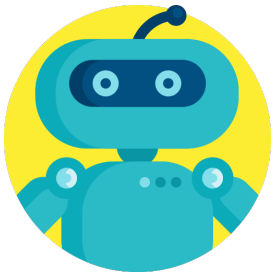


MEC4127F: Introduction to Robotics

Chapter 3: Describing pose

Table of Contents

MEC4127F: Introduction to Robotics....1
Chapter 3: Describing pose....1
3.1 Introduction....2
3.1.1 Representing orientation....2
3.2 Euler angles....3
3.2.1 Rotation sequence....3
3.2.2 Gimbal lock....7
Definition: Gimbal lock....7
3.2.3 Extracting Euler angles....8
3.2.4 Intrinsic and extrinsic rotations....10
Example: Intrinsic and extrinsic rotations....10
3.3 Quaternions....12
3.3.1 Definition....12
3.3.2 Quaternion multiplication....16
3.3.3 Rotating a frame....20
3.3.4 Double cover....21
Example: Double cover....22
3.3.5 Rotating vectors....23
3.4 Exponential coordinates of rotations....24
3.4.1 Definition....24
Definition: Skew-symmetric matrix....24
3.4.2 Exponential coordinates of SO(3)....24
Example: Using Rodrigues' formula....26
3.4.3 Principal axis rotations....27
3.4.4 Matrix logarithms of rotations....28
Example: Matrix logarithm....28
3.4.5 The dot product and cross product....29
Example: Cross product and rotation matrix....30
3.5 Angular velocity....31
3.5.1 Rotation matrix derivative....31
3.5.2 Quaternion derivative....33
3.6 Obtaining orientation from angular velocity....34
3.6.1 Rotation matrix formulation....34
3.6.2 Quaternion formulation....35
Example: Determining quaternion-based orientation from angular velocity....35
3.7 Exponential coordinates in SE(3)....38

3.7.1 Definition....38
3.7.2 Left Jacobian of SO(3)....39
3.7.3 Body twist....39
Example: Transformation matrix from body twist....40

3.1 Introduction

This chapter will largely focus on the interrelation between angular velocity and orientation of different representations. The latter part of this chapter will then develop an exact mapping that can be used to relate pose-rate information (translational and rotational velocity) to pose.

Based on our results in **Chapter 2**, we now have a means of encoding the orientation of our robot body with respect to a reference frame of interest. The rotation matrix, $\mathbf{R}$, is a useful starting point in containing orientation and also plays a pivotal role in mapping vectors between frames. However, our prior knowledge of mathematics means that we are probably more comfortable with visualising rotations using individual angles. In fact, while the rotation matrix requires 9 elements to be fully populated based on the direction cosine formulation of

$$ {}^{W}\mathbf{R}_{B} = \begin{bmatrix} {}^{W}\hat{\mathbf{x}}_{B} & {}^{W}\hat{\mathbf{y}}_{B} & {}^{W}\hat{\mathbf{z}}_{B} \end{bmatrix} = \begin{bmatrix} \cos(\alpha_{xx}) & \cos(\alpha_{yx}) & \cos(\alpha_{zx}) \\ \cos(\alpha_{xy}) & \cos(\alpha_{yy}) & \cos(\alpha_{zy}) \\ \cos(\alpha_{xz}) & \cos(\alpha_{yz}) & \cos(\alpha_{zz}) \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}, $$

we actually only need to know as little as 3 distinct values to define the orientation. This follows from the fact that an orthogonal matrix imposes 6 constraints on the 9-element matrix $\mathbf{R}$; 3 from the unit-norm requirement of

$$ r_{11}^2 + r_{21}^2 + r_{31}^2 = 1, \quad r_{12}^2 + r_{22}^2 + r_{32}^2 = 1, \quad r_{13}^2 + r_{23}^2 + r_{33}^2 = 1, $$

and 3 from the orthogonality condition of

$$ r_{11}r_{12} + r_{21}r_{22} + r_{31}r_{32} = 0, \quad r_{12}r_{13} + r_{22}r_{23} + r_{32}r_{33} = 0, \quad r_{11}r_{13} + r_{21}r_{23} + r_{31}r_{33} = 0. $$

With 9 unknown terms in $\mathbf{R}$ and 6 constraints, this results in 3 free parameters. This result should not be terribly surprising considering that we are working in a three-dimensional space with three axes.

Note that in the special case of planar rotation, as discussed in **Section 2.3.3**, the orientation corresponding to the rotation matrix

$$ \mathbf{R} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}, $$

is uniquely described by the single angle θ.

3.1.1 Representing orientation

There are a number of different ways to represent orientation. While rotation matrices have their uses in performing vector or frame mappings, it is not always obvious what orientation is being described within the 3×3 matrix form. Additionally, when visualising orientation on a time-domain plot, for example to assess the attitude of a spacecraft, vector descriptions of orientation make more sense. We will also see later in this course that vector-based orientation descriptors fit better in state estimation and control design techniques.

We will consider four main methods of representing orientation (directly or indirectly) in this course:

1. Rotation matrices
2. Euler angles
3. Quaternions
4. Exponential coordinates

Notably, each of these approaches has relative strengths and weaknesses. We will see later in this Chapter that Euler angles and exponential coordinates are a useful means of *directly* describing orientation, whereas quaternions and rotation matrices *indirectly* describe orientation and are ways to encode orientation in a functional way to facilitate vector or frame mappings.

3.2 Euler angles

Euler angles are arguably the most common representation used when describing orientation in robotics and can be used to encode/decode orientation information from the rotation matrix. In the case of capturing the frame $\{B\}$ orientation with respect to $\{W\}$, the Euler angle vector is given by

$$^{W}\boldsymbol{\eta}_{B} = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix},$$

which uniquely describes the relative orientation using three angles:

- roll (ϕ),
- pitch (θ),
- yaw (ψ).

These three angles have the following bounds:

- $\{\phi, \psi\} \in [-\pi, \pi]$,
- $\theta \in [-\frac{\pi}{2}, \frac{\pi}{2}]$.

Euler angles describe a particular orientation using three disparate rotations. The basic concept is that we can construct the orientation of one reference frame relative to another by using three sequential, distinct rotations about the principal axes of one of the frames.

3.2.1 Rotation sequence

In the case of the single rigid-body problem, and using our two frames of interest, $\{W\}$ and $\{B\}$, we are going to make three sequential rotations about our *current* reference frame, moving "backwards" from frame $\{W\}$, eventually to frame $\{B\}$. We begin in frame $\{W\}$ with a null rotation matrix of $\mathbf{R} = \mathbf{I}$ (our current reference frame that is attached to the robot body and $\{W\}$ are aligned). With reference to Figure 3.1, we start by first rotating about the current z -axis, $\hat{\mathbf{z}}_W$, with angle ψ (shown in the leftmost image).

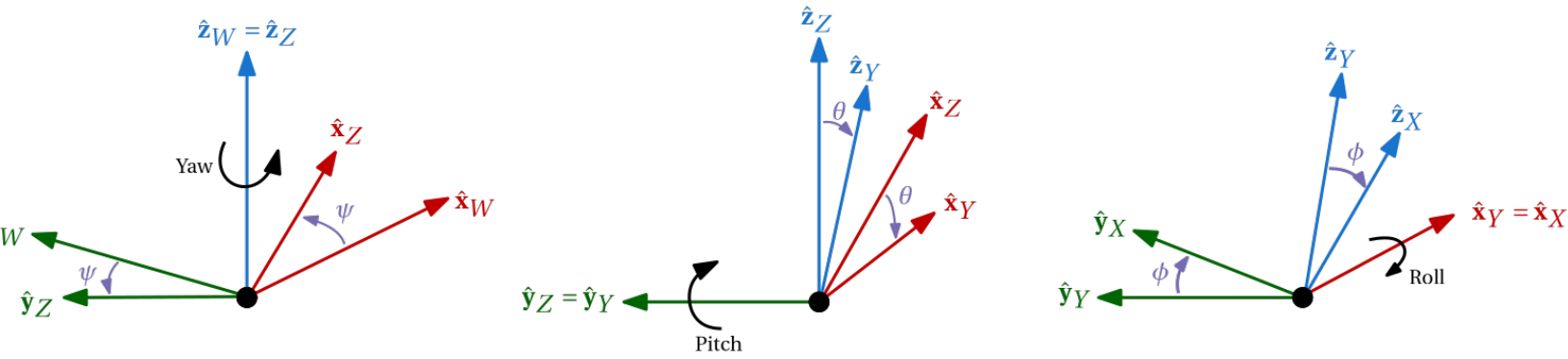


Figure 3.1: Intrinsic Euler ZYX rotation sequence.

The rotation matrix that corresponds to this rotation about $\hat{\mathbf{z}}_W$, rotating from $\{W\}$ to the new frame $\{Z\}$, is

$$ {}^{W}\mathbf{R}_Z = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}. $$

${}^{W}\mathbf{R}_Z$ is known as an elementary rotation and notably only affects vectors in the $\hat{\mathbf{x}}_W$ - $\hat{\mathbf{y}}_W$ plane (the z component is unaffected). Note that we have implicitly introduced a new reference frame, $\{Z\}$, from this rotation, with the above equation describing the orientation of $\{Z\}$ relative to $\{W\}$. Following this, we then rotate about the y-axis of $\{Z\}$, $\hat{\mathbf{y}}_Z$, with angle θ. The incremental rotation matrix that rotates $\{Z\}$ to $\{Y\}$ is

$$ {}^{Z}\mathbf{R}_Y = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}. $$

We have again introduced a new reference frame, $\{Y\}$, using an elementary rotation. The final step is to rotate about $\hat{\mathbf{x}}_Y$ with angle ϕ. The corresponding incremental rotational mapping from $\{Y\}$ to $\{X\} = \{B\}$ (our body-frame of interest) is

$$ {}^{Y}\mathbf{R}_B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix}. $$

We can now stitch together the sequential rotations (keeping an eye on the subscripts and superscripts to aid the multiplication order) to describe our body-frame orientation, relative to the world frame:

$$ {}^{W}\mathbf{R}_B = {}^{W}\mathbf{R}_Z {}^{Z}\mathbf{R}_Y {}^{Y}\mathbf{R}_B. $$

This representation makes use of *Z-Y-X Euler angles,* with the"Z-Y-X" term indicating the rotation order. The rotation sequence that we have just followed can be thought of as so-called *intrinsic,* as the three rotations are performed around one of the "new" reference frame principal axes. *Extrinsic Euler angles* by contrast result from performing three rotations about different combinations of the world-frame axes.

The form derived above is commonly referred to as *Tait-Bryan Angles*, or *nautical angles*. That is, we first rotated about $\hat{\mathbf{z}}_W$, followed by a rotation about $\hat{\mathbf{y}}_Z$, and finishing with a rotation about $\hat{\mathbf{x}}_Y$. Explicitly, our rotation matrix, parameterised by the Z-Y-X Euler angles, is written as

$$
{}^{W}\mathbf{R}_B = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix},
$$

and after performing the matrix multiplication yields

$$
{}^{W}\mathbf{R}_B = \begin{bmatrix} \cos\psi\cos\theta & \cos\psi\sin\theta\sin\phi - \cos\phi\sin\psi & \sin\psi\sin\phi + \cos\psi\cos\phi\sin\theta \\ \cos\theta\sin\psi & \cos\psi\cos\phi + \sin\psi\sin\theta\sin\phi & \cos\phi\sin\psi\sin\theta - \cos\psi\sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\phi\cos\theta \end{bmatrix}.
$$

```
syms phi alpha psi real

Rz = [cos(psi) -sin(psi) 0;
      sin(psi) cos(psi) 0;
      0          0       1];
Ry = [cos(alpha)   0   sin(alpha);
      0          1       0;
     -sin(alpha)   0   cos(alpha)];
Rx = [1     0        0;
      0 cos(phi) -sin(phi);
      0 sin(phi) cos(phi)];

R = Rz*Ry*Rx
```

```
R =
```

$$
\begin{pmatrix} \cos(\alpha)\cos(\psi) & \cos(\psi)\sin(\alpha)\sin(\phi) - \cos(\phi)\sin(\psi) & \sin(\phi)\sin(\psi) + \cos(\phi)\cos(\psi)\sin(\alpha) \\ \cos(\alpha)\sin(\psi) & \cos(\phi)\cos(\psi) + \sin(\alpha)\sin(\phi)\sin(\psi) & \cos(\phi)\sin(\alpha)\sin(\psi) - \cos(\psi)\sin(\phi) \\ -\sin(\alpha) & \cos(\alpha)\sin(\phi) & \cos(\alpha)\cos(\phi) \end{pmatrix}
$$

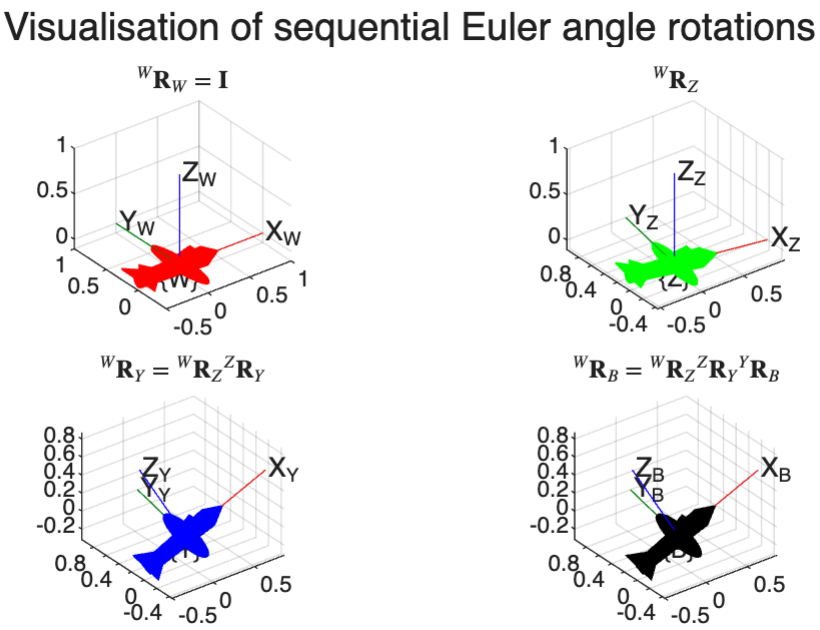
While the matrix above may seem a bit overwhelming, it is important to note that we only require three variables to fully define it, namely the Euler angles - $\{\phi, \theta, \psi\}$, which respectively describe roll, pitch, and yaw — see Figure 3.2 for a useful mnemonic to remember the rotation axes.

