Christopher Lum
lum@uw.edu

Lecture 03b
Quaternions

The YouTube video entitled 'Computing Euler Angles: The Euler Kinematical Equations and Poisson's Kinematical Equations' that covers this lecture is located at https://youtu.be/9GZjtfYOXao.

Outline

-References
-Quaternions Introduction
-Quaternion Mathematics
    -Norm
    -Conjugate
    -Inverse
    -Multiplication
-Attitude Representation with Quaternions
    -Direction Cosine Matrix (DCM) to Quaternion (dcm2quat)
    -Quaternion to Direction Cosine Matrix (DCM) (quat2dcm)
    -Angle to Quaternion (angle2quat)
    -Quaternion to Angle (quat2angle)
-Attitude Kinematics
-Simulink

References

https://en.wikipedia.org/wiki/Gimbal_lock
http://www.tu-berlin.de/fileadmin/fg169/miscellaneous/Quaternions.pdf
https://en.wikipedia.org/wiki/Quaternions_and_spatial_rotation
https://www.mathworks.com/discovery/quaternion.html

Quaternions Introduction

Gimbal lock is a problem with Euler angles. This leads to investigation of quaternions.

Instead of using 3 Euler angles to represent the relative attitude of two frames, an alternative formulation to keep track of the attitude of a rigid body is to use quaternions.

A quaternion is an in a 4x1 vector where elements consists of a scalar part s and a vector part $\overline{v}$.

$$q = \begin{pmatrix} s \\ \overline{v} \end{pmatrix} = \begin{pmatrix} s \\ v_x \\ v_y \\ v_z \end{pmatrix} = \begin{pmatrix} q_s \\ q_x \\ q_y \\ q_z \end{pmatrix} = \begin{pmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{pmatrix} \qquad \textbf{(Eq.1.1)}$$

where s = scalar part of quaternion
$\overline{v}$ = vector part of quaternion

The quaternion representation is based on Euler's rotational theorem which states that the relative orientation of two coordinate systems can be described by only one rotation about a fixed axis (see 'Euler Angles and the Euler Rotation Sequence' at 49:57 - https://youtu.be/GJBc6z6p0KQ?t=2997).

<Revisit previous example showing unique rotation>

We can use a quaternion embed the information about the axis of rotation and the rotation/transformation angle using

$$q = \begin{pmatrix} q_s \\ q_x \\ q_y \\ q_z \end{pmatrix} = \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) \\ \overline{e}\sin\left(\frac{\theta}{2}\right) \end{pmatrix} \qquad \textbf{(Eq.1.2)}$$

where θ = rotation/transformation angle
$\overline{e}$ = normalized rotational axis (vector)

Note that in this discussion, q refers to the quaternion, not the pitch rate. Furthermore, θ is the transformation/rotation angle of the rotation, not the pitch Euler angle.

Note that the notation of quaternions may differ depending on the context or programming language. The aforementioned convention is used by the Matlab Aerospace Toolbox.

Example

Consider the DCM we considered in the Euler Angle lecture

$$\phi = 25\,\frac{\pi}{180}$$
$$\theta = 130\,\frac{\pi}{180}$$

$$\psi = 70\,\frac{\pi}{180}$$

We earlier determined that the axis of rotation for this rotation was given by

$$\overline{n} = \begin{pmatrix} -0.4845 \\ 0.8706 \\ 0.0851 \end{pmatrix} \qquad \text{(axis of rotation obtained from eigenvalue/eigenvector analysis)}$$

We can normalize to compute $\overline{e}$

$$\overline{e} = \frac{\overline{n}}{||\overline{n}||} = \begin{pmatrix} -0.4845 \\ 0.8706 \\ 0.0851 \end{pmatrix} \qquad (\overline{n} \text{ was already a unit vector})$$

In the previous example, we somewhat blindly rotated through angles using Rodrigues' Rotation Formula (using a left handed rotation μ) until we achieved good alignment between the two frames and we determined that the appropriate transformation/rotation angle was

$$\mu \approx 233.5\,\frac{\pi}{180} \qquad \text{(this was a left handed rotation, it is approximate)}$$

