SATELLITE AOCS – HOMEWORK #2

M2TSI - M11 Orbital Mechanics II

Benjamin AKERLUND benker-3@student.ltu.se

M2 - Space Technology and Instrumentation (TSI)
Faculty of Science and Engineering (*Faculté sciences et ingénierie*)
Université Toulouse III - Paul Sabatier (UPS) January 5, 2025

Preface

This report is a submission for the *M2TSI - M11 Orbital Mechanics II* course at UPS during the autumn semester of 2024. The submission is for the *SATELLITE AOCS – HOMEWORK #2* assignment as part of the graded assignments for this course.

The documentation for this file is uploaded per e-mail to the course lecturer, Cloé Alcaria, and the further source files and code can be found uploaded to my GitHub at: https://github.com/benjaminakerlund/M11-Orbital-Mechanics-Assignment-2.

Contents

1 Local orbital frame

- **Give the definition of a geocentric inertial reference frame**

  Geocentric refers to the centre of the reference frame being the centre of the Earth. Reference frames can typically be divided into *inertial* and *non-inertial* reference frames, where *inertial* means that the frame is not accelerating (or rotating) and is fixed. The orientation of the coordinate axes is fixed relative to distant stars, typically defined by the vernal equinox and the celestial equator. Such frames are more often used due to simplification of the calculation of motion laws.

  An example of an often-used geocentric inertial reference frame is the Earth-Centric Inertial (ECI).

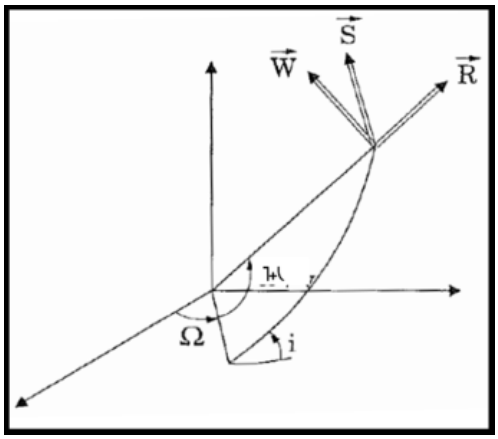
- **Give the definition of the local orbital frame R_{LOF}**

  A local orbit reference frame Z-, Y- and X-axis are defined as follows:

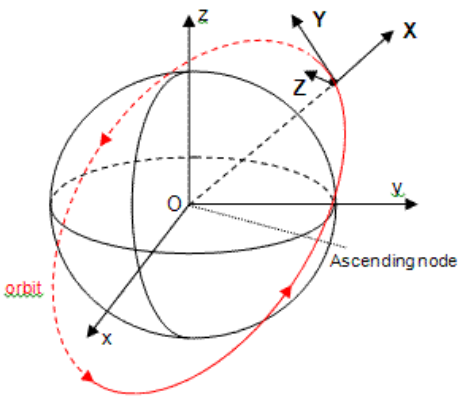
  – Z_{ol} or **R**: points directly from the centre of the central body (in this case Earth) towards the orbiting object (in this case the satellite), Zenith direction.

  – Y_{ol} or **W**: points in the direction of the normal positive to the orbital plane.

  – X_{ol} or **S**: completes the orthonormal trihedron and in this case points in the direction of the satellite's movement along its orbit.



(a) From lecture slides, (R,S,W) vectors (1-2-3)



(b) qsw local orbital reference frame

Figure 1: The LOF axis can be defined in different ways, such as in (b), where it is the X-axis that follows the radial or zenith direction [3]

- **Compute the Cubesat local orbital frame vector directions in your geocentric reference frame RE (theoretical computation)**

  The transformation matrix for converting LOF to ECI can be found in the lecture slides (1-2-3, slide 49):

$$
\begin{aligned}
&(X,Y,Z)|_{ECI} = (x,y,z)|_{LOF} \cdot \mathbf{M}_{\omega+\theta} \cdot \mathbf{M}_i \cdot \mathbf{M}_\Omega \\
&= (x,y,z)|_{LOF} \begin{bmatrix} \cos(\omega+\theta) & \sin(\omega+\theta) & 0 \\ -\sin(\omega+\theta) & \cos(\omega+\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(i) & \sin(i) \\ 0 & -\sin(i) & \cos(i) \end{bmatrix} \cdot \begin{bmatrix} \cos(\Omega) & \sin(\Omega) & 0 \\ -\sin(\Omega) & \cos(\Omega) & 0 \\ 0 & 0 & 1 \end{bmatrix}
\end{aligned} \tag{1}
$$

  where:

