UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa

Applied Subsystems Design
Master's degree in Space and Aeronautical Engineering
Universitat Politècnica de Catalunya

Final Project

ADCS Final Project: Tracking "El Niño"

González Martínez, Víctor

Contents

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa

1 Introduction

The objective of this project is to design and simulate an **Attitude Determination and Control System (ADCS)** capable of orienting a satellite so that one of its body axes (specifically, the **+Z body axis**) continuously points toward a **Point of Interest (POI)** located on Earth's surface — in this case, the city of **Quito (Ecuador)** — while the satellite completes an orbital revolution around the planet.

To achieve this, a complete **attitude dynamics and control simulation** was developed in **MATLAB**. The implemented model includes the following main elements:

- The **orbital motion** of the satellite in a **700 km circular equatorial orbit**, expressed in an **Earth-Centered Inertial (ECI)** frame.
- The **rotation of the Earth** at its real angular velocity, causing the position of the POI to evolve within the inertial frame over time.
- The **rotational dynamics** of the satellite, taking into account its **inertia tensor**, the presence of **constant disturbance torques**, and the **control torques** generated by the ADCS.
- A **Proportional–Derivative–Integral (PDI)** controller implemented using **quaternions**, which avoids the singularities associated with Euler angles and ensures smooth orientation transitions.

The purpose of the controller is to minimize both the attitude error (represented by the vector part of a quaternion) and the angular rate error, maintaining the satellite's +Z body axis aligned with the line-of-sight vector from the satellite to the POI at all times.

To obtain a physically consistent and visually illustrative simulation, the code includes:

- Realistic **orbital and rotational kinematics** for both the satellite and the Earth.
- A **feedforward rate estimation** term that anticipates the necessary rotational velocity, improving transient response.
- A **3D graphical visualization** showing the Earth, the orbit, the satellite, and its body-frame axes, making it possible to observe the pointing behavior in real time.
- A **video export function** that generates a complete orbital animation of approximately **10 seconds**, preserving the true dynamics but accelerating playback for improved visualization.

2 Results and Analysis

The simulation demonstrates the correct performance of the implemented ADCS. Throughout the orbital revolution, the satellite successfully maintains its +Z axis continuously pointing toward the Point of Interest (Quito), compensating for both its own orbital motion and the Earth's rotation.

2.1 Attitude Behavior

During the initial phase of the simulation, the controller progressively adjusts the satellite's orientation to align the +Z body axis with the line-of-sight vector toward the POI. The quaternion-based PDI controller provides a **stable, accurate, and oscillation-free** response, allowing smooth transitions between orientations without any singularities.

The **feedforward term** enhances the system's responsiveness by predicting the angular rate variations induced by orbital motion, thus reducing tracking delay.

2.2 Visual Tracking

In the 3D representation (Fig. 1) generated in MATLAB:

- The **Earth** is displayed as a rotating sphere, with its rotation axis aligned with the inertial Z-axis.
- The **satellite** moves along its orbit, represented as a dashed circular path.
- The **body-frame axes** (X, Y, and Z) are shown as colored arrows. Throughout the entire orbit, the **+Z axis** remains directed toward Quito, fulfilling the tracking objective.
- The **line-of-sight vector (Sat→POI)** updates dynamically, confirming the continuous alignment between the two directions.

The **control torques** remain within the defined saturation limits (±0.8 N·m), demonstrating a realistic actuator capability. Despite the presence of a **constant disturbance torque**, the integral term of the controller effectively eliminates steady-state error, maintaining accurate and stable pointing throughout the simulation.

