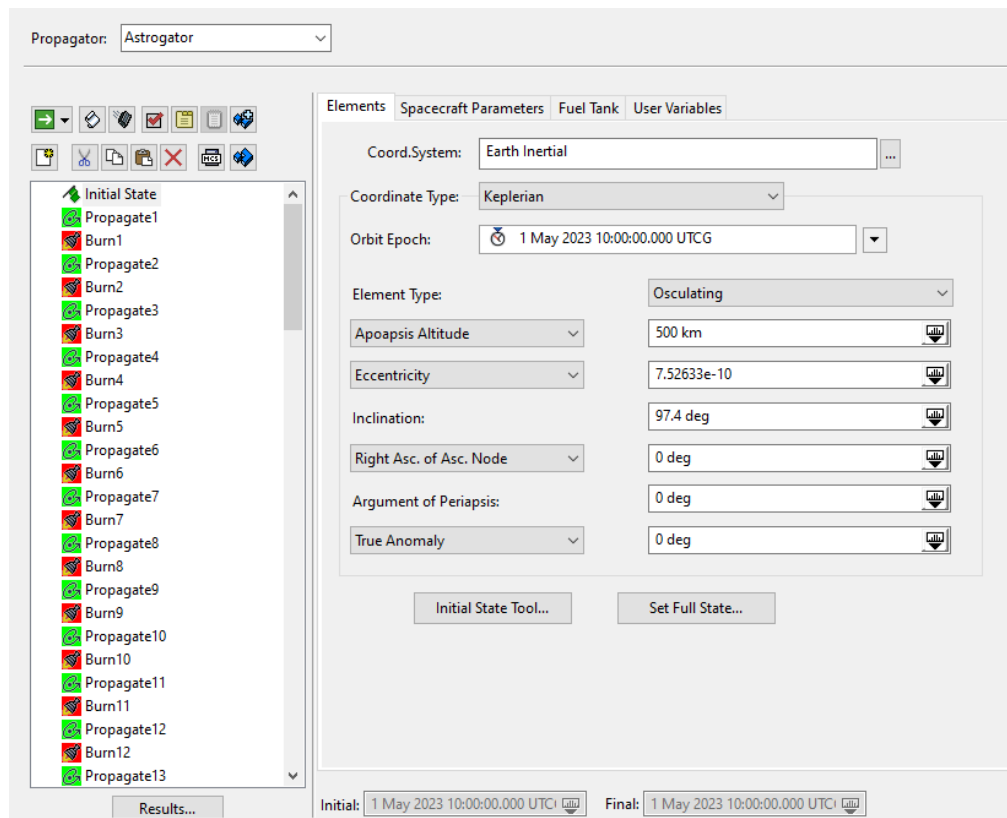


Figure 4: Exemplary "Initial State" segment

In the ‘Spacecraft Parameters’ tab, located in the initial state of the satellite, the mass of the satellite and parameters such as the area affected by drag and the area affected by radiation pressure, which directly affects the torque tool, are parametrized. These significant parameters chosen values for STK simulations is shown in Figure 5.

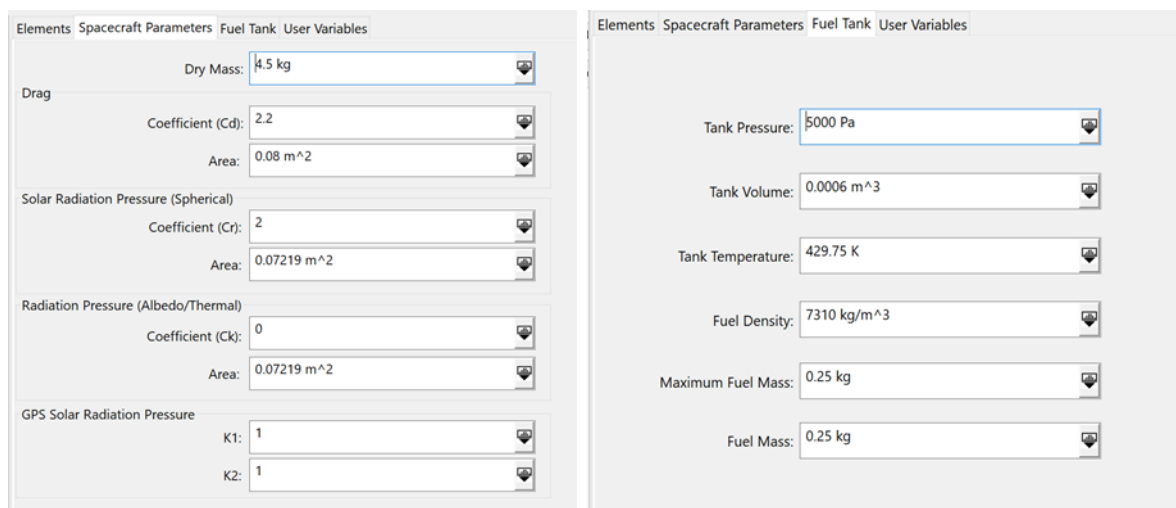


Figure 5: Satellite parameters, which influence torque tool and maneuvers.

Following the initial state declaration, the satellite propagates for one orbit stopping at the next perigee pass. The propagator used here is HPOP (High-Precision Orbit Propagator) which takes into account the forces acting on the satellite from earth’s unequal mass distribution, the sun, moon and other planets, atmospheric drag, solar radiation pressure and other factors.

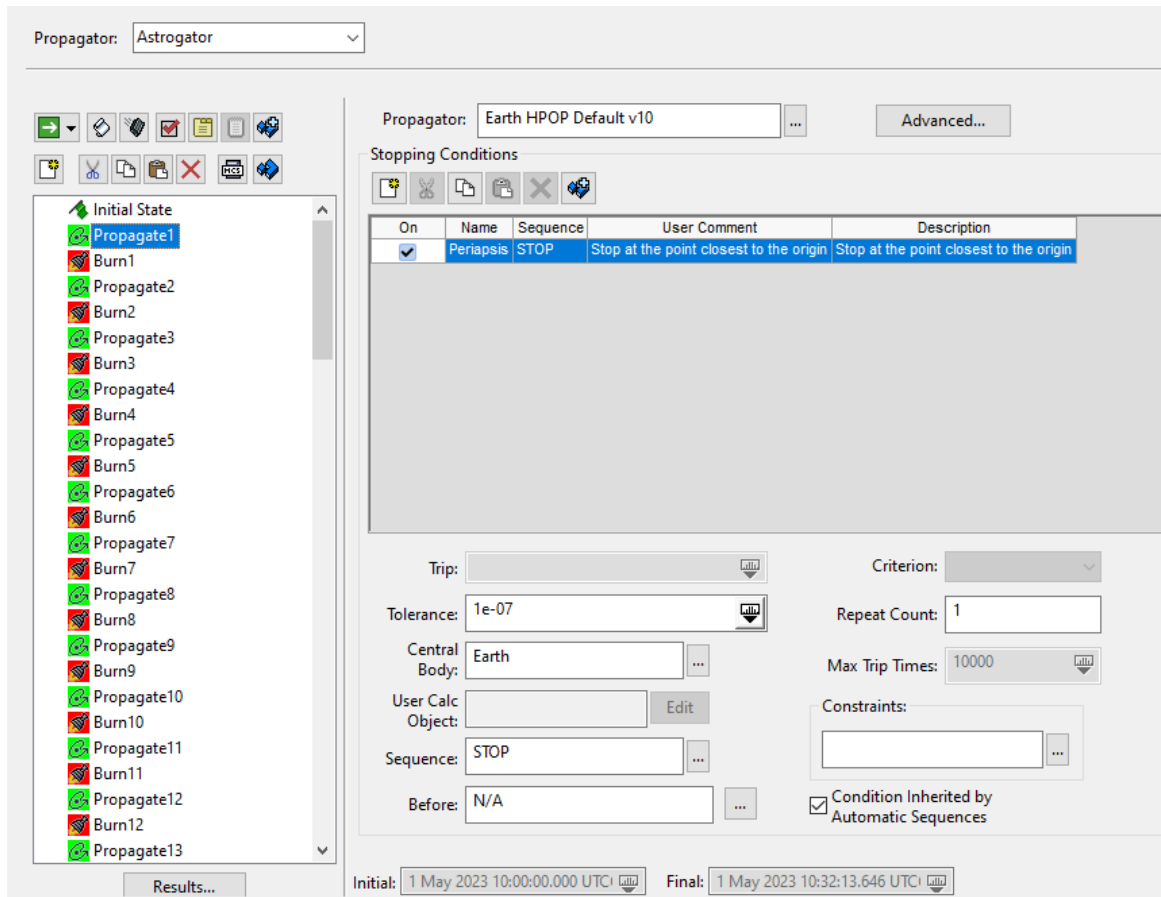


Figure 6: Propagation segment

This segment is repeated after every thrust maneuver of CLIMB. Figure 6 shows such a section. The stopping condition which can be seen here describes the segment coming to an end at the next periapsis as described above.

Once the satellite reaches perigee, a thrust maneuver of 10 min is carried out with the electrical Enpulsion thruster. It is important to note, that the thrust duration is centered around periapsis, meaning that half of the burn is performed before periapsis and the other half after this point. This is done to ensure an as close to optimal as possible performance of the spacecraft. This segment also defines the engine model used by STK, which, in this case, is taken from the Enpulsion thruster. Also, the direction of the burn is set directly opposite the velocity vector, thereby assuming the optimal thrust direction for an orbit raising maneuver. Figure 7 shows such a thrust segment.

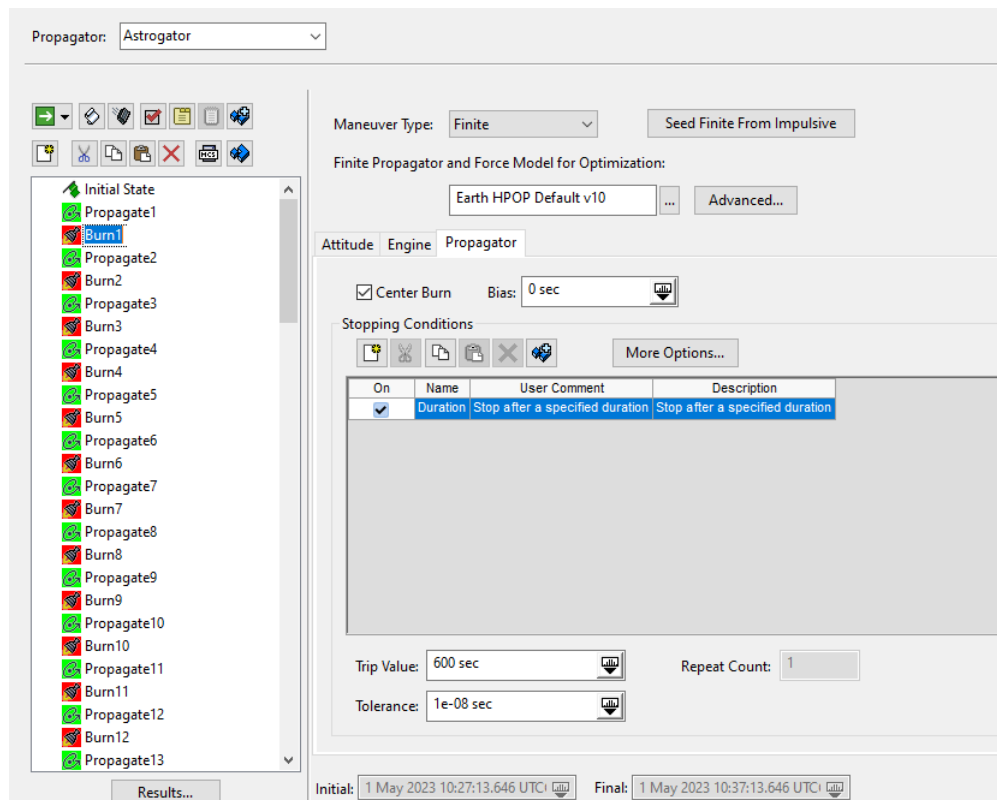


Figure 7: Thrust segment

All propagation and thrust segments are identical, except for the time of execution. In all ‘Burn’ segments, a FEPP thruster with 350  $\mu\text{N}$  thrust and 1900 s Isp was used, as shown in Figure 8.

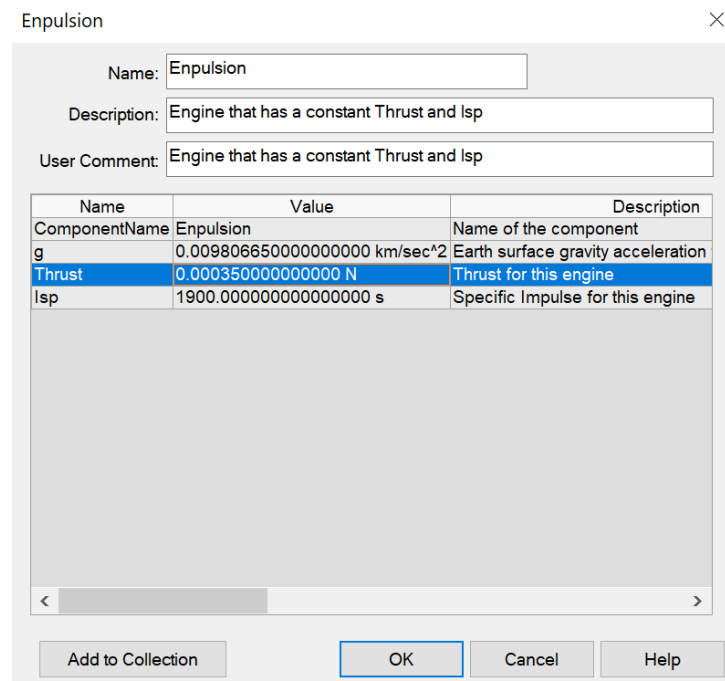


Figure 8: Engine model used for thrust maneuvers.

After the simulation is automatically carried out, the perigee passes, and communication windows are recorded and saved as separate .csv files for the following MATLAB interface to access.

## 3.2 MATLAB GUI

The MATLAB GUI serves as the central interface for the operator to plan the upcoming maneuvers of CLIMB. It is laid out as can be seen in Figure 9:

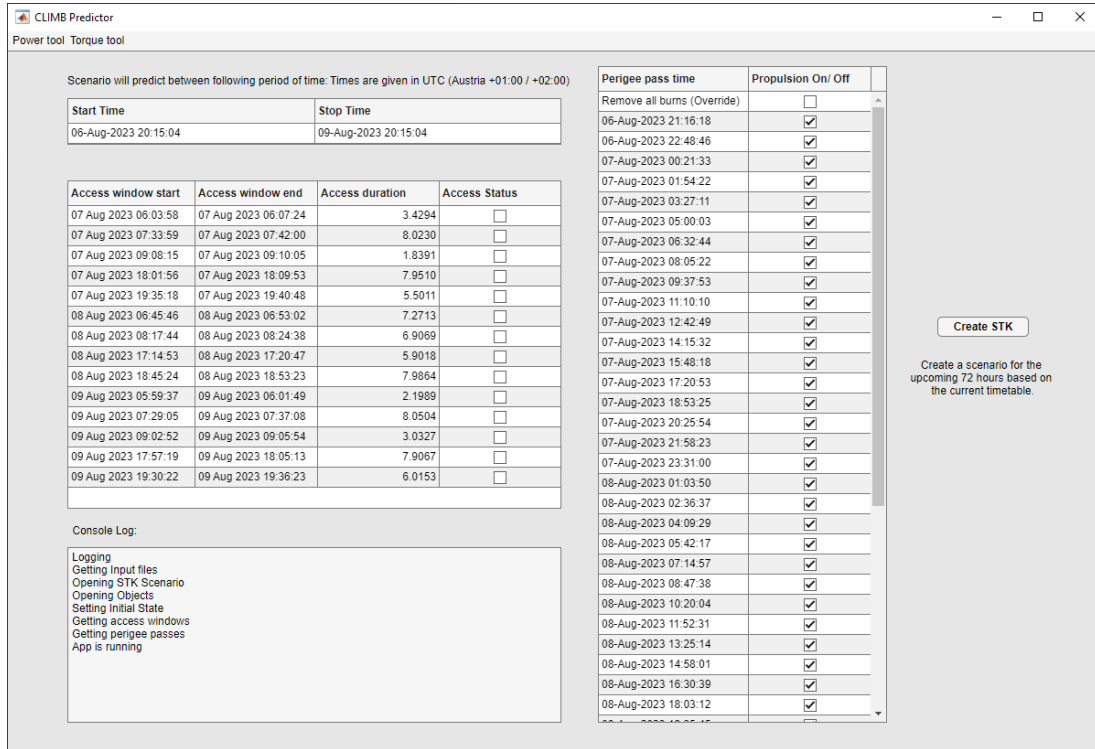


Figure 9: Layout of the MATLAB GUI

The toolbar on top gives the user direct access to the variable parameters of the power- and torque tools. On the top left corner, the time window for the simulation is given. Note that times and dates are set in the UTC (Coordinated Universal Time) format. To adjust for Austrian time, one or two hours have to be added (depending on summer- or wintertime). Below that, the access windows and their respective durations are displayed, allowing the operator to easily select the desired ones. On the right side are the possible thrust opportunities which the user can select. These are, unlike the communication maneuvers, preselected as a default. If the operator wishes to deselect all of them at once, they can override the selection in the first row.