  - ω is the argument of perigee (40.8630° for the ISS as of Jan 3rd 2025) [1]
  - θ is the true anomaly (68.2039° for the ISS as of Jan 3rd 2025) [1]
  - i is the inclination of the orbit (51.6°)
  - Ω is the RAAN of the orbit (40.3677° for the ISS as of Jan 3rd 2025) [1]

  The above transformation matrix can be simplified for calculation by setting $\omega = 0$ and $\theta = 0$, however, this can also be calculated in Python with all values. An example of these vectors is generated in the Python code in Appendix A on line 94 using the `compute_LOF()` function with the output:

```
Transformation Matrix:
[[-0.62913233  0.23570859  0.7406983 ]
 [-0.58867996 -0.76675579 -0.25601066]
 [ 0.5075908  -0.59709883  0.62114778]]

Vector in Inertial Frame:
[ 0.34727457 -1.61144642  0.53163974]
```

- **Make a plot of the local orbital frame vector directions (as a function of time)**

  The script for plotting the LOF vector directions can be seen in the Python code in Appendix A on lines 95 using the `computeandplot_LOF` function. Figure 2 shows how the components of the CubeSat's local orbital frame vectors (R, W, S) evolve over time during one orbital period in a circular orbit.

  The R vector points directly from the Earth's center toward the CubeSat. Since the CubeSat orbits in a circular path, the components of R oscillate sinusoidally. The W vector is perpendicular to the orbital plane and for a circular orbit in the equatorial plane, the W vector remains constant along the z-axis. Since the S vector is aligned with the velocity direction, tangent to the orbital path, it also varies sinusoidally, but in a phase shift relative to R since velocity and position are perpendicular.

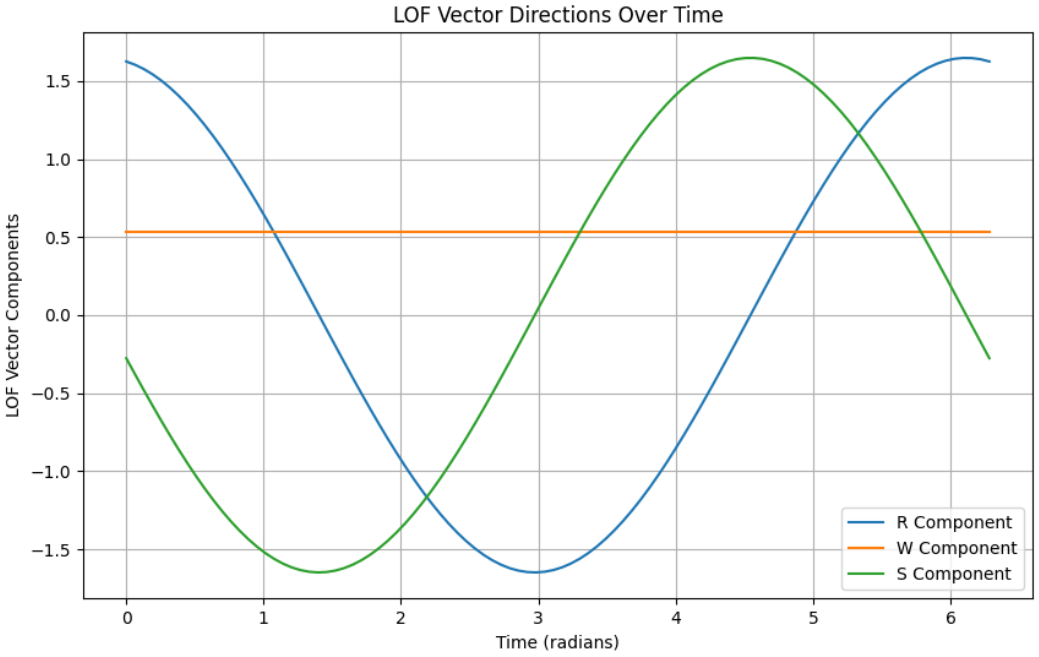


Figure 2: Plot of LOF vector directions over one orbit

2 Visibility of a sub satellite point

Let's consider the point P on Earth located at *P* (0° latitude, 0°, longitude).

- **Describe (in plain language) the conditions for the point P to be visible from the CubeSat.**

  In general, many aspects need to be taken into consideration when determining a point's visibility from a satellite. Most importantly, the point P needs to be within *Line of Sight* and inside the circular *view factor area* of the satellite. The CubeSat must have a clear and unobstructed line of sight to the point P, which means that no part of Earth's surface should be blocking the view. This can be analysed by means of the horizon distance, $D_H = \sqrt{2\,R_E\,h}$, which forms a triangle between the satellite, the point in the horizon and Earth's centre. By looking at the maximum case of the triangle, we can determine a trigonometric relationship and find the maximum angle, α, in both latitude and longitude:

$$\begin{aligned} \alpha &= arcsin\left(\frac{D_H}{R_e + h}\right) \\ &= arcsin\left(\frac{\sqrt{2\,R_E\,h}}{R_e + h}\right) \qquad ||R_E = 6378\,km,\ h = 408\,km \\ &= 19.644385^\circ \approx 19.6^\circ \end{aligned} \tag{2}$$

  In other words, for a point P to be visible from the satellite, the satellite's footprint needs to be between -19.6° and 19.6° in both longitude and latitude.