So to convert to a right handed rotation we have

$$\theta = -\mu = -233.5\,\frac{\pi}{180} = 126.5\,\frac{\pi}{180}$$

We can calculate the quaternion for this rotation

$$s = \cos\left(\frac{126.5\,\frac{\pi}{180}}{2}\right) = 0.4501$$

$$\overline{v} = \begin{pmatrix} -0.4845 \\ 0.8706 \\ 0.0851 \end{pmatrix} \sin\left(\frac{126.5\,\frac{\pi}{180}}{2}\right) = \begin{pmatrix} -0.4327 \\ 0.7774 \\ 0.0760 \end{pmatrix}$$

$$q = \begin{pmatrix} 0.4501 \\ -0.4327 \\ 0.7774 \\ 0.0760 \end{pmatrix}$$

Note that q represents rotating about $\overline{e}$ through $126.5\,\frac{\pi}{180}$. This may not exactly correspond to the Euler angle sequence of $\phi = 25\,\frac{\pi}{180}$, $\theta = 130\,\frac{\pi}{180}$, and $\psi = 70\,\frac{\pi}{180}$ because we simply guessed at the total transformation/rotation angle of $126.5\,\frac{\pi}{180}$.

We will revisit this example later when we are able to compute the exact transformation/rotation angle θ

Quaternion Mathematics

There are many mathematical operators that are defined for quaternions

Norm

The norm of a quaternion is given by

$$| q | = \sqrt{{q_s}^2 + {q_x}^2 + {q_y}^2 + {q_z}^2} \qquad \textbf{(Eq.2.1)}$$

A quaternion with norm $| q | = 1$ is called a **unit quaternion**. All quaternions for attitude representation are unit quaternions.

One can normalize a quaternion by dividing by its norm

$$|| q || = \frac{q}{|q|} \qquad \textbf{(Eq.2.2)}$$

Conjugate

The conjugate quaternion has an inverted vector part

$$q^* = \begin{pmatrix} q_s \\ -q_x \\ -q_v \\ -q_z \end{pmatrix} \qquad \textbf{(Eq.2.3)}$$

Inverse

The inverse of a quaternion is obtained by normalizing its conjugate

$$q^{-1} = \frac{q^*}{|q|} \qquad \textbf{(Eq.2.4)}$$

Multiplication

The product, q, of two quaternions, q_1 and q_2, is defined by

$$q = q_1 \otimes q_2 \triangleq \begin{pmatrix} s_1 s_2 - < \overline{v}_1, \overline{v}_2 > \\ s_1 \overline{v}_2 + s_2 \overline{v}_1 + \overline{v}_1 \times \overline{v}_2 \end{pmatrix}$$

where $< \overline{v}_1, \overline{v}_2 >$ dot product of $\overline{v}_1$ and $\overline{v}_2$

Attitude Representation with Quaternions

We now explore how to use quaternions to represent the attitude of a rigid body.

For the following section, consider the DCM

$$C_{b/v} = \begin{pmatrix} C_{11} & C_{12} & C_{13} \\ C_{23} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{pmatrix} \qquad \textbf{(Eq.3.1)}$$

Note that some references (such as http://www.tu-berlin.de/fileadmin/fg169/miscellaneous/Quaternions.pdf) develops equations based on $C_{v/b}$ instead of $C_{b/v}$ as we present here.

Direction Cosine Matrix (DCM) to Quaternion (dcm2quat)

We can compute the quaternion associated with a given direction cosine matrix using a multi-step process. First, we calculate the quantity

$$\tilde{q} = \begin{pmatrix} \tilde{q}_s \\ \tilde{q}_x \\ \tilde{q}_y \\ \tilde{q}_z \end{pmatrix} = \begin{pmatrix} \sqrt{\frac{1}{4} \times (1 + C_{11} + C_{22} + C_{33}} \\ \sqrt{\frac{1}{4} \times (1 + C_{11} - C_{22} - C_{33}} \\ \sqrt{\frac{1}{4} \times (1 - C_{11} + C_{22} - C_{33}} \\ \sqrt{\frac{1}{4} \times (1 - C_{11} - C_{22} + C_{33}} \end{pmatrix} \qquad \textbf{(Eq.3.2)}$$

Next, we find the maximum entry of $\tilde{q}$

$$\tilde{q}_{\max} = \max(\tilde{q}_s, \tilde{q}_x, \tilde{q}_y, \tilde{q}_z) \qquad \textbf{(Eq.3.3)}$$