On the lower left side of the window, the GUI logs its actions for the user to see. These automatic actions start from getting the input TLE files all the way to outputting the STK2 reports and using them in the different tools. This allows the user to effectively simulate everything they need to know with the click of a button. The “Create STK” button (middle right) starts STK with the given inputs from the user, simulates the scenario, runs the tools, and creates all output files.

A more detailed flowchart of the MATLAB GUI is explained in Figure 10. In the flowchart, three main actions set the calculation run: start the app and press the “Create STK” button (painted red in the flowchart). Two main pre-processing, torque and power tools, are marked in red, and their calculation will be explained in more detail in the following chapters. Documents marked in green are the main outputs in CSV format, which will be displayed in the Grafana display. “Connect to STK” in the process in amber color bridges the gap between the two softwares, which includes building a connection between the MATLAB app and STK, reading scenario data, getting access to satellite parameters, getting access to the report generated, and exporting these reports.



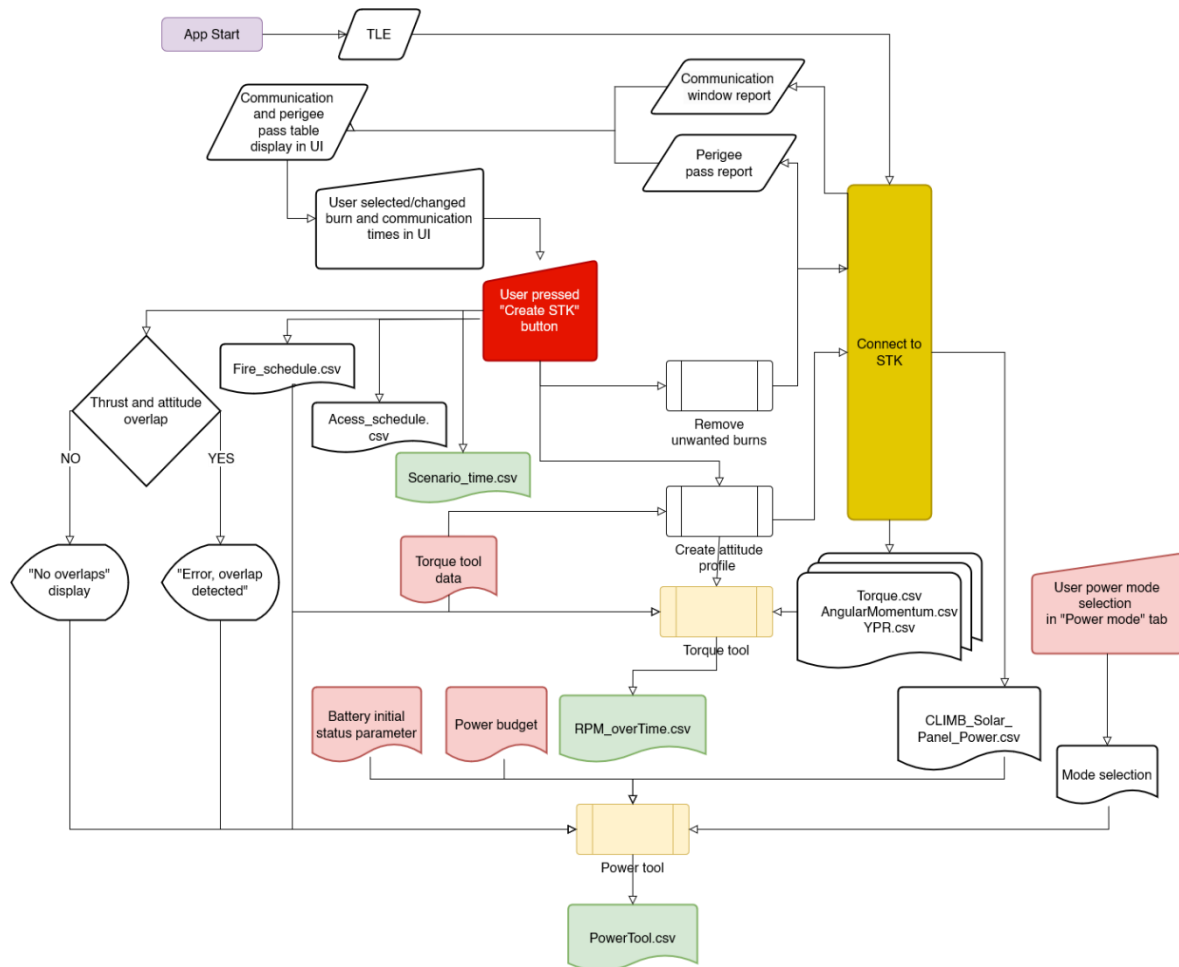


Figure 10: MATLAB GUI flowchart

Focusing on the top toolbar, displayed in Figure 11, interactions with the power tool is split into three different categories:

- The power mode selection: Here the power mode can be chosen, indicating if the spacecraft is operating in cruise mode (default), science mode, recovery mode, or safe mode.
- The power consumption data: This opens a small table where the current battery status parameters, voltage, capacity, and cycle information can be altered.
- The battery model data: This gives the user access to the extensive model of CLIMB's Power budget parameters.

The tab for the torque tool opens a table where the model parameters of this tool can be modified in addition to information on the ADCS, and the CG (Center of Gravity) and Moments of inertia of the spacecraft as well as possible misalignments.

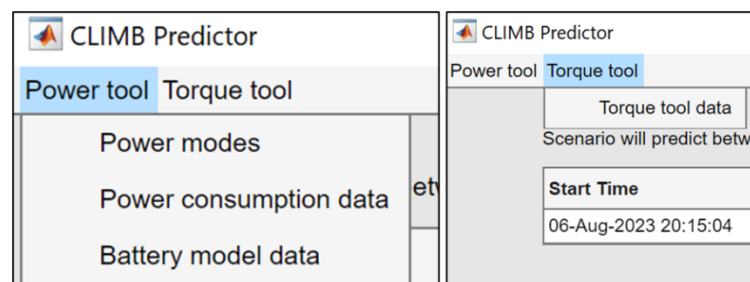


Figure 11: MATLAB GUI tabs expanded.

### 3.3 STK 2

The second STK scenario is created automatically when the operator presses the “Create STK” button in the GUI. It is not actually a “new” scenario, but it expands on the already existing STK1 file. Therefore, any changes made to the STK1 scenario regarding, for example spacecraft parameters are adopted. It takes the additional input for the chosen communication windows and thrust maneuvers and implements them in the simulation. This implementation is split into two parts: the thrust maneuvers, and the attitude segments. They operate mostly independent of each other.

Since all possible thrust maneuvers are already activated in the STK1 scenario, only unwanted burns are removed. This results in two (or more) propagation maneuvers scheduled back to back, resulting in CLIMB not thrusting for however many orbits are propagated after one another. Figure 12 shows such a scenario. Here the 2<sup>nd</sup> and 3<sup>rd</sup> burn are removed, allowing CLIMB to just orbit the earth for three consecutive rotations.

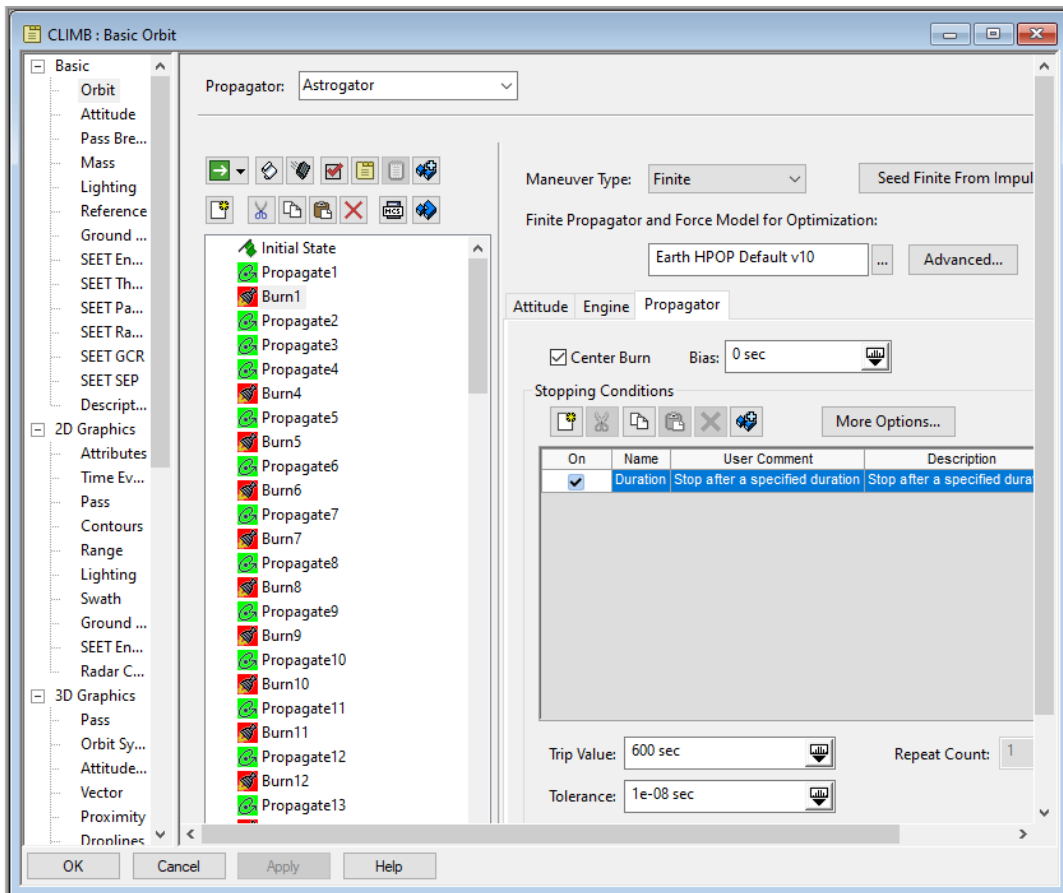


Figure 12: Removed burns in Astrogator

The attitude profiles are inserted automatically through MATLAB. Also, to accommodate for the capabilities of CLIMB’s ADCS system, a slew segment is added before and after each targeted attitude. Such an exemplary attitude profile is shown in Figure 13. Notice that between each “Thrust”, “Sun”, and “Communication” segment there is a “Slew” segment.

Using this information, the GUI checks for overlaps in the attitude profile (even for the “Slew” segments). If such an overlap is detected, the operator is informed in the console log of the GUI and can see the potential error in the relevant Grafana dashboard.

Name	Source	Number	Start Time	Stop Time
Sun_t	Basic	25	7 Aug 2023 05:15:03.00	7 Aug 2023 05:58:58.00
Slew_c0	Basic	26	7 Aug 2023 05:58:58.00	7 Aug 2023 06:03:58.00
Communication	Basic	27	7 Aug 2023 06:03:58.00	7 Aug 2023 06:07:24.00
Slew_c	Basic	28	7 Aug 2023 06:07:24.00	7 Aug 2023 06:12:24.00
Sun_c	Basic	29	7 Aug 2023 06:12:24.00	7 Aug 2023 06:17:44.00
Slew_t0	Basic	30	7 Aug 2023 06:17:44.00	7 Aug 2023 06:22:44.00
Thrust	Basic	31	7 Aug 2023 06:22:44.00	7 Aug 2023 06:42:44.00
Slew_t	Basic	32	7 Aug 2023 06:42:44.00	7 Aug 2023 06:47:44.00
Sun_t	Basic	33	7 Aug 2023 06:47:44.00	7 Aug 2023 07:28:59.00
Slew_c0	Basic	34	7 Aug 2023 07:28:59.00	7 Aug 2023 07:33:59.00
Communication	Basic	35	7 Aug 2023 07:33:59.00	7 Aug 2023 07:42:00.00
Slew_c	Basic	36	7 Aug 2023 07:42:00.00	7 Aug 2023 07:47:00.00
Sun_c	Basic	37	7 Aug 2023 07:47:00.00	7 Aug 2023 07:50:22.00
Slew_t0	Basic	38	7 Aug 2023 07:50:22.00	7 Aug 2023 07:55:22.00
Thrust	Basic	39	7 Aug 2023 07:55:22.00	7 Aug 2023 08:15:22.00
Slew t	Basic	40	7 Aug 2023 08:15:22.00	7 Aug 2023 08:20:22.00

Figure 13: Example attitude profile

## 4. Tools

The overall planning tool was divided in the above-mentioned STK scenario and three separate tools which would process the outputs from STK and custom made GUI.

Once CLIMB is in orbit, the Torque Tool and Power Tool will be both planning and monitoring tools. In contrast, the Propulsion Tool will be mainly a monitoring tool that could feed data to the other tools to improve their accuracy.

The monitoring part of each tool consists of the processing of telemetry data. For example, the Torque related data from ADCS for the torque tool and power-related data from the EPS and other monitored power supply rails or subsystems. Given the complexity and number of parameters to be scanned in the ENPULSION thruster, it was decided to dedicate a specific tool for its monitoring: the Propulsion tool.

However, in practice, this distinction in the monitoring part of the tools will not be significant because it would be achieved by the same methods for all tools as described in section 5. In fact, all telemetry data should be included in a Grafana visualization on a subsystem level.

In summary, this is the reason why this project focused on the planning part of each tool, rather than the monitoring one, with the exception of the propulsion tool. Therefore on providing the user with tools to validate a certain maneuver set based on the propagation of resources such as ADCS reaction wheels RPMs or Battery capacity available.

As will be mentioned within each tool, we used CSV files to exchange data both between the STK scenario and the tools and between the different tools (e.g. Torque Tool and Power Tool).

### 4.1 Torque tool

During its operation, the satellite would need to control its attitude for various reasons, such as attitude mode switches, environmental disturbances, and internal reasons, which occur when propulsion operates, such as CG and Geometrical center mismatch, thrust vector misalignment, and are shown in Figure 14. The effects of each of these elements on planning CLIMB operation will be discussed in this chapter.

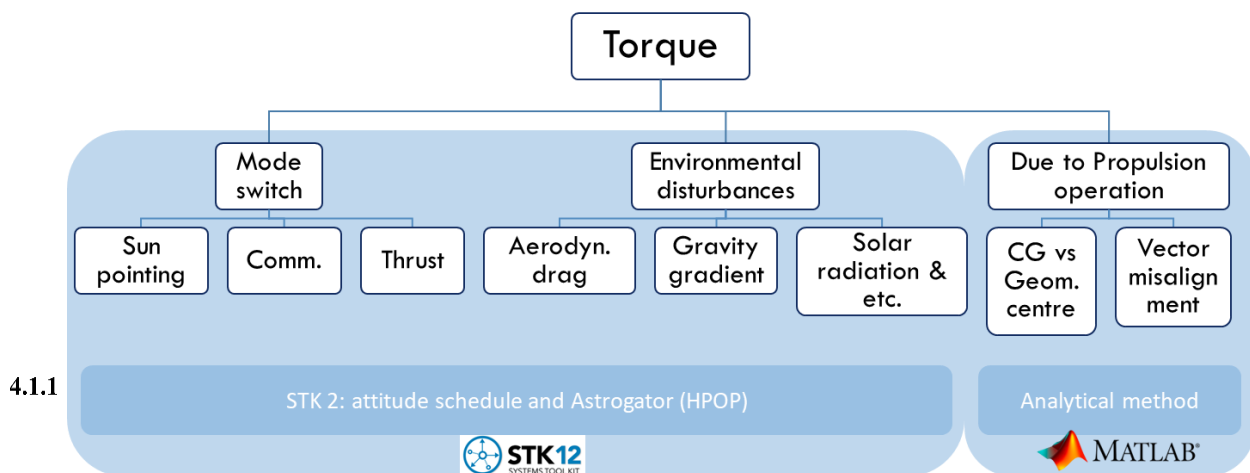


Figure 14: Torque types on CLIMB

#### Attitude mode control

The satellite needs to be aligned differently as required from a functional point of view for its sub-systems. As it was defined during JTP, CLIMB is planned to have four main attitude modes (in STK,

attitude control aligned means two vectors strictly coincide, constrained means the angle between two vectors is minimized)

- Communication mode:
  - +Y axis, where the S-band is located, is aligned with the nadir to face the ground station due to the unidirectional property of the S-band antenna.
  - -Z axis, opposite to the direction where the thruster is located, is constrained with the velocity vector: due to FEEP thruster requirements and in JTP, it was analyzed that this setting keeps the +Z axis in 15 degrees half angle cone to the direction opposite to the Velocity vector.
- Thrust mode:
  - -Z axis is aligned with the Velocity vector to raise the orbit
  - Vector 45 degrees between +X and +Y is aligned to the Sun: the solar panel's normal vector is directly aligned with the Sun.
- Sun mode:
  - Vector 45 degrees between +X and +Y is aligned to the Sun: the solar panel's normal vector is directly aligned with the Sun.
  - -Z axis, opposite to the direction where the thruster is located, is constrained with the velocity vector: due to FEEP thruster requirements and in JTP, it was analyzed that this setting keeps the +Z axis in 15 degrees half angle cone to the direction opposite to the Velocity vector.
- De-tumble mode: this mode would directly depend on the ADCS sub-system's magnetorquer and Earth's magnetic field and is not covered for attitude analysis of this STP due to its complexity.