  The *Orbital Path and Period* thus play a vital role in the visibility of point P. The orbital path must be such that it brings the satellite into line of sight with the point P. E.g. very high inclination orbits, such as polar orbits, or geostationary orbits could be obtained in such a way that the point P would never be visible. Since the CubeSat has a circular orbit and an inclination of 51,6°, it will travel between 51.6° North and 51.6° South, crossing the equator in between. This means that at some points during its orbits, the satellite will cross over the point P (between -19.6° and 19.6° in both longitude and latitude). Figure 3 shows a Molniya orbit and the real-world location of the point P.

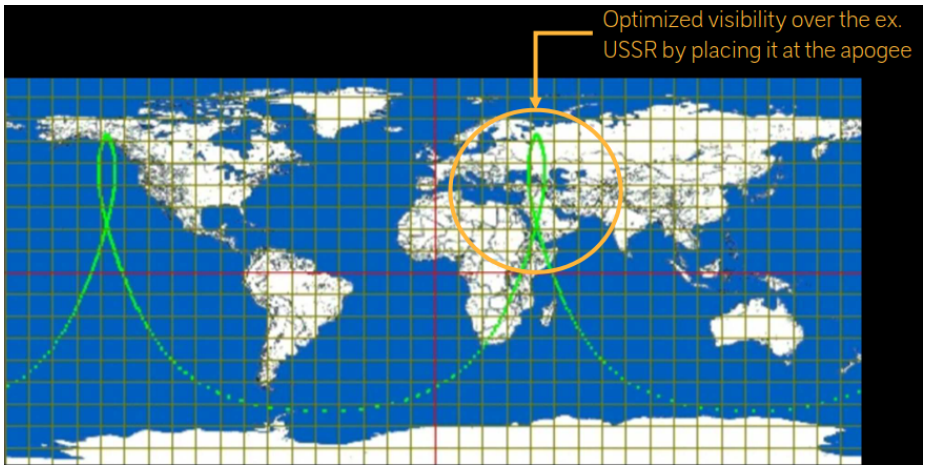


Figure 3: Groundtrack of a Molniya orbit. The equator and prime meridian intersect at Null island (in red).

Realistically, *Night and Day visibility* would also play a role. While technically the point would be visible as long as the Line of Sight requirement is met, it is also important to remember in real-life applications that the point P might not be visible to the CubeSat during night-time when it is dark. Depending on the instruments onboard the CubeSat and what is meant by "visible", i.e. visible to a communications array or visible in the sense that the satellite can take a clear image of the point P. With the given orbital parameters, and an assumed orbital speed similar to the ISS (7.66 km/s), the CubeSat's orbital period would be about 92 min, meaning that it completes a whole orbit around Earth every 92 minutes. As such, it would pass over the equator often enough that it would intersect with daytime.

- **Describe an algorithm that allows to determine if the point P is visible from the CubeSat.**

For point P to be visible from the satellite, the norm of the vector between P and the satellite footprint point P_f coordinates needs to be smaller or equal to the norm of a vector between P and P_{max} (19.7°, 19.7°). This can be mathematically expressed as below:

$$\begin{aligned} \sqrt{P_{f,lat}^2 + P_{f,long}^2} &\leq \sqrt{P_{max,lat}^2 + P_{max,long}^2} \\ \sqrt{P_{f,lat}^2 + P_{f,long}^2} &\leq \sqrt{(19.7°)^2 + (19.7°)^2} \end{aligned} \tag{3}$$

- **Code this algorithm and show, on a ground map, the points of the orbit which are visible from this point P.**

  For this exercise, the code for plotting a ground track from Homework1 will be partially reused and modified in such a way that it checks for coordinates over 100 orbits with the condition from Equation 3 and appends these into a new list and plots them on a ground map. The code implementation can be seen in Appendix A using the `plot_groundtrack_visible()` function and the output plot can be seen below in Figure 4.