Depending on which entry is the maximum entry, the others entries are re-calculated as shown below. The first columns lists all values from Eq.3.2, while the columns 2 through 5 list the re-calculated values for all other elements

maximum	q_s	q_x	q_y	q_z
$\tilde{q}_s = \tilde{q}_{\max}$	$\tilde{q}_s$	$\frac{C_{23}-C_{32}}{4\tilde{q}_s}$	$\frac{C_{31}-C_{13}}{4\tilde{q}_s}$	$\frac{C_{12}-C_{21}}{4\tilde{q}_s}$
$\tilde{q}_x = \tilde{q}_{\max}$	$\frac{C_{23}-C_{32}}{4\tilde{q}_x}$	$\tilde{q}_x$	$\frac{C_{12}-C_{21}}{4\tilde{q}_x}$	$\frac{C_{31}-C_{13}}{4\tilde{q}_x}$
$\tilde{q}_y = \tilde{q}_{\max}$	$\frac{C_{31}-C_{13}}{4\tilde{q}_y}$	$\frac{C_{12}+C_{21}}{4\tilde{q}_y}$	$\tilde{q}_y$	$\frac{C_{23}+C_{32}}{4\tilde{q}_y}$
$\tilde{q}_z = \tilde{q}_{\max}$	$\frac{C_{12}-C_{21}}{4\tilde{q}_z}$	$\frac{C_{31}+C_{13}}{4\tilde{q}_z}$	$\frac{C_{23}+C_{32}}{4\tilde{q}_z}$	$\tilde{q}_z$

(Eq.3.4)

Matlab provides the function 'dcm2quat' to perform this operation.

Quaternion to Direction Cosine Matrix (DCM) (quat2dcm)

If we know the quaternion, we can calculate the direction cosine matrix using

$$C_{b/v} = \begin{pmatrix} q_s^2 + q_x^2 - q_y^2 - q_z^2 & 2(q_x q_y + q_z q_s) & 2(q_x q_z - q_y q_s) \\ 2(q_x q_y - q_z q_s) & q_s^2 - q_x^2 + q_y^2 - q_z^2 & 2(q_y q_z + q_x q_s) \\ 2(q_x q_z + q_y q_s) & 2(q_y q_z - q_x q_s) & q_s^2 - q_x^2 - q_y^2 + q_z^2 \end{pmatrix}$$

Matlab provides the function 'quat2dcm' to perform this operation.

Angle to Quaternion (angle2quat)

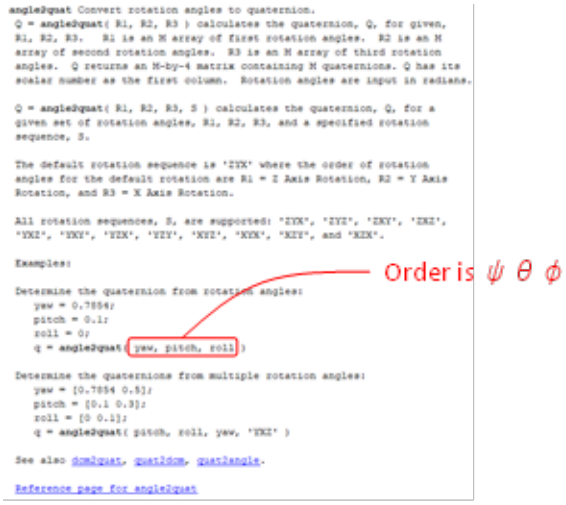
We can easily convert a set of Euler angles to a DCM (this was covered in a previous lecture and is repeated here for convenience)

$$C_{b/v} = \begin{pmatrix} \cos(\theta)\cos(\psi) & \cos(\theta)\sin(\psi) & -\sin(\theta) \\ -\cos(\phi)\sin(\psi) + \sin(\phi)\sin(\theta)\cos(\psi) & \cos(\phi)\cos(\psi) + \sin(\phi)\sin(\theta)\sin(\psi) & \sin(\phi)\cos(\theta) \\ \sin(\phi)\sin(\psi) + \cos(\phi)\sin(\theta)\cos(\psi) & -\sin(\phi)\cos(\psi) + \cos(\phi)\sin(\theta)\sin(\psi) & \cos(\phi)\cos(\theta) \end{pmatrix}$$

Once we have the DCM, we can use the algorithm outlined in Eq.3.2 - Eq.3.4 to convert from a DCM to a quaternion.