Figure 3.2: Mnemonics to remember the roll, pitch and yaw angle rotation axes [5].

The code block below can be used to visualise the workings of Euler angles in the four different frames that show the progression from $\{W\}$ to $\{B\}$, namely

$$\{W\} \rightarrow \{Z\} \rightarrow \{Y\} \rightarrow \{B\}.$$

```
phi = 0*pi/180;    %define roll angle about x axis of {Y}
the = -30*pi/180;     %define pitch angle about y axis of {Z}
psi = -10*pi/180;    %define yaw angle about z axis of {W}

Rz = [cos(psi) -sin(psi) 0;                %principal z-axis rotation matrix
      sin(psi) cos(psi) 0;
      0          0       1];
Ry = [cos(the)   0   sin(the);             %principal y-axis rotation matrix
      0          1       0;
     -sin(the)   0   cos(the)];
Rx = [1     0        0;                    %principal x-axis rotation matrix
      0 cos(phi) -sin(phi);
      0 sin(phi) cos(phi)];

%plot figures
figure,hold on
subplot(2,2,1),plotTransforms([0 0
0],rotm2quat(eye(3)),"MeshFilePath",'fixedwing.stl',"MeshColor","red","FrameAxisLabe
ls","on","FrameLabel","W")
axis equal, title('$^W{\bf R}_W={\bf I}$',Interpreter='latex'),grid on
subplot(2,2,2),plotTransforms([0 0
0],rotm2quat(Rz),"MeshFilePath",'fixedwing.stl',"MeshColor","green","FrameAxisLabels
","on","FrameLabel","Z")
axis equal, title('$^W{\bf R}_Z$',Interpreter='latex'),grid on
subplot(2,2,3),plotTransforms([0 0
0],rotm2quat(Rz*Ry),"MeshFilePath",'fixedwing.stl',"MeshColor","blue","FrameAxisLabe
ls","on","FrameLabel","Y")
axis equal, title('$^W{\bf R}_Y={^W{\bf R}_Z}{^Z{\bf R}_Y}
$',Interpreter='latex'),grid on
subplot(2,2,4),plotTransforms([0 0
0],rotm2quat(Rz*Ry*Rx),"MeshFilePath",'fixedwing.stl',"MeshColor","black","FrameAxis
Labels","on","FrameLabel","B")
axis equal, title('$^W{\bf R}_B={^W{\bf R}_Z}{^Z{\bf R}_Y}{^Y{\bf R}_B}
$',Interpreter='latex'),grid on

sgtitle('Visualisation of sequential Euler angle rotations')
```



While the *Z-Y-X* rotation order is fairly common in engineering (especially aeronautics and aerial robotics), we could have actually chosen from a multitude of rotation orders. Our only constraint is that we cannot rotate about the same axis sequentially, as that would be equivalent to a single rotation by the sum of the two angles. The twelve valid rotation orders are:

$$XYX, XZX, YXY, YZY, ZXZ, ZYZ, XYZ, XZY, YZX, YXZ, ZXY, ZYX$$

3.2.2 Gimbal lock

Different industries and manufacturers will use different rotation conventions, selecting one from above, but it is important to note there is technically no right or wrong version. However, one crucial point to be aware of is that an Euler angle representation of orientation always exhibits a singularity when any two rotation axes become parallel or anti-parallel. This phenomenon is commonly referred to as **gimbal lock**.

Definition: Gimbal lock

Gimbal lock is the loss of one degree of freedom in a three-dimensional, three-gimbal mechanism that occurs when the axes of two of the three gimbals are driven into a parallel configuration, "locking" the system into rotation in a degenerate two-dimensional space.

For the set of Euler angle rotation orders with repeated axes (sometimes referred to as *Eulerian*), namely

$$XYX, XZX, YXY, YZY, ZXZ, ZYZ,$$

gimbal lock will occur when the second angle of rotation equals $0 + k\pi$, where $k \in \mathbb{Z}$, as this will make the first and third rotation axis align.

For the set of Euler angles with no repeated axes (sometimes referred to as *Cardian*), namely

$$XYZ, XZY, YZX, YXZ, ZXY, ZYX,$$

gimbal lock will occur when the second angle of rotation equals $\pm\frac{\pi}{2}$, as this will make the first and third rotation axis align.

In summary, regardless of the Euler rotation convention, gimbal lock will occur when the first and third rotation axes are aligned. Physically, this means that rotations about the first and third axis cannot be differentiated apart, and are instead summed, as a result of degenerate 2D space. Figure 3.3 below gives a visualisation of gimbal lock.

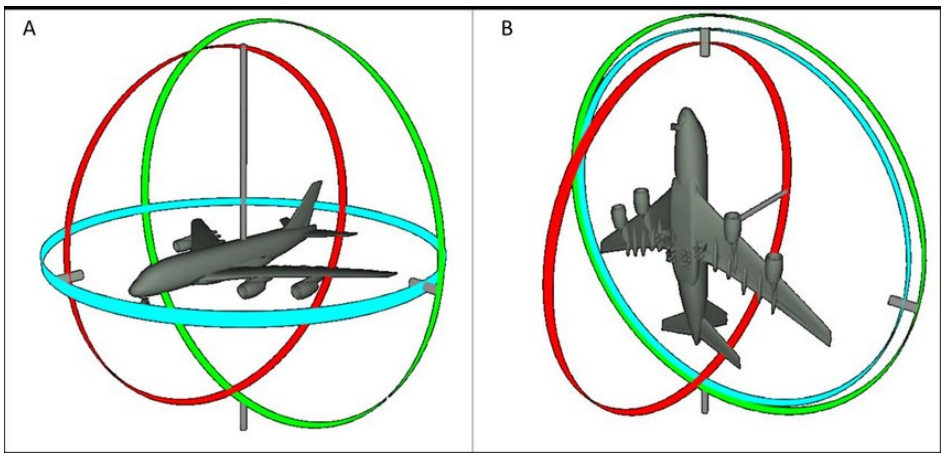


Figure 3.3: Visualisation of the three rotational axes (facilitated by imaginary gimbals) attached to an airplane. Left: nominal condition with none of the axes aligned. Right: Two of the gimbals become aligned causing a loss of information in one of the axes and resulting in a singularity [6]

So if you plan to use Euler angles, you need to be cognizant of how the angular behaviour is going to evolve. If the operating region contains Euler angles that cause the first and last rotation axes to become parallel, then consider re-framing how you have defined your reference frame, or use a different rotation order to circumvent the potential for gimbal lock. This video provides a good explanation of gimbal lock and the singularity problem.

3.2.3 Extracting Euler angles

The Euler-parameterised rotation matrix of

$${}^{W}\mathbf{R}_{B} = \begin{bmatrix} \cos\psi\cos\theta & \cos\psi\sin\theta\sin\phi - \cos\phi\sin\psi & \sin\psi\sin\phi + \cos\psi\cos\phi\sin\theta \\ \cos\theta\sin\psi & \cos\psi\cos\phi + \sin\psi\sin\theta\sin\phi & \cos\phi\sin\psi\sin\theta - \cos\psi\sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\phi\cos\theta \end{bmatrix}$$

shows us how to populate a rotation matrix given our Z-Y-X ordering, ${}^{W}\mathbf{R}_{B} = \mathbf{R}({}^{W}\boldsymbol{\eta}_{B})$, but we can also extract Euler angle information from a rotation matrix if required:

$${}^{W}\boldsymbol{\eta}_{B} = \mathbf{R}^{-1}({}^{W}\mathbf{R}_{B}).$$

We may, for example, need Euler angle information to visualise how our robot is oriented, or we may want to make use of that information within a control loop (e.g. regulating the yaw angle of a manipulator arm). Denoting the row x and column y element of the rotation matrix $\mathbf{R}$ as r_{xy}, namely

$$ {}^{W}\mathbf{R}_{B} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} = \begin{bmatrix} \cos\psi\cos\theta & \cos\psi\sin\theta\sin\phi - \cos\phi\sin\psi & \sin\psi\sin\phi + \cos\psi\cos\phi\sin\theta \\ \cos\theta\sin\psi & \cos\psi\cos\phi + \sin\psi\sin\theta\sin\phi & \cos\phi\sin\psi\sin\theta - \cos\psi\sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\phi\cos\theta \end{bmatrix}. $$

the corresponding Euler angles can be extracted using

$$ \phi = \arctan(\frac{r_{32}}{r_{33}}), $$

$$ \theta = -\arcsin r_{31}, $$

$$ \psi = \arctan(\frac{r_{21}}{r_{11}}). $$

Importantly, this result is only valid for *Z-Y-X* intrinsic Euler angles.

```
phi = -40*pi/180;    %set roll angle about x axis of {Y}
the = 90*pi/180;     %set pitch angle about y axis {Z}
psi = -30*pi/180;    %set yaw angle about z axis of {W}

Rz = [cos(psi) -sin(psi) 0; %principle rotation about z-axis
      sin(psi) cos(psi) 0;
      0          0       1];
Ry = [cos(the)   0   sin(the);  %principle rotation about y-axis
      0          1       0;
     -sin(the)   0   cos(the)];
Rx = [1     0        0;     %principle rotation about x-axis
      0 cos(phi) -sin(phi);
      0 sin(phi) cos(phi)];
R = Rz*Ry*Rx;    %combining all three rotations into 3D rotation matrix

%extracting Euler angles using formula above.
phi_ = atan2( R(3,2),R(3,3) )*180/pi
```

```
phi_ =
-40
```

```
theta_ = -asin( R(3,1) )*180/pi
```

```
theta_ =
90
```

```
psi_ = atan2( R(2,1),R(1,1) )*180/pi
```

```
psi_ =
-30.0000
```

Note that in practice we make use of `atan2` instead of `atan` when performing the calculations to determine ϕ and ψ, as this takes into account the full possible range of $\{\phi, \psi\} \in [-\pi, \pi]$.

3.2.4 Intrinsic and extrinsic rotations

Recall our Euler angle result of

$$^{W}\mathbf{R}_{B} = {}^{W}\mathbf{R}_{Z}{}^{Z}\mathbf{R}_{Y}{}^{Y}\mathbf{R}_{B},$$

which we assembled from left to right using the ordering of

$$\{W\} \rightarrow \{Z\} \rightarrow \{Y\} \rightarrow \{B\}.$$

Notably, we can also think of constructing the result above, building instead from *right to left*, by starting in $\{B\}$ and then:

1. performing a rotation of ϕ about $\hat{\mathbf{x}}_{W}$,
2. followed by a rotation of θ about $\hat{\mathbf{y}}_{W}$,
3. and then finally, a rotation of ψ about $\hat{\mathbf{z}}_{W}$.

This gives us a clue about intrinsic and extrinsic rotations:

- when a frame is rotated about a vector in $\{B\}$, we should *post-multiply* the current orientation with the matrix describing the new rotation,
- when a frame is rotated about a vector in $\{W\}$, we should *pre-multiply* the current orientation with the matrix describing the new rotation.

Example: Intrinsic and extrinsic rotations

In this example the robot has a initial orientation described by

$$\mathbf{R} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}.$$

If the robot then experiences a rotation of ψ about $\hat{\mathbf{z}}_{B}$, we can describe the resulting orientation as

$$\mathbf{R}\mathbf{R}_{z} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

which is obtained by post-multiplying $\mathbf{R}$ by $\mathbf{R}_{z}$. This constitutes an intrinsic rotation.

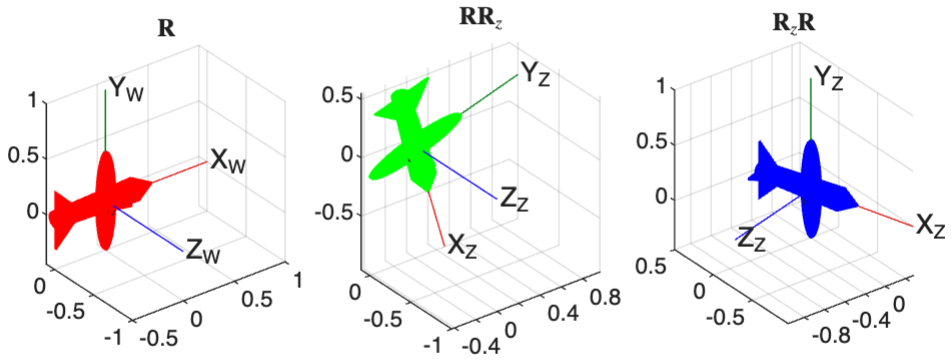
If instead, the robot with initial orientation of $\mathbf{R}$ experiences a rotation of ψ about $\hat{\mathbf{z}}_{W}$, we can describe the resulting orientation as

$$\mathbf{R}_z\mathbf{R} = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix},$$

which is obtained by pre-multiplying $\mathbf{R}$ by $\mathbf{R}_z$. This constitutes an extrinsic rotation.