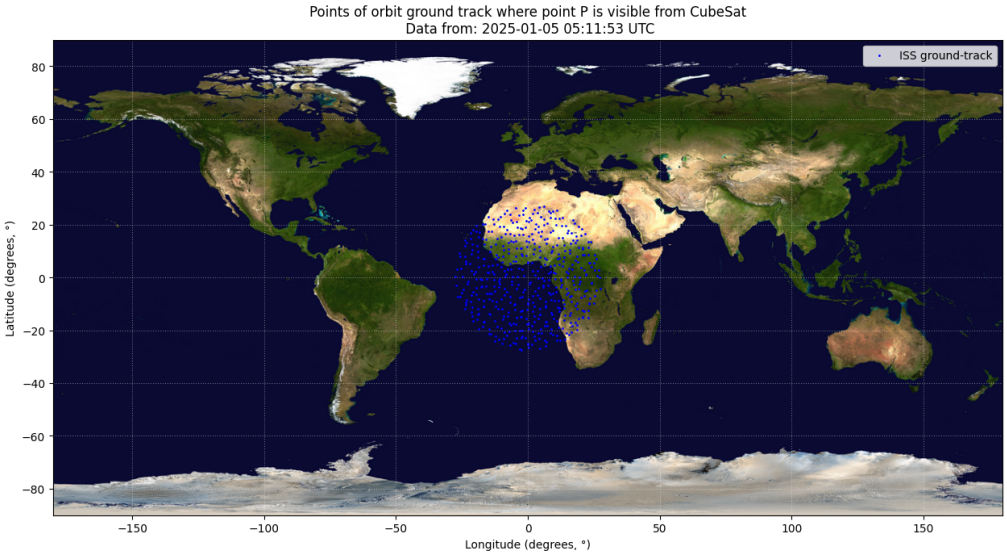


Figure 4: Point P visibility over 100 orbits

- **What is the duration of the visibility for a satellite passing at the zenith of P?**

  When the satellite passes over the zenith of point P, the point is visible from opposite ends of the horizon, i.e., at a total distance of $2D_H$. Since the ISS/CubeSat has an approximately circular orbit, the duration of the visibility for the CubeSat can be calculated based on the orbital period of the ISS and Earth's circumference:

$$\begin{aligned} T_{vis} &= \frac{2D_H}{2\pi R_E} T \\ &= 10.5725 \approx 10.6\, min \end{aligned} \tag{4}$$

  The code implementation can be seen in Appendix A in the `visibility_duration()`: function.

- **Express the vector (CubeSat, P) in satellite local orbital frame.**

  When the satellite is directly above the point P, i.e. at the zenith, the vector $\vec{C}$ = (CubeSat, P) can be expressed in LOF as (-408km, 0, 0).

  At any point during the orbit, the vector $\vec{C}$ can be expressed as the sum of the (negative) vector from the centre of the Earth to the CubeSat, $\vec{R_S}$, and the vector from the centre of the Earth to the point P (which will always be the same), $\vec{R_P}$, as follows:

$$\vec{C} = -\vec{R_S} + \vec{R_P} \tag{5}$$

- **Describe an algorithm to compute the direction of the point P in local orbital reference frame.**

  To compute the direction of the vector between the CubeSat and the point P, $\vec{C}$, we start with the unit vector in LOF pointing straight down (-1, 0, 0). We then take the angle, ϕ, that is formed between the vector from the CubeSat to the centre of the Earth and between the CubeSat and the point P and rotate the unit vector in the orbital plane by this angle. To find the angle, ϕ, we must first solve the hypotenuse, d, of the triangle formed by these points via the law of cosines:

$$\begin{aligned} \phi &= \arccos\left(\frac{d^2 + (R_E + h)^2 + R_E}{2 \cdot d \cdot (R_E + h)}\right) \\ d &= \sqrt{(R_E + h)^2 + R_E^2 - 2 \cdot (R_E + h) \cdot R_E \cdot \cos\theta} \end{aligned} \tag{6}$$

  where θ is the true anomaly, R_E is the radius of the Earth and h is the satellite orbit altitude as before.

- **Code this algorithm and show, on a 3D plot, the vector wrt time over one orbit.**

  The vector, $\vec{C}$, can be transformed to ECI with the transformation matrix from Equation 1 as expressed before.

3 Attitude Determination and Control System (AOCS)

The payload onboard your satellite has a pointing requirement of 0,5°. You have to design a AOCS for the cubesat, with a budget of about 100 kEuros. The sensors and actuators that you can use are to be found on https://www.cubesatshop.com/

- **Make a schematics diagram (such as the one found on slide 61 of "9-10.pdf") of the AOCS control system with the (real) elements found here.**