Matlab provides the function 'angle2quat' to perform this operation.

WARNING! Ensure you read the documentation and understand the order of input/outputs from the various Matlab functions



Quaternion to Angle (quat2angle)

We can use the previous relationship (quat2dcm) to convert the quaternion to a DCM

Once we have the DCM, we can use the same relationship we derived in a previous lecture to convert the DCM to Euler angles (repeated here for convenience)

$$\phi = \text{atan2}(c_{23}, c_{33}) \qquad \textbf{(Eq.1.2)}$$

$$\theta = -\sin^{-1}(c_{13}) \qquad \textbf{(Eq.1.3)}$$

$$\psi = \text{atan2}(c_{12}, c_{11}) \qquad \textbf{(Eq.1.4)}$$

Matlab provides the function 'quat2angle' to perform this operation.

WARNING! Ensure you read the documentation and understand the order of input/outputs from the various Matlab functions

```
quat2angle Convert quaternion to rotation angles.
 [R1, R2, R3] = quat2angle( Q ) calculates the set of
 rotation angles, R1, R2, R3, for a given quaternion, Q.  Input Q is an
 M-by-4 matrix containing M quaternions.  R1 returns an M array of
 first rotation angles.  R2 returns an M array of second rotation
 angles.  R3 returns an M array of third rotation angles. Each element
 of Q must be a real number.  Additionally, Q has its scalar number as
 the first column. Rotation angles are output in radians.

 [R1, R2, R3] = quat2angle( Q, S ) calculates the set of rotation
 angles, R1, R2, R3, for a given quaternion, Q, and a
 specified rotation sequence, S.

 The default rotation sequence is 'ZYX' where the order of rotation
 angles for the default rotation are R1 = Z Axis Rotation, R2 = Y Axis
 Rotation, and R3 = X Axis Rotation.

 All rotation sequences, S, are supported: 'ZYX', 'ZYZ', 'ZXY', 'ZXZ',
 'YXZ', 'YXY', 'YZX', 'YZY', 'XYZ', 'XYX', 'XZY', and 'XZX'.

 Examples:                                          Order is ψ θ ϕ

 Determine the rotation angles from q = [1 0 1 0]:
    [yaw, pitch, roll] = quat2angle( [1 0 1 0] )

 Determine the rotation angles from multiple quaternions:
    q = [1 0 1 0; 1 0.5 0.3 0.1];
    [pitch, roll, yaw] = quat2angle( q, 'YXZ' )

 See also angle2dcm, dcm2angle, dcm2quat, angle2quat, quat2dcm.

 Reference page for quat2angle
```

Example

We can now revisit the previous example.

1. Use 'angle2quat' to get the quaternion associated with the Euler angles.
2. Extract the scalar component of the quaternion and compute the associated transformation/rotation angle using the definition of

$$q_s = \cos\left(\frac{\theta}{2}\right) \quad \Rightarrow \quad \theta = 2\cos^{-1}(q_2)$$

3. Extract the vector component of the quaternion, $\overline{v}$, and compute the associated axis of rotation using the definition of

$$\overline{v} = \overline{e}\sin\left(\frac{\theta}{2}\right) \quad \Rightarrow \quad \overline{e} = \overline{v}\left(\frac{1}{\sin(\theta/2)}\right)$$

```
%Compute the actual quaternion associated with the Euler rotation
q_matlab = angle2quat(psi, theta, phi); %Note order of inputs!

%Compute the exact value of the transformation/rotation angle
qs_matlab = q_matlab(1);
theta_exact = 2*acos(qs_matlab);
rad2deg(theta_exact)

%Compare the axis of rotation (e vs. eigenvector)
v_matlab = q_matlab(2:4);
e_matlab = v_matlab*(1/sin(theta_exact/2))

%cross product between the axis of rotation from the quaternion and the
%axis of rotation from the eigenvector analysis should be the same
sum(abs(cross(e_matlab, n)))
```

Evaluates to effectively 0

This yields $\theta = 126.449 \frac{\pi}{180}$ which is fairly close to our estimated value.

