Assignment 3: Attitude Dynamics and Control

Abstract

In this assignment, the student will learn how to model attitude dynamics of rigid bodies. Specifically, the student will implement quaternion representation of orientation of a satellite, and study how it rotates relative to the orbit frame and relative to the inertial frame. Further, the student will implement simple PD controller to point the satellite in a desired direction and show the results through plotting using Matplotlib and animation through Pyvista.

1 Objective

The objective of this assignment is to learn about attitude dynamics and control and see how the attitude of a satellite in elliptical orbit can be controlled.

2 Attitude Dynamics

The attitude of a spacecraft can be represented by the quaternion $\mathbf{q}_{i,b} := \begin{bmatrix}\eta_{i,b} & \boldsymbol{\epsilon}_{i,b}^\top\end{bmatrix}^\top$ describing the orientation of the body frame, relative to the inertial frame (ECI). The kinematics of the quaternion can be written as

$$\dot{\mathbf{q}}_{i,b} = \frac{1}{2}\mathbf{q}_{i,b} \otimes \begin{bmatrix}0 \\ \boldsymbol{\omega}_{i,b}^b\end{bmatrix} = \frac{1}{2}\mathbf{T}(\mathbf{q}_{i,b}) \begin{bmatrix}0 \\ \boldsymbol{\omega}_{i,b}^b\end{bmatrix} \tag{1}$$

where $\otimes$ is the quaternion product defined as (Egeland and Gravdahl, 2002)

$$\mathbf{q} = \begin{bmatrix}\eta_1 \\ \boldsymbol{\epsilon}_1\end{bmatrix} \otimes \begin{bmatrix}\eta_2 \\ \boldsymbol{\epsilon}_2\end{bmatrix} = \begin{bmatrix}\eta_1\eta_2 - \boldsymbol{\epsilon}_1^\top\boldsymbol{\epsilon}_2 \\ \eta_1\boldsymbol{\epsilon}_2 + \eta_2\boldsymbol{\epsilon}_1 + \mathbf{S}(\boldsymbol{\epsilon}_1)\boldsymbol{\epsilon}_2\end{bmatrix} \tag{2}$$

where $\mathbf{S}(\cdot)$ is the cross-product operator defined for a vector $\mathbf{v} = \begin{bmatrix}v_1 & v_2 & v_3\end{bmatrix}^\top$ as

$$\mathbf{S}(\mathbf{v}) = \begin{bmatrix}0 & -v_3 & v_2 \\ v_3 & 0 & -v_1 \\ -v_2 & v_1 & 0\end{bmatrix}. \tag{3}$$

The quaternion product can also be written in vector form where $\mathbf{q}\otimes\mathbf{u} = \mathbf{T}(\mathbf{q})\mathbf{u}$ where

$$\mathbf{T}(\mathbf{q}) = \begin{bmatrix}\eta & -\boldsymbol{\epsilon}^\top \\ \boldsymbol{\epsilon} & \eta\mathbf{I} + \mathbf{S}(\boldsymbol{\epsilon})\end{bmatrix} \in \mathbb{R}^{4\times 4}. \tag{4}$$

The rotation matrix from the body frame to the inertial frame can be constructed given the quaternion $\mathbf{q}_{i,b}$ as

$$\mathbf{R}_b^i = \mathbf{I} + 2\eta_{i,b}\mathbf{S}(\boldsymbol{\epsilon}_{i,b}) + 2\mathbf{S}^2(\boldsymbol{\epsilon}_{i,b}), \tag{5}$$

while composite rotations can be found through the quaternion product as

$$\mathbf{q}_{a,d} = \mathbf{q}_{a,b} \otimes \mathbf{q}_{b,c} \otimes \mathbf{q}_{c,d} = \mathbf{T}(\mathbf{q}_{a,b})\mathbf{T}(\mathbf{q}_{b,c})\mathbf{q}_{c,d}. \tag{6}$$