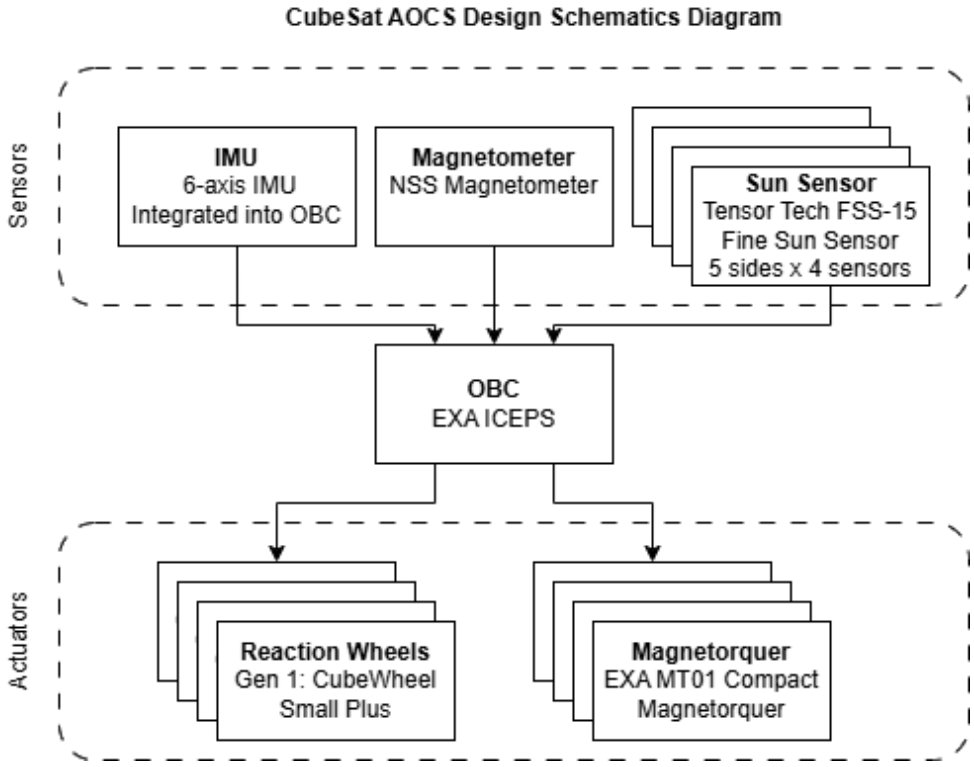


Figure 5: CubeSat AOCS Design Schematics Diagrame made with draw.io

Position	Part Name	Unit Price	Amount	Total Price
OBC / IMU	EXA ICEPS	20 500€	1	20 500€
Magnetometer	NSS Magnetometer	15 000€	1	15 000€
Sun Sensor	Tensor tech fine Sun Sensor	1 800€	4x4	28 800€
Reaction Wheels	CubeWheel small +	6550€	4	26 200€
Magnetorquers	EXA MT01 Compact Magnetorquer	1200€	4	4 800€
				95 300€

Table 1: Cost estimation for CubeSat AOCS design

- **Explain why you chose each element and how many of each are needed.**

  Since we are conducting a CubeSat mission which is being released from the ISS we can make a few assumptions about the requirements that will drive the design of the AOCS system.

  Assumptions

  – *CubeSat size:* Since the ISS CubeSat deployers (J-SSOD and NRCSD) seem to accomodate up to 6U [4] cubesats and most recent CubeSats deployed from the ISS [2] have been up to 3U we will assume 3U as a reasonable size and design driver for the mission. This size limitation of 30cm x 30cm x 30cm for the CubeSat wall designs will dictate the size and amount of systems the satellite can accomodate

  – Assuming the CubeSat is used for Earth-related observations or measurements, we will leave the Nadir face empty of AOCS systems to leave space for the payload. Thus, an earth sensor is out of question for attitude sensing.

  – We also assume that we do not need to leave any budget margins or save any money to accommodate for possible cost overruns for the other satellite systems and as such can freely use the full 100 k€ budget.

  Based on these assumptions, we design the AOCS according to the needs of such a system and an ample amount of redundancy.

  Needs:

  – **Attitude Sensors:** The AOCS needs both an internal and an external attitude sensor: one internal sensor for the CubeSat actuators to base their control algorithm on and an external sensor as a reference system. I chose the *EXA ICEPS Onboard Computer* as it contains a 6-axis Internal Measurement Unit.

    For the external sensor, I chose the *NSS Magnetometer*. This provides quite accurate external attitude measurements based on the Earth's magnetic field and it would be mounted on the outside of the CubeSat, however not on a Boom as it is optimised for, since it will work in complement with other sensors.

    To complement this sensor, I also added 4 *Tensor Tech fine Sun Sensor* on the sun side of the CubeSat and three other sides (not the nadir side) to be mounted on the outside.

  – **Attitude Actuators:** Since we might be dealing with an up to 6U CubeSat, I wanted to include a robust and fine actuation system to also comply with the pointing requirement of 0.5°. For this purpose, I deemed reaction wheels the best choice. I chose the 2nd smallest of the CubeWheel family, the *CubeWheel*

Small +. The minimum requirement would be three of these and I chose 4 for redundancy. When using reaction wheels, it is also good practice to add external actuators to offload the wheels and thus enable a longer mission lifetime.