As mentioned in Chapter 3 previously, the availability of these modes results from the first iteration in STK1, especially the presence of the CubeSat in line of sight of the ground station and perigee pass timestamps for the next 72 hours. In detail:

Table 2: Attitude modes' characteristics

Attitude modes	Availability	Duration	Controllability
<b>Communication</b>	When in line of sight of the ground station	Depends on the orbital position	Active/ passive communication window selection controllable by Schedule GUI
<b>Thrust</b>	When passing through the perigee	20 minutes: 5 minutes before the burn 5 minutes burn before perigee pass 5 minutes after the perigee pass 5 minutes after the burn	The duration of attitude setting before and after the burn is controllable in the 'Torque tool' tab in Schedule GUI. Burn duration could be changed in STK1.
<b>Sun</b>	At all times (Sun pointing no matter if in the sun or shadow of the earth)	At all available times when two previous modes are not activated/chosen by the user.	Automatic

One of the most critical aspects of this STP was to develop a method for selecting communication windows and firing opportunities during perigee passes from all communication and firing modes available in STK1. Using a MATLAB tool would allow the chosen attitude modes to be inserted into STK2. The main command to control this attitude schedule is shown in Figure 15 :

```
baseString = string(strjoin(['AddAttitude */Satellite/', 'CLIMB', ' Profile ', profileName, ' ', startTime, ' ', profileType], ''));
```

Figure 15: Snip from Schedule GUI code

Figure 15 describes the main elements that were required to set attitude mode in the attitude schedule of the satellite:

- Profile name is used to distinguish between different modes of the same type. For our case, we required the following profiles:
  - Slew\_c0: Slew into communication mode
  - Communication: Communication mode
  - Slew\_c: Slew from communication mode
  - Sun\_c: Sun mode after communication mode
  - Slew\_t0: Slew into thrust mode
  - Thrust: Thrust mode
  - Slew\_t: Slew from thrust mode
  - Sun\_t: Sun mode after thrust mode
- Profile type:
  - AlignConstrain: profile type, where one body vector is aligned, and the other is constrained. As mentioned, all three main attitude modes are modeled with this profile type.
  - FixedTimeSlew: profile type, which describes how fast the satellite switches from one mode to another. Maximum angular speed and velocity depend on reaction wheel selection.
- Start time describes the start of the profile, which puts modes in the correct sequence. Duration is defined as the time between two neighboring profiles' start times.
  - Communication mode and thrust mode start times are inputs from user-selected active profiles' start times.
  - All slews start times are defined by user-selected active mode profiles' (communication and thrust) stop times.
  - Sun mode start time is defined by the slew length past since the start of the user-selected active profiles' start times.

Attitude schedules are modeled in the way that the duration of the slew is also controllable from Schedule GUI. Modifying attitude schedules to control the duration of slews can be done through the Schedule GUI. However, the current model only provides fixed-time profiles for slews to ensure seamless integration with the Schedule GUI. Other types of slews with fixed speed or angular acceleration profiles exist, but integrating them with different profile types may pose a more significant challenge. The exemplary schedule for the first 6 hours is displayed in Figure 16.

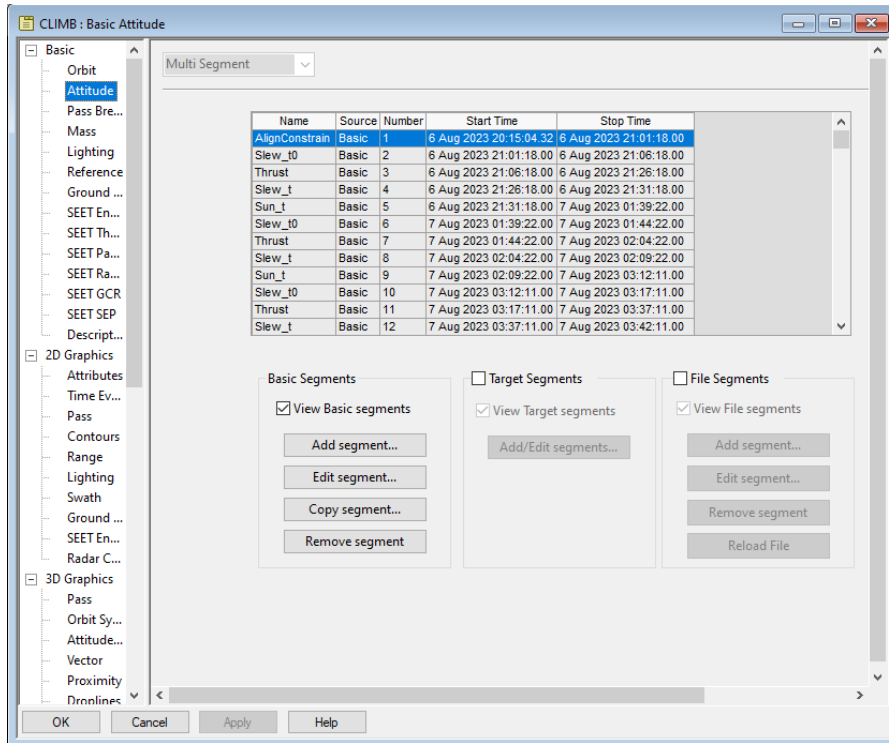


Figure 16: Exemplary attitude schedule with custom profiles.

## Environmental disturbances

### 4.1.2

Numerous scenarios with the same satellite physical model (mass, moments of inertia, etc.) but different propagators and altitudes were investigated over six months to examine the impact of the environmental disturbances.

First, the torque and angular momentum of the CLIMB at 1000 km altitude with the fixed attitude were investigated over six months with two orbit propagation models. Satellite attitude was ‘fixed,’ meaning the  $-Z$  axis was aligned with the velocity vector, and the  $+X$  axis was fixed toward the position vector. This attitude profile would focus only on environmental disturbances and not consider other factors, such as mode switches or sun positioning. The attitude description of this scenario is visually represented in Figure 18, with reference axis VNC shown in Figure 17.

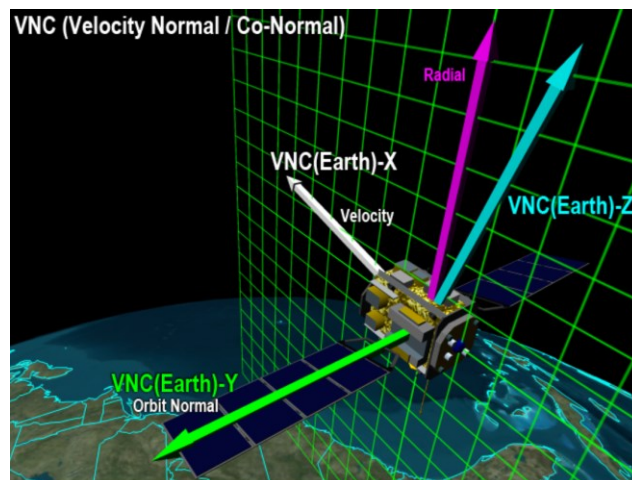


Figure 17: VNC (Velocity Normal Co-Normal) reference system visualization [1]

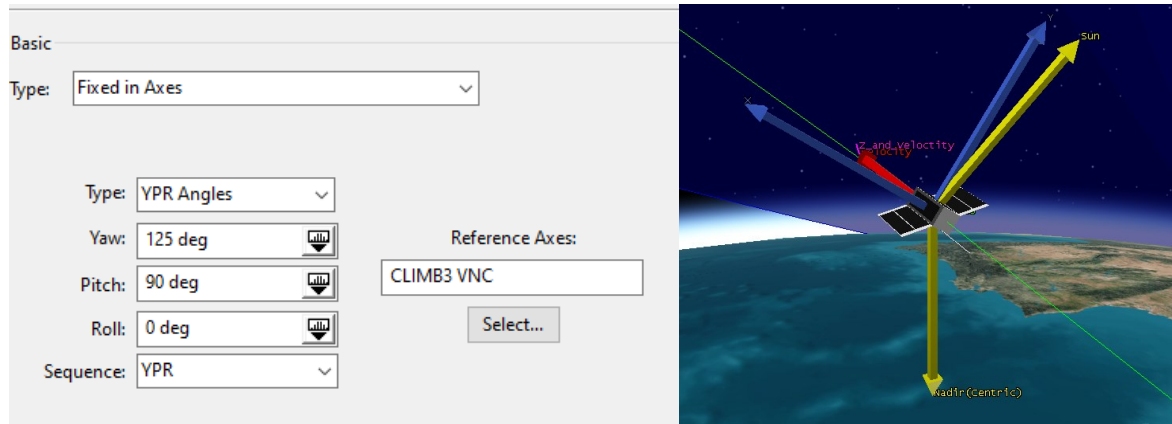


Figure 18: Fixed axis description in VNC reference system and visualization.

The first scenario had a simple ‘Two body’ propagator, and the second was assigned an ‘HPOP’ propagator. According to the STK bulletin, the HPOP model includes central body gravity, solar radiation pressure, drag/atmospheric density, third body gravity, and covariance (a statistical measure of the joint variability of two random variables) [2]. As a result of these two scenarios, it was confirmed that having an HPOP propagator would include environmental disturbances.

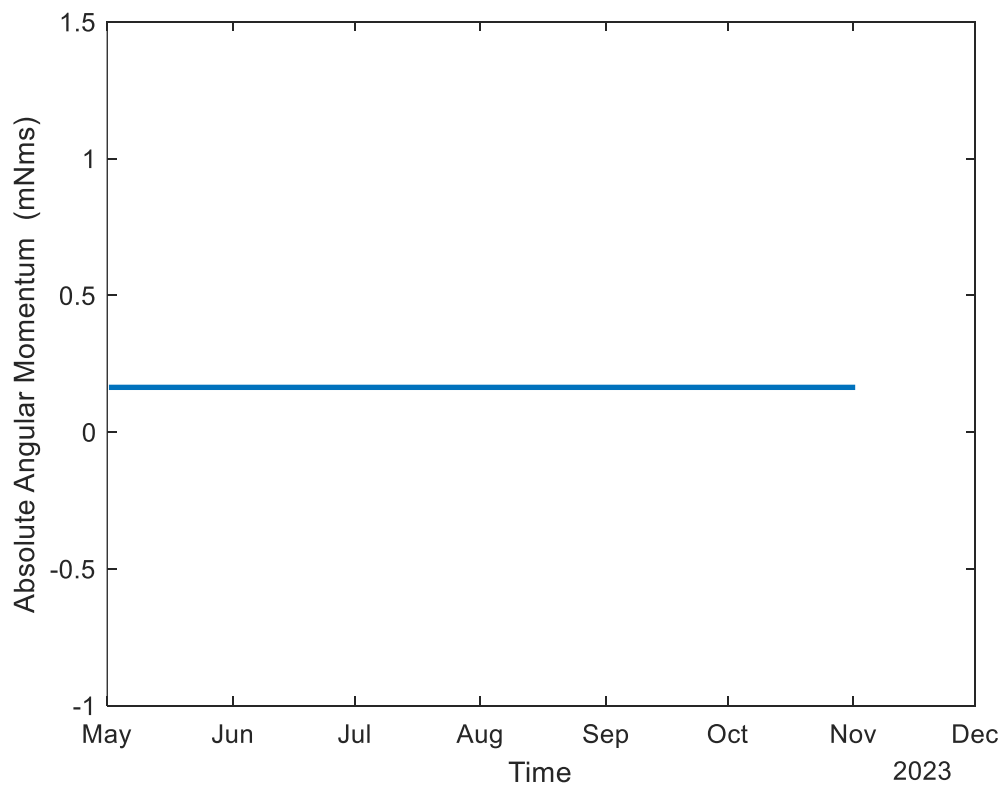


Figure 19: Absolut angular momentum using the ‘Two body’ propagator.

Graphs Figure 19 and Figure 20 were obtained by extracting angular momentum reports in the satellite's X, Y, and Z body axes in STK and post-processing in MATLAB. Code lines for the post-processing data from the STK model over six months are attached in Appendix 1. As a result, we can notice how the satellite in the ‘Two-body’ model maintains a constant angular momentum equal to 0.16323 mNms; hence, it has constant angular velocity.



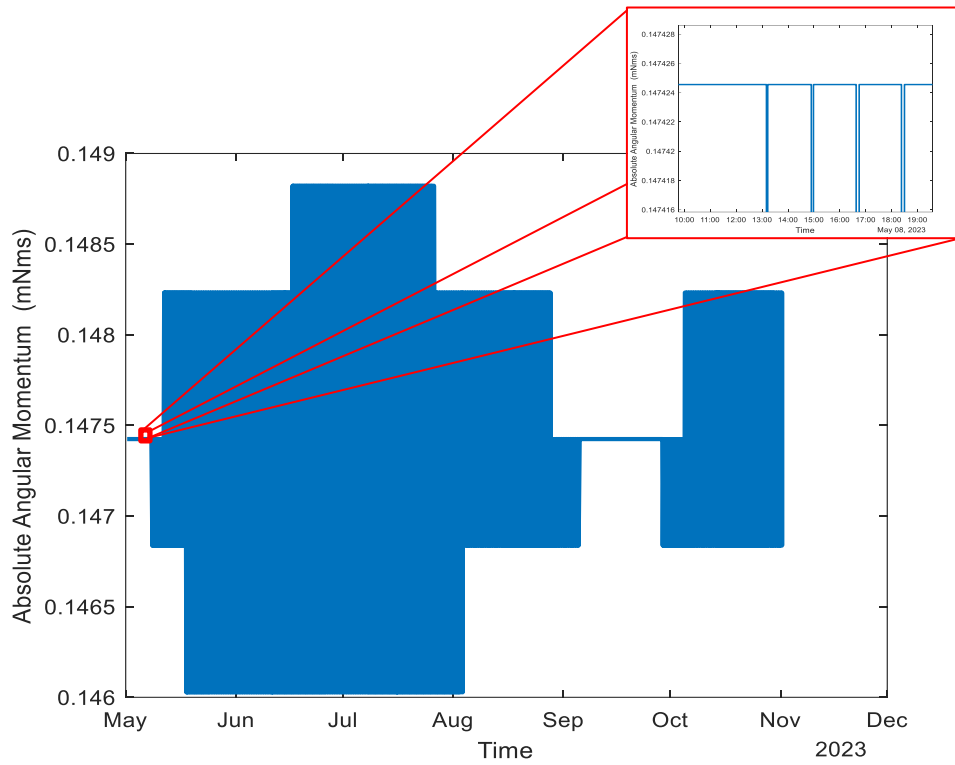


Figure 20: Absolute angular momentum using ‘HPOP (High Precision Orbit Propagator).’

Meanwhile, Figure 20 demonstrates that the satellite in the HPOP scenario experiences a change in angular momentum, which comes from non-constant angular velocity. Upon closer inspection, it is apparent that periodic variations are occurring consistently in the specific area that has been magnified. These fluctuations are observable at all times.

Further, the third scenario was conducted similarly to the last scenario, but at a different altitude, to show environmental disturbances at the start of the CLIMB mission, meaning at a low altitude of 500 km, as shown in Figure 21.

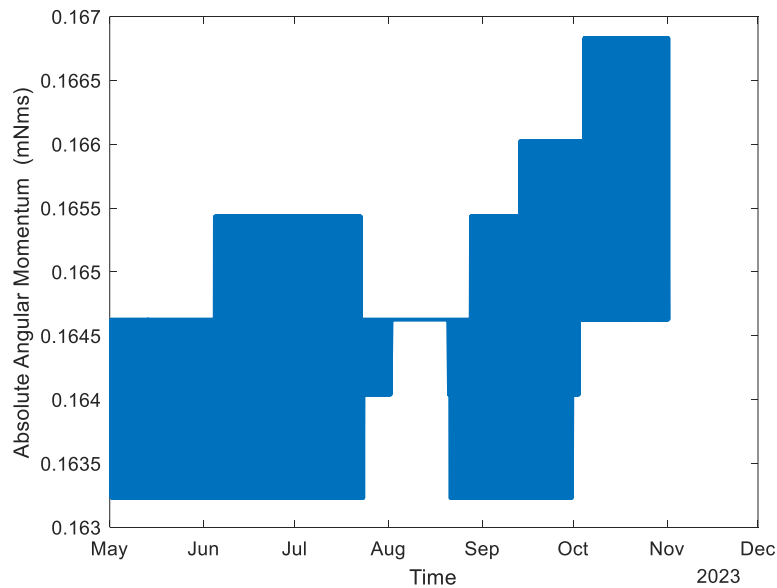


Figure 21: Absolute angular momentum using ‘HPOP (High Precision Orbit Propagator) at low altitude.