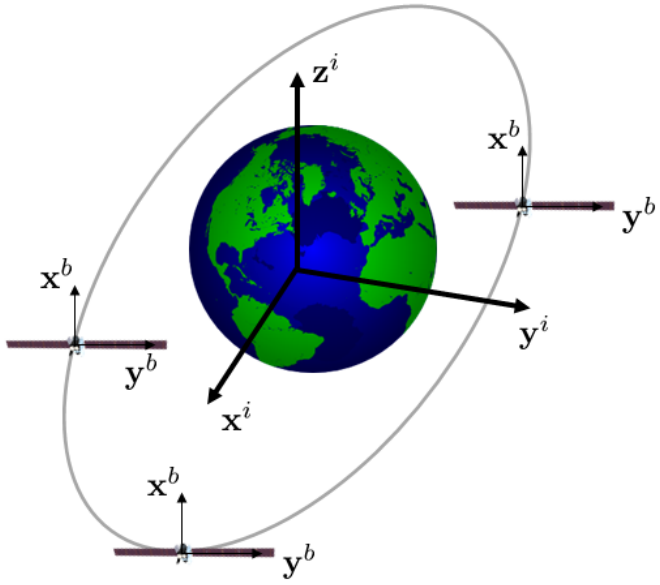

Figure 1: Pointing the $\mathbf{x}^b$ axis along the $\mathbf{z}^i$ direction. It is evident that it is not very useful with regard to pointing sensors or aligning solar panels towards the Sun.

The inverse quaternion that rotates the opposite way can be found by simply taking a negative sign on the vector part (i.e. axis of rotation).

$$\mathbf{q}_{b,a} = \begin{bmatrix} \eta_{a,b} & -\boldsymbol{\epsilon}_{a,b}^\top \end{bmatrix}^\top. \tag{7}$$

The rotational dynamics of a rigid body can be found by using Euler's moment equation as

$$\mathbf{J}\dot{\boldsymbol{\omega}}_{i,b}^b = -\mathbf{S}(\boldsymbol{\omega}_{i,b}^b)\mathbf{J}\boldsymbol{\omega}_{i,b}^b + \boldsymbol{\tau}_a^b + \boldsymbol{\tau}_p^b \tag{8}$$

where $\mathbf{J} \in \mathbb{R}^{3\times3}$ is the inertia matrix of the spacecraft, $\boldsymbol{\omega}_{i,b}^b$ is the angular velocity of the body frame relative to the inertial frame, $\boldsymbol{\tau}_a^b$ is the torque produced by actuators on the spacecraft, while $\boldsymbol{\tau}_p^b$ is the perturbing torques acting on the spacecraft, e.g. gravitational torque, solar wind moments, dipole moments, aerodynamic drag moments and third body perturbations. For now the perturbation torques are set to zero and are not modeled.

Now assume that the objective is to align the $\mathbf{x}^b$ axis along the $\mathbf{z}^i$ axis to point the sensor in that direcion. Figure 1 shows how the satellite will be aligned during one orbit. It is evident that if the satellite has a sensor, e.g. camera for Earth observations or similar, then it is not very useful to control relative to the inertial frame. Instead, it is desirable to control the orientation relative to the orbit frame.

The quaternion representing the orientation of the body frame relative to the orbit frame can be found through the composite rotation

$$\mathbf{q}_{o,b} = \mathbf{q}_{o,i} \otimes \mathbf{q}_{i,b} \tag{9}$$

which holds the kinematics as

$$\dot{\mathbf{q}}_{o,b} = \frac{1}{2}\mathbf{T}(\mathbf{q}_{o,b}) \begin{bmatrix} 0 \\ \boldsymbol{\omega}^b_{o,b} \end{bmatrix} \tag{10}$$

where the angular velocity can be found as

$$\boldsymbol{\omega}^b_{o,b} = \boldsymbol{\omega}^b_{o,i} + \boldsymbol{\omega}^b_{i,b} \tag{11}$$

$$\boldsymbol{\omega}^b_{o,b} = \boldsymbol{\omega}^b_{i,b} - \mathbf{R}^b_i \boldsymbol{\omega}^i_{i,o} \tag{12}$$

where the latter term was found in Assignment 1. Hence, an expression for the angular velocity of the body frame relative to the orbit frame has been developed. Now, to find the dynamics relative to the orbit frame, pre-multiply each term in Equation 12 by the inertia matrix $\mathbf{J}$, such that

$$\mathbf{J}\boldsymbol{\omega}^b_{o,b} = \mathbf{J}\boldsymbol{\omega}^b_{i,b} - \mathbf{J}\mathbf{R}^b_i \boldsymbol{\omega}^i_{i,o}. \tag{13}$$