I also added 4 *EXA MT01 Compact Magnetorquers* as an external actuation method to complement the reaction wheels.

- **Redundancy:** Redundancy is present both in the amount of components chosen and the numbers of sensing methods and actuation. The OBC/IMU has 6 axis internal attitude sensing, providing at least one form of redundancy per axis. The magnetometer provides no redundancy but serves as the main external attitude sensor and is proficiently radiation protected.

  The sun sensors provide an additional external attitude sensing method with plenty of redundancy with 1 extra sun sensor per side (minimum of three) and one redundant side mounted with the sensors.

  The reaction wheels include one additional wheel for redundancy (minimum of three) and external actuation methods with the magnetorquers to offload the reaction wheels. The magnetorquers also include one redundant component (minimum of three)

- **Accuracy:** Given the chosen components and their accuracies, redundancies and error rates the pointing requirement of 0.5° will be obtained.

4 Satellite control

- **Draw the (detailed) control loop of the proposed control system, and precise which data you will pass at the interfaces between the boxes representing the control system.**

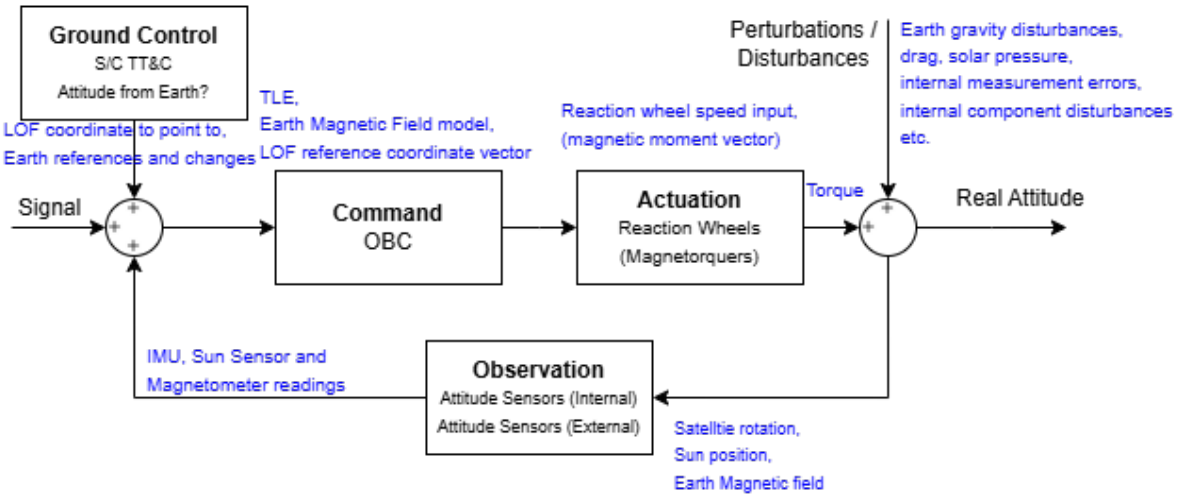


Figure 6: AOCS control loop with data passing indicated in blue

- **What is the data that you need to compute on-board and what is the data that you need to give to the satellite from the ground?**

  On-Board Data: Since no information was given on what the CubeSat would point to (and measure) I have to assume that pointing needs to be possible at any stage in the orbit. As such, the CubeSat shall be able to operate semi-independently of ground control, as having ground stations available for continuous CubeSat monitoring and command can be tedious and costly. Any changes to the CubeSat modes would thus be made only when the CubeSat would be in range to the main Ground Station and the precise attitude of the CubeSat would need to be calculated on-board based off the two external sensor readings. Also the control for the actuation would need to be computed onboard based on the data from the internal attitude measurements.

  Ground Data: From the ground we would at least need to give the TT&C commands to change the CubeSat to pointing mode and possibly to a correction mode for offloading the reaction wheels with the magnetorquers. The Earth Magnetic Field model should also be updated regularly to the CubeSat from ground in addition to any corrections to the satellite's attitude as compared to the perceived attitude from Earth.

  The desired attitude, as a LOF vector and durations for pointing would also need to be updated from the ground.

References

[1] *CelesTrak: Space Stations*. URL: https://celestrak.org/Norad/elements/table.php?GROUP=stations&FORMAT=tle (visited on 01/05/2025).

[2] Erik Kulu. *Nanosatellite & CubeSat Database*. Nanosats Database. URL: https://www.nanosats.eu/database.html (visited on 01/04/2025).

[3] *Local frames*. URL: https://sourceforge.isae.fr/svn/dcas-soft-espace/support/softs/CelestLab/trunk/help/en_US/scilab_en_US_help/Local%20frames.html (visited on 01/03/2025).