The code block below provides a visual representation of this.

```
psi = -74*pi/180;

Rz = [cos(psi) -sin(psi) 0; %principle rotation about z-axis
      sin(psi) cos(psi) 0;
      0          0       1];

R = [1 0  0;    %initial orientation of robot
     0 0 -1;
     0 1  0];

R1 = R*Rz;    %new orientation after intrinsic rotation about z axis
R2 = Rz*R;    %new orientation after extrinsic rotation about z axis

%plot figures
figure,hold on
subplot(1,3,1),plotTransforms([0 0
0],rotm2quat(R),"MeshFilePath",'fixedwing.stl',"MeshColor","red","FrameAxisLabels","
on","FrameLabel","W")
axis equal, title('${\bf R}$',Interpreter='latex'),grid on
subplot(1,3,2),plotTransforms([0 0
0],rotm2quat(R1),"MeshFilePath",'fixedwing.stl',"MeshColor","green","FrameAxisLabels
","on","FrameLabel","Z")
axis equal, title('${\bf R}{\bf R}_z$',Interpreter='latex'),grid on
subplot(1,3,3),plotTransforms([0 0
0],rotm2quat(R2),"MeshFilePath",'fixedwing.stl',"MeshColor","blue","FrameAxisLabels"
,"on","FrameLabel","Z")
axis equal, title('${\bf R}_z{\bf R}$',Interpreter='latex'),grid on
sgtitle('Comparison of intrinsic and extrinsic rotations')
```



3.3 Quaternions

Euler angles are appealing as an introduction to understanding orientation, as they are intuitively pleasing and relatively easy to visualise. However, the potential for our orientation representation becoming degenerate and resulting in kinematic singularities raises a red flag for many robotic applications. Additionally, the idea of describing an orientation using three sequential rotations does not follow how rigid-body robot motion will evolve in general. Instead, we would expect pure rotations about some arbitrary axis — enter the quaternion!

The **quaternion** is a popular choice of encoding attitude that can completely avoid the issue of gimbal lock by rather describing the orientation as a single rotation about some defined axis that intercepts the relevant reference frame origin. At first glance, the quaternion can seem a bit daunting, but the benefits of understanding and being able to utilise it in robotics (among other fields) is worth the time investment that you may need to understand it. We will also see later in Section 3.4 that quaternions are closely related to exponential coordinates — another powerful and yet intuitive way to describe orientation.

3.3.1 Definition

The quaternion, $\mathbf{q}$, is a 4-element vector (also known as a 4-tuple) and is structured as

$$\mathbf{q} = \begin{bmatrix} q_0 \\ q_x \\ q_y \\ q_z \end{bmatrix},$$

in which $\{q_0, q_x, q_y, q_z\}$ are real numbers. In the context of describing orientation, we only consider *unit quaternions*, which gives us the unit-norm constraint of

$$|\mathbf{q}| = q_0^2 + q_x^2 + q_y^2 + q_z^2 = 1.$$

Explicitly, the unit quaternion follows an *angle-axis* structure of

$$\mathbf{q} = \begin{bmatrix} \cos\frac{\alpha}{2} \\ \sin\frac{\alpha}{2}\hat{\mathbf{n}} \end{bmatrix} = \begin{bmatrix} \cos\frac{\alpha}{2} \\ \sin\frac{\alpha}{2}v_x \\ \sin\frac{\alpha}{2}v_y \\ \sin\frac{\alpha}{2}v_z \end{bmatrix},$$

where $\hat{\mathbf{n}} = [n_x\, n_y\, n_z]^T$ is the unit-length ($|\hat{\mathbf{n}}| = 1$) rotation axis, and $\alpha \in [-\pi, \pi)$ is the rotation angle, as shown in Figure 3.4 for the case when the rotation vector is described with respect to $\{W\}$.

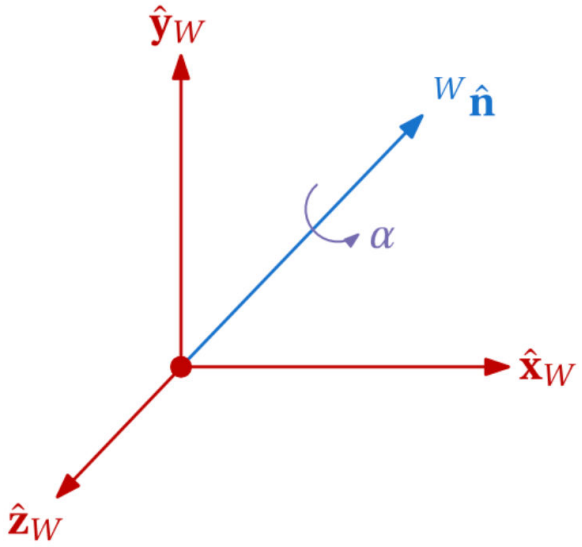
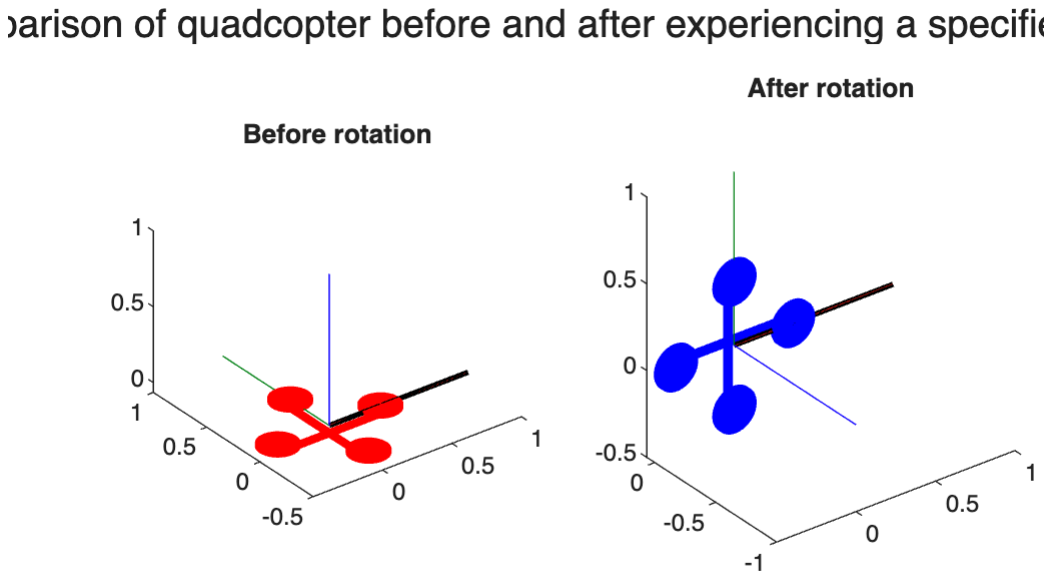


Figure 3.4: Visualisation of a quaternion-based rotation about vector $^W\hat{\mathbf{v}}$ with rotational angle α.

The code block below also shows how the quaternion axis-angle form can be used to rotate a rigid body, such as a quadcopter. The black line shows the rotation axis, $\hat{\mathbf{n}}$.

```
n_x = 1;
n_y = 0;
n_z = 0;
n = [n_x n_y n_z]';
n = n/norm(n);  %normalise v so that it always has unit length.

alpha = 90*pi/180;
q = [cos(alpha/2); sin(alpha/2)*n];

figure
subplot(1,2,1),hold on
plotTransforms([0 0 0],[1 0 0 0],"MeshFilePath",'multirotor.stl',"MeshColor","red")
```

```
line([0 n(1)],[0,n(2)],[0,n(3)],'Color','black','LineStyle','-','LineWidth',2)
axis equal,title('Before rotation')

subplot(1,2,2)
plotTransforms([0 0 0],q',"MeshFilePath",'multirotor.stl',"MeshColor","blue")
line([0 n(1)],[0,n(2)],[0,n(3)],'Color','black','LineStyle','-','LineWidth',2)
axis equal,title('After rotation')
sgtitle('Comparison of quadcopter before and after experiencing a specified
rotation')
```



The explicit quaternion form of

$$\mathbf{q} = \begin{bmatrix} \cos\frac{\alpha}{2} \\ \sin\frac{\alpha}{2}\hat{\mathbf{n}} \end{bmatrix} = \begin{bmatrix} \cos\frac{\alpha}{2} \\ \sin\frac{\alpha}{2}n_x \\ \sin\frac{\alpha}{2}n_y \\ \sin\frac{\alpha}{2}n_z \end{bmatrix} = \begin{bmatrix} q_0 \\ q_x \\ q_y \\ q_z \end{bmatrix}$$

is a way of *encoding* the 3D rotation of α about $\hat{\mathbf{v}}$ in a useful format, analogous to how rotation matrices encode attitude. Assuming we are given $\mathbf{q} = [q_0 \; q_x \; q_y \; q_z]^T$, we can always extract the angle-axis information from our quaternion representation using

$$\alpha = 2\cos^{-1}(q_0),$$

and

$$\hat{\mathbf{n}} = \frac{1}{\sin\frac{\alpha}{2}}\begin{bmatrix} q_x \\ q_y \\ q_z \end{bmatrix} = \frac{1}{\sqrt{1-q_0^2}}\begin{bmatrix} q_x \\ q_y \\ q_z \end{bmatrix}.$$

When $q_0 = 1$, (which implies that $\alpha = 0$), the equation above tends to infinity, making the rotation axis become undefined. However, this is not problematic, as $\alpha = 0$ corresponds to the null rotation matrix of $\mathbf{R} = \mathbf{I}$. In essence, we cannot determine the rotation vector because there is no rotation taking place. In this case, we would describe the identity (or null) quaternion as $\mathbf{q_I} = [1\,0\,0\,0]^T$. Given an angle-axis combination, we can now describe the corresponding quaternion

$$\{\alpha, \hat{\mathbf{n}}\} \rightarrow \mathbf{q},$$

and similarly, given a quaternion, we can recover our angle-axis pair

$$\mathbf{q} \rightarrow \{\alpha, \hat{\mathbf{n}}\}.$$

For example, if we wanted to rotate about the $\hat{\mathbf{z}}_W$-axis by an angle $\alpha = \psi$, our rotation axis would be $\hat{\mathbf{n}} = \hat{\mathbf{z}}_W = [0\,0\,1]^T$, resulting in

$${}^W\mathbf{q}_Z = \begin{bmatrix} \cos\frac{\psi}{2} \\ 0 \\ 0 \\ \sin\frac{\psi}{2} \end{bmatrix}.$$

Note that this quaternion is analogous to the principle z-axis rotation matrix of

$${}^W\mathbf{R}_Z = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

The method of encoding the orientation is just different. Fortunately, there does exist a one-to-one mapping between the quaternion representation and rotation matrix, given by

$$\mathbf{R} = \begin{bmatrix} 2(q_0^2+q_x^2)-1 & 2(q_xq_y-q_0q_z) & 2(q_xq_z+q_0q_y) \\ 2(q_xq_y+q_0q_z) & 2(q_0^2+q_y^2)-1 & 2(q_yq_z-q_0q_x) \\ 2(q_xq_z-q_0q_y) & 2(q_yq_z+q_0q_x) & 2(q_0^2+q_z^2)-1 \end{bmatrix},$$

and unsurprisingly, by substituting our result for ${}^W\mathbf{q}_Z$ into $\mathbf{R}$ above, we obtain ${}^W\mathbf{R}_Z$.

For $q_0 \neq 0$, which implies that $\alpha \neq \pm\pi$, extraction of the quaternion from a rotation matrix is achieved using

$$q_0 = \frac{1}{2}\sqrt{1+\mathrm{tr}\mathbf{R}}, \qquad q_x = \frac{r_{32}-r_{23}}{4q_0}, \qquad q_y = \frac{r_{13}-r_{31}}{4q_0}, \qquad q_z = \frac{r_{21}-r_{12}}{4q_0},$$

where

$$\mathbf{R} = \begin{bmatrix} 2(q_0^2 + q_x^2) - 1 & 2(q_x q_y - q_0 q_z) & 2(q_x q_z + q_0 q_y) \\ 2(q_x q_y + q_0 q_z) & 2(q_0^2 + q_y^2) - 1 & 2(q_y q_z - q_0 q_x) \\ 2(q_x q_z - q_0 q_y) & 2(q_y q_z + q_0 q_x) & 2(q_0^2 + q_z^2) - 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}.$$

If $q_0 = 0$, implying that that $\alpha = \pm\pi$, the equation above for determining the vector part of $\mathbf{q}$ will result in a singularity. In this case, we can instead determine $\{q_x, q_y, q_z\}$ using

$$q_x = \sqrt{\frac{r_{11} + 1}{2}}, \quad q_y = \sqrt{\frac{r_{22} + 1}{2}}, \quad q_z = \sqrt{\frac{r_{33} + 1}{2}}.$$

3.3.2 Quaternion multiplication

While the mapping from $\mathbf{q}$ to $\mathbf{R}$ is a nice sanity check on things, we are losing one of the benefits of the quaternion, and that is its mathematical efficiency. Just as we were able to sequentially construct our body-frame orientation by using the elementary rotations when dealing with Euler angles, we can follow a similar process for the quaternion. However, we first need to cover how quaternion multiplication works. Given two quaternions, $\mathbf{q}$ and $\mathbf{r}$, the quaternion product (or Hamilton Product), indicated using the symbol $\otimes$, is defined as

$$\mathbf{q} \otimes \mathbf{r} = \begin{bmatrix} q_0 r_0 - q_x r_x - q_y r_y - q_z r_z \\ q_0 r_x + r_0 q_x + q_y r_z - r_y q_z \\ q_0 r_y + r_0 q_y + r_x q_z - q_x r_z \\ q_0 r_z + r_0 q_z + q_x r_y - r_x q_y \end{bmatrix}.$$

This stems from the fact that $\mathbf{q}$ and $\mathbf{r}$ can be written as

$$\mathbf{q} = q_0 + q_x\hat{\mathbf{i}} + q_y\hat{\mathbf{j}} + q_z\hat{\mathbf{k}},$$

$$\mathbf{r} = r_0 + r_x\hat{\mathbf{i}} + r_y\hat{\mathbf{j}} + r_z\hat{\mathbf{k}},$$

respectively, where $\{\hat{\mathbf{i}}, \hat{\mathbf{j}}, \hat{\mathbf{k}}\}$ are the basis vectors indicating direction. Using the relationship of

$$\hat{\mathbf{i}}^2 = \hat{\mathbf{j}}^2 = \hat{\mathbf{k}}^2 = \hat{\mathbf{i}}\hat{\mathbf{j}}\hat{\mathbf{k}} = -1$$

yields the above-mentioned result of

$$\begin{aligned} \mathbf{q} \otimes \mathbf{r} &= \left(q_0 + q_x\hat{\mathbf{i}} + q_y\hat{\mathbf{j}} + q_z\hat{\mathbf{k}}\right)\left(r_0 + r_x\hat{\mathbf{i}} + r_y\hat{\mathbf{j}} + r_z\hat{\mathbf{k}}\right) \\ &= (q_0 r_0 - q_x r_x - q_y r_y - q_z r_z) + (q_0 r_x + r_0 q_x + q_y r_z - r_y q_z)\hat{\mathbf{i}} + (q_0 r_y + r_0 q_y + r_x q_z - q_x r_z)\hat{\mathbf{j}} + (q_0 r_z + r_0 q_z + q_x r_y - r_x q_y)\hat{\mathbf{k}} \\ &= \begin{bmatrix} q_0 r_0 - q_x r_x - q_y r_y - q_z r_z \\ q_0 r_x + r_0 q_x + q_y r_z - r_y q_z \\ q_0 r_y + r_0 q_y + r_x q_z - q_x r_z \\ q_0 r_z + r_0 q_z + q_x r_y - r_x q_y \end{bmatrix}. \end{aligned}$$

The result above can be written compactly as

$$\mathbf{q} \otimes \mathbf{r} = \begin{bmatrix} q_0 r_0 - \vec{\mathbf{q}} \cdot \vec{\mathbf{r}} \\ q_0 \vec{\mathbf{r}} + r_0 \vec{\mathbf{q}} + \vec{\mathbf{q}} \times \vec{\mathbf{r}} \end{bmatrix},$$

where $\vec{\mathbf{q}} = [q_x \; q_y \; q_z]^T$ and $\vec{\mathbf{r}} = [r_x \; r_y \; r_z]^T$.