Now, assume that the inertia matrix is constant (i.e. we do not consider changes in mass and inertia), then equation (13) can be differentiated using ($\dot{\mathbf{R}}^b_i = -\mathbf{S}(\boldsymbol{\omega}^b_{i,b})\mathbf{R}^b_i$) as

$$\mathbf{J}\dot{\boldsymbol{\omega}}^b_{o,b} = \mathbf{J}\dot{\boldsymbol{\omega}}^b_{i,b} + \mathbf{J}\mathbf{S}(\boldsymbol{\omega}^b_{i,b})\mathbf{R}^b_i \boldsymbol{\omega}^i_{i,o} - \mathbf{J}\mathbf{R}^b_i \dot{\boldsymbol{\omega}}^i_{i,o}. \tag{14}$$

By inserting (8) into (14), it is obtained that

$$\mathbf{J}\dot{\boldsymbol{\omega}}^b_{o,b} = -\mathbf{S}(\boldsymbol{\omega}^b_{i,b})\mathbf{J}\boldsymbol{\omega}^b_{i,b} + \boldsymbol{\tau}^b_a + \boldsymbol{\tau}^b_p + \mathbf{J}\mathbf{S}(\boldsymbol{\omega}^b_{i,b})\mathbf{R}^b_i \boldsymbol{\omega}^i_{i,o} - \mathbf{J}\mathbf{R}^b_i \dot{\boldsymbol{\omega}}^i_{i,o}. \tag{15}$$

where $\dot{\boldsymbol{\omega}}^i_{i,o}$ and $\boldsymbol{\omega}^i_{i,o}$ were calculated in Assignment 2, representing the angular acceleration and velocity of the orbit frame relative to inertial frame.

Attitude Dynamics

1. Create a new file called `attitude_dynamics.py` and create the following functions
   - S(w)
   - T(q)
   - calculate_rotation_matrix_from_quaternion(q_ab)
   - calculate_inverse_quaternion(q_ab)
   - quaternion_kinematics(q_ob, w_ob_b)
   - attitude_dynamics(J, q_ob, w_ob_b, w_io_i, w_io_i_dot, R_i_o, tau_a_b, tau_p_b)

The transformation matrix T() can be implemented based on the following code:

```
def T(q):
    # Creating the transformation matrix
    eta = q[0]
    epsilon = np.array([q[1], q[2], q[3]])
    I = np.eye(3)
```

```
    top_left = np.array([[eta]])
    top_right = -epsilon.reshape(1,3)
    bottom_left = epsilon.reshape(3,1)
    bottom_right = eta*I + S(epsilon)

    T_mat = np.block([
                    [top_left, top_right],
                    [bottom_left, bottom_right]
                    ])

    return T_mat
```

3 Proportional Derivative Attitude Controller

A proportional derivative attitude controller can be created to make the satellite point in a desired direction. As mentioned above, we operate the satellite attitude relative to the orbit frame to make meaningful maneuvers. Let a desired quaternion and angular velocity be defined as $\mathbf{q}_{o,d}, \boldsymbol{\omega}_{o,d}^{d} \in \mathcal{L}_{\infty}$, which represent a desired attitude and angular velocity relative to the orbit frame and can be chosen by the designer. Note that if we apply a desired angular velocity, we require an update equation for the desired quaternion such that for now we are only interested in constant attitudes.

The quaternion error which we want to make go to zero can be written as

$$\mathbf{q}_{d,b} = \mathbf{q}_{d,o} \otimes \mathbf{q}_{o,b} = \mathbf{T}(\mathbf{q}_{d,o})\mathbf{q}_{o,b} \tag{16}$$

where $\mathbf{q}_{d,o}$ can be found as the inverse quaternion

$$\mathbf{q}_{d,o} = \begin{bmatrix} \eta_{o,d} & -\boldsymbol{\epsilon}_{o,d}^{\top} \end{bmatrix}^{\top}. \tag{17}$$

Now the angular velocity error can be found as

$$\boldsymbol{\omega}_{d,b}^{b} = \boldsymbol{\omega}_{d,o}^{b} + \boldsymbol{\omega}_{o,b}^{b} \tag{18}$$
$$\boldsymbol{\omega}_{d,b}^{b} = -\boldsymbol{\omega}_{o,d}^{b} + \boldsymbol{\omega}_{o,b}^{b} \tag{19}$$
$$\boldsymbol{\omega}_{d,b}^{b} = \boldsymbol{\omega}_{o,b}^{b} - \boldsymbol{\omega}_{o,d}^{b} \tag{20}$$
$$\boldsymbol{\omega}_{d,b}^{b} = \boldsymbol{\omega}_{o,b}^{b} - \mathbf{R}_{d}^{b}\boldsymbol{\omega}_{o,d}^{d} \tag{21}$$