The full animation of the satellite attitude control simulation can be viewed at the following link:
`Google Drive { G07`$_{F}inalProject.$

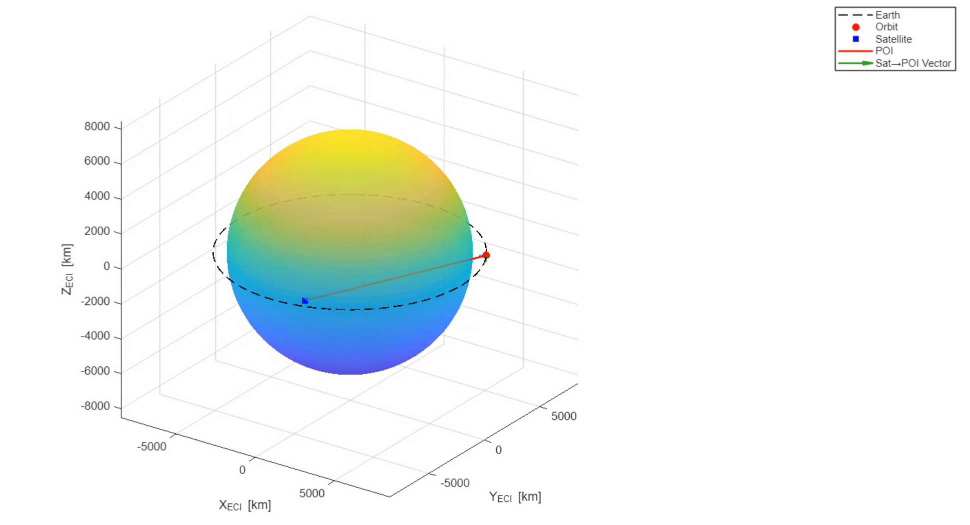

Figure 1: $+Z_{\text{body}}$ following POI (Quito).

2.3 Performance and Stability

The attitude error (vector part of the quaternion) decreases rapidly during the initial transient and remains close to zero for the rest of the simulation, confirming the **stability and robustness** of the control system. Similarly, the angular velocity smoothly converges to the desired value, and the torque commands remain **continuous and bounded**.

The final animation clearly illustrates the result: the satellite maintains its +Z axis directed toward the POI at all times, even as the Earth rotates and the satellite completes its orbital revolution. This demonstrates the correct performance of the implemented control algorithm.

3 Conclusions

The project successfully meets its main objective: developing a quaternion-based attitude control system capable of **tracking a fixed target on the Earth’s surface** during the satellite’s orbital motion.

The simulation accurately couples orbital kinematics, Earth’s rotation, and satellite attitude dynamics, providing both **numerical** and **visual** validation of the control system’s performance.

The use of quaternions was essential to avoid singularities and guarantee smooth attitude representation, while the PDI control law ensured **precision, stability, and smooth tracking**. The inclusion of the feedforward term improved transient response, reducing delay and enhancing control efficiency.

Both the numerical results and the visual animation confirm that the satellite continuously maintains accurate pointing toward the Point of Interest throughout the entire orbit, satisfying the performance requirements expected from a functional ADCS.