Continuing with our exercise of emulating the sequential construction of our body-frame orientation as was done using ZYX Euler angles, we begin by configuring the two reference frames such that they are aligned and coincident. We then apply a rotation about $\hat{\mathbf{z}}_W = [0\,0\,1]^T$, with an angle of ψ, which gives us

$${}^W\mathbf{q}_Z = \begin{bmatrix} \cos\frac{\psi}{2} \\ 0 \\ 0 \\ \sin\frac{\psi}{2} \end{bmatrix}.$$

We next rotate about $\hat{\mathbf{y}}_Z = [0\,1\,0]^T$ with an angle of θ, which is encoded by the incremental quaternion of

$${}^Z\mathbf{q}_Y = \begin{bmatrix} \cos\frac{\theta}{2} \\ 0 \\ \sin\frac{\theta}{2} \\ 0 \end{bmatrix}.$$

Lastly, we rotate about $\hat{\mathbf{x}}_Y = [1\,0\,0]^T$, with an angle of ϕ, which is described by the incremental quaternion of

$${}^Y\mathbf{q}_B = \begin{bmatrix} \cos\frac{\phi}{2} \\ \sin\frac{\phi}{2} \\ 0 \\ 0 \end{bmatrix}.$$

Our full description of the body-frame orientation, using the quaternion as our *encoder* follows as

$${}^W\mathbf{q}_B = {}^W\mathbf{q}_Z \otimes {}^Z\mathbf{q}_Y \otimes {}^Y\mathbf{q}_B,$$

and is analogous to the rotation matrix equivalent of ${}^W\mathbf{R}_B = {}^W\mathbf{R}_Z {}^Z\mathbf{R}_Y {}^Y\mathbf{R}_B$. The code block below shows how to perform quaternion multiplication in `MATLAB` and also illustrates that the quaternion representation of attitude matches that of the rotation matrix counterpart.

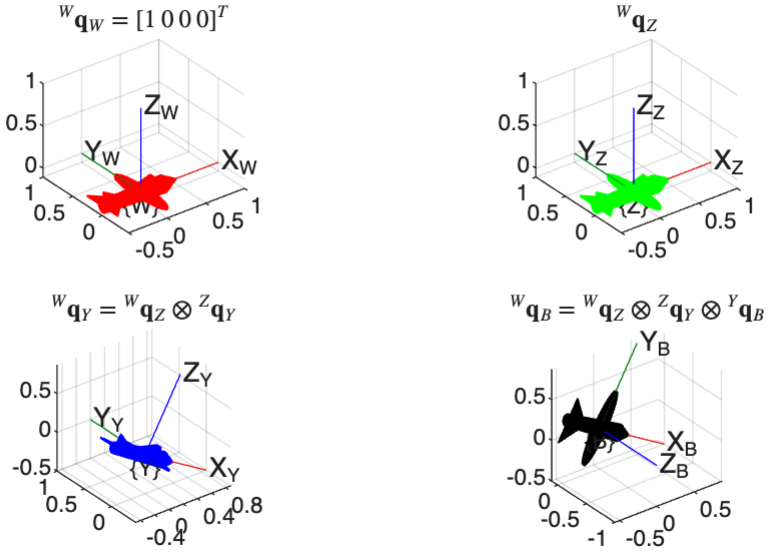
```
phi = 90*pi/180;    %set roll angle about x axis of {Y}
the = 30*pi/180;     %set pitch angle about y axis of {Z}
psi = 0*pi/180;    %set yaw angle about z axis of {W}
```

```
q_z2w = [cos(psi/2) 0 0 sin(psi/2)];    %quaternion representation of yaw rotation
about z in {W}
q_y2z = [cos(the/2) 0 sin(the/2) 0];    %quaternion representation of pitch
rotation about y in {Z}
q_b2y = [cos(phi/2) sin(phi/2) 0 0];    %quaternion representation of roll rotation
about x in {Y}

q_b2w = quatmultiply( quatmultiply(q_z2w,q_y2z), q_b2y )    %full quaternion
describing orientation of {B} wrt {W}
```

```
q_b2w = 1×4
    0.6830    0.6830    0.1830   -0.1830
```

```
%plot figures
figure,hold on,sgtitle('Representing Euler angle sequence using quaternions')
subplot(2,2,1),plotTransforms([0 0 0],[1 0 0
0],"MeshFilePath",'fixedwing.stl',"MeshColor","red","FrameAxisLabels","on","FrameLab
el","W")
axis equal, title('$^W{\bf q}_W=[1~0~0~0]^T$',Interpreter='latex'),grid on
subplot(2,2,2),plotTransforms([0 0
0],q_z2w,"MeshFilePath",'fixedwing.stl',"MeshColor","green","FrameAxisLabels","on","
FrameLabel","Z")
axis equal, title('$^W{\bf q}_Z$',Interpreter='latex'),grid on
subplot(2,2,3),plotTransforms([0 0
0],quatmultiply(q_z2w,q_y2z),"MeshFilePath",'fixedwing.stl',"MeshColor","blue","Fram
eAxisLabels","on","FrameLabel","Y")
axis equal, title('$^W{\bf q}_Y={^W{\bf q}_Z}\otimes {^Z{\bf q}_Y}
$',Interpreter='latex'),grid on
subplot(2,2,4),plotTransforms([0 0
0],quatmultiply( quatmultiply(q_z2w,q_y2z),q_b2y),"MeshFilePath",'fixedwing.stl',"Me
shColor","black","FrameAxisLabels","on","FrameLabel","B")
axis equal, title('$^W{\bf q}_B={^W{\bf q}_Z}\otimes {^Z{\bf q}_Y}\otimes {^Y{\bf
q}_B}$',Interpreter='latex'),grid on
```

Representing Euler angle sequence using quaternions



As with the Euler angles, we have followed the ordering of $\{W\} \rightarrow \{Z\} \rightarrow \{Y\} \rightarrow \{B\}$. Quaternions can also be thought of as being intrinsic or extrinsic — when post-multiplying by a quaternion, we can imagine the rotation vector being described in the current frame (as above), whereas when pre-multiply the vector is described in $\{W\}$.

Note that in this particular case, our quaternions are still parameterised by the Euler angles. This is not how we will usually make use of quaternions; rather, this is a demonstration of how quaternion rotation works, and its relationship with Euler angles and rotation matrices.

It is worth highlighting the numerical efficiency of the quaternion — while multiplying two rotation matrices requires 27 multiplications and 18 additions, quaternion multiplication only requires 16 multiplications and 12 additions. This may seem insignificant, but in certain applications where processing power is limited, or a large number of coordinate mappings are required (e.g. in the video game industry) this FLOP saving is invaluable!

As with the rotation matrix, the quaternion has an inverse, defined as

$$\mathbf{q}^{-1} = \begin{bmatrix} q_0 \\ -q_x \\ -q_y \\ -q_z \end{bmatrix}.$$

We can think of this inverse in one of two ways: either

- (i) the rotation vector changes direction and points in the opposite direction, or
- (ii) the rotation angle changes direction.

In either case, the quaternion inverse can be thought of as "undoing" a rotation.

If we rotated our robot such that our quaternion was $\mathbf{a} = [a_0\, a_x\, a_y\, a_z]^T$, followed by multiplication of the inverse, $\mathbf{a}^{-1} = [a_0\ -a_x\ -a_y\ -a_z]^T$, we would return to our identity quaternion, $\mathbf{a}^{-1} \otimes \mathbf{a} = [1\,0\,0\,0]^T$.

```
alpha = 45*pi/180;    %set rotation angle
nx = [1 1 1];     %define arbitrary vector
nx = nx/norm(nx);    %set vector norm to one

q = [cos(alpha/2) sin(alpha/2)*nx];    %quaternion describing rotation

q_I = quatmultiply( quatinv(q),q )    %quaternion multiplied by its inverse, which
always returns the null quaternion
```

```
q_I = 1×4
     1     0     0     0
```

3.3.3 Rotating a frame

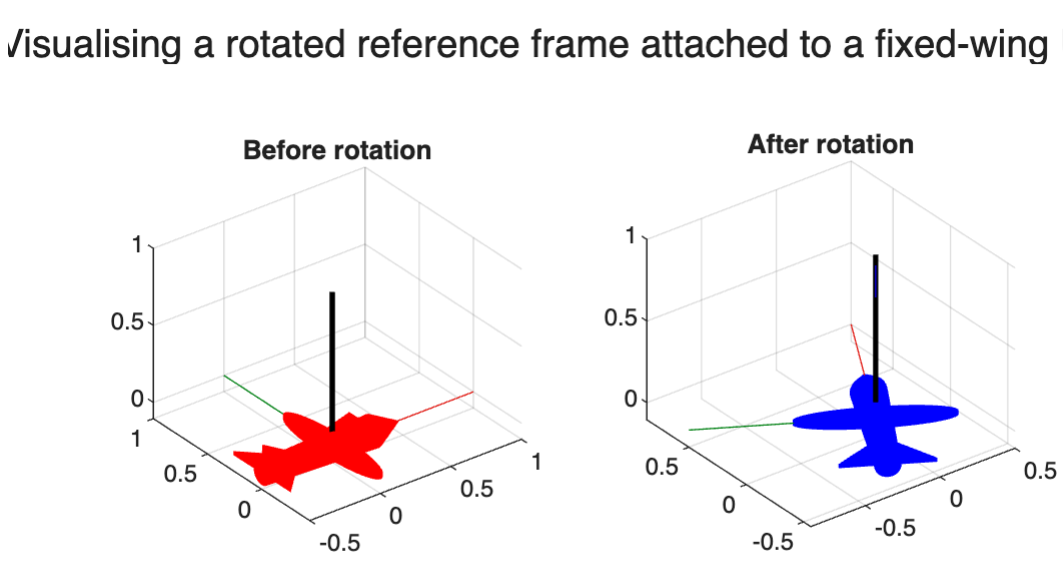
The sequential process of building a quaternion using the principal axis rotations is useful for demonstrating the parallel between rotation matrices and quaternions, but there is actually no need to ever use Euler angles when incorporating quaternions. As previously mentioned, the orientation can be defined using a rotation vector, $\hat{\mathbf{n}}$, and rotation angle α about $\hat{\mathbf{n}}$. In other words, we can define a mapping between two reference frames by either rotating about a single vector with a specific rotation angle (encoded as a quaternion), or equivalently performing three sequential rotations about the intrinsic axes (Euler-parameterised rotation matrix).

The quaternion approach is generally quite appealing for robotics, as it matches the "natural" rotational behaviour of rigid bodies. That is, incremental rotations will occur as a single motion, not three discrete motions. Later Chapters will also demonstrate that quaternions are highly beneficial when confronting the rigid-body robotic trajectory generation, orientation estimation, and control problem, whereas Euler angles can become problematic in most cases.