The rotation matrix $\mathbf{R}_{d}^{b}$ can be constructed using the quaternion $\mathbf{q}_{b,d}$. We now have the error in attitude ($\mathbf{q}_{d,b}$) of the body frame relative to the desired frame, and the error in angular velocity $\boldsymbol{\omega}_{d,b}^{b}$. If both these errors go to zero, the satellite will point in a desired direction relative to the orbit frame. A simple proportional derivative controller can be constructed as

$$\boldsymbol{\tau}_{d}^{b} = -k_p\boldsymbol{\epsilon}_{d,b} - k_d\boldsymbol{\omega}_{d,b} \tag{22}$$

where $k_p, k_d > 0$ are two positive gains, while $\boldsymbol{\epsilon}_{d,b}$ is the vector part of the error quaternion. Quaternions have four elements, while the torque is defined as a vector of three elements, such that only the vector part must be used. Also due

to the nature of quaternions being unit length of one, it follows that as $\boldsymbol{\epsilon}_{d,b}$ goes to zero, that $\eta_{d,b}$ goes to ± 1.

The notation used here for the control law $\boldsymbol{\tau}_d^b$ where the subscript "d" denotes that it is a desired torque. Later when we introduce actuator dynamics, there will be a mapping from the desired torque over to the actuator torque $\boldsymbol{\tau}_a^b$ where actuator constraints can be enforced. For now, we simply let $\boldsymbol{\tau}_a^b = \boldsymbol{\tau}_d^b$.

Controller Design

1. Create a new file called `controllers.py` and implement the PD attitude controller. Let it have the inputs: $\mathbf{q}_{o,b}, \boldsymbol{\omega}_{o,b}^b, \mathbf{q}_{o,d}, \boldsymbol{\omega}_{o,d}^d, k_p, k_d$ and return the desired torque $\boldsymbol{\tau}_d^b$. Use the above calculations to find the error quaternion and angular velocity error before finding the control signal.

4 Solver

We are now ready to implement the changes into our solver by including the attitude dynamics.

```
def satellite_dynamics_loop(t, state, params):

    # Extracting state vector
    theta = state[0]
    q_ob = np.array([state[1], state[2], state[3], state[4]])
    w_ob_b = np.array([state[5], state[6], state[7]])

    # ... old code

    # Calculate tau_a_b
    # Calculate q_ob_b_dot
    # Calculate w_ob_b_dot

    # Expand logging data to include the new variables q_ob and
     w_ob_b

    # Populating the state vector derivative
    state_dot[0] = theta_dot
    state_dot[1] = q_ob_b_dot[0]
    state_dot[2] = q_ob_b_dot[1]
    state_dot[3] = q_ob_b_dot[2]
    state_dot[4] = q_ob_b_dot[3]
    state_dot[5] = w_ob_b_dot[0]
    state_dot[6] = w_ob_b_dot[1]
    state_dot[7] = w_ob_b_dot[2]


    #Returning state vector derivative and data_log
    return state_dot, data_entry
```

Satellite Dynamics Loop

1. Expand the initial condition to contain an initial quaternion of the body relative to the orbit frame, as well as an initial angular velocity. Let $\mathbf{q}_{o,b}(0) = \begin{bmatrix}0 & 1 & 0 & 0\end{bmatrix}^\top$ and $\boldsymbol{\omega}^b_{o,b} = \begin{bmatrix}0.1 & 0.2 & -0.3\end{bmatrix}^\top$.
2. Expand the satellite_dynamics_loop with the additional inputs/outputs and implement the functions created through this assignment.
3. Let the desired quaternion be defined as $\mathbf{q}_{o,d} = \begin{bmatrix}1 & 0 & 0 & 0\end{bmatrix}^\top$ and the desired angular velocity as $\boldsymbol{\omega}^d_{o,d} = \begin{bmatrix}0 & 0 & 0\end{bmatrix}^\top$.
4. Expand on the logging to include $\mathbf{q}_{o,b}$ and $\boldsymbol{\omega}^b_{o,b}$ and plot the quaternion and angular velocity. How do they compare to what you are expecting ?
5. Select different gains for the controller, and change the initial conditions. How well does the controller work?

