**Problem I**

1. Given Yaw = 30°, Pitch = 30°, and Roll = 20°

This is a 3-2-1 rotation because the standard notation is that roll is a rotation about , pitch is a rotation about , and yaw is a rotation about . Also, it is known that “the standard yaw-pitch-roll (ψ; θ; φ) angles are the (3-2-1) set of Euler angles” (Junkins and Schaub, p. 87).

1. The DCM for a 3-2-1 rotation is known and shown in Figure 1.

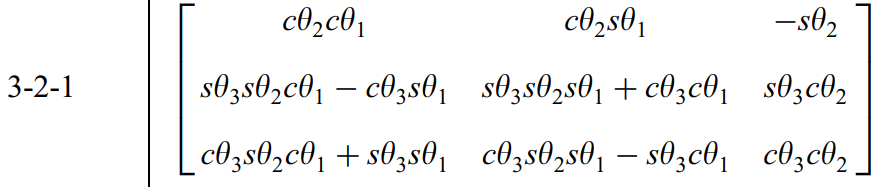


Figure 1: 3-2-1 rotation matrix. Source: Junkins and Schaub, Appendix B

where is the yaw, is the pitch, and is the roll. Plugging the numbers in yields the following DCM:

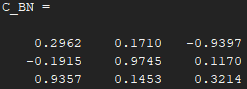


Figure 2: DCM that describes the attitude of the body WRT the inertial frame

1. A math equations on a white background

   Description automatically generatedConverting this DCM into a quaternion using Sheppard’s algorithm:

Figure 3: Conversion equations from DCM to Quaternion using Sheppard’s method.  
Source: Junkins and Schaub, p. 105

Plugging in the DCM from part 2 yields

1. The angular velocity was given in the *SensorData.csv* file, and the data was extracted using the provided pseudocode. This data is plotted blow:

Figure 4: Angular velocity components plotted over time

1. A graph of earth-mars transfer

   Description automatically generatedThe frequency of the sensor input can be found by looking at the values of the time variable, *t*, in Matlab. It shows that data is, generally, being taken every 0.01 seconds. However, it is not always at equal time intervals; Sometimes the timing is off by 0.001 seconds. Below is a snippet of some of the time values of *t* that shows this:

|  |  |
| --- | --- |
| Index | Value |
| 162 | 1.6100 |
| 163 | 1.6200 |
| 164 | 1.6310 |
| 165 | 1.6410 |
| 166 | 1.6500 |
| 167 | 1.6600 |

**Problem 1 Code**

% This is the code for Aersp 450, HW 4, Question I

% Made by Nicholas Luis (PSU ID 930841391)

clc

clear

%% Provided Skeleton Code

T = readtable('SensorData.csv');

wx = T.wx;

wy = T.wy;

wz = T.wz;

% Step 1: Convert the time strings into datetime format

timeData = datetime(T.time, 'InputFormat', 'yyyy-MM-dd''T''HH:mm:ss.SSS''Z''',...

'TimeZone', 'UTC');

% Step 2: Calculate time differences from the first time in the list

timeDifferences = timeData - timeData(1);

% Step 3: Convert the differences to seconds

t = seconds(timeDifferences);

%%

% Given euler angle rotations

theta1 = 30; % yaw

theta2 = 30; % pitch

theta3 = 20; % theta3l

% DCM rotation based on a 3-2-1 rotation

C\_BN = [cosd(theta2)\*cosd(theta1), cosd(theta2)\*sind(theta1), -sind(theta2);

sind(theta3)\*sind(theta2)\*cosd(theta1)-cosd(theta3)\*sind(theta1), sind(theta3)\*sind(theta2)\*sind(theta1)+cosd(theta3)\*cosd(theta1), sind(theta3)\*cosd(theta2);

cosd(theta3)\*sind(theta2)\*cosd(theta1)+sind(theta3)\*sind(theta1), cosd(theta3)\*sind(theta2)\*sind(theta1)-sind(theta3)\*cosd(theta1), cosd(theta3)\*cosd(theta2);