The resulting axis of rotation from the quaternion is

$$\bar{e} = \begin{pmatrix} -0.4845 \\ 0.8706 \\ 0.0851 \end{pmatrix}$$

This exactly agrees with our previous axis of rotation obtained from eigenvalue/eigenvector analysis of the DCM.

Attitude Kinematics

We can now formulate an alternative method to relate the body rotation rate (measured by gyros) to the attitude of the vehicle.

$$\dot{q} = \frac{1}{2} \Omega q$$

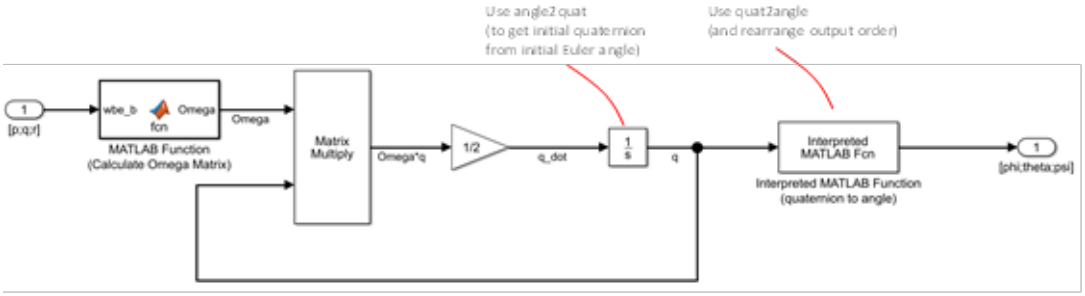
where $$\Omega = \begin{pmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & \omega_z & -\omega_y \\ \omega_y & -\omega_z & 0 & \omega_x \\ \omega_z & \omega_y & -\omega_x & 0 \end{pmatrix}$$

Note that we use notation of ω_x, ω_y, ω_z instead of p, q, r to avoid the naming collision of q = quaternion vs. q = pitch rate.

Simulink

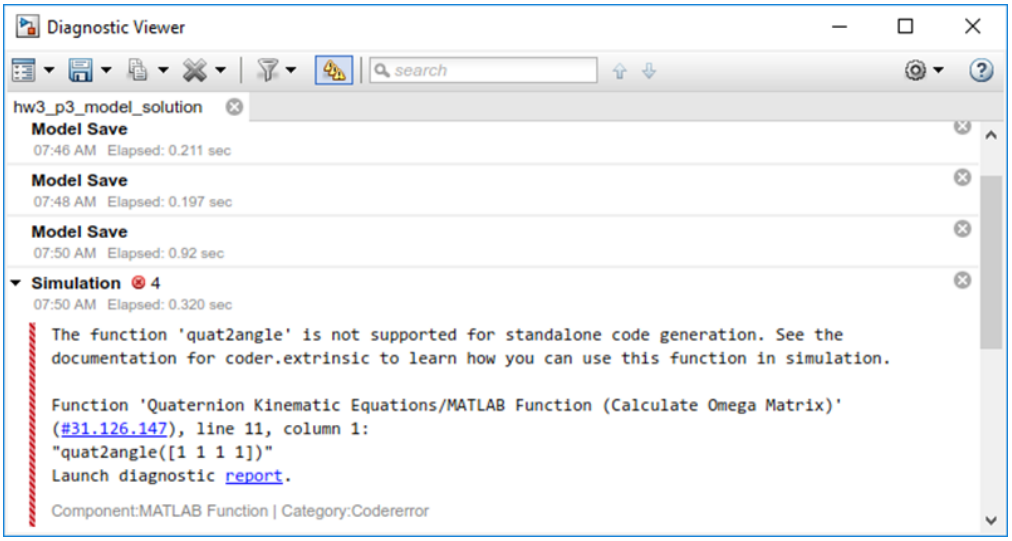
As we saw previously, Matlab provides many functions (dcm2quat, quat2angle, etc.) to perform quaternion mathematics. The next logical step is to integrate these into Simulink models. This requires some discussion.

Armed with this knowledge, we can sketch a Simulink diagram that can maintain Euler angles using quaternions



As of Matlab R2018b, many of these quaternion related functions are not supported for standalone

code generation. This means that you may encounter issues if you use them inside ‘MATLAB Function’ blocks within a Simulink Model.



The workaround to this is to instead use an ‘Interpreted MATLAB Function’ block which then calls a normal Matlab function which can call quaternion related functions.