The code block below shows how rotation about a vector can be visualised, with the resulting pose after the rotation facilitated using quaternions.

```
n_x = 0;    %x component of rotation vector
n_y = 0;    %y component of rotation vector
n_z = 1;    %z component of rotation vector
n = [n_x n_y n_z]';  %full rotation vector
n = n/norm(n);  %normalise vector so that it always has unit length

alpha = 60*pi/180;    %rotation angle
q = [cos(alpha/2); sin(alpha/2)*n];    %quaternion, formed from combination of
rotation angle and rotation vector

%plot figures
figure
subplot(1,2,1),hold on
plotTransforms([0 0 0],[1 0 0 0],"MeshFilePath",'fixedwing.stl',"MeshColor","red")
line([0 n(1)],[0,n(2)],[0,n(3)],'Color','black','LineStyle','-','LineWidth',2)
axis equal,title('Before rotation'),grid on
```

```
subplot(1,2,2)
plotTransforms([0 0 0],q',"MeshFilePath",'fixedwing.stl',"MeshColor","blue")
line([0 n(1)],[0,n(2)],[0,n(3)],'Color','black','LineStyle','-','LineWidth',2)
axis equal,title('After rotation'),grid on

sgtitle('Visualising a rotated reference frame attached to a fixed-wing UAV')
```



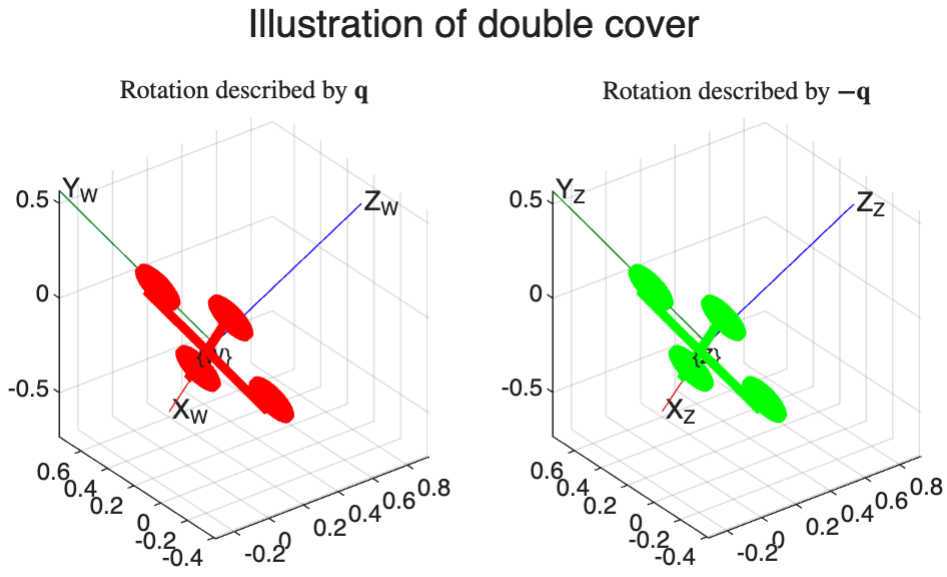
3.3.4 Double cover

There is one *minor* caveat that should be made clear when using quaternions. We are operating in a 3D Euclidean space, but the quaternion is a 4-element vector. So why does the quaternion have 4 parameters to specify a 3D orientation? The simple answer is that a single quaternion does not uniquely define an orientation. That is, the orientation derived using a rotation about some axis is equivalent to a negative rotation about that same axis after it has been inverted. This results in a **double cover** phenomenon where $\mathbf{q}$ and $-\mathbf{q}$ represent the same rotation.

The code block demonstrates this result in general, showing that orientation $\mathbf{q}$ and $-\mathbf{q}$ are equivalent.

```
q0 = 1; %set scalar component of quaternion
qx = 0.2; %set x component of quaternion
qy = 0.7; %set y component of quaternion
qz = 0.4; %set z component of quaternion
q = [q0 qx qy qz]; %unnormalised quaternion
q = q/norm(q); %normalised quaternion

%plot figures
figure,hold on
```

```
subplot(1,2,1),plotTransforms([0 0
0],q,"MeshFilePath",'multirotor.stl',"MeshColor","red","FrameAxisLabels","on","Frame
Label","W")
axis equal, title(['Rotation described by ','$\bf q$'],Interpreter='latex'),grid on
subplot(1,2,2),plotTransforms([0 0 0],-
q,"MeshFilePath",'multirotor.stl',"MeshColor","green","FrameAxisLabels","on","FrameL
abel","Z")
axis equal, title(['Rotation described by ','$\bf -q$'],Interpreter='latex'),grid on
sgtitle('Illustration of double cover')
```



Example: Double cover

You rotate an arbitrary reference frame about a specified rotation vector, $\hat{\mathbf{n}}$, by some angle ϕ, giving us the quaternion

$$\mathbf{q}_1 = [\cos(\phi/2)\hat{\mathbf{n}}^T \sin(\phi/2)]^T .$$

If we had decided to instead rotate about the same rotation vector but in the opposite direction with a rotation angle of $\phi - 2\pi$, our quaternion would follow as

$$\mathbf{q}_2 = [\cos(\frac{\phi - 2\pi}{2}) \quad \hat{\mathbf{n}}^T \sin(\frac{\phi - 2\pi}{2})]^T = [\cos(\phi/2 - \pi) \quad \hat{\mathbf{n}}^T \sin(\phi/2 - \pi)]^T.$$

Noting the periodic nature of the cos and sin functions, we can make use of $\sin(x - \pi) = -\sin x$ and $\cos(x - \pi) = -\cos x$ to reduce $\mathbf{q}_2$ to

$$\mathbf{q}_2 = [-\cos(\phi/2) \quad -\hat{\mathbf{n}}^T \sin(\phi/2)]^T = -\mathbf{q}_1 .$$

As a result of this double covering, every orientation will map to two quaternions; one that describes the minimum angle rotation, $|\alpha| \leq \pi$, and the other describing the maximum angle rotation, $|\alpha| \geq \pi$. The only exception is the special case where the rotation angle is 180°, making both rotations equivalent in terms of angular distance. This disparity between a minimum and maximum angle rotation becomes important when our robot is trying to follow a path or correct an angular error, as we commonly desire minimal angular changes that result in smaller time and energy requirements (the minimum angle rotation). This is often referred to as the geodesic.

3.3.5 Rotating vectors

We have already shown that quaternions can be used to

1. represent orientation and
2. rotate frames.

Analogous to rotation matrices, quaternions also possess the third property of being able to map vectors between frames. We will not explore this in detail in this course, but the following formulation can be used to rotate vector $\mathbf{p}$ between two frames, such as from $\{B\}$ to $\{W\}$:

$$\begin{bmatrix} 0 \\ {}^W\mathbf{p} \end{bmatrix} = {}^W\mathbf{q}_B \otimes \begin{bmatrix} 0 \\ {}_B\mathbf{p} \end{bmatrix} \otimes {}^W\mathbf{q}_B^{-1}.$$

Note that this is equivalent to the rotation matrix form of

$${}^W\mathbf{p} = {}^W\mathbf{R}_B {}^B\mathbf{p},$$

albeit less convenient to express mathematically when chaining together multiple sequential rotations.

```
pb = [1 2 3]';    %p wrt {B}

q = [1/sqrt(2) 0 1/sqrt(2) 0];    %quaternion describing orientation of {B} wrt {W}
R = quat2rotm(q);

pw = R*pb    %p wrt {W} using rotation matrix approach
```

```
pw = 3×1
    3.0000
    2.0000
   -1.0000
```

```
pb_q = [0 pb']; %"quaternionised" p wrt {B}
pw_q = quatmultiply( quatmultiply(q, pb_q),quatinv(q) );  %calculate
"quaternionised" p wrt {W}
pw_ = pw_q(2:4)'    %p wrt {W} using quaternion approach
```

```
pw_ = 3×1
     3
     2
    -1
```

3.4 Exponential coordinates of rotations

3.4.1 Definition

Directly related to quaternions, **exponential coordinates** parameterise an orientation in terms of a rotation axis, represented by a unit vector $\hat{\mathbf{n}}$, and an angle of rotation α; the vector $\alpha\hat{\mathbf{n}} \in \mathbb{R}^3$ then serves as the three-parameter exponential coordinate representation of the rotation. Writing $\hat{\mathbf{n}}$ and α individually is known as the **axis-angle** representation of a rotation and can be recovered from exponential coordinates of $\alpha\hat{\mathbf{n}} = [\theta_x \ \theta_y \ \theta_z]^T$ using

$$\alpha = |\alpha\hat{\mathbf{n}}| = \sqrt{\theta_x^2 + \theta_y^2 + \theta_z^2},$$

$$\hat{\mathbf{n}} = \frac{1}{\alpha}[\theta_x \ \theta_y \ \theta_z]^T = \frac{1}{\sqrt{\theta_x^2 + \theta_y^2 + \theta_z^2}}[\theta_x \ \theta_y \ \theta_z]^T.$$

Note that the rotation angle in this case will always be positive.

Exponential coordinates can be used to describe orientation and can also be encoded in rotation matrices or quaternions to map vectors between frames. However, before we can show this result, we need to introduce the skew-symmetric matrix.

Definition: Skew-symmetric matrix

A **skew-symmetric matrix** is a square matrix whose transpose equals its negative: $\mathbf{S}^T = -\mathbf{S}$.

By this definition, the diagonal elements of $\mathbf{S}$ must all equal zero in order to satisfy $\mathbf{S}^T = -\mathbf{S}$. Skew-symmetric matrices are useful in reposing a vector cross product operation as a matrix-vector multiplication, as we will show in a latter section. Given an arbitrary vector $\mathbf{v} = [v_x \, v_y \, v_z]^T$, the corresponding skew-symmetric matrix is

$$\lfloor \mathbf{v} \rfloor = \begin{bmatrix} 0 & -v_z & v_y \\ v_z & 0 & -v_x \\ -v_y & v_x & 0 \end{bmatrix},$$

which shows the symmetry across the diagonal, albeit with a sign change. Note the use of $\lfloor \ldots \rfloor$ to indicate the skew-symmetric mapping.

As we will explore later in detail, if we wanted to evaluate a cross product operation such as $\mathbf{v} \times \mathbf{p}$, we could repose it as $\mathbf{v} \times \mathbf{p} = \lfloor \mathbf{v} \rfloor \mathbf{p}$, which is a vector multiplying into a matrix (not a cross product of two vectors). However, the result would remain the same.

3.4.2 Exponential coordinates of SO(3)

It can be shown that a rotation matrix, parameterised using exponential coordinates, is given by the matrix exponential mapping

$$\mathbf{R} = \mathrm{e}^{\lfloor \alpha \hat{\mathbf{n}} \rfloor} = \mathrm{e}^{\lfloor \hat{\mathbf{n}} \rfloor \alpha}.$$

Note that the matrix exponential differs from the standard single-element matrix exponential and in general is described by

$$\mathrm{e}^{\mathbf{A}t} = I + t\mathbf{A} + \frac{t^2}{2!}\mathbf{A}^2 + \frac{t^3}{3!}\mathbf{A}^3 + \dots,$$

where $\mathbf{A} \in \mathbb{R}^{n \times n}$ is constant. For the special case when $\mathbf{A}$ is skew-symmetric, for example, $\mathbf{A} = \lfloor \hat{\mathbf{n}} \rfloor \alpha$, the equation above can be reduced without loss of generality as

$$\mathbf{R} = \mathrm{e}^{\lfloor \hat{\mathbf{n}} \rfloor \alpha} = I + \sin\alpha \lfloor \hat{\mathbf{n}} \rfloor + (1 - \cos\alpha) \lfloor \hat{\mathbf{n}} \rfloor^2.$$

The equation above is known as **Rodrigues' formula** for rotations and is a convenient means of building a rotation matrix from exponential coordinates without having to evaluate the matrix exponential (an operation that can be computationally expensive on embedded hardware).

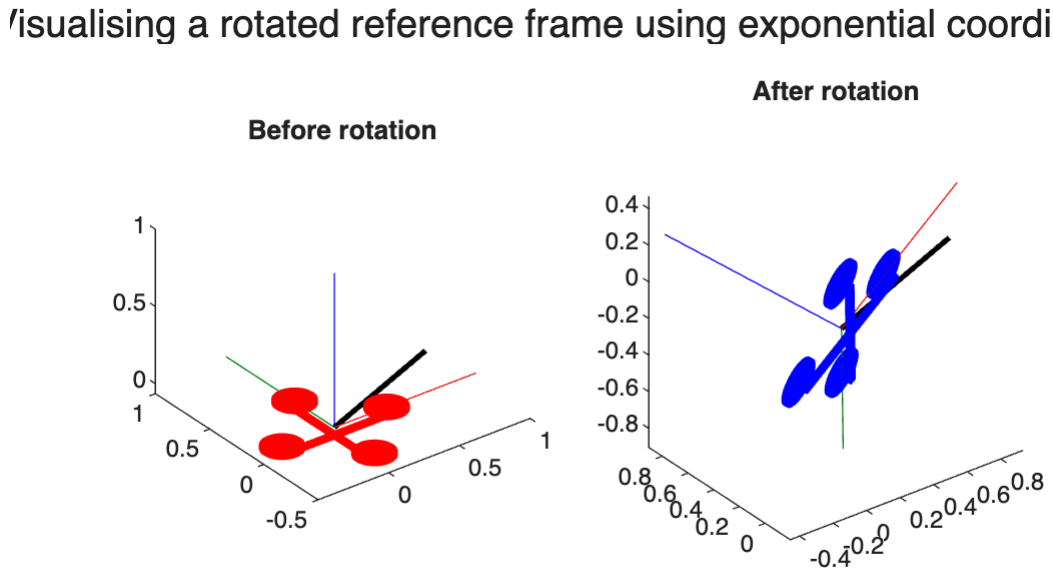
Writing out the above equation explicitly yields

$$\mathbf{R} = \begin{bmatrix} \mathrm{c}_\alpha + \hat{n}_x^2(1-\mathrm{c}_\alpha) & \hat{n}_x\hat{n}_y(1-\mathrm{c}_\alpha) - \hat{n}_z\mathrm{s}_\alpha & \hat{n}_x\hat{n}_z(1-\mathrm{c}_\alpha) + \hat{n}_y\mathrm{s}_\alpha \\ \hat{n}_x\hat{n}_y(1-\mathrm{c}_\alpha) + \hat{n}_z\mathrm{s}_\alpha & \mathrm{c}_\alpha + \hat{n}_y^2(1-\mathrm{c}_\alpha) & \hat{n}_y\hat{n}_z(1-\mathrm{c}_\alpha) - \hat{n}_x\mathrm{s}_\alpha \\ \hat{n}_x\hat{n}_z(1-\mathrm{c}_\alpha) - \hat{n}_y\mathrm{s}_\alpha & \hat{n}_y\hat{n}_z(1-\mathrm{c}_\alpha) + \hat{n}_x\mathrm{s}_\alpha & \mathrm{c}_\alpha + \hat{n}_z^2(1-\mathrm{c}_\alpha) \end{bmatrix},$$

where $\hat{\mathbf{n}} = [\hat{n}_x\ \hat{n}_y\ \hat{n}_z]^T$, using the shorthand notation of $\mathrm{s}_\alpha = \sin\alpha$ and $\mathrm{c}_\alpha = \cos\alpha$.

```
n_x = 1;    %x component of rotation vector
n_y = 0.4;    %y component of rotation vector
n_z = 0;    %z component of rotation vector
n = [n_x n_y n_z]';
n = n/norm(n);  %normalised rotation vector

alpha = -70*pi/180;    %rotation angle

S = [0   -n_z  n_y;    %skew-symmetric form of rotation vector
     n_z  0   -n_x;
    -n_y  n_x  0];

% R = expm(S*alpha);
R = eye(3)+sin(alpha)*S+( 1-cos(alpha) )*S^2;    %Rodrigues' formula to determine R

%plot figures
figure
subplot(1,2,1),hold on
plotTransforms([0 0 0],[1 0 0 0],"MeshFilePath",'multirotor.stl',"MeshColor","red")
line([0 n(1)],[0,n(2)],[0,n(3)],'Color','black','LineStyle','-','LineWidth',2)
axis equal,title('Before rotation')

subplot(1,2,2)
```

```
plotTransforms([0 0
0],rotm2quat(R),"MeshFilePath",'multirotor.stl',"MeshColor","blue")
line([0 n(1)],[0,n(2)],[0,n(3)],'Color','black','LineStyle','-','LineWidth',2)
axis equal,title('After rotation')

sgtitle('Visualising a rotated reference frame using exponential coordinates')
```



Note that when dealing with exponential coordinates in SO(2), the exponential coordinates collapses to $\alpha = \theta$, where θ corresponds to the rotation experienced by one frame with respect to another whilst bound to a plane.