A Appendix

Matlab code used:

```
clear; clc; close all;

%% ADCS Final Project - Víctor González Martínez

% Global constants and parameters
G    = 6.67430e-11;            % [m^3/(kg s^2)]
M    = 5.972e24;               % [kg]
mu   = G*M;                    % [m^3/s^2]
Re   = 6371e3;                 % [m]
h    = 700e3;                  % [m]
r    = Re + h;                 % [m]
wE   = 2*pi/86400;             % [rad/s]

% POI: Quito (lat=0°, lon=-78.467834°)
lat = deg2rad(0.0);
lon0 = deg2rad(-78.467834);

% Orbital dynamics
T_orbit = 2*pi*sqrt(r^3/mu);       % [s]
w = 2*pi/T_orbit;                  % [rad/s]

% Time
dt = 0.5;                          % [s]
t_end = T_orbit;                   % one complete orbit
t = 0:dt:t_end;

% Reference kinematics (ECI)
poi_ecef = Re * [cos(lat)*cos(lon0); cos(lat)*sin(lon0); sin(lat)];     % Quito in ECEF (spherical Earth)

Rz = @(ang)[cos(ang) -sin(ang) 0;                           % Rotation around Z
            sin(ang)  cos(ang) 0;
                 0         0   1];

r_sat_ECI = @(tt) r * [cos(w*tt); sin(w*tt); 0*tt+0];       % Satellite position in ECI (equatorial orbit, 0° inclination)
poi_ECI = @(tt) Rz(wE*tt) * poi_ecef;                       % POI in ECI (Earth rotates with wE)
v_sat_ECI = @(tt) w * r * [-sin(w*tt); cos(w*tt); 0];       % Satellite velocity in ECI

% Inertia of satellite (cube + 2 panels)
m_c = 500;                              % Cube mass
L = 2;                                  % Cube sides
I_c = (1/6)*m_c*L^2 * eye(3);           % Main body: cube 2x2x2 m
```

```
m_p = 50;                                    % Panel mass
ax = 0.1; by = 4; cz = 2;                    % Panel dimensions
Ipx = (1/12)*m_p*(by^2 + cz^2);
Ipy = (1/12)*m_p*(ax^2 + cz^2);
Ipz = (1/12)*m_p*(ax^2 + by^2);
Ip_centroid = diag([Ipx, Ipy, Ipz]);

d = 1 + ax/2;                              % 1.05 m from body center
Ip_shift = diag([0, m_p*d^2, m_p*d^2]);    % translation in X -> adds inertia on Y,Z
Ip_body_each = Ip_centroid + Ip_shift;

I_total = I_c + 2*Ip_body_each;            % approx ~ diag(500,477,577) kg m^2
I = I_total;
Iinv = diag(1./diag(I));

% Control parameters
Kp = 0.6;
Kd = 12.0;
Ki = 0.005;

tau_max = 0.8;                             % [N·m]
tau_dist = [0.01; -0.02; 0.005];           % [N·m]
int_err = [0;0;0];

% Initial attitude
q = [1;0;0;0]; q = q/ norm(q);             % quaternion [qw;qx;qy;qz]
omega = [0; 0; 0.05];                      % rad/s
omega_d = [0;0;0];                         % desired angular rate

% === Figure setup ===
fig = figure('Color','w','Name','ADCS POI Tracking','NumberTitle','off');
ax = gca; hold(ax,'on'); grid(ax,'on'); axis(ax,'equal')
xlabel('X_{ECI} [km]'); ylabel('Y_{ECI} [km]'); zlabel('Z_{ECI} [km]');
title('+Z_{body} following POI (Quito)');

earth_T = hgtransform('Parent', ax);

[Ns,~] = deal(50);
[xe, ye, ze] = sphere(Ns);
earth_surf = surf(Re*xe/1e3, Re*ye/1e3, Re*ze/1e3, ...
                  'EdgeColor','none', 'FaceAlpha',0.7, 'Parent', earth_T);
colormap(parula);

theta = linspace(0, 2*pi, 400);
plot3(ax, (r/1e3)*cos(theta), (r/1e3)*sin(theta), 0*theta, 'k--', 'LineWidth',1);