[4] *Nanoracks CubeSat Deployer*. In: *Wikipedia*. Page Version ID: 1220805250. Apr. 26, 2024. URL: https://en.wikipedia.org/w/index.php?title=Nanoracks_CubeSat_Deployer&oldid=1220805250 (visited on 01/04/2025).

Appendices

A Python code

```
import matplotlib
matplotlib.use('QtAgg')
import numpy as np
import matplotlib.pyplot as plt
import math
import pytz
from dateutil.relativedelta import relativedelta
from skyfield.api import load, wgs84, EarthSatellite
from datetime import timedelta

def compute_LOF(omega, theta, i, OMEGA):
    # Defining transformation matrixes
    R1 = np.array([
        [np.cos(omega + theta), np.sin(omega + theta), 0],
        [-np.sin(omega + theta), np.cos(omega + theta), 0],
        [0, 0, 1]
    ])

    R2 = np.array([
        [1, 0, 0],
        [0, np.cos(i), np.sin(i)],
        [0, -np.sin(i), np.cos(i)]
    ])

    R3 = np.array([
        [np.cos(OMEGA), np.sin(OMEGA), 0],
        [-np.sin(OMEGA), np.cos(OMEGA), 0],
        [0, 0, 1]
    ])

    # Compute the overall transformation matrix
    transformation_matrix = R1 @ R2 @ R3

    # Example vector in the Local Orbital Frame (LOF)
    vector_LOF = np.array([1, 1, 1])

    # Transform the vector to the inertial frame
    vector_inertial = transformation_matrix @ vector_LOF

    print("Transformation Matrix:")
    print(transformation_matrix)
    print("\nVector in Inertial Frame:")
    print(vector_inertial)

def computeandplot_LOF(omega, theta, i, OMEGA):
```

```
    # Re-Define the transformation matrices
    R3_omega_theta = lambda t: np.array([
        [np.cos(omega + t), np.sin(omega + t), 0],
        [-np.sin(omega + t), np.cos(omega + t), 0],
        [0, 0, 1]
    ])

    R1_i = np.array([
        [1, 0, 0],
        [0, np.cos(i), np.sin(i)],
        [0, -np.sin(i), np.cos(i)]
    ])

    R3_OMEGA = np.array([
        [np.cos(OMEGA), np.sin(OMEGA), 0],
        [-np.sin(OMEGA), np.cos(OMEGA), 0],
        [0, 0, 1]
    ])

    # Generate time values and calculate LOF vectors over time
    time_steps = np.linspace(0, 2 * np.pi, 100)
    lof_vectors = []

    for t in time_steps:
        transformation_matrix = R3_omega_theta(t) @ R1_i @ R3_OMEGA
        vector_LOF = np.array([1, 1, 1])
        vector_inertial = transformation_matrix @ vector_LOF
        lof_vectors.append(vector_inertial)

    lof_vectors = np.array(lof_vectors)

    # Plotting the LOF vector components over time
    plt.figure(figsize=(10, 6))
    plt.plot(time_steps, lof_vectors[:, 0], label='R-Component')
    plt.plot(time_steps, lof_vectors[:, 2], label='W-Component')
    plt.plot(time_steps, lof_vectors[:, 1], label='S-Component')
    plt.xlabel('Time-(radians)')
    plt.ylabel('LOF-Vector-Components')
    plt.title('LOF-Vector-Directions-Over-Time')
    plt.legend()
    plt.grid(True)
    plt.savefig('../doc/Graphics/LOF_vector_directions_plot')
    plt.show()

def calculate_max_view_angle(h, R_E):
    # Calculate alpha using the formula
    D_H = math.sqrt(2 * R_E * h)
    alpha = math.asin(D_H / (R_E + h))

    # Convert alpha from radians to degrees
    alpha_deg = math.degrees(alpha)
```

```
97
98      # Display the result
99      print(f"\nAlpha (in radians): {alpha:.6f}")
100     print(f"Alpha (in degrees): {alpha_deg:.6f}")
101
102 def plot_groundtrack_visible():
103     '''
104     Parts of this software was reused from homework1
105     Ground track computation
106     * get starting time from epoch
107     * set timescale and calculate subpoints (latitude and longitude)
108     * Plot the groundtrack onto an existing map
109         * image credits: https://upload.wikimedia.org/wikipedia/commons
              /2/23/Blue_Marble_2002.png
110     '''
111     timescale = load.timescale()
112
113     # two-line elements taken from https://celestrak.org/Norad/elements/
          table.php?GROUP=stations&FORMAT=tle
114     # taken on jan 5th
115     line1 = '1 25544U 98067A   25005.21659662  .00022613  00000+0  39198-3 0
          9995'
116     line2 = '2 25544  51.6390  34.3594 0006245  46.9928  71.6484