Comparing two similar scenarios at two different altitudes in Figure 20 and Figure 21 suggests that there is no noticeable significant difference. However, monitoring data shows that at 1000 km and 500 km altitudes, the largest absolute angular momentum is 0.0028 and 0.0036 mNm, respectively. It is suggested that this minor change in two scenarios comes from the fact that CLIMB size is only 3U and doesn't have a sizeable effective area susceptible to aerodynamic drag.

### **Torque due to the shift of the center of mass and misalignment of the thrust vector**

As observed from JTP and previous sub-chapters, attitude mode shifts directly depend on the slew rate and do not contribute to cumulative angular momentum due to their periodic nature. Moreover, the environmental disturbances effect is complicated to analyze by looking at the torque and angular momentum per body axis report from STK; hence, it was decided to extract yaw, pitch, and roll in degrees from STK and post-process similar to JTP, but with changes to eliminate cyclic lines, which was learned due to yaw, pitch, and roll (YPR) being in -180 to +180 limit. It should be noted that Yaw corresponds to Z, pitch corresponds to Y, and roll corresponds to the X axis of the satellite. This sub-chapter will discuss the main steps of the version of the torque tool integrated into the final Schedule GUI.

Figure 22 displays YPR extracted directly from STK2.

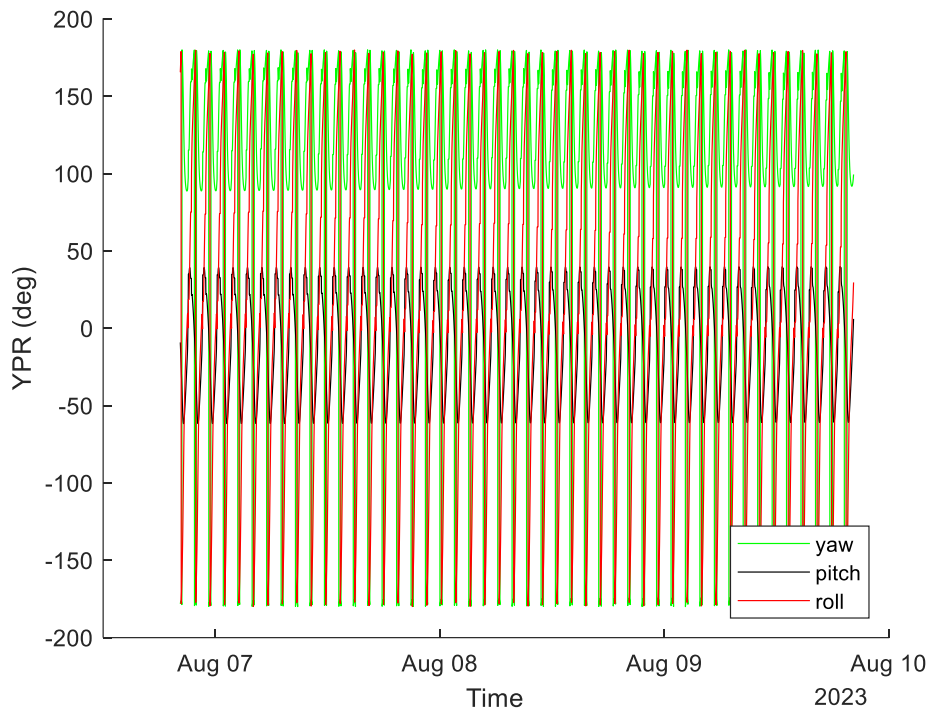


Figure 22: Yaw, pitch and roll over time from STK2.

Data extracted from STK2 was post-processed to remove periodic lines due to the -180 to +180 limit, and YPR, after removing the -180 to +180 limit, is shown in Figure 23.

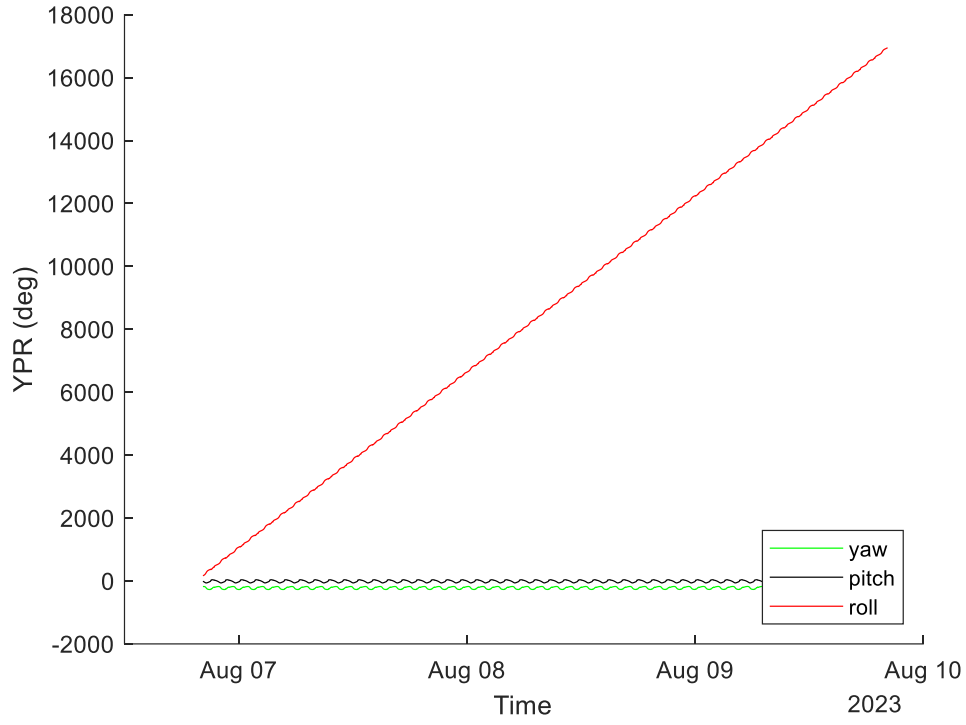


Figure 23: Yaw, pitch and roll over time after removing -180 to +180 limit.

Further, YPR is differentiated to find yaw, pitch and roll rates, and  $\omega$  in (deg/sec), which correspond to the angular velocity.

$$\omega = \frac{d(YPR)}{dt} \quad (1)$$

Angular momentum for yaw, pitch, and roll,  $L_i$ , using angular momentum,  $I_i$ , values, which are mentioned in Table 1, is calculated as follows:

$$L_{cum\_i} = \omega \cdot \frac{\pi}{180} \cdot I_i \quad (2)$$

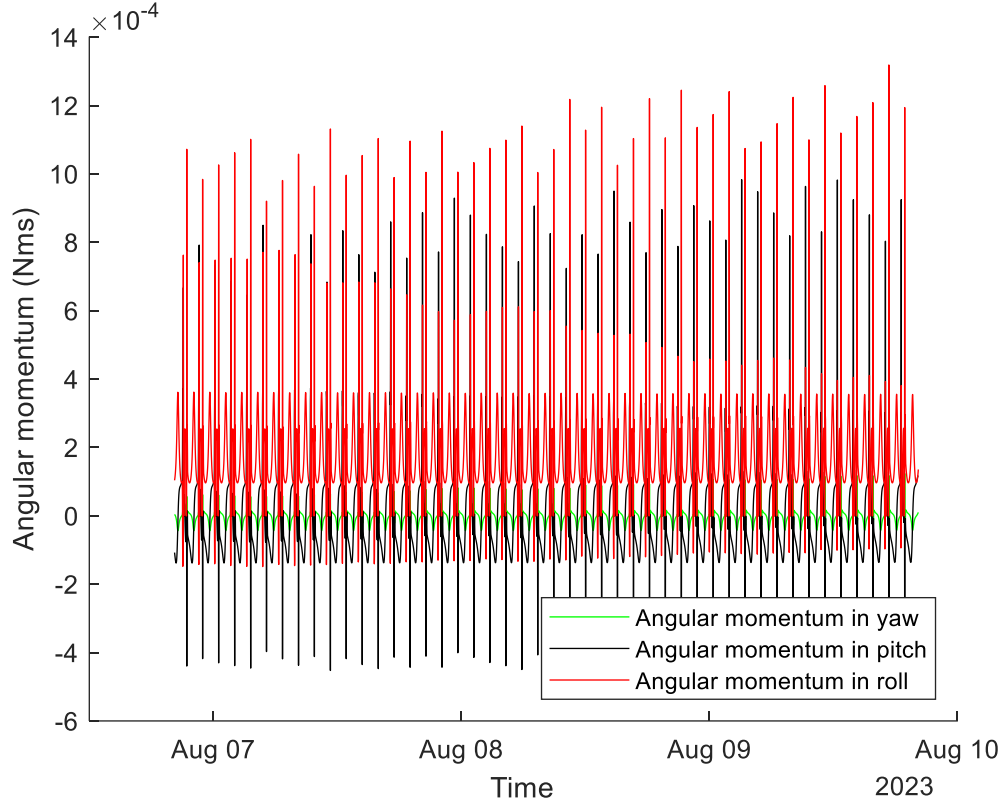


Figure 24: Angular momentum due to mode switch and environmental disturbances.

After these steps, angular momentum covers all the attitude changes covered in STK2: attitude mode switches and environmental disturbances. Afterward, the torque due to the center of the mass shift from the geometrical center and thrust vector misalignment needs to be added to the MATLAB code. The torque due to the center of mass shift from the geometrical center is calculated as follows:

$$T_{cm\_shift} = F \cdot \Delta d \quad (3)$$

Where  $F$  is thrust equal to  $350 \mu N$  and  $\Delta d$  values from the recent CAD model of the CLIMB are shown in Table 3.

Table 3: Center of mass shift in CLIMB

Center of Gravity shift, $\Delta d$	
<b>x</b>	0.0027
<b>y</b>	0.0047

Angular momentum is calculated by integrating torque over time and multiplied to burn schedule array to get angular momentum over time array in MATLAB, which is shown below.

$$L_{cum\_cm\_shift} = \text{cumulative}(T_{cm\_shift} \cdot \Delta t \cdot \text{Burn\_time\_status}) \quad (4)$$

Where, *Burn\_time\_status* is output from STK2 consist of binary array. 0 indicates no burn, and 1 indicates burn. As a result of this calculation, cumulative angular momentum is shown in Figure 25.

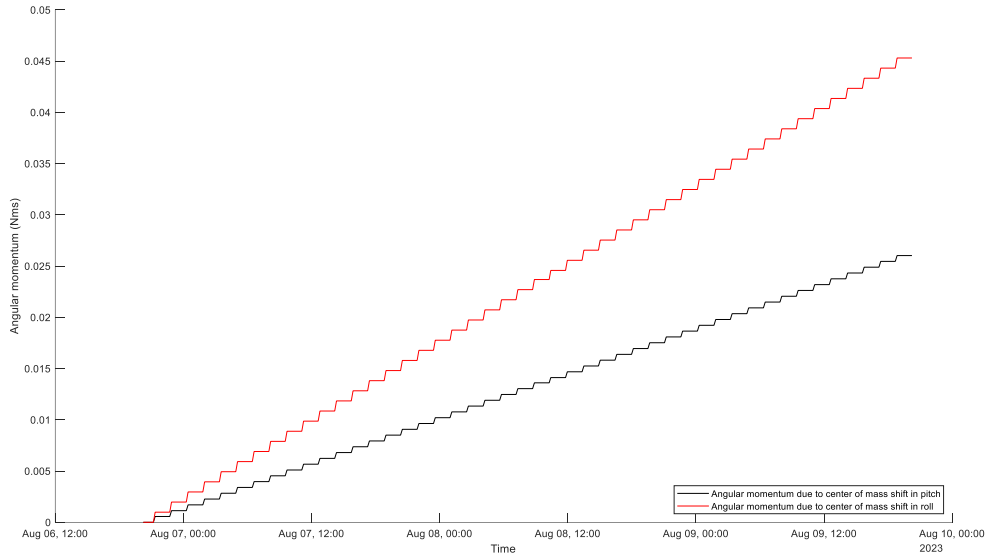


Figure 25: Cumulative angular momentum due to center of mass shift in pitch and roll.

Furthermore, the torque due to thrust vector misalignment is calculated:

$$T_{\text{vector\_mis}} = F \cdot \Delta z \cdot \sin(5^\circ) \quad (5)$$

5-degree misalignment is obtained from the FEEP thruster manufacturer, and increment z is the distance from the center of the CubeSat to the thruster and is assumed to be 0.013 m. Angular momentum contribution could be calculated similarly for the center of mass shift. And

$$L_{\text{cum\_vector\_mis}} = \text{cumulative}(T_{\text{vector\_mis}} \cdot \Delta t \cdot \text{Burn\_time\_status}) \quad (6)$$

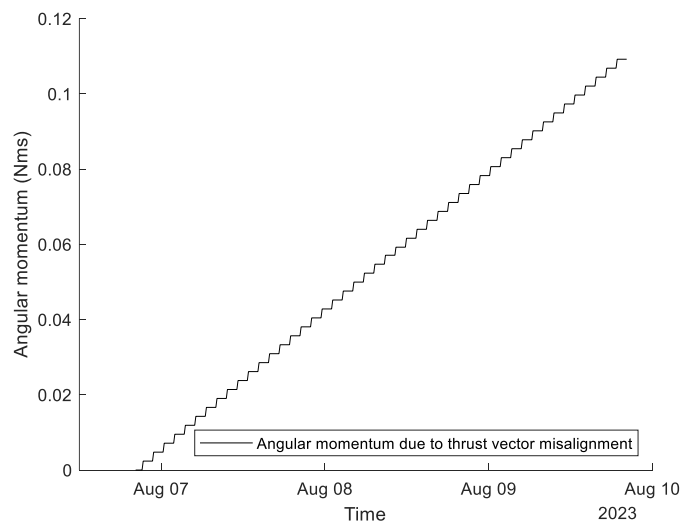
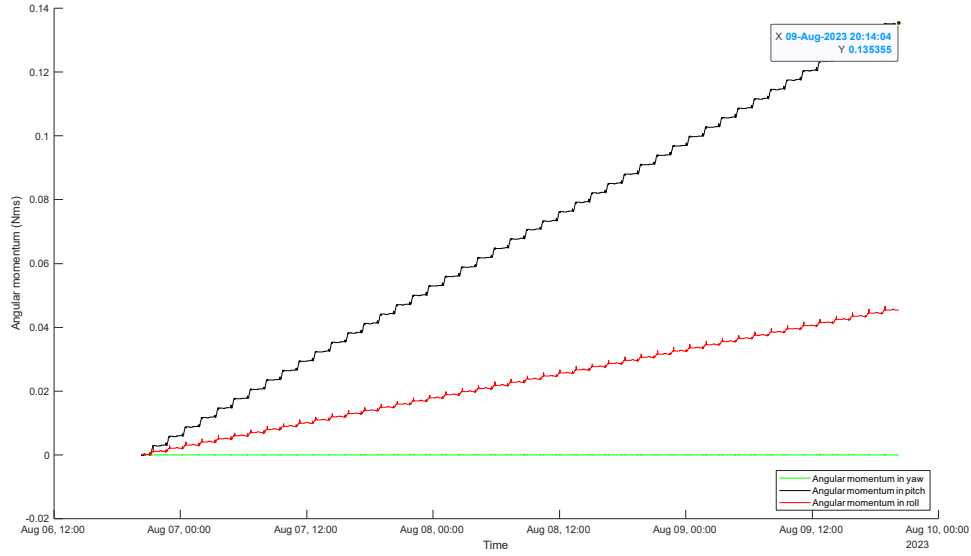


Figure 26: Cumulative angular momentum due to vector misalignment added to pitch.

Cumulative angular momentum due to vector misalignment is added to pitch (Y axis) or roll (X-axis), and as the worst case is considered, it is added to pitch.

Lastly, after summing all the cumulative angular momentum, the result suggests that reaction wheels accumulate 0.045 Nms daily. In contrast, the reaction wheel under consideration CW0057 has 0.0057 mNms total angular momentum storage, which is not a good solution for the satellite



operation.

Figure 27: Total angular momentum.

Finally, the rpm of the flywheel is calculated as follows:

$$\text{rpm}_i = \text{rpm}_0 + \frac{L_i}{I_{fw\_i}} * \frac{30}{\pi} \quad (7)$$

Where  $I_{fw\_i}$  indicated moment of inertia of the reaction wheel and assumed to be equal to 0.00007 kg·m<sup>2</sup>.  $\text{rpm}_0$  is the last downloaded rpm of the reaction wheel. Finally, torque tool outputs, RPM, and RPM limit due to the chosen ADCS (currently set for CubeSpace Gen 2 with CW0057 reaction wheel) will be shown in the Grafana display, as demonstrated in Figure 28.