sat_h = plot3(ax, 0,0,0, 'ro', 'MarkerFaceColor','r', 'MarkerSize',6);
poi_h = plot3(ax, 0,0,0, 'bs', 'MarkerFaceColor','b', 'MarkerSize',6);
los_h = plot3(ax, [0 0],[0 0],[0 0], 'r-', 'LineWidth',1.5);
```

```
L_brf = 400;
brf_x = quiver3(ax, 0,0,0, 0,0,0, 'LineWidth',1.5, 'MaxHeadSize',1.5);
brf_y = quiver3(ax, 0,0,0, 0,0,0, 'LineWidth',1.5, 'MaxHeadSize',1.5);
brf_z = quiver3(ax, 0,0,0, 0,0,0, 'LineWidth',1.5, 'MaxHeadSize',1.5);

legend({'Earth','Orbit','Satellite','POI','Sat→POI Vector'}, 'Location','bestoutside');
view(35,25);
xlim(ax,[-1.2*r, 1.2*r]/1e3); ylim(ax,[-1.2*r, 1.2*r]/1e3); zlim(ax,[-1.2*r, 1.2*r]/1e3);

% VIDEO RECORDING SETUP
outputPath = 'C:\Users\Víctor\Desktop\Máster\1º Cuatri\Applied Subsystems Design\Assignments\Final Project\simulation_ADCS.mp4';
v = VideoWriter(outputPath, 'MPEG-4');
v.FrameRate = 30;    % playback fps
open(v);

% we want ~10 s total duration => 10*30 = 300 frames
% number of steps in sim = numel(t)
frameSkip = ceil(numel(t)/300);

% Simulation
err_hist = zeros(3, numel(t));
tau_hist = zeros(3, numel(t));

for k = 1:numel(t)
    tk = t(k);

    % Positions and velocities
    rs = r_sat_ECI(tk);
    rp = poi_ECI(tk);
    vs = v_sat_ECI(tk);

    % Earth rotation (visual)
    set(earth_T, 'Matrix', makehgtform('zrotate', wE*tk));

    % [Sat → POI] vector and desired +Z axis
    los = (rp - rs);  if norm(los) < eps, los = [0;0;1]; end
    z_d = los / norm(los);

    % Coherent yaw: x_d aligned with projected velocity
    v_par  = (dot(vs, z_d)) * z_d;
    v_perp = vs - v_par;
    if norm(v_perp) < 1e-6
        up_ref = [0;0;1]; if abs(dot(z_d, up_ref)) > 0.97, up_ref = [1;0;0]; end
        x_d = cross(up_ref, z_d); x_d = x_d / norm(x_d);
    else
        x_d = v_perp / norm(v_perp);
    end
```

```
    y_d = cross(z_d, x_d);
    R_d = [x_d, y_d, z_d];
    q_d = rotm2quat(R_d); q_d = q_d / norm(q_d);

    % Feedforward for desired rate
    rs_n = r_sat_ECI(tk+dt);
    rp_n = poi_ECI(tk+dt);
    vs_n = v_sat_ECI(tk+dt);
    los_n = rp_n - rs_n; if norm(los_n) < eps, los_n = [0;0;1]; end
    z_d_n = los_n / norm(los_n);
    v_par_n  = (dot(vs_n, z_d_n)) * z_d_n;
    v_perp_n = vs_n - v_par_n;
    if norm(v_perp_n) < 1e-6
        up_ref = [0;0;1]; if abs(dot(z_d_n, up_ref)) > 0.97, up_ref = [1;0;0]; end
        x_d_n = cross(up_ref, z_d_n); x_d_n = x_d_n / norm(x_d_n);
    else
        x_d_n = v_perp_n / norm(v_perp_n);
    end
    y_d_n = cross(z_d_n, x_d_n);
    R_d_n = [x_d_n, y_d_n, z_d_n];
    q_d_n = rotm2quat(R_d_n); q_d_n = q_d_n / norm(q_d_n);

    q_rel = quatMultiply(q_d_n, quatConj(q_d));
    if q_rel(1) < 0, q_rel = -q_rel; end
    omega_d = (2/dt) * q_rel(2:4);

    % Attitude error
    q_e = quatMultiply(q_d, quatConj(q)); q_e = q_e / norm(q_e);
    if q_e(1) < 0, q_e = -q_e; end
    e = q_e(2:4);

    % Control: P(e) - D(omega - omega_d) + I(e)
    int_err = int_err + e*dt;
    rate_err = omega - omega_d;
    tau_c = Kp*e - Kd*rate_err + Ki*int_err;

    % Saturation + anti-windup
    tau_c = max(min(tau_c, tau_max), -tau_max);
    for i=1:3
        if abs(tau_c(i)) >= tau_max*0.999