          15.50776974489887'
117     satellite = EarthSatellite(line1, line2, 'ISS (ZARYA)', timescale)
118
119     # Orbital elements
120     eccentricity = line2[26:33]
121     inclination = line2[8:16]
122     right_ascension = line2[17:25]
123     argument = line2[34:42]
124     mean_anomaly = line2[43:51]
125     mean_motion = line2[52:63]
126
127     # Calculate semi-major axis
128     T = 24 * 60 * 60 / float(mean_motion)
129     mu = 3.986 * 10 ** 14
130     a = (((T ** 2) * mu) / (4 * (math.pi ** 2))) ** (1 / 3)
131
132     # Get current time in a timezone-aware fashion from the epoch
133     tz = pytz.timezone('UTC')
134     dt = satellite.epoch.astimezone(tz)
135     print()
136     print(satellite)
137     print(f"Exectution time: {dt:%Y-%m-%d %H:%M:%S %Z}\n")
138
139     # Split 3 orbits (3*92.94061233 minutes) into 400 evenly spaced
          Timescales as indicated by points
140     # 400 chosen for map visibility reasons, could be 101 ==> one plot every
           2 min + endpoints
141     orbits_min = 100 * T / 60
```

```
    t0 = timescale.utc(dt)
    t1 = timescale.utc(dt + relativedelta(minutes=orbits_min))
    timescales = timescale.linspace(t0, t1, 10000)

    # calculate the latitude and longitude subpoints.
    geocentrics = satellite.at(timescales)
    subpoints = wgs84.subpoint_of(geocentrics)
    latitude = subpoints.latitude.degrees
    longitude = subpoints.longitude.degrees

    latitude_filtered = []
    longitude_filtered = []

    # Filter through groundtrack points
    alpha = 19.644385
    for p in range(len(latitude)):
        '''if latitude[p] > -alpha and latitude[p] < alpha:
            if longitude[p] > -alpha and longitude[p] < alpha:
                latitude_filtered.append(latitude[p])
                longitude_filtered.append(longitude[p])
        '''
        if math.sqrt(
            (latitude[p])**2 + (longitude[p])**2
        ) <= math.sqrt(
            (alpha)**2 + (alpha)**2
        ):
            latitude_filtered.append(latitude[p])
            longitude_filtered.append(longitude[p])



    # Load background image
    background_image_path = r'C:\Users\benja\PycharmProjects\groundtrack\
        earth.jpg'
    background_img = plt.imread(background_image_path)

    # Create the plot
    plt.figure(figsize=(15.2, 8.2))
    plt.imshow(background_img, extent=[-180, 180, -90, 90])

    # Plot the ground track
    title = f"Points of orbit ground track where point P is visible from
        CubeSat\n" \
            f" Data from: {dt:%Y-%m-%d %H:%M:%S %Z}" #TODO change this
    #plt.scatter(longitude, latitude, label="ISS ground-track", color='red',
         marker='o', s=1)
    plt.scatter(longitude_filtered, latitude_filtered, label="ISS ground-
        track", color='blue', marker='o', s=1)
    plt.xlabel("Longitude (degrees, \N{DEGREE SIGN})")
    plt.ylabel("Latitude (degrees, \N{DEGREE SIGN})")
```

```
189     plt.title(title)
190
191     # Show the plot
192     plt.legend()
193     plt.grid(True, color='w', linestyle=":", alpha=0.4)
194     plt.savefig("../doc/Graphics/P_visible_groundtrack")
195     plt.show()
196
197 def visibility_duration(h, R_E, T):
198     D_H = math.sqrt(2 * R_E * h)
199     T_vis = ((2*D_H) / (2*math.pi * R_E)) * T
200     print("Visibility duration (in minutes):", T_vis)
201
202 # Initial parameters taken from ISS data: https://celestrak.org/Norad/
        elements/table.php?GROUP=stations&FORMAT=tle
203 omega = np.radians(40.8630)
204 theta = np.radians(68.2039)
205 i = np.radians(51.6)
206 OMEGA = np.radians(40.3677)
207 h = 408
208 R_E = 6378  # Radius of Earth in km
209 orbital_period = 92.86 # ISS orbital period in minutes
210
211 compute_LOF(omega, theta, i, OMEGA)
212 if input("Do you want to plot LOF directions [y/n]?") == "y":
213     computeandplot_LOF(omega, theta, i, OMEGA)
214 calculate_max_view_angle(h, R_E)
215 if input("Do you want to plot groundtrack [y/n]?") == "y":
216     plot_groundtrack_visible()
217 visibility_duration(h, R_E, orbital_period)
```