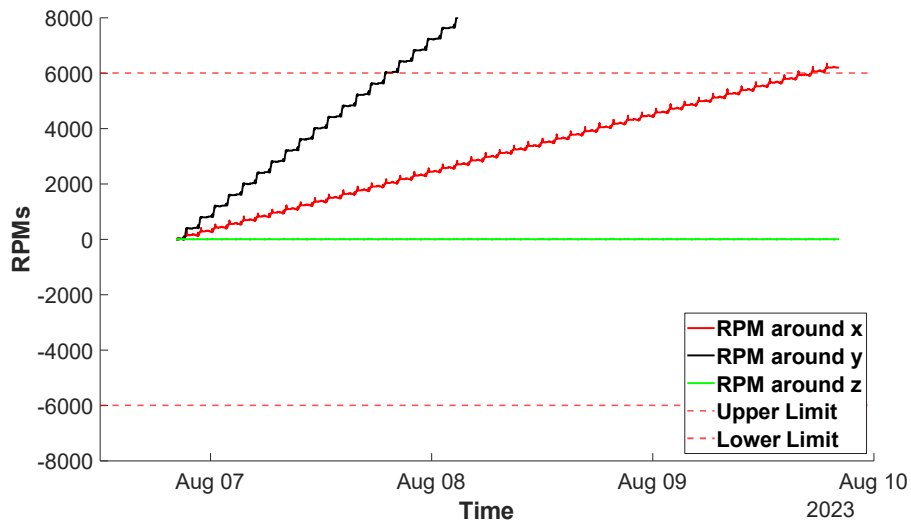


Figure 28: Total angular momentum.

The main parameters used in the Torque tool and which might change before the satellite launch or could be corrected during the commissioning phase after launch are collected in the Torque tool data Excel file and can be easily found in the 'Torque tool data' tab of the Predictor app. These parameters are shown in Table 4.

Table 4: Torque tool inputs: Torque\_tool\_data.xlsx

CubeSat CAD data			ADCS data sheet		
Moment of Inertia			Max Torque	2	mNm
I <sub>xx</sub>	0.1432	kg*m <sup>2</sup>	Max Ang. Momentum	5.7	mNms
I <sub>yy</sub>	0.14601	kg*m <sup>2</sup>	RPM limit	6000	
I <sub>zz</sub>	0.0235	kg*m <sup>2</sup>			
			Moment if Inertia	0.00007	kgm <sup>2</sup>
Center of Gravity shift					
x	0.0027	m			
y	0.0047	m			
I <sub>z</sub>	0.13	m			
Operational values			ADCS data: last update		
Thrust	0.00035	N	RPM	0	
Thrust duration	600	s	Ang.Mom X	0	
Thrust vector misalignment	5	deg	Ang.Mom Y	0	
			Ang.Mom Z	0	
Slew length	5	minutes			
Thrust half time	5	minutes			
Lead length before thrust	5	minutes			

## 4.2 Power tool

### Power tool overview

4.2.1 The Power tool shall provide the operation planning team with an overview of the power and energy resources of the spacecraft. This is one of the most important resources to be monitored and planned for, as it is required by every subsystem. The main output of this planning tool is the feasibility verification of the planned maneuvers as they were selected in the GUI (firings, downlink opportunities used) for the given orbit (TLE and subsequent STK II propagation). From the “CLIMB power budget V2” document, it can be seen that many power relevant “modes” have been considered for the CLIMB mission, namely:

- Propulsion mode
- Downlink mode
- Cruise mode
- Science mode
- De-tumbling mode
- Safe mode
- Recovery mode
- De-saturation mode (not present in the document)

However, it was decided that the scope of this project should be the initial study and development of these planning tools, and that they may be limited to consider the main operating modes of the satellite during normal operation avoiding more rare cases such as De-tumbling, safe and recovery mode. Nevertheless, it should be mentioned, that for a full overview of the operational scenario, the de-saturation mode should be included in a second iteration of the tool. In fact, according to the results from the Torque tool, the de-saturation maneuver using magneto-torquers may be required more often than originally anticipated, having therefore a higher impact on the power budget. Summarizing, the following were the modes considered in the Power tool:

- Propulsion mode
- Downlink mode (S-band communication only)
- Cruise mode
- Science mode

4.2.2 The default mode considered between maneuvers is the cruise mode.

### Input data

The following table summarizes the relevant input data for the Power Tool and their sources:

Table 5: Power tool input data

Input data	Source (e.g., software, STK scenario)	File
Perigee timestamps	STK II	Fire_schedule.csv
Access windows start-end timestamps	STK II	Access_schedule.csv
Background/idle power consumption value for the satellite	GUI-power tool tab	ModeSelection.csv



based on operational mode (e.g. cruise, science)		
Generated power from solar arrays in Watts	STK II	CLIMB_Solar_Panel_Power.csv
Power consumption values for each subsystem and for each mode	CLIMB Power Budget Excel sheet	CLIMB Power Budget V2.xlsx
Battery status at the beginning of scenario (Voltage, Cycles) and battery linear models coefficients	GUI-power tool tab	Battery_InitialStatus_Model_Parameters.xlsx
Reaction wheels RPM over time	Torque Tool	RPM_overTime.csv

Ideally the Propulsion Tool shall also help to increase the accuracy of the Power Tool by automatically correcting the power consumption value for the propulsion mode based on actual thruster telemetry data. However, this automation was not achieved during the STP project. Nevertheless, the user could apply such changes manually by monitoring thruster telemetry and editing the Power Budget table.

#### 4.2.3

### Battery models

One of the main objectives of the Power Tool was the simulation of the Battery capacity over time, considering realistic charging and discharging due to subsystem power consumptions and assess whether the given set of maneuvers was feasible from an energy perspective. This is important both from the battery management perspective and for the overall satellite operation safety. In fact, SOC should be maintained as high as possible to extend battery life and a very low SOC should be avoided to allow for one or more emergency reboots of the satellite, during eclipse, which is the worst case from the battery point of view in terms of SOC and temperature.

Two main battery models were used:

1. Capacity vs Voltage
2. Total Capacity vs Cycles

The first model was necessary because telemetry data only contains battery voltage and not capacity, therefore this model was needed to translate the measured or extrapolated initial voltage to an initial battery capacity as a starting point for the subsequent propagation. The model was derived from extrapolation of experimental data contained in the battery manufacturer datasheet for a single cell.

The second model was necessary to model the life-time degradation of the battery with respect to charge-discharge cycles. The actual behavior of the battery is a function of the way the battery is discharged or in other words the average DOD (Depth of Discharge). To simplify the analysis, since no experimental data was available at the time of writing, intermediate values on a single cell basis as per battery manufacturer datasheet [3], were used.

The graphs from which the analytical models were extrapolated, as well as the extrapolated linear equations are provided below. It can be easily seen that the best fit for the experimental data is in both cases and in first approximation, a linear model. For the Voltage vs Capacity graph, it should be noted that a steep non-linear drop occurs around 30% SOC. The errors associated with this non-linearity were assumed to be small since the satellite should avoid operations in this battery region for the safety reasons provided above.

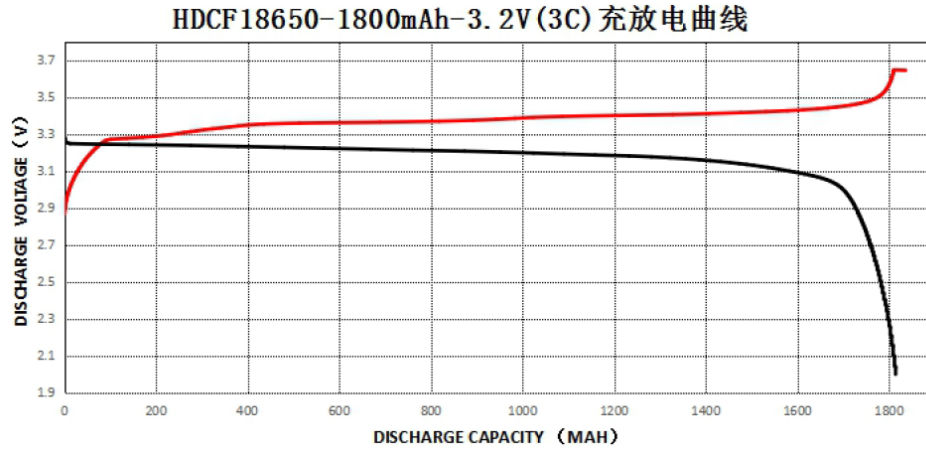


Figure 29: Voltage vs capacity graph for a single cell

$$V = 3.27 - 0.0625 * C$$

\*with  $C$  in Ah

The linear equation was multiplied by the number of cells in series (2S) to obtain the final equation used for the first battery model:

$$V = 6.54 - 0.125 * C$$

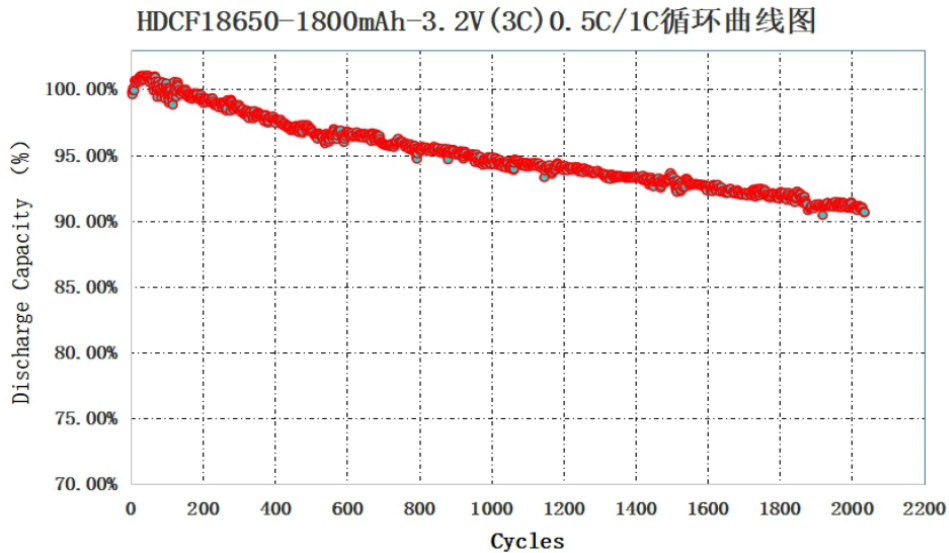


Figure 30: Capacity vs cycles graph for a single cell

Equation of the second battery model:

$$C_{total} = C_{nominal} * (1 - 0.0000375 * N_{cycles})$$

All these parameters can be easily edited to change the model by editing the “Battery\_InitialStatus\_Model\_Parameters.xlsx” file.

Battery status parameters	
Initial battery voltage (at start of scenario, measured or extrapolated from telemetry) [V]	6.54
Initial cycles	100
Battery Voltage vs Capacity model parameters (linear model)	
V0 or Voltage at zero-discharge capacity [V]	6.54
Voltage gradient (slope of graph) [V/Ah]	-0.125
Battery Capacity vs Cycles model parameters	
degradation factor (ABS of slope of graph) [1/cycle]	0.0000375

Figure 31: screenshot of battery status and models parameters Excel sheet

The actual battery cells used for CLIMB are the 2000 mAh from the same company (ENERPOWER Lithium-Iron Phosphate 18650 3.2V 2000mAh), however on the company website only datasheets for cells up to 1800 mAh were available. Therefore, the data for this cell was for this cell as an approximation of the behavior of the actual cells. Moreover, the CLIMB battery pack consists of 2 cells in series and 4 in parallel (6.4V 8000mAh). Again, the single cell performance was assumed to be representative of the whole battery pack, but early reports from battery pack testing suggested a more pronounced non-linear behavior which should be investigated and ultimately implemented into a further iteration of the tool.

However, the actual battery pack voltage and capacity were used for all subsequent calculations.

4.2.4 The voltage used for charging calculations was 7.3 V as per Datasheet (3.65V per cell).

## Methods

### 4.2.4.1 Python code structure

The following chart explains the structure of the code used for the Power Tool which is fully contained in the python script “PowerTool.py” (see installation guide). Python was used for this part of the project mainly due to the ease of treating many dependencies with other file formats and due to a subjective choice from the Author of this specific tool (Giacomo Scomparin). However, the same tasks could be achieved with a Matlab script in a very similar way. This choice was possible also because there was no direct connection between this tool and STK (which requires Matlab) and because it was possible to run the python script from the main Matlab GUI with a specific Matlab function. In hindsight, Matlab could have been a better choice only because it eliminates the need for a Python installation on the final usage machine.

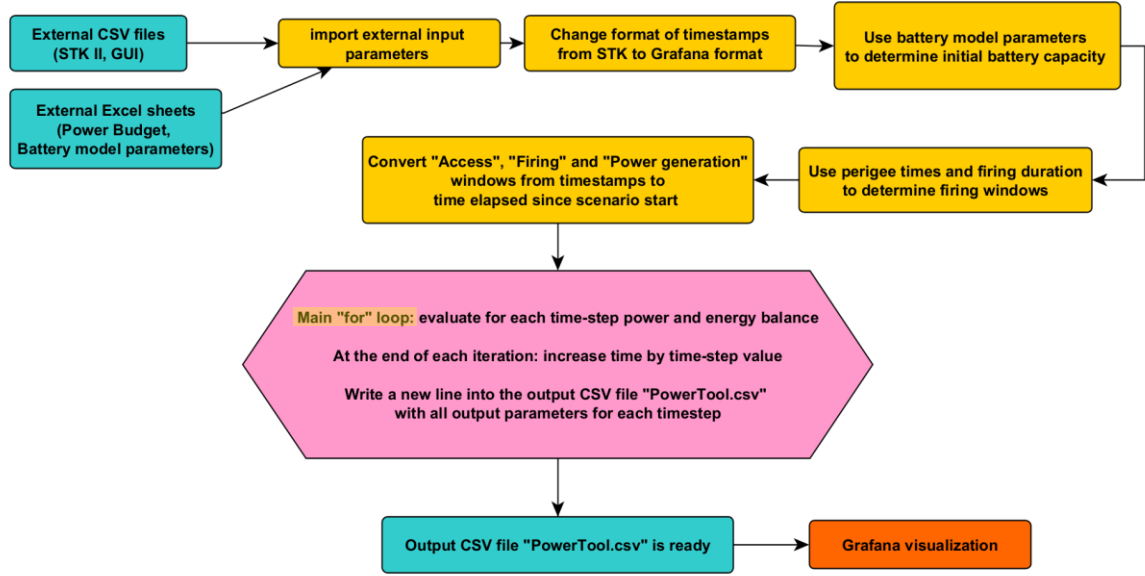


Figure 32: Simplified Power tool code structure and main functions

The “main *for* loop” highlighted in the above chart of the key features necessary to calculate the power and energy balances in such a way that they could be plotted against time in a graph for the given input files. In fact, the given inputs from STK only contained timestamps of the events of interest, therefore the main idea was to perform a “scan” of the time domain between scenario-start and scenario-end evaluating at each time instant which power consumption mode was applicable between propulsion mode, downlink mode and background power consumption (cruise or science mode).

#### 4.2.4.2 Calculations

Within the above-mentioned for loop a few calculations were performed to obtain the following parameters for each time instant:

- **Power output from battery:** if positive, it represents the output power the battery needs to provide at a given time, whereas if negative, it represents battery charging:

$$P_{battery_{out}} = P_{consumed} - P_{generated}$$

Where:

- $P_{consumed}$  : Total power consumption at a given instant depending on current mode or maneuver
- $P_{generated}$  : Total power generated by solar arrays at the given time

- **Battery capacity:**

$$C_{battery} = C_{initial} + C_{charging} - C_{discharge}$$

Where:

- $C_{initial}$ : initial battery capacity at the start of the scenario calculated from initial measured voltage using the above-mentioned battery models (section 4.2.3)
- $C_{charging}$ : charging capacity due to solar arrays.

$$C_{charging} = I_{charging} * \Delta t$$

Where :

- $I_{charging}$ : charging current;  $I_{charging} = P_{solar\ arrays}/V_{charging}$
- $V_{charging}$ : charging voltage;  $V_{charging} = 3.65 * N_{cells-series}$
- $N_{cells-series} = 2$
- $\Delta t$  = time-step value

- $C_{discharge}$ : discharging capacity due to subsystems power consumption.

$$C_{discharge} = C_{discharge-propulsion} + C_{discharge-downlink} + C_{discharge-background}$$

To calculate each of the discharge components, the currents from each rail (3.3V, 5V, 7.4V, 12V) were obtained by dividing the total loads in Watts for that rail and divide by the respective voltage. The currents were then all summed to provide the total current draw for a given mode. The current draw was then multiplied by the time-step to obtain the discharge capacity due to a given mode, similarly to the charging capacity calculations. In the actual code implementation, the formula was written in such a way that currents were not counted twice (e.g. the discharge due to propulsion mode includes the idle power consumption, therefore when the thruster was on, the background discharge was not considered, same for the downlink discharge)

Loads		Cruise mode (W)			
		12V	7V4	3V3	5V
OBC		0.00	0.00	0.33	0.00
PPU		0.00	0.17	0.00	0.00
STACIE-D	Rx	0.00	0.00	0.00	0.20
	Tx	0.00	0.00	0.00	0.00
	Beacon	0.00	0.00	0.00	0.26
STACIE-S	Rx	0.00	0.00	0.00	0.00
	Tx	0.00	0.00	0.00	0.00
	TRx	0.00	0.00	0.00	0.00
ADCS		0.00	0.00	0.87	0.00
Thruster	Cold-standby	0.00	0.00	0.00	0.00
	Hot-standby	4.50	0.00	0.00	0.00
	Active	0.00	0.00	0.00	0.00
Payload (Magnetometer, dosimeter)		0.00	0.00	0.00	1.00
Sum loads (W)		4.50	0.17	1.20	1.46
Efficiency (%)		80.00	90.00	60.00	80.00
Power Consumed (W)		5.63	0.18	2.00	1.83

Figure 33: Screenshot from "CLIMB Power Budget V2" Excel sheet

- **Battery State of Charge (SOC)**

$$SOC = \left( \frac{C_{battery-current}}{C_{battery-total}} \right) * 100$$

Where:

- $C_{battery-current}$ : current battery capacity at a given time
- $C_{battery-total}$ : total battery capacity for a given number of cycles (calculated using the life-time degradation model)

## Results and graphical output

The following are the results for the chosen exemplary scenario using PEGASUS TLE data. The data refers to the 6<sup>th</sup> of August 2023:

- Eccentricity: 0.00043
- Inclination: 97.1425°
- 4.2.5 • Altitude: approx. 400 km

The assumed initial battery parameters were the following:

- initial SOC: 90%
- Initial cycles: 3000

The graphs provided were taken directly from the grafana interface which is what the operator would see. The peaks in the red area show that the battery is subject to the maximum power draw due to the use of the thruster (propulsion mode) during eclipse. Whereas the valleys in the green area represents areas where the battery would be charged due to the excess power produced by the solar arrays. The smaller peaks and valleys represent combinations of these extreme cases for example propulsion mode in the sun or idle power consumption during eclipse.