]

DCMcheck(C\_BN);

% Quaternion based on the DCM matrix using Sheppard Algo

Beta = SheppardAlgo(C\_BN)

% Plotting the angular velocities as a function of time

figure(1)

hold on

plot(t, wx, LineWidth=2)

plot(t, wy, LineWidth=2)

plot(t, wz, LineWidth=2) % Only plotting half of the transfer orbit

title('Earth-Mars Transfer')

xlabel("Time (s)")

ylabel("Angular Velocity (deg / s)")

legend('wx', 'wy', 'wz')

hold off

exportgraphics(gca,"HW4\_Problem1\_AngVeloPlots.jpg");

%% Functions

function isDCM = DCMcheck(A)

% This function checks if a matrix is a DCM

isDCM = true;

% Checks if the rows and columns are unit vectors

for i = 1:3

if (round(norm(A(i,:)), 10) ~= 1)

isDCM = false;

fprintf("The DCM is not valid!");

return;

end

if (round(norm(A(:,i)), 10) ~= 1)

isDCM = false;

fprintf("The DCM is not valid!");

return;

end

end

% Checks if the DCM is orthonormal

if (round(A\*A',10) ~= eye(3))

isDCM = false;

fprintf("The DCM is not valid!");

return;

end

end

function BetaVec = SheppardAlgo(C)

% This funciton inputs some matrix C and does Sheppard's algorithm to

% compute the quaternion

% Equation 3.95

B0 = sqrt(0.25\*(1+trace(C)));

B1 = sqrt(0.25\*(1+2\*C(1,1)-trace(C)));

B2 = sqrt(0.25\*(1+2\*C(2,2)-trace(C)));

B3 = sqrt(0.25\*(1+2\*C(3,3)-trace(C)));

BetaVec = [B0; B1; B2; B3];

biggestB = max(BetaVec);

% Equation 3.96

if (B0 == biggestB)

BetaVec(1) = 0.25\*(C(2,3)-C(3,2))/B0;

BetaVec(2) = 0.25\*(C(3,1)-C(1,3))/B0;

BetaVec(3) = 0.25\*(C(1,2)-C(2,1))/B0;

return;

elseif (B1 == biggestB)

BetaVec(0) = 0.25\*(C(2,3)-C(3,2))/B1;

BetaVec(2) = 0.25\*(C(1,2)+C(2,1))/B1;

BetaVec(3) = 0.25\*(C(3,1)+C(1,3))/B1;

return;

elseif (B2 == biggestB)

BetaVec(0) = 0.25\*(C(3,1)-C(1,3))/B2;

BetaVec(1) = 0.25\*(C(1,2)+C(2,1))/B2;

BetaVec(3) = 0.25\*(C(2,3)+C(3,2))/B2;

return;

else

BetaVec(0) = 0.25\*(C(1,2)-C(2,1))/B3;

BetaVec(1) = 0.25\*(C(3,1)+C(1,3))/B3;

BetaVec(2) = 0.25\*(C(2,3)+C(3,2))/B3;

return;

end

end

. .

**Problem II**

1. Numerically propagating the DCM

This was done using the following equation:

At every time step, the rate of change in the DCM, , was calculated and multiplied by the timestep to get the absolute change of the DCM. Then, this change was added to the DCM, , to get the new DCM at that timestep. This process repeated for every timestep.

1. Analytically propagating the DCM

This was done using the following equation:

At every time step, the rate of change in the DCM, , was calculated and multiplied by the timestep to get the absolute change of the DCM. Then, this change was added to the DCM, , to get the new DCM at that timestep. This process repeated for every timestep.

References:

Junkins, J., Schaub, H., *Analytical Mechanics of Space Systems, Fourth Edition (AIAA Education Series)*, AIAA American Institute of Aeronautics & Ast, April 2018