Example: Using Rodrigues' formula

Determine the pose of $\{B\}$ with respect to $\{W\}$ when a frame initially aligned with $\{W\}$ experiences a rotation about ${}^{W}\hat{\mathbf{n}} = [0\ 0.866\ 0.5]^T$ with an angle of $\alpha = 30^\circ = 0.524$ rad.

Using Rodrigues' formula, we obtain

$$
\begin{aligned}
{}^{W}\mathbf{R}_B &= \mathrm{e}^{\lfloor {}^{W}\hat{\mathbf{n}} \rfloor \alpha}, \\
&= I + \sin\alpha \lfloor {}^{W}\hat{\mathbf{n}} \rfloor + (1 - \cos\alpha) \lfloor {}^{W}\hat{\mathbf{n}} \rfloor^2, \\
&= I + 0.5 \begin{bmatrix} 0 & -0.5 & 0.866 \\ 0.5 & 0 & 0 \\ -0.866 & 0 & 0 \end{bmatrix} + 0.134 \begin{bmatrix} 0 & -0.5 & 0.866 \\ 0.5 & 0 & 0 \\ -0.866 & 0 & 0 \end{bmatrix}^2, \\
&= \begin{bmatrix} 0.866 & -0.250 & 0.433 \\ 0.250 & 0.967 & 0.058 \\ -0.433 & 0.058 & 0.899 \end{bmatrix}.
\end{aligned}
$$

```
n = [0 0.866 0.5]';    %rotation vector from text above
alpha = 30*pi/180;     %rotation angle from text above
```

```
S = [0 -n(3) n(2);    %skew-symmetric form of rotation vector
     n(3) 0 -n(1);
     -n(2) n(1) 0];

R = expm(S*alpha)    %rotation matrix determined from matrix exponential
```

```
R = 3×3
    0.8660   -0.2500    0.4330
    0.2500    0.9665    0.0580
   -0.4330    0.0580    0.8995
```

```
R_ = eye(3)+sin(alpha)*S+( 1-cos(alpha) )*S^2    %rotation matrix determined using
Rodrigues' formula
```

```
R_ = 3×3
    0.8660   -0.2500    0.4330
    0.2500    0.9665    0.0580
   -0.4330    0.0580    0.8995
```

3.4.3 Principal axis rotations

Recalling that our rotation matrix can be parameterised by the axis-angle representation as

$$\mathbf{R} = \begin{bmatrix} \mathrm{c}_\alpha + \hat{n}_x^2(1-\mathrm{c}_\alpha) & \hat{n}_x\hat{n}_y(1-\mathrm{c}_\alpha) - \hat{n}_z\mathrm{s}_\alpha & \hat{n}_x\hat{n}_z(1-\mathrm{c}_\alpha) + \hat{n}_y\mathrm{s}_\alpha \\ \hat{n}_x\hat{n}_y(1-\mathrm{c}_\alpha) + \hat{n}_z\mathrm{s}_\alpha & \mathrm{c}_\alpha + \hat{n}_y^2(1-\mathrm{c}_\alpha) & \hat{n}_y\hat{n}_z(1-\mathrm{c}_\alpha) - \hat{n}_x\mathrm{s}_\alpha \\ \hat{n}_x\hat{n}_z(1-\mathrm{c}_\alpha) - \hat{n}_y\mathrm{s}_\alpha & \hat{n}_y\hat{n}_z(1-\mathrm{c}_\alpha) + \hat{n}_x\mathrm{s}_\alpha & \mathrm{c}_\alpha + \hat{n}_z^2(1-\mathrm{c}_\alpha) \end{bmatrix},$$

we can easily form the three principal axis rotations of:

- $\mathbf{R}_x = \mathbf{R}(\alpha, \hat{\mathbf{x}})$ — rotation of α about x axis,
- $\mathbf{R}_y = \mathbf{R}(\alpha, \hat{\mathbf{y}})$ — rotation of α about y axis,
- $\mathbf{R}_z = \mathbf{R}(\alpha, \hat{\mathbf{z}})$ — rotation of α about z axis,

by adjusting the components of $\hat{\mathbf{n}} = [n_x\, n_y\, n_z]^T$ in the rotation matrix equation above.

Specifically, if we set $\hat{\mathbf{n}} = [1\,0\,0]^T$ in the rotation matrix equation above, we obtain

$$\mathbf{R}_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix},$$

if we set $\hat{\mathbf{n}} = [0\,1\,0]^T$ we obtain

$$\mathbf{R}_y = \begin{bmatrix} \cos\alpha & 0 & \sin\alpha \\ 0 & 1 & 0 \\ -\sin\alpha & 0 & \cos\alpha \end{bmatrix},$$

and if we set $\hat{\mathbf{n}} = [0\,0\,1]^T$ we get

$$\mathbf{R}_z = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

These results match of the Euler angle formulation supplied in Section 3.2.1.

3.4.4 Matrix logarithms of rotations

We can recover the exponential coordinates, $\alpha\hat{\mathbf{n}}$ from a rotation matrix $\mathbf{R}$ using a **matrix logarithm** operation, which is the *inverse* of the matrix exponential. Specifically, if the rotation matrix is given by

$$\mathbf{R} = \mathbf{e}^{\lfloor \alpha\hat{\mathbf{n}} \rfloor} = \mathbf{e}^{\lfloor \hat{\mathbf{n}} \rfloor \alpha},$$

then the matrix logarithm of the rotation matrix yields

$$\log \mathbf{R} = \lfloor \hat{\mathbf{n}} \rfloor \alpha.$$

Based on this, we can think of the matrix exponential as in *integration operator* for the exponential coordinates, and the matrix logarithm as a *differentiation operator*.

Example: Matrix logarithm

Using MATLAB, determine the rotation matrix corresponding to $\hat{\mathbf{n}} = [0 \ 0.866 \ 0.6]^T$ and $\alpha = 30^\circ$, and show that the exponential coordinates can be recovered using the matrix logarithm.

```
n = [0 0.866 0.5]'    %rotation vector
```

```
n = 3×1
         0
    0.8660
    0.5000
```

```
alpha = 30*pi/180     %rotation angle
```

```
alpha =
0.5236
```

```
S = [0 -n(3) n(2);    %skew-symmetric form of rotation vector
     n(3) 0 -n(1);
     -n(2) n(1) 0];

R = expm(S*alpha);    %rotation matrix from matrix exponential

S_ = logm(R);         %matrix logarithm of rotation matrix
va = [-S_(2,3) S_(1,3) -S_(1,2)]';    %recovered exponential coordinates
alpha_ = norm( va )   %recovered rotation angle
```

```
alpha_ =
0.5236
```

```
v_ = va/alpha         %recovered rotation vector
```

```
v_ = 3×1
    0.0000
    0.8660
    0.5000
```

Instead of having to use the matrix logarithm, which is computationally heavy on embedded systems, an algorithm exists that can determine the exponential coordinates. The rotation angle can be determined from the rotation matrix using

$$\alpha = \cos^{-1}\left(\frac{\mathrm{tr}\mathbf{R} - 1}{2}\right),$$

where $\mathrm{tr}\mathbf{R}$ corresponds to the trace of matrix $\mathbf{R}$, namely, summing the diagonal entries of $\mathbf{R}$:

$$\mathrm{tr}\mathbf{R} = r_{11} + r_{22} + r_{33}.$$

For $\alpha \neq \pi$, the rotation vector (in skew-symmetric form) is then determined by

$$\lfloor \hat{\mathbf{n}} \rfloor = \frac{1}{2\sin\alpha}(\mathbf{R} - \mathbf{R}^T).$$

For the special case of $\alpha = \pi$, the rotation vector is one of the following solutions, which would need to be verified using $\mathbf{R} = \mathrm{e}^{\lfloor \hat{\mathbf{n}} \rfloor \alpha} = I + \sin\alpha \lfloor \hat{\mathbf{n}} \rfloor + (1 - \cos\alpha)\lfloor \hat{\mathbf{n}} \rfloor^2$:

$$\hat{\mathbf{n}} = \frac{1}{\sqrt{2(1+r_{33})}}\begin{bmatrix} r_{13} \\ r_{23} \\ 1 + r_{33} \end{bmatrix}, \qquad \hat{\mathbf{n}} = \frac{1}{\sqrt{2(1+r_{22})}}\begin{bmatrix} r_{12} \\ 1 + r_{22} \\ r_{32} \end{bmatrix}, \qquad \hat{\mathbf{n}} = \frac{1}{\sqrt{2(1+r_{11})}}\begin{bmatrix} 1 + r_{11} \\ r_{21} \\ r_{31} \end{bmatrix}.$$

3.4.5 The dot product and cross product

Recall that the cross product of two unitary vectors is described by

$$\hat{\mathbf{a}} \times \hat{\mathbf{b}} = \sin\alpha \hat{\mathbf{n}},$$

where α describes the angle between vectors $\hat{\mathbf{a}}$ and $\hat{\mathbf{b}}$, and $\hat{\mathbf{n}}$ defines the vector perpendicular to $\hat{\mathbf{a}}$ and $\hat{\mathbf{b}}$. Based on this description, if we were to rotate vector $\hat{\mathbf{a}}$ about vector $\hat{\mathbf{n}}$ with a rotation of α units, we would obtain $\hat{\mathbf{b}}$, namely

$$\begin{aligned} \hat{\mathbf{b}} &= \mathbf{R}\hat{\mathbf{a}}, \\ &= \mathbf{e}^{\lfloor \alpha \hat{\mathbf{n}} \rfloor}\hat{\mathbf{a}}. \end{aligned}$$

As such, $\alpha\hat{\mathbf{n}}$ can be thought of as exponential coordinates describing the mapping from $\hat{\mathbf{a}}$ and $\hat{\mathbf{b}}$. This relationship is useful when dealing with multivariable control design (e.g. thrust vectoring control) and state estimation (comparing vector directions) — both of which will be explored later in this course. Note that when α is sufficiently small, the exponential coordinates can be suitably approximated by $\hat{\mathbf{a}} \times \hat{\mathbf{b}} = \sin\alpha\hat{\mathbf{n}} \approx \alpha\hat{\mathbf{n}}$.

Recall the dot product of two unitary vectors:

$$\hat{\mathbf{a}} \cdot \hat{\mathbf{b}} = \cos\alpha.$$

Given known $\hat{\mathbf{a}}$ and $\hat{\mathbf{b}}$, the rotation angle can be recovered by

$$\alpha \ = \cos^{-1}\left(\hat{\mathbf{a}} \cdot \hat{\mathbf{b}}\right),$$

where $\alpha \in [0, \pi]$. The rotation vector is then determined by

$$\hat{\mathbf{n}} = \frac{1}{\sin\alpha}\hat{\mathbf{a}} \times \hat{\mathbf{b}}.$$

Note that the inverse cosine function will return a value of α between 0-π. Additionally, if $\hat{\mathbf{a}}$ and $\hat{\mathbf{b}}$ are anti-parallel, determination of $\hat{\mathbf{n}}$ using the equation above will be incorrect, as $\hat{\mathbf{a}} \times \hat{\mathbf{b}} = \mathbf{0}$ and $\alpha = \pi$. Specifically, when vectors $\hat{\mathbf{a}}$ and $\hat{\mathbf{b}}$ are collinear (form a line), any rotation vector and rotation angle $\alpha = \pi$ will map $\hat{\mathbf{a}}$ to $\hat{\mathbf{b}}$. In this case, one could arbitrarily choose a vector that lies in the plane orthogonal to this line.