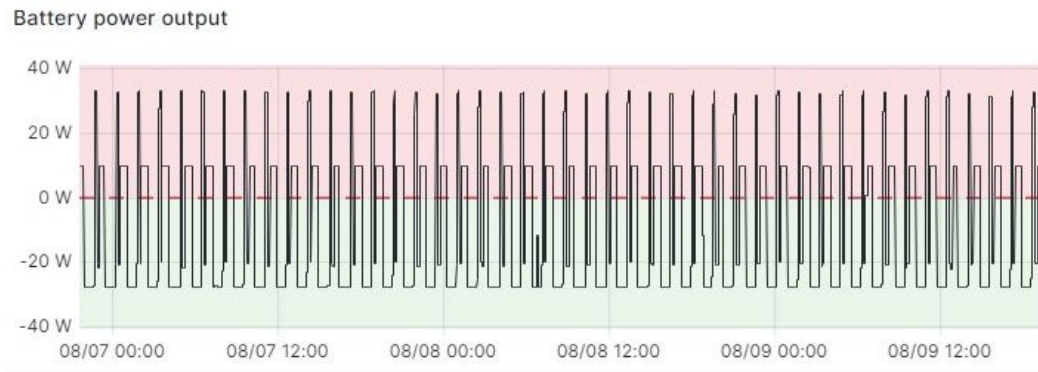


Figure 34: Battery output Grafana visualization

The following is a detailed view of the same scenario but on a smaller timeframe.

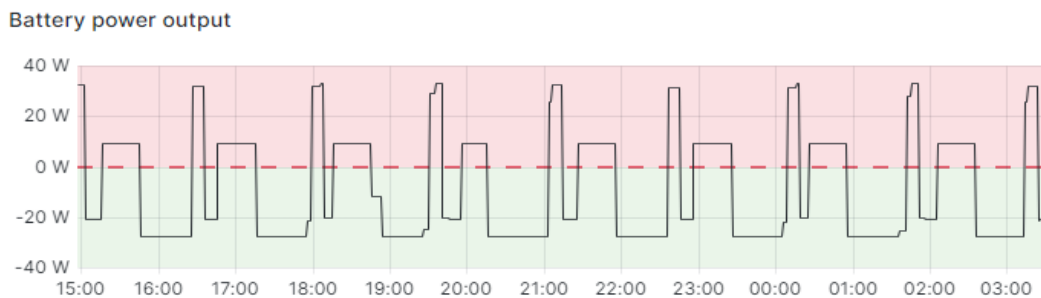


Figure 35: Battery power output Grafana visualization (shorter time-frame)

The following are the results of the battery charge-discharge propagation. These would be probably the graphs of greatest interest for the operator since they easily provide a sense of the feasibility of the planned maneuver from the battery perspective.

The red dashed line represents 33% of nominal battery capacity which corresponds to the steep non-linear drop in voltage shown in the voltage vs capacity graph.

Battery capacity

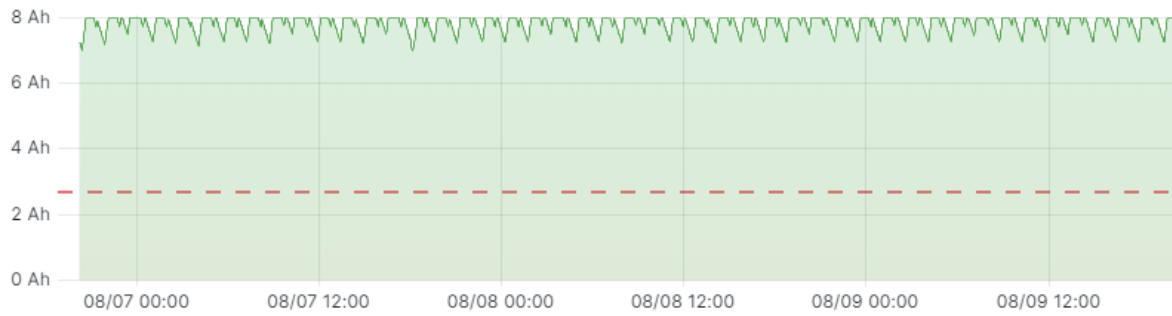


Figure 37: Battery capacity Grafana plot through entire scenario

Battery SOC

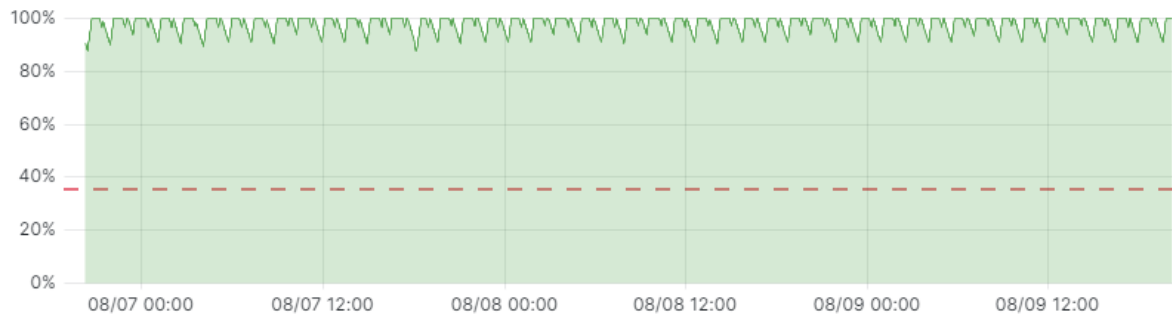


Figure 36: Battery SOC Grafana plot through entire scenario

It can be verified that the system is not particularly sensitive to the initial battery SOC as the generated power is sufficient to recharge the batteries and maintain the SOC above 90% even when starting from a low SOC (e.g. 45%):

Battery SOC

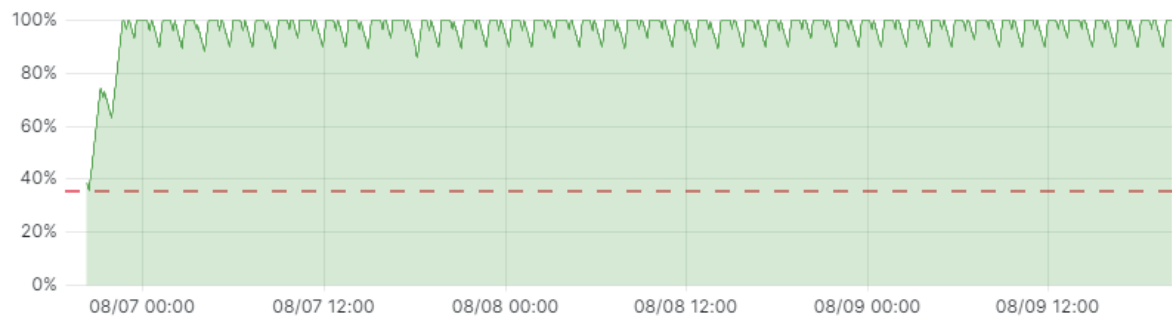


Figure 38: Battery SOC visualization with lower initial SOC

On the other hand the energy balance is very sensitive with respect to idle ADCS power consumption, due to the relatively high power consumption at low voltage resulting in a relatively high current draw (around 1 Ampere) which is present at all times. Therefore more accurate data on ADCS power consumption can make the model significantly more reliable.

The following is an overview of the proposed planning dashboard. It includes all the above-mentioned plots, as well as additional ones to visually check when firings or S-band downlink would occur. It also shows the output of the Torque tool, namely the RPM versus time graph with the saturation levels displayed.



Figure 39: Screenshot of the planning dashboard

## Future Power tool improvements

### 4.2.6

Summarizing what could be improved in the Power tool to make it more reliable and accurate:

- Verification of critical consumption data (e.g. ADCS)
- Improve the battery model by extrapolating data points from actual battery-pack testing
- If possible, validation of overall calculations, code and models by using a test-bench



## 4.3 Propulsion tool

The propulsion tool, along with the torque and power tools, is a tool to assist in mission status assessment and future planning for both short- and long-term operations. The propulsion tool, also called the thruster tool interchangeably, has a higher focus on assessment than future planning compared to the other tools. This tool aims to receive, handle, and deliver the data from the thruster to the other tools and present it graphically. This dedication to a single subsystem, unlike the others, is due to its importance and critical role in the CLIMB mission. For example, the orbital maneuvers are performed by the thruster, and while on, it is the highest power consumer in the whole system. This thruster data can be fed back to the power tool to calculate actual power consumption, which, again, is mission assessment and future planning. Therefore, a seamlessly automated data process and handling are required for the tool and for the operator. Lastly, a graphical tool is used in conjunction to provided the operator and planner to better understand the conditions of the thruster.

The CLIMB on-board thruster is a NANO by Enpulsion. It is a FEEP thruster controlled by the electrode voltages. [4] The firmware within the thruster provides the status and conditions of the thruster through parameters. Figure 39 shows a document received from Enpulsion listing parameters and their information.


 ENPULSION		<b>Interface Control Document</b> <b>ENPULSION NANO R<sup>3</sup> / IR<sup>3</sup></b>		Doc. No.:	<b>ENP2020-005</b>
				Iss./Rev.:	<b>G.1</b>
				Page 17 of 23	
0x0A	Fuse Status (LSB)	read / write	0 to 0xFFFFFFFF*	0	Shows which fuses have been triggered. Writing 0 (all 4 bytes) resets the triggered fuses.
0x0B	Fuse Status				
0x0C	Fuse Status				
0x0D	Fuse Status (MSB)				
0x0E	Operational Mode	read / write	0 to 2	0	0: Disabled, all power sections are disabled 1: Enabled, reservoir is heated up automatically 2: Enabled, reservoir is heated up automatically returns to operational mode (1) after a cycle
0x0F	Status	read	0 to 3	0	0: Idle, all power sections are disabled 1: Heating, reservoir temperature is still rising 2: Standby, reservoir temperature is ok, thruster is not operating 3: Operation, thruster is operating

Figure 39: Parameter Document from Enpulsion

The raw data from the thruster is delivered to the ground station for decoding and to the mission planning team. Python code processes the received data, and the visualization tool Grafana displays the results.

## Creating Imaginary Received Data

4.3.1 In order to start with the propulsion tool, a set of data is needed for establishing the framework. However, an example of real data is not available until after the operation takes place. Therefore, contact was made with Enpulsion, the thruster manufacturer. Mr. Federico Toller provided a document called “ENP2020-005 G005.G ENPULSION NANO R3\_IR3 - ICD (extract).pdf” This is the document shown in Figure 39 previously. The parameter names, value ranges, default values, and explanations were extracted from the document. The parameters were chosen based on advice from Mr. Toller. Then, these parameters were declared as variables in Python. Figure 40 presents a portion of these parameters.

```
79 #Thrust (LSB), read only
80 #: 0~1 [mN]
81 Thrust= emptyZeroes
82 Thrust[1800:2400]= 1- abs(var2[0:600])
83 df.insert(5,"Thrust", Thrust)
84
85
86 #Specific Impulse (LSB), read only
87 #: 1000~7000 [s]
88 Isp= emptyZeroes
89 Isp[1800:2400]= 7000*(1- abs(var2[0:600]) )
90 df.insert(6,"Isp", Isp)
91 #df.loc[1800:2400,"Isp"]= 1- abs(ref["Uniform"][0:599])
92
93 #Bus Voltage (LSB), read only
94 #: nominal vss= 1 [mV]
95 Bus_V= emptyOnes* (12+ var1)
96 df.insert(7,"Bus_V", Bus_V)
97
98 #Bus Current (LSB), read only
99 #: 1~5 [A]
100 Bus_I= emptyOnes* (4.8 + var2) #var2 so uniform random
101 df.insert(8,"Bus_I", Bus_I)
102
```

Figure 40: Examples of Declared Parameters

A total of 3600 seconds were used, which is one hour worth of data. This data had 3600 timesteps and was used as a dataframe with the Pandas library in Python. This is shown in Figure 41. During the 1 hour of operation, the thruster operated for 10 minutes, starting from the 30 minute mark to the 40 minute mark. This is represented between the indexes 1800 and 2400.

```

18 # Defining Received Values/Data #####
19 #in lieu of actual data, a 0-50% variation to the nominal values are randomly
20 #created. this section of the code must be deleted if actual data file is
21 #received from the thruster via the ground station team.
22
23 # Get the current date and time
24 DT= pd.date_range(start="2024/01/11",periods=3600,freq="S")
25 #period in DT is seconds, thus 1 hour of data is generated as
26 #the thruster operated 10 minutes between 30 & 40 minute mark.
27
28 # Format the datetime object as a string
29 fmDT = DT.strftime("%Y-%m-%d %H:%M:%S")
30
31
32 #converting variable type to pandas DataFrame
33 df= pd.DataFrame(fmDT,columns=["Date & Time"])

```

Figure 41: Pandas Dataframe and Date\_Range

Additionally, two different noises were introduced to imitate real-world operation and fluctuation. First, a randomly generated normal distribution with a mean of zero and a standard deviation of 0.15 was created. (The normal distribution is also called the Gaussian distribution.) The ‘bell curve’ is centered at zero and spread in such a way that 68.2% of random values fall within the (-0.15, 0.15) range. [ numpy.org] Secondly, a uniform random distribution was used to create a second noise. In figure 42, two variables called ‘var1’ and ‘var2’ represent the noises written in lines 41 and 43. Var1 is used to create a larger noise, while var2 is used to create a smaller noise. And these were added to the parameters.

```

40 #generating {gaussian} distribution centered at 0 with std dvi of 0.1
41 var1= np.random.normal(0, 0.15, dfsize) #large variation
42 #generating {uniform} random values between -0.05 and 0.05, i.e 5% noise
43 var2= np.random.uniform(-0.05, 0.05, dfsize) #small variation

```

Figure 42: Noise Variables ‘var1’ and ‘var2’

There were some parameters, such as "status," which show the conditions, like on or off. These parameters were kept as ‘off’ conditions as the baseline without noise.

Of course, these parameters were switched to the value equivalent of ‘on’ when the thruster was turned on. Figure 43 shows this setting.

```

52 #Defining Nominal Values based on Received Spec. Sheet #####
53 #Reset Cycle Counter (LSB), read only
54 #: after each power/reset cycle the non-volatile counter is incremented.
55 #: LSB stands for Least Significant Byte
56 RCC= emptyZeroes
57 df.insert(1,"RCC",RCC)
58 df.loc[ref["Uniform"]> 0.04, "RCC"]= 1
59
60 #Fuse Status (LSB), read&write
61 #: 0 to 0x1F, shows which fuses have been triggered. Writing 0 to all 4 bytes
62 #resets. default value (df) is zero
63 FS= emptyZeroes
64 df.insert(2,"FS",FS)
65
66 #Operational Mode, read&write
67 #: 0=disabled (df), 1=enabled reservoir heats up,
68 #2=enabled reservoir heats up, returns to operational mode 1.
69 OpMode= emptyZeroes
70 df.insert(3,"Op Mode",OpMode)
71 df.loc[1800:2400, "Op Mode"]= 1 #could replace with "OpMode[1800:2400]= 1"
72
73 #Status, read only
74 #: 0=idle (df), 1=heating reservoir, 2=standby, 3=operation(thrust on)
75 Status= emptyZeroes
76 df.insert(4,"Status", Status)
77 df.loc[1800:2400, "Status"]= 3
78

```

Figure 43: Non-Range Parameters

This imaginary data from the thruster was used as the starting point of the propulsion tool. However, it will be obsolete in the future as real and actual data is expected.

## Data Handling

The handling of data is the most significant task for the propulsion tool. However, it is also the most uncertain. This is because the format of the received data is unknown. Although the imaginary data is made in Part 1, the tool must be able to read and handle the data universally. Therefore, in this part, the tool is built based on the idea that it does not know the shape and indexes of the data.