            int_err(i) = int_err(i) - e(i)*dt*0.5;
        end
    end

    % Rotational dynamics
    omega_dot = Iinv * (tau_c + tau_dist - cross(omega, I*omega));
    omega = omega + omega_dot*dt;

    % Quaternion kinematics
    q_dot = 0.5 * [  0      -omega';
                    omega   -skew(omega) ] * q;
```

```
    q = q + q_dot*dt; q = q / norm(q);

    % --- Graphics update ---
    set(sat_h, 'XData', rs(1)/1e3, 'YData', rs(2)/1e3, 'ZData', rs(3)/1e3);
    set(poi_h, 'XData', rp(1)/1e3, 'YData', rp(2)/1e3, 'ZData', rp(3)/1e3);
    set(los_h, 'XData', [rs(1) rp(1)]/1e3, ...
               'YData', [rs(2) rp(2)]/1e3, ...
               'ZData', [rs(3) rp(3)]/1e3);

    R_cb = quat2rotm(q); ex = R_cb(:,1); ey = R_cb(:,2); ez = R_cb(:,3);
    x0 = rs(1)/1e3; y0 = rs(2)/1e3; z0 = rs(3)/1e3;
    set(brf_x, 'XData', x0, 'YData', y0, 'ZData', z0, ...
               'UData', L_brf*ex(1), 'VData', L_brf*ex(2), 'WData', L_brf*ex(3));
    set(brf_y, 'XData', x0, 'YData', y0, 'ZData', z0, ...
               'UData', L_brf*ey(1), 'VData', L_brf*ey(2), 'WData', L_brf*ey(3));
    set(brf_z, 'XData', x0, 'YData', y0, 'ZData', z0, ...
               'UData', L_brf*ez(1), 'VData', L_brf*ez(2), 'WData', L_brf*ez(3));

    err_hist(:,k) = e;
    tau_hist(:,k) = tau_c;

    % draw frame occasionally (skip frames to make 10s video)
    if mod(k, frameSkip) == 0
        frame = getframe(fig);
        writeVideo(v, frame);
    end
end

% END VIDEO RECORDING
close(v);
disp(' Simulation video saved successfully at:');
disp(outputPath);

% Helper functions
function S = skew(v)
S = [  0    -v(3)  v(2);
      v(3)    0   -v(1);
     -v(2)  v(1)    0  ];
end

function qout = quatMultiply(q1, q2)
w1=q1(1); x1=q1(2); y1=q1(3); z1=q1(4);
w2=q2(1); x2=q2(2); y2=q2(3); z2=q2(4);
qout = [ w1*w2 - x1*x2 - y1*y2 - z1*z2;
         w1*x2 + x1*w2 + y1*z2 - z1*y2;
         w1*y2 - x1*z2 + y1*w2 + z1*x2;
         w1*z2 + x1*y2 - y1*x2 + z1*w2 ];
end
```

```
function qconj = quatConj(q)
qconj = [q(1); -q(2); -q(3); -q(4)];
end

function q = rotm2quat(R)
tr = trace(R);
if tr > 0
    S = sqrt(tr+1.0)*2;
    qw = 0.25*S;
    qx = (R(3,2)-R(2,3))/S;
    qy = (R(1,3)-R(3,1))/S;
    qz = (R(2,1)-R(1,2))/S;
else
    if (R(1,1) > R(2,2)) && (R(1,1) > R(3,3))
        S = sqrt(1.0 + R(1,1) - R(2,2) - R(3,3))*2;
        qw = (R(3,2)-R(2,3))/S;
        qx = 0.25*S;
        qy = (R(1,2)+R(2,1))/S;
        qz = (R(1,3)+R(3,1))/S;
    elseif (R(2,2) > R(3,3))
        S = sqrt(1.0 - R(1,1) + R(2,2) - R(3,3))*2;
        qw = (R(1,3)-R(3,1))/S;
        qx = (R(1,2)+R(2,1))/S;
        qy = 0.25*S;
        qz = (R(2,3)+R(3,2))/S;
    else
        S = sqrt(1.0 - R(1,1) - R(2,2) + R(3,3))*2;
        qw = (R(2,1)-R(1,2))/S;
        qx = (R(1,3)+R(3,1))/S;
        qy = (R(2,3)+R(3,2))/S;
        qz = 0.25*S;
    end
end
q = [qw; qx; qy; qz];
if q(1) < 0, q = -q; end
q = q / norm(q);
end

function R = quat2rotm(q)
qw=q(1); qx=q(2); qy=q(3); qz=q(4);
R = [1-2*(qy^2+qz^2), 2*(qx*qy - qz*qw), 2*(qx*qz + qy*qw);
     2*(qx*qy + qz*qw), 1-2*(qx^2+qz^2), 2*(qy*qz - qx*qw);
     2*(qx*qz - qy*qz), 2*(qy*qz + qx*qw), 1-2*(qx^2+qy^2)];
end
```