Example: Cross product and rotation matrix

```
a = [1 2 3]';        %arbitrary vector a
a = a/norm(a);        %arbitrary normalised vector a
b = [-1 1 3]';        %arbitrary vector b
b = b/norm(b)        %arbitrary normalised vector b
```

```
b = 3×1
   -0.3015
    0.3015
    0.9045
```

```
n = cross(a,b);    %cross product of vectors a and b
alpha = acos( dot(a,b));    %rotation angle between a and b
c = n/sin(alpha);    %rotation vector perpendicular to a and b

phi = alpha*c;    %exponential coordinates
S = [0 -phi(3) phi(2);    %skew-symmetric form of exponential coordinates
    phi(3) 0 -phi(1);
    -phi(2) phi(1) 0];

R = expm(S);    %rotation matrix
b_ = R*a        %determining vector b using rotation matrix
```

```
b_ = 3×1
   -0.3015
    0.3015
    0.9045
```

```
%plot figures
figure, hold on
quiver3(0,0,0,a(1),a(2),a(3))
quiver3(0,0,0,b(1),b(2),b(3))
```

```
quiver3(0,0,0,c(1),c(2),c(3))
legend('$\hat{a}$','$\hat{b}$','$\hat{c}$','Interpreter','Latex')
axis equal
sgtitle('Relationship between the cross product of two vectors and their relative
orientation')

%set x-y axis limits and view angle
xlim([-0.60 0.94])
ylim([-0.74 0.47])
view([-22.493 9.992])
```

3.5 Angular velocity

Angular velocity will dictate how our orientation changes over time, so we are therefore interested in how the two are related when dealing with multiple reference frames. This is important when mathematically modelling systems (for example when designing simulation environments) and also when angular rate information is available from a sensor (for example a gyroscope) and we want to estimate orientation from it.

In the case of planar rotations (resulting in rotations about a fixed principle axis) the relationship is trivial, but we require more scrutiny for the generalised spatial case when up to three degrees of rotation freedom are at work.

3.5.1 Rotation matrix derivative

To understand the relationship between angular velocity and orientation, we need to develop a better understanding of reference frame motion as they rotate. Consider the motion of a point fixed to the end of ${}^{W}\hat{\mathbf{x}}_{B}$ when $\{B\}$ experiences and angular velocity of ${}^{W}\boldsymbol{\omega}$, as shown in Figure 3.5.

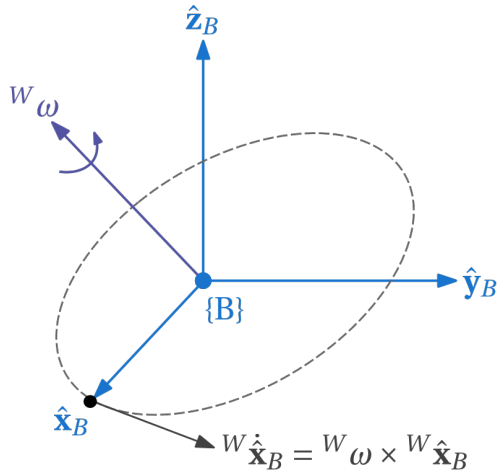


Figure 3.5: Reference frame $\{B\}$ experiencing angular velocity. ${}^W\hat{\mathbf{x}}_B$ experiences translational velocity based on angular velocity ${}^W\boldsymbol{\omega}$.

Recall that ${}^W\hat{\mathbf{x}}_B$ describes the x-axis of the body frame when described in the world frame, which is obtained by

$$\begin{aligned} {}^W\hat{\mathbf{x}}_B &= {}^W\mathbf{R}_B\hat{\mathbf{x}}_B, \\ &= {}^W\mathbf{R}_B\begin{bmatrix}1\\0\\0\end{bmatrix}. \end{aligned}$$

Given the orthogonality property of ${}^W\mathbf{R}_B$, ${}^W\hat{\mathbf{x}}_B$ will always have unit length. This means that during rotation, only the direction of ${}^W\hat{\mathbf{x}}_B$ will change — not the length. Given that the angular velocity of the body frame is described in the world frame as ${}^W\boldsymbol{\omega}$, we can determine the rate of change of ${}^W\hat{\mathbf{x}}_B$ based on elementary mechanics, namely

$${}^W\dot{\hat{\mathbf{x}}}_B = {}^W\boldsymbol{\omega} \times {}^W\hat{\mathbf{x}}_B.$$

We can repeat this process for the other two axes of $\{B\}$without loss of generality, which yields

$${}^W\dot{\hat{\mathbf{y}}}_B = {}^W\boldsymbol{\omega} \times {}^W\hat{\mathbf{y}}_B.$$

$${}^W\dot{\hat{\mathbf{z}}}_B = {}^W\boldsymbol{\omega} \times {}^W\hat{\mathbf{z}}_B.$$

Recalling that

$${}^W\mathbf{R}_B = \begin{bmatrix} {}^W\hat{\mathbf{x}}_B & {}^W\hat{\mathbf{y}}_B & {}^W\hat{\mathbf{z}}_B \end{bmatrix},$$

the derivative of the rotation matrix follows as

$$\begin{aligned} {}^W\dot{\mathbf{R}}_B &= \begin{bmatrix} {}^W\dot{\hat{\mathbf{x}}}_B & {}^W\dot{\hat{\mathbf{y}}}_B & {}^W\dot{\hat{\mathbf{z}}}_B \end{bmatrix}, \\ &= \begin{bmatrix} {}^W\boldsymbol{\omega} \times {}^W\hat{\mathbf{x}}_B & {}^W\boldsymbol{\omega} \times {}^W\hat{\mathbf{y}}_B & {}^W\boldsymbol{\omega} \times {}^W\hat{\mathbf{z}}_B \end{bmatrix}. \end{aligned}$$

Making use of our skew-symmetric form for the cross product, namely

$$\boldsymbol{\omega} \times \mathbf{r} = \lfloor \boldsymbol{\omega} \rfloor \mathbf{r},$$

we can express the rotation matrix derivative as

$$\begin{aligned} {}^{W}\dot{\mathbf{R}}_{B} &= \begin{bmatrix} {}^{W}\boldsymbol{\omega} \times {}^{W}\hat{\mathbf{x}}_{B} & {}^{W}\boldsymbol{\omega} \times {}^{W}\hat{\mathbf{y}}_{B} & {}^{W}\boldsymbol{\omega} \times {}^{W}\hat{\mathbf{z}}_{B} \end{bmatrix}, \\ &= \lfloor {}^{W}\boldsymbol{\omega} \rfloor \begin{bmatrix} {}^{W}\hat{\mathbf{x}}_{B} & {}^{W}\hat{\mathbf{y}}_{B} & {}^{W}\hat{\mathbf{z}}_{B} \end{bmatrix}, \\ &= \lfloor {}^{W}\boldsymbol{\omega} \rfloor {}^{W}\mathbf{R}_{B}. \end{aligned}$$

This result is quite interesting, as it suggests that the rate of change of our orientation encoder will be a function of both the angular velocity vector and current orientation. It can be shown that for any $\mathbf{R} \in \text{SO(3)}$ and $\boldsymbol{\omega} \in \mathbb{R}^3$,

$$\lfloor \mathbf{R}\boldsymbol{\omega} \rfloor = \mathbf{R} \lfloor \boldsymbol{\omega} \rfloor \mathbf{R}^{T}.$$

Recalling that we can relate the angular velocity between different frames using

$${}^{W}\boldsymbol{\omega} = {}^{W}\mathbf{R}_{B} {}^{B}\boldsymbol{\omega},$$

and this, along with the result above of $\lfloor \mathbf{R}\boldsymbol{\omega} \rfloor = \mathbf{R} \lfloor \boldsymbol{\omega} \rfloor \mathbf{R}^{T}$ can be used to determine the rotation matrix derivative as a function of ${}^{B}\boldsymbol{\omega}$:

$$\begin{aligned} {}^{W}\dot{\mathbf{R}}_{B} &= \lfloor {}^{W}\boldsymbol{\omega} \rfloor {}^{W}\mathbf{R}_{B}, \\ &= \lfloor {}^{W}\mathbf{R}_{B} {}^{B}\boldsymbol{\omega} \rfloor {}^{W}\mathbf{R}_{B}, \\ &= {}^{W}\mathbf{R}_{B} \lfloor {}^{B}\boldsymbol{\omega} \rfloor {}^{W}\mathbf{R}_{B}^{T} {}^{W}\mathbf{R}_{B}, \\ &= {}^{W}\mathbf{R}_{B} \lfloor {}^{B}\boldsymbol{\omega} \rfloor. \end{aligned}$$

Note the similarity when determining ${}^{W}\dot{\mathbf{R}}_{B}$ as a function of ${}^{W}\boldsymbol{\omega}$ and ${}^{B}\boldsymbol{\omega}$.

3.5.2 Quaternion derivative

The relationship between angular velocity and the quaternion follows a similar pattern to that of the rotation matrix counterpart. The full derivation will be spared here for sake of time. Analogous to the result of ${}^{W}\dot{\mathbf{R}}_{B} = \lfloor {}^{W}\boldsymbol{\omega} \rfloor {}^{W}\mathbf{R}_{B}$, the quaternion derivative can be written as

$${}^{W}\dot{\mathbf{q}}_{B} = \frac{1}{2} {}^{W}\boldsymbol{\Omega} \otimes {}^{W}\mathbf{q}_{B},$$

where

$${}^{W}\boldsymbol{\Omega} = \begin{bmatrix} 0 \\ {}^{W}\boldsymbol{\omega} \end{bmatrix}$$

is a "pure" quaternion formed from the angular velocity vector in frame $\{W\}$. Using appropriate identities, the quaternion derivative can be written as

$${}^{W}\dot{\mathbf{q}}_{B} = \frac{1}{2} {}^{W}\mathbf{q}_{B} \otimes {}^{B}\boldsymbol{\Omega},$$

where

$$^{B}\boldsymbol{\Omega} = \begin{bmatrix} 0 \\ ^{B}\boldsymbol{\omega} \end{bmatrix},$$

which follows a similar structure but now using the angular velocity in $\{B\}$.

3.6 Obtaining orientation from angular velocity

If the angular velocity is known exactly, we can use this information to determine the corresponding orientation at a given time using integration. We must, however, take into account the particular reference frame mapping, such as from $\{B\}$ to $\{W\}$ when the velocity is described in $\{B\}$, before performing the integration operation — otherwise the resulting "orientation" described in $\{B\}$ is meaningless in general.

3.6.1 Rotation matrix formulation

The incremental orientation can be determined in rotation matrix form using exponential coordinates. Recall that

$$\mathbf{R} = \mathbf{e}^{\lfloor \alpha \hat{\mathbf{n}} \rfloor} = \mathbf{e}^{\lfloor \hat{\mathbf{n}} \rfloor \alpha},$$

where $\alpha\hat{\mathbf{n}}$ are the exponential coordinates describing a generalised 3D rotation. Assuming a body experiences an angular velocity vector of $^{B}\boldsymbol{\omega}$, expressed in $\{B\}$, for a duration of Δt units of time, we can determine the incremental exponential coordinates corresponding to rotating about vector $^{B}\boldsymbol{\omega}$ for Δt units of time (usually seconds) using

$$^{B}\boldsymbol{\theta} = {}^{B}\boldsymbol{\omega}\Delta t.$$

The incremental rotation angle and normalised rotation vector follow respectively as

$$\Delta\alpha = |^{B}\boldsymbol{\omega}|\Delta t,$$

$$^{B}\hat{\boldsymbol{\omega}} = \frac{^{B}\boldsymbol{\omega}}{|^{B}\boldsymbol{\omega}|}.$$

The corresponding incremental rotation matrix is given by

$$\begin{aligned} \Delta\mathbf{R} &= \mathbf{e}^{\lfloor {}^{B}\boldsymbol{\theta} \rfloor}, \\ &= \mathbf{e}^{\lfloor {}^{B}\boldsymbol{\omega}\Delta t \rfloor}, \\ &= \mathbf{I} + \sin\Delta\alpha \lfloor {}^{B}\hat{\boldsymbol{\omega}} \rfloor + (1 - \cos\Delta\alpha)\lfloor {}^{B}\hat{\boldsymbol{\omega}} \rfloor^{2}, \\ &= \mathbf{I} + \sin(|{}^{B}\boldsymbol{\omega}|\Delta t)\lfloor {}^{B}\hat{\boldsymbol{\omega}} \rfloor + (1 - \cos(|{}^{B}\boldsymbol{\omega}|\Delta t))\lfloor {}^{B}\hat{\boldsymbol{\omega}} \rfloor^{2}. \end{aligned}$$

If our orientation is originally described as $^{W}\mathbf{R}_{\{0\}}$ prior to experiencing the angular velocity, we can then determine our updated orientation, $^{W}\mathbf{R}_{\{1\}}$, by post-multiplying $^{W}\mathbf{R}_{\{0\}}$ by the incremental rotation matrix, namely

$$^{W}\mathbf{R}_{\{1\}} = {}^{W}\mathbf{R}_{\{0\}}{}^{\{0\}}\mathbf{R}_{\{1\}},$$

where $\Delta\mathbf{R} = {}^{\{0\}}\mathbf{R}_{\{1\}}$ is the incremental orientation as a result of the body-frame angular velocity. This process can be repeated any number of times, assuming one is able to capture each incremental rotation matrix that corresponds to the angular velocity that was experienced for the given time frame. This is generalised as

$$ {}^{W}\mathbf{R}_{\{n\}} = {}^{W}\mathbf{R}_{\{0\}}{}^{\{0\}}\mathbf{R}_{\{1\}} \dots {}^{\{n-1\}}\mathbf{R}_{\{n\}}. $$

Note that we are post-multiplying in each instance, as the rotation vector is described in $\{B\}$. If instead we had information of ${}^{W}\boldsymbol{\omega}$, we would need to pre-multiply in each instance.