### 4.3.2

To begin, the tool must import and read the data from the thruster via the ground station. This data is called 'received data.' The task is done in lines 202–223. The received data file, in CSV format, is given an arbitrary name "exemplary\_CLIMB\_random\_telemetry.csv" and is stored in the folder called "Received Data." This data is imported and stored as 'rawData.' (line 205) The name of the received file and the location relative to the propulsion tool (python code) must be changed in the future with the real data file name and location.

```
202 # Data/File Importing #####
203
204 #importing/reading the received data from the Ground Station
205 rawData= pd.read_csv("Received Data/exemplary_CLIMB_random_telemetry.csv")
206 #the file name and its path must be changed for the real data
207
```

Figure 44: Importing the Thruster Data as 'rawData'

Then, the column headers and shape of the received data are printed for the user. The shape is printed as (number of rows, number of columns).

```
204 # Data/File Importing #####
205
206 #importing/reading the received data from the Ground Station
207 rawData= pd.read_csv("Received Data/exemplary_CLIMB_random_telemetry.csv")
208 #the file name and its path must be changed for the real data
209
210 #printing first row or the column label/name
211 print(rawData.columns)
212 print("\n")
213
214 #printing the size of the received data
215 print (rawData.shape)
216
217 """
218 #activate the following lines (except the comment) if data file is a tab seperated file.
219 #adding the delimiter in order to recognize the tab seperation
220 rawData= pd.read_csv("Received Data/exemplary_CLIMB_random_telemetry.csv",\
221                      delimiter="\t")
222 print(rawData.head)
223 print("\n")
224 """
225
```

Figure 45: Column Headers and Table Shape

However, it is possible that it could be a tab-separated file instead of a comma-separated file (CSV). Therefore, as a backup, the code from lines 215 to 222 was written to read a tab-separated file and store it as 'rawData.' (Figure ) This portion of code is commented out for now.

```

228 # Data Selecting & Recording #####
229
230 #creating a empty list; it's a place holder
231 ""
232 ph= list()
233
234 #assigning(copy) wanted columns to the place holder list
235 #column number is recorded in a list called "cn."
236 #it contains the column number or INDEXs of wanted data
237 #cn= [1,31,44,28,29,18,37,39,43,39,8,11,15,16,25,16,24,5,6]
238 cn= [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24]
239 for i in cn:
240     ph.append( rawData.iloc[:,i] )
241 ""
242

```

Figure 46: Importing a Tab Separated File (Backup)

The next section is data selection and recording. During the early stages of development, the “ENP2020-005 G005.G ENPULSION NANO R3\_IR3 - ICD (extract).pdf” document was unavailable. Therefore, an assumption was made that the received data contained irrelevant information about or from the thruster. As a result, this block of code was written to create an empty list and fill the empty list with the wanted data using the column index number. The line 228 creates an empty list, and 'cn' is the column number the user has to enter in order to save in the 'rawData' dataframe. However, this section was commented out once the imaginary data was created. Should this function be needed with the actual data, this section of code can be uncommented and used.

The thruster provides many important data, but it does not include all the necessary information. For instance, it does not provide the power in watts on the bus lane or fuel parameters. In this section, the tool is designed to calculate the missing data.

```

245 # Data Calculation #####
246
247 # Bus Power calculation
248 Bus_P= rawData.iloc[:,8] * rawData.iloc[:,9]
249 rawData.loc[:, "Bus_P"] = Bus_P
250 # Fuel Flow calculation
251 m_dot= ( rawData.iloc[:,6]/ (rawData.iloc[:,7]* 0.908665) ) * 10**3 #[mg/s]
252 rawData.loc[:, "m_dot"] = m_dot
253

```

Figure 47: Data Calculation

The power calculation is straightforward; it is the voltage multiplied by current.

$$P = V \times I$$

Line 234 performs a power calculation for the bus. Here, numbers 8 and 9 are the numbers of the voltage and current columns. (One must remember that the numbers start at zero in the code world instead of one.) These numbers must be updated if the received data has the voltage and current in different columns.

Next, index searching and recording are performed to find when the thruster was on. The lines 250 and 251 perform this task in Figure above. An empty list is created, and the indexes are recorded (added). The condition for this action is when the thrust status was one, which is greater than zero.

```

254 #Index searching & recording when the thrust was on
255 idx=[]
256 idx= df[df['Thrust']>0].index.tolist()
257

```

Figure 48: Index Searching and Recording

Continuously, empty variables were pre-declared to store the average and the last values for the thruster operation duration. Then, a for loop iterates the 'idx' list and adds all the values. Once again, this process only adds values when the thruster was on.

```

258 #Pre declanation to store the Average and Last Values
259 #These will be displayed on Grafana
260 md=0 # short for mass flow= m dot
261 thr=0 # short for thrust
262 isp=0
263 emtv=0 # emitter voltage
264 emti=0 # emitter current
265 emtp=0 # emitter power
266 extv=0 # extractor voltage
267 exti=0 # extractor current
268 extp=0 # extractor power
269 neuv=0 # neutralizer voltage
270 neui=0 # neutralizer current
271 neup=0 # Neutralizer power
272 rsvp=0 # Reservoir heat power
273
274 #look to sum all the values when the thrust was on
275 for i in idx:
276     thr= thr+ rawData.iloc[i,6]
277     isp= isp+ rawData.iloc[i,7]
278
279     emtv= emtv+ rawData.iloc[i,10]
280     emti= emti+ rawData.iloc[i,11]
281     emtp= emtp+ rawData.iloc[i,12]
282
283     extv= extv+ rawData.iloc[i,13]
284     exti= exti+ rawData.iloc[i,14]
285     extp= extp+ rawData.iloc[i,15]
286
287     neuv= neuv+ rawData.iloc[i,19]
288     neui= neui+ rawData.iloc[i,20]
289     neup= neup+ rawData.iloc[i,21]
290
291     rsvp= rsvp+ rawData.iloc[i,16]
292
293     md= md+ m_dot[i]
294

```

Figure 49: For-loop Value Addition

Lastly, the average values are calculated simply by dividing the total number of values. The total number of values found by adding the total number of indexes, shown in line 292; this value is stored as the 'ilen' variable. The last value is the value before the thruster switched off (line 294).

For example, the calculated average thrust value is stored in the 0th row under a new column called 'thr' and the last thrust value in the very last row by using minus indexing. The lines 299 and 300

show this process. This process is repeated for the other parameters. As before, the column number must be updated based on the received data. For example, line 300 reads the data from column number 6. This column corresponds to the thrust [mN].

The final section of data calculation is to export the data. Therefore, the 'rawDate' dataframe is exported as "Test\_Prpln\_Tool.csv", as shown in Figure .

```
340 # Data Exporting #####
341 # Saving the propulsion dataframe to a CSV file
342 rawData.to_csv("CLIMB_Grafana_Prpln_Tool.csv", index=False)
343 rawData.to_csv("Test_Prpln_Tool.csv", index=False)
344
345
```

Figure 50: Exportation

## Grafana Visualization

4.3.3 Grafana is visualization software using a web browser. It is open, free software implemented by many companies not only in the space sector but also in various areas that require monitoring. For example, FHWN's previous cubesat PEGASUS's data are presented using Grafana in the SatNOGS database.

Once installed (explained in another part of this report), Grafana has a straightforward interface to create simple graphs such as line and gauge graphs. However, the documentation falls behind for advanced graphing. Also, as Mr. Emmeric Vitztum advised, Grafana was purely used for visualization only. Figure shows the propulsion tool dashboard in Grafana. This is based on the local file from the propulsion tool, and Grafana is also locally assembled. At the top, various statuses are displayed. Followed by line plots, average values, and last values. Rows are created for the bus, emitter, extractor, neutralizer, etc.

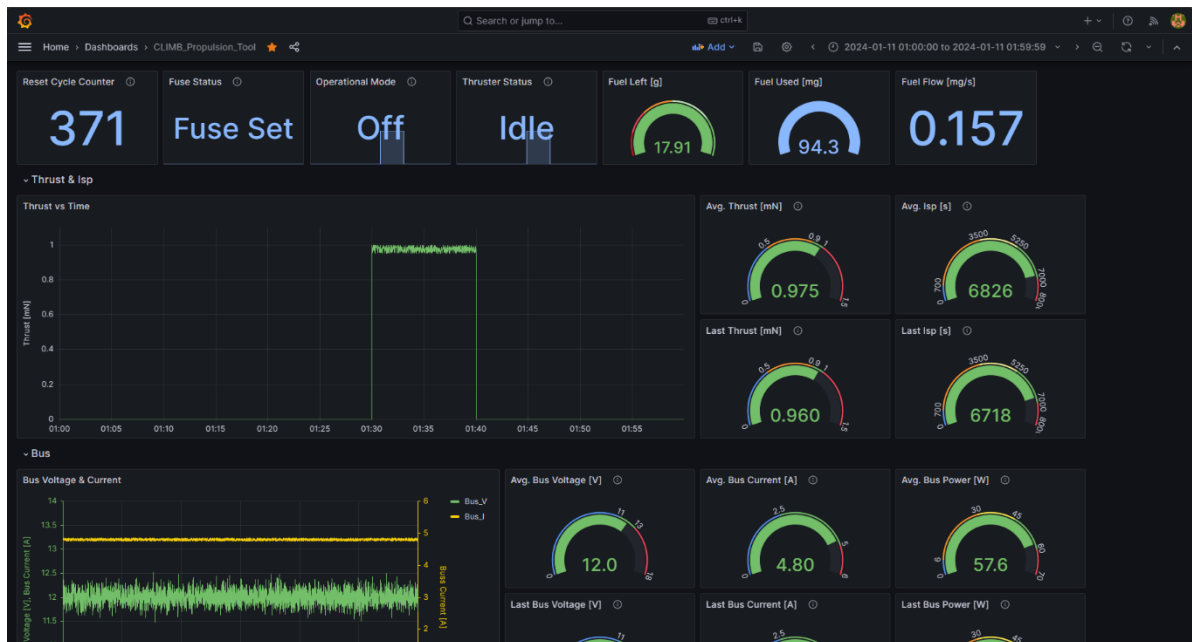


Figure 51: Propulsion Tool Grafana Main Page



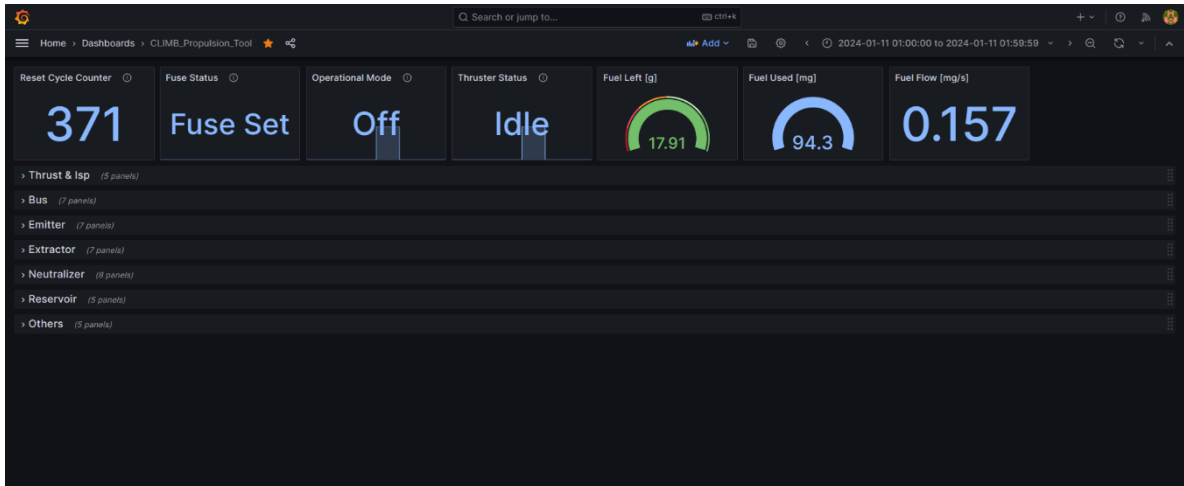


Figure 52: Main Page Rows Closed

Looking at Figure below, the desired plot type is selected on the top left. Then, the wanted data column heading is selected; here, “thr” is selected. Since the data is numerical, the type is selected to be a number. Within the data, Grafana can do a simple selection or calculation; therefore, the first value in the column “thr” is selected. Once done, a gauge graph and a numerical value show up in the center. And the value marking and coloring can be set up (not shown).

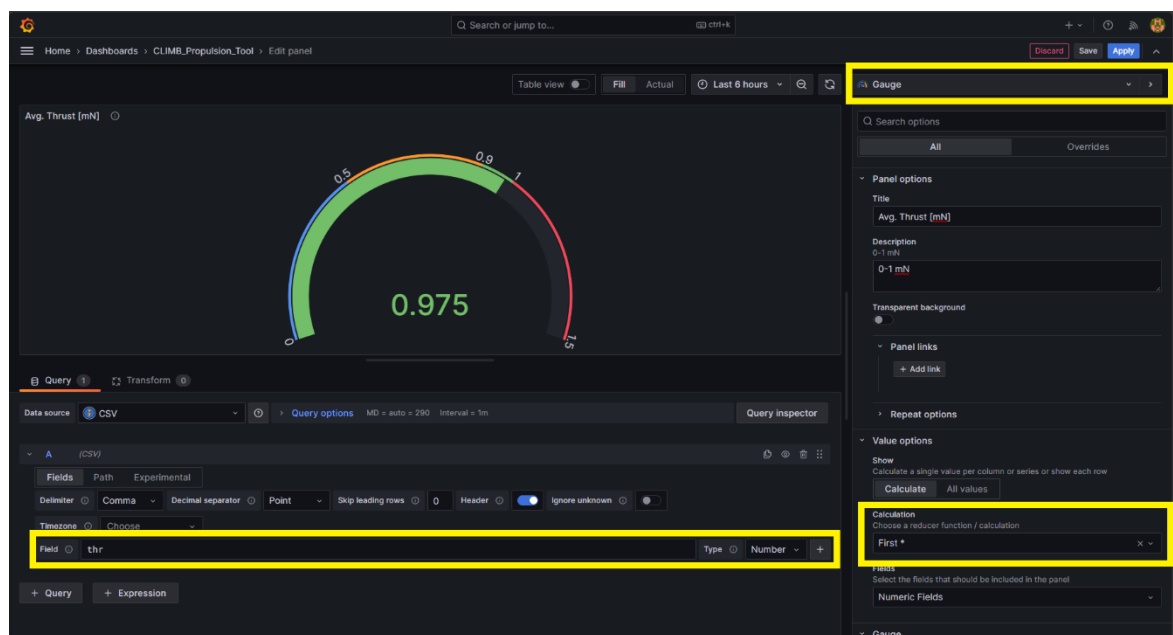


Figure 53: Creating a gauge plot

Moving up in a hierarchy, a general dashboard directory is built for the future. Figure 54 shows this dashboard. For example, the propulsion row presents a few important graphs and pinned notes.

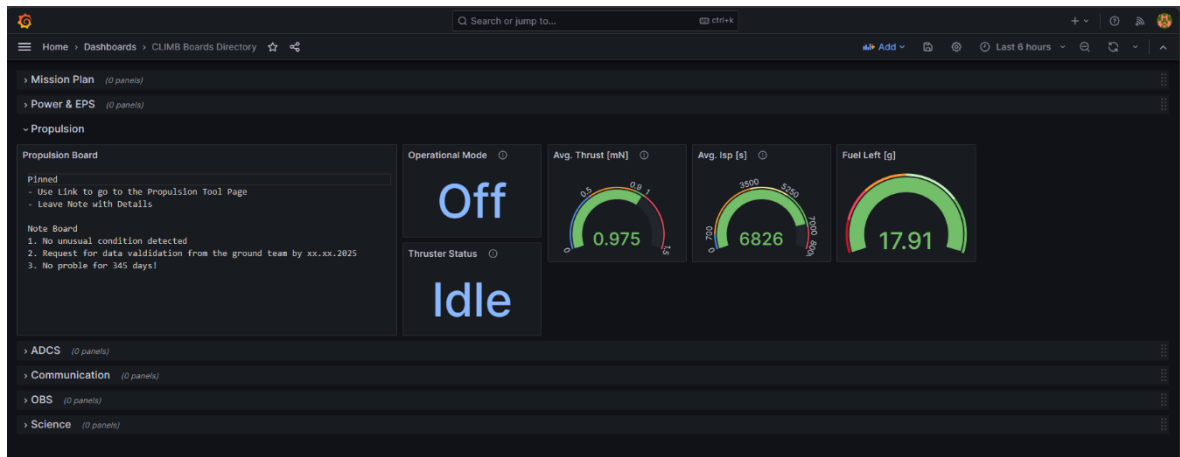


Figure 54: Propulsion Tool Grafana Main Page

## 5. Graphical display - Grafana

The already mentioned Grafana software is a free partially web-based application which offers great potential for telemetry monitoring. In fact grafana can plot large amounts of data on several dashboards with many possibilities for customization and alerting.