5 Revisiting the Orbital Mechanics

The quaternion $\mathbf{q}_{i,b}$ is required for visualization of results and can be found through a combination of the quaternion $\mathbf{q}_{o,b}$ and the rotation matrix $\mathbf{R}^o_i$. Ideally, both should be expressed as quaternions, as this allows easier manipulations, and the fact that to go from a rotation matrix to quaternion often leads to discontinuities that are unwanted. To that end we know that the matrix $\mathbf{R}^o_i$ is constructed as a z rotation with Ω, an x rotation by the inclination i followed by a z rotation by $\omega + \theta$. This can be written as three individual quaternions

$$\mathbf{q}_\Omega = \begin{bmatrix}\cos(\frac{\Omega}{2}) & 0 & 0 & \sin(\frac{\Omega}{2})\end{bmatrix}^\top \tag{23}$$

$$\mathbf{q}_i = \begin{bmatrix}\cos(\frac{i}{2}) & \sin(\frac{i}{2}) & 0 & 0\end{bmatrix}^\top \tag{24}$$

$$\mathbf{q}_{\omega+\theta} = \begin{bmatrix}\cos(\frac{\omega+\theta}{2}) & 0 & 0 & \sin(\frac{\omega+\theta}{2})\end{bmatrix}^\top \tag{25}$$

where

$$\mathbf{q}_{i,o} = \mathbf{q}_{\omega+\theta} \otimes \mathbf{q}_i \otimes \mathbf{q}_\Omega \tag{26}$$

which can be calculated using the $\mathbf{T}()$ operator.

Orbit to Inertial Quaternion

1. Create the following function inside the `orbital_mechanics.py` file: calculate_quaternion_from_orbital_parameters(omega, Omega, i, theta)
2. Create the rotation matrix $\mathbf{R}^i_o$ using $\mathbf{q}_{i,o}$ (note the order of subscripts). Then compare it with the original $\mathbf{R}^i_o$ matrix to show that they are identical.

6 Animation of Attitude Dynamics

Quaternions are difficult to visualize and Pyvista requires Euler angles as inputs to perform rotations. To that end, the quaternion can be related to Euler angles through the following definitions. Let $\mathbf{q}_{i,b} := \begin{bmatrix} q_0 & q_1 & q_2 & q_3 \end{bmatrix}^\top$, then

$$\phi = \text{atan2}\left(2(q_0 q_1 + q_2 q_3), 1 - 2(q_1^2 + q_2^2)\right) \tag{27}$$

$$\theta = \begin{cases} \frac{\pi}{2}\text{sign}\left(2(q_0 q_2 - q_1 q_3)\right) & \text{if } |2(q_0 q_2 - q_1 q_3)| \geq 1 \\ \sin^{-1}\left(2(q_0 q_2 - q_1 q_3)\right) & \text{otherwise} \end{cases} \tag{28}$$

$$\psi = \text{atan2}(2(q_0 q_3 + q_1 q_2), 1 - 2(q_2^2 + q_3^2)) \tag{29}$$

which also accounts for gimbal lock. The Euler angles can then be constructed as $\mathbf{\Theta}_{i,b} = \begin{bmatrix} \phi & \theta & \psi \end{bmatrix}^\top$.

Animation of Attitude Dynamics

1. Create the function calculate_euler_angles_from_quaternion(q_ab) inside `attitude_dynamics.py`.
2. Update the animation to also contain the orbit frame, located at the satellite location and moving with the satellite.
3. Calculate $\mathbf{q}_{i,b}$ inside the satellite_dynamics_loop and add it to be logged. This quaternion describes the attitude of the satellite relative to the inertial frame and is what we desire to visualize.
4. Extract the quaternion inside the animate_satellite() function as

```
def animate_satellite_with_attitude(t, data_log):
    #... old code

    q_ib_array = np.array(data_log["q_ib"])

    for i in range(len(t)):
        Theta_ib =
    calculate_euler_angles_from_quaternion(q_ib_array
    [i])*180/np.pi
        # old code ...
```

   and then observe that the satellite is able to control its attitude.
5. Do one animation with a circular trajectory, and another animation with a highly elliptical trajectory. What do you see?
6. Document your results and findings in the report.