Grafana was already implemented in the PEGASUS telemetry visualization at the FHWN ground station, however for PEGASUS, the Grafana platform was integrated within the SatNogs web-application, with the main downsides of being public, and also not the most up to date version of the Grafana software, therefore lacking some functionalities and customization options.

For the CLIMB mission it will be also necessary to share some basic information as prescribed by the Austrian and International law. However, it is also planned to keep most of the telemetry data confidential also because of the use of the ENPULSION thruster.

To visualize data, Grafana uses the concept of “connections” which stands for data sources. Many possible data sources are provided, most of which are related to server-based databases but also some local ones like locally stored CSV files via a specific user-developed plugin.

We chose to use locally stored CSV files mainly because of ease of use, especially during the development of the different tools, compared to the setup of a server-based database and the upload of data to such database.

However, from discussions with the ground station (Alexander Spaniol) it was decided that the best option for telemetry data within the real operation of the satellite, in terms of backup capabilities would be to use a server-based infrastructure and move away from locally stored CSV files.

The future concept for telemetry processing would have a decoding script to translate the demodulated data into readable data, then this script or a dedicated one would need to upload the decoded data into a private server-based database dedicated for the CLIMB operation telemetry, Grafana would then be connected to this database to visualize the data in appropriate dashboards.

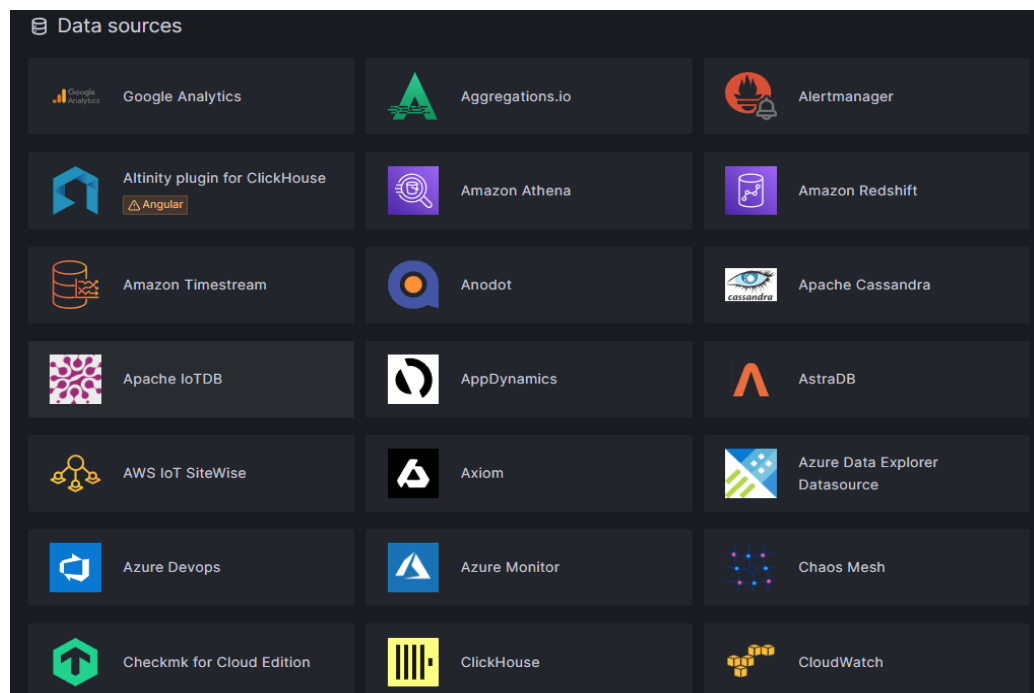


Figure 55: Some of Grafana's data sources possibilities

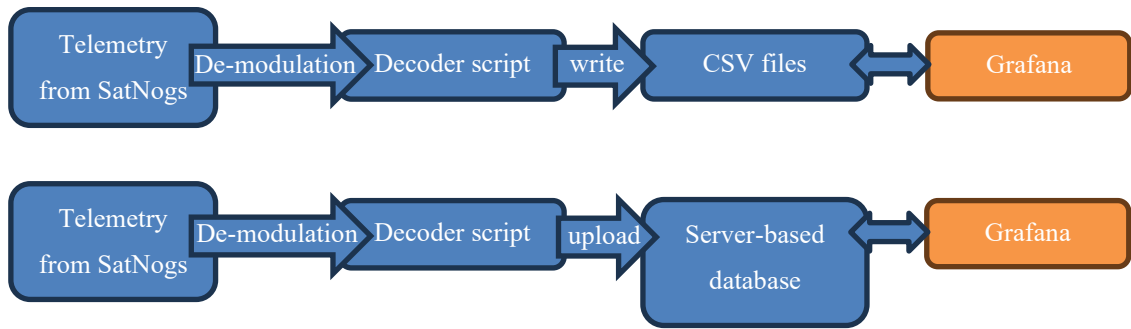


Figure 56: Two different approaches for telemetry visualization using Grafana

## 6. Installation guide

### 6.1 Local PC planning tools installation steps

Required software on PC : Matlab app designer, STK version 12 , Python

On top of the main required software additional Python modules shall be installed via command line (pip-install): “pandas”, “numpy”, “openpyxl”

A folder dedicated to all the necessary files shall be created with this exact path and name due to references within the different codes: “C:\CLIMB Operation”

Within this folder the following files shall be present before running the Predictor\_app. .mlapp tool:

- Battery\_InitialStatus\_Model\_Parameters.xlsx
- CLIMB Power Budget V2.xlsx
- PowerTool.py
- power\_tool\_modeSelection.mlapp
- CLIMB\_scenario1 (folder)
- Torque\_tool\_data
- TLE.txt (currently TLE\_PEGASUS\_07082023.txt)
- 

Name	Änderungsdatum	Typ	Größe
PowerTool.csv	29.01.2024 17:26	Microsoft Excel-C...	332 KB
RPM_grafana_format.csv	29.01.2024 17:26	Microsoft Excel-C...	190 KB
RPM_overTime.csv	29.01.2024 17:26	Microsoft Excel-C...	305 KB
YPR.csv	29.01.2024 17:25	Microsoft Excel-C...	512 KB
AngularMomentum.csv	29.01.2024 17:25	Microsoft Excel-C...	66 KB
CLIMB_Solar_Panel_Power.csv	29.01.2024 17:25	Microsoft Excel-C...	20 KB
Torque.csv	29.01.2024 17:25	Microsoft Excel-C...	68 KB
Access_schedule.csv	29.01.2024 17:25	Microsoft Excel-C...	1 KB
Fire_schedule.csv	29.01.2024 17:25	Microsoft Excel-C...	2 KB
Scenario_time.csv	29.01.2024 17:25	Microsoft Excel-C...	1 KB
ModeSelection.csv	29.01.2024 17:25	Microsoft Excel-C...	1 KB
Predictor_app.mlapp	29.01.2024 17:24	MATLAB App	148 KB
PowerTool.py	29.01.2024 16:26	Python File	14 KB
Torque_tool_data.xlsx	29.01.2024 14:48	Microsoft Excel-Ar...	12 KB
CLIMB Power Budget V2.xlsx	21.01.2024 18:55	Microsoft Excel-Ar...	137 KB
Battery_InitialStatus_Model_Parameters.x...	21.01.2024 18:55	Microsoft Excel-Ar...	16 KB
power_tool_modeSelection.mlapp	21.01.2024 17:19	MATLAB App	90 KB
TLE_PEGASUS_07082023.txt	06.12.2023 20:14	Textdokument	1 KB
CLIMB_scenario1	29.01.2024 14:40	Dateiordner	

\*NOTE: The picture shows all the files related to the tools, including outputs, for the first run only the above-mentioned files are required which are also contained in the orange rectangle

Figure 57: Two different approaches for telemetry visualization using Grafana

## 6.2 Local Grafana installation and setup

Grafana is a web-based application which runs on all main browsers, however, it requires also a local installation and local service running on the PC where the grafana dashboards shall be accessed.

**Passwords can be easy, must be tracked! Extremely hard to recover unlike other web services, therefore don't lose otherwise all the dashboards will be inaccessible** (it's possible to back them up locally by exporting them to a file)

Main installation steps:

1. Download and install the latest **local** grafana installation from the official "Grafanalabs" website: <https://grafana.com/grafana/download?platform=windows>  
For more details, please visit the following link:  
<https://devconnected.com/how-to-install-grafana-on-windows-8-10/>
2. Once installed, login at <http://localhost:3000>  
Default user and password are "admin", password can be changed on the first login.  
**It's very important to keep this password tracked and safe, this is because Grafana does not provide a reliable password recovery system unlike typical web-services. The recommendation is to keep it simple and safe (e.g. printed or written as a sticker in the ground station)!**
3. Before importing or creating dashboards, the CSV plugin must be installed and local CSV files must be allowed by adding to "Defaults.ini" or preferably "Custom.ini" (see details provided in the Grafana installation at the above link) text lines to the code:

```
[plugin.marcusolsson-csv-datasource]
allow_local_mode = true
```

4. **Close all windows and restart the Grafana service (go to task manager-services)**
5. The input of local path in Grafana (when creating a new CSV data source) should look like this (c must be lower case and forward slash must be used): `c:/temp/grafana_test.csv`

## 7. Conclusion

This project demonstrated that it's possible to integrate STK with in-house developed tools to have an overview of critical resources across the scenario duration (e.g. 72h time-span). The objectives of these STP could have been realistically achieved also using proprietary software such as the STK extension "STK Scheduler", however that solution may be too expensive and maybe not flexible enough to model some of the satellite resources. In fact, especially with respect to the torque tool it was necessary to make external calculations to account for parasitic torque phenomena. Also in the Power tool, developing an in-house tool in Python allowed to have full control over the calculation methods, models and input data used.

The following are some possible general improvements to the tools:

- Include de-saturation mode across all tools
- Include firing time as an input parameter in the GUI (at the moment fixed across all tools at 10 minutes)
- Implement accuracy and reliability improvements mentioned in the Power tool section 4.2.6
- Improve mode-selection GUI (currently only working for science and cruise mode)
- More development in Grafana (e.g. adding alerts for monitoring, Server-based data sources)

## 8. References

- [1] ANSYS, "STK Help," [Online]. Available:  
<https://help.agi.com/stk/index.htm#stk/referenceframesvgt.htm?Highlight=reference%20axis>.
- [2] ANSYS, "Detailed Ansys STK Capabilities," [Online]. Available:  
<https://www.ansys.com/content/dam/amp/2023/november/asset-creation/stk-capabilities.pdf>.
- [3] ENERPOWER-HAIDI, *HDCE18650-1800mAh-3.2V Datasheet*.
- [4] numpy.org., "NumPy v1.26 Manual," [Online]. Available:  
<https://numpy.org/doc/stable/reference/random/generated/numpy.random.normal.html#numpy.random.normal> . [Accessed 28 January 2024].



## 9. Appendix

### 9.1 Appendix 1:

```
clc
close all
clear all
%% Constants
ADCS_max_L=1.77; % mNms
ADCS_max_T=2; % mNm
RPM_wheel_initial=0;
L_x_cubesat_initial=0;
L_y_cubesat_initial=0;
L_z_cubesat_initial=0;
I_fw=2.4E-05; %kg*m^2 2.4E-05 for CW0017 and 7E-05 for CW0057 and CS has 0.1432 kg*m^2
RPM_limit=6000;% 8000 for CW0017 and 10000 for CW0057for reaction
F=0.00035;% N
t_thrust=600;% s
cm_x=0.0027;
cm_y=0.0047;
T_cm_x=cm_x*F;
T_cm_y=cm_y*F;
L_cm_x=T_cm_x*t_thrust;
L_cm_y=T_cm_y*t_thrust;
%% Input reading
Torque_data= readtable("CLIMB_Torque1.csv", 'HeaderLines', 1);
Torque_time=Torque_data(:,1);
Torque_time = table2cell(Torque_time);
Torque_time= datetime(Torque_time, 'InputFormat', 'd MMM yyyy HH:mm:ss.SSS');
T_x=1000*table2array(Torque_data(:,2)); % mNm
T_y=1000*table2array(Torque_data(:,3)); % mNm
T_z=1000*table2array(Torque_data(:,4)); % mNm
T=sqrt(T_x.^2+T_y.^2+T_z.^2);
Max_Torque=max(T);
disp(['Maximum Instantenious Torque (mNm): ', num2str(Max_Torque)]);
cum_T_x=sum(T_x);
cum_T_y=sum(T_y);
cum_T_z=sum(T_z);
Total_torque=sqrt(cum_T_x^2+cum_T_y^2+cum_T_z^2);
disp(['Total Torque (mNm): ', num2str(Max_Torque)]);

L_data= readtable("CLIMB_Momentum1.csv", 'HeaderLines', 1);
L_time=L_data(:,1);
L_time = table2cell(L_time);
L_time= datetime(L_time, 'InputFormat', 'd MMM yyyy HH:mm:ss.SSS');
L_x=1000*table2array(L_data(:,2)); % 1000*kg*m^2/sec= mNms % + L_cm_x*
L_y=1000*table2array(L_data(:,3)); % 1000*kg*m^2/sec= mNms
L_z=1000*table2array(L_data(:,4)); % 1000*kg*m^2/sec= mNms
L=sqrt(L_x.^2+L_y.^2+L_z.^2);
Max_L=max(L);
disp(['Maximum instantaneous Angular Momentum (mNms) ', num2str(Max_L)]);

delta_L_x=zeros(size(L_x, 1), 1);
for i = 2:length(delta_L_x)
    delta_L_x_i = L_x(i);
    delta_L_x_iminus1 = L_x(i-1);
    delta_L_x(i) = L_x(i)-L_x(i-1);
end
L_x(1)=L_x_cubesat_initial+L_x(1);
Cum_L_x=L_x(1)+cumsum(delta_L_x);

delta_L_y=zeros(size(L_y, 1), 1);
for i = 2:length(delta_L_y)
    delta_L_y_i = L_y(i);
    delta_L_y_iminus1 = L_y(i-1);
    delta_L_y(i) = L_y(i)-L_y(i-1);
end
L_y(1)=L_y_cubesat_initial+L_y(1);
Cum_L_y=L_y(1)+cumsum(delta_L_y);

delta_L_z=zeros(size(L_z, 1), 1);
```

```

for i = 2:length(delta_L_z)
    delta_L_z_i = L_z(i);
    delta_L_z_iminus1 = L_z(i-1);
    delta_L_z(i) = L_z(i)-L_z(i-1);
end
L_z(1)=L_z_cubesat_initial+L_z(1);
Cum_L_z=L(1)+cumsum(delta_L_z);

L_limit_pos=ADCS_max_L*ones(size(L_time));
L_limit_neg=-ADCS_max_L*ones(size(L_time));

time_L_limit_x =max( L_time(abs(Cum_L_x) <= L_limit_pos));
time_L_limit_y =max( L_time(abs(Cum_L_y) <= L_limit_pos));
time_L_limit_z =max( L_time(abs(Cum_L_z) <= L_limit_pos));

% time_L_limit = min(time_L_limit_x, time_L_limit_y, time_L_limit_z, 'omitnan');
% time_unil_desaturation=days(duration((time_L_limit-min(L_time))));
% disp(['saturation occurs on: ', datestr(time_L_limit)]);
% disp(['max days without desaturation: ', num2str(time_unil_desaturation)]);

delta_RPM_wheel=-delta_L_x/I_fw*(30/pi);
RPM_wheel=cumsum(delta_RPM_wheel);
RPM_wheel(1,1)=RPM_wheel_initial+RPM_wheel(1,1);

time_RPM_limit=max( L_time(abs(RPM_wheel) <= RPM_limit));
time_unil_desaturation_maxRPM=days(duration((time_RPM_limit-min(L_time))));
disp(['saturation occurs on (due to RPM): ', datestr(time_RPM_limit)]);
disp(['max days without desaturation (due to RPM): ', num2str(time_unil_desaturation_maxRPM)]);
%% Plots
figure Name AbsTorque
plot(Torque_time,T, 'LineWidth',2)
xlabel('Time')
ylabel('Absolute Torque (mNm)')

figure Name AngularMomentum
plot(L_time,L, 'LineWidth',2)
xlabel('Time')
ylabel('Absolute Angular Momentum (mNms)')

figure Name CumAngularMomentum
hold on
plot(L_time,Cum_L_x,'.', 'LineWidth',2)
plot(L_time,Cum_L_y,'.', 'LineWidth',2)
plot(L_time,Cum_L_z,'.', 'LineWidth',2)
plot(L_time,L_limit_pos, '--', 'LineWidth',2, 'Color', 'red')
plot(L_time,L_limit_neg, '--', 'LineWidth',2, 'Color', 'red')

xlabel('Time')
ylabel('Cumulative angular momentum (mNms)')
legend({'Cumulative angular momentum X axis','Cumulative angular momentum Y axis','Cumulative angular momentum Z axis', ...
    'Upper Limit' , 'Lower Limit'}, 'Location', 'southeast')

hold off

figure Name FlyWheelRPM
plot(L_time,RPM_wheel, 'LineWidth',2)
xlabel('Time')
ylabel('Reaction wheel RPM')

```