3.6.2 Quaternion formulation

Recall that the quaternion encodes the exponential coordinates. Given our previously defined incremental axis-angle representation from Section 3.6.1 of

$$ \Delta\alpha = |{}^{B}\boldsymbol{\omega}|\Delta t, \qquad {}^{B}\widehat{\boldsymbol{\omega}} = \frac{{}^{B}\boldsymbol{\omega}}{|{}^{B}\boldsymbol{\omega}|}, $$

we can determine the corresponding incremental quaternion as

$$ \Delta\mathbf{q} = \begin{bmatrix} \cos\frac{\Delta\alpha}{2} \\ \sin\frac{\Delta\alpha}{2}\,{}^{B}\widehat{\boldsymbol{\omega}} \end{bmatrix} = \begin{bmatrix} \cos\frac{|{}^{B}\boldsymbol{\omega}|\Delta t}{2} \\ \sin\frac{|{}^{B}\boldsymbol{\omega}|\Delta t}{2}\,\widehat{\omega}_x \\ \sin\frac{|{}^{B}\boldsymbol{\omega}|\Delta t}{2}\,\widehat{\omega}_y \\ \sin\frac{|{}^{B}\boldsymbol{\omega}|\Delta t}{2}\,\widehat{\omega}_z \end{bmatrix}. $$

If our orientation is originally described by ${}^{W}\mathbf{q}_{\{0\}}$ prior to experiencing the angular velocity, we can then determine our updated orientation, ${}^{W}\mathbf{q}_{\{1\}}$, by post-multiplying ${}^{W}\mathbf{q}_{\{0\}}$ by the incremental quaternion, namely

$$ {}^{W}\mathbf{q}_{\{1\}} = {}^{W}\mathbf{q}_{\{0\}} \otimes {}^{\{0\}}\mathbf{q}_{\{1\}}, $$

where $\Delta\mathbf{q} = {}^{\{0\}}\mathbf{q}_{\{1\}}$ is the incremental orientation as a result of the body-frame angular velocity. This process can be repeated any number of times, assuming one is able to capture each incremental rotation matrix that corresponds to the angular velocity that was experienced for the given time frame. This is expressed as

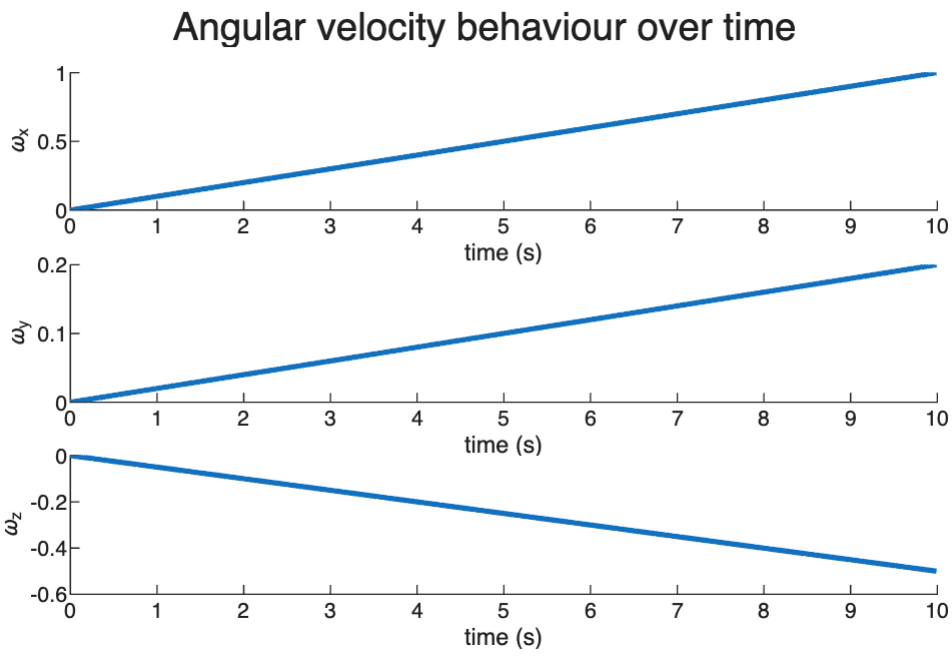
$$ {}^{W}\mathbf{q}_{\{n\}} = {}^{W}\mathbf{q}_{\{0\}} \otimes {}^{\{0\}}\mathbf{q}_{\{1\}} \otimes \dots \otimes {}^{\{n-1\}}\mathbf{q}_{\{n\}}. $$

As with the rotation matrix formulation, we are post-multiplying in each instance, as the rotation vector is described in $\{B\}$. If instead we had information of ${}^{W}\boldsymbol{\omega}$, we would need to pre-multiply in each instance.

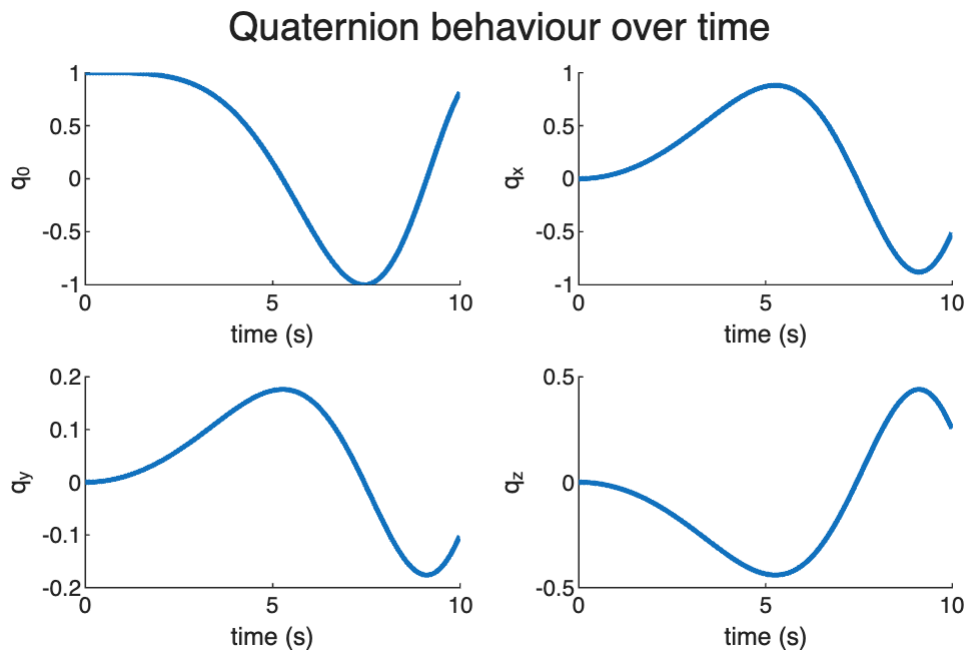
Example: Determining quaternion-based orientation from angular velocity

The code block below provides a simulation of how angular velocity information can be correctly integrated and mapped to obtain a quaternion-based representation of orientation.

```
dT = 0.01;  %sampling duration
numSamples = 1000;      %number of samples
t = (0:numSamples-1)*dT;    %time vector

omega1 = linspace(0,1,numSamples);
omega2 = linspace(0,0.2,numSamples);
omega3 = linspace(0,-0.5,numSamples);
omega = [omega1' omega2' omega3'];  %generate arbitrary 3D angular velocity


q = zeros(numSamples,4);    %initialise quaternion array
q(1,1:4) = [1 0 0 0];       %initialise first sample of quaternion to be null
quaternion

for k=1:numSamples-1
    %actual
    dtheta = omega(k,:)*dT; %determine incremental exponential coordinates
    dalpha = norm(dtheta);  %determine incremental rotation angle
    if( dalpha == 0)        %determine incremental rotation vector
        n = [0 0 0];
    else
        n = dtheta/dalpha;
    end
    dq = [cos(dalpha) sin(dalpha)*n];   %populate incremental quaternion
    q(k+1,1:4) = quatmultiply(q(k,1:4),dq); %calculate updated quaternion at sample
instance k+1
    q(k+1,1:4) = q(k+1,1:4)/norm( q(k+1,1:4) ); %normalise quaternion
end
%plot angular velocity figures
figure, sgtitle('Angular velocity behaviour over time')
subplot(3,1,1),hold on
plot(t,omega(:,1),lineWidth=2),xlabel('time (s)'),ylabel('\omega_x')
subplot(3,1,2),hold on
plot(t,omega(:,2),lineWidth=2),xlabel('time (s)'),ylabel('\omega_y')
subplot(3,1,3),hold on
plot(t,omega(:,3),lineWidth=2),xlabel('time (s)'),ylabel('\omega_z')
```




```
%plot quaternion figures
figure, sgtitle('Quaternion behaviour over time')
subplot(2,2,1),hold on
plot(t,q(:,1),lineWidth=2),xlabel('time (s)'),ylabel('q_0')
subplot(2,2,2),hold on
plot(t,q(:,2),lineWidth=2),xlabel('time (s)'),ylabel('q_x')
subplot(2,2,3),hold on
plot(t,q(:,3),lineWidth=2),xlabel('time (s)'),ylabel('q_y')
subplot(2,2,4),hold on
plot(t,q(:,4),lineWidth=2),xlabel('time (s)'),ylabel('q_z')
```



3.7 Exponential coordinates in SE(3)

3.7.1 Definition

Recall that the exponential map is used to map exponential coordinates to rotation matrices using

$$\mathbf{R} = \mathrm{e}^{\lfloor \hat{\mathbf{n}} \rfloor \alpha} = I + \sin\alpha \lfloor \hat{\mathbf{n}} \rfloor + (1 - \cos\alpha) \lfloor \hat{\mathbf{n}} \rfloor^2.$$

Inversely, rotation matrices can be mapped to exponential coordinates in SO(3) using the (matrix) logarithmic mapping,

$$\log \mathbf{R} = \lfloor \hat{\mathbf{n}} \rfloor \alpha.$$

Analogous to how exponential coordinates of rotations can be mapped to/from rotation matrices (or quaternions), there exist exponential coordinates of pose (position and orientation) that can be mapped to transformation matrices. The exponential coordinates in SE(3) are defined as

$$\boldsymbol{\xi} = \begin{bmatrix} \boldsymbol{\theta} \\ \Delta\mathbf{p} \end{bmatrix} = \begin{bmatrix} \alpha\hat{\mathbf{n}} \\ \Delta\mathbf{p} \end{bmatrix},$$

where $\boldsymbol{\theta} = \alpha\hat{\mathbf{n}}$ represents the rotational exponential coordinates, based on Section 3.4, and $\Delta\mathbf{p}$ describes the translational exponential coordinates, which is the displacement experienced within the vector space. The vector above is often also referred to as the twist vector.

Importantly, this information will have different geometric meanings depending on in which frame it is described in. The tangent space representation of the exponential coordinates in SE(3) is given by

$$
\begin{aligned}
\lfloor \boldsymbol{\xi} \rfloor &= \begin{bmatrix} \lfloor \boldsymbol{\theta} \rfloor & \Delta\mathbf{p} \\ \mathbf{0} & 1 \end{bmatrix}, \\
&= \begin{bmatrix} \lfloor \alpha \widehat{\mathbf{n}} \rfloor & \Delta\mathbf{p} \\ \mathbf{0} & 1 \end{bmatrix},
\end{aligned}
$$

which can be thought of as the encoded matrix form of the exponential coordinates. Finally, the transformation matrix describing the pose is obtained by mapping the tangent space representation through the exponential mapping:

$$
\begin{aligned}
\mathbf{T} &= \mathbf{e}^{\lfloor \boldsymbol{\xi} \rfloor} \\
&= \mathbf{e}^{\begin{bmatrix} \lfloor \boldsymbol{\theta} \rfloor & \Delta\mathbf{p} \\ \mathbf{0} & 1 \end{bmatrix}}.
\end{aligned}
$$

3.7.2 Left Jacobian of SO(3)

To avoid having to mathematically evaluate the matrix exponential every time we need to evaluate the equation above, we can instead make use of a compact closed form solution. Recall that the exponential mapping in SO(3) is given by

$$
\mathbf{R} = \mathrm{e}^{\lfloor \widehat{\mathbf{n}} \rfloor \alpha} = I + \sin\alpha \lfloor \widehat{\mathbf{n}} \rfloor + (1 - \cos\alpha) \lfloor \widehat{\mathbf{n}} \rfloor^2,
$$

which provides us a direct means of determining the rotation matrix if we know the rotational exponential coordinates. Analogous to this, the transformation matrix can be determined by

$$
\begin{aligned}
\mathbf{T} &= \mathbf{e}^{\lfloor \boldsymbol{\xi} \rfloor}, \\
&= \mathbf{e}^{\begin{bmatrix} \lfloor \boldsymbol{\theta} \rfloor & \Delta\mathbf{p} \\ \mathbf{0} & 0 \end{bmatrix}}, \\
&= \begin{bmatrix} \mathbf{R} & \mathbf{V}\Delta\mathbf{p} \\ \mathbf{0} & 0 \end{bmatrix},
\end{aligned}
$$

where $\mathbf{V}$ is known as the left Jacobian of SO(3) and is given by

$$
\mathbf{V} = \mathbf{I} + \frac{\mathbf{I} - \mathbf{R}}{\alpha} \lfloor \widehat{\mathbf{n}} \rfloor,
$$

and

$$
\mathbf{R} = \mathrm{e}^{\lfloor \widehat{\mathbf{n}} \rfloor \alpha} = I + \sin\alpha \lfloor \widehat{\mathbf{n}} \rfloor + (1 - \cos\alpha) \lfloor \widehat{\mathbf{n}} \rfloor^2.
$$

Importantly, matrix $\mathbf{V}$ accounts for the fact that the robot is rotating *while* it is translating, creating a curved path rather than a straight line. The combination of Rodrigues' formula and the left Jacobian of SO(3) means that the transformation matrix can be easily determined if one knows the twist vector.

3.7.3 Body twist

In order to make use of the exponential coordinates in SE(3) in a meaningful way, we need to be aware of which reference frame is being used to represent the information. In this course we will predominantly base

exponential coordinates in SE(3) in $\{B\}$. As such, we can represent the body-frame 6-vector exponential coordinates using

$$ {}^{B}\boldsymbol{\xi} = \begin{bmatrix} {}^{B}\boldsymbol{\theta} \\ {}^{B}\Delta\mathbf{p} \end{bmatrix} = \begin{bmatrix} \alpha {}^{B}\widehat{\mathbf{n}} \\ {}^{B}\Delta\mathbf{p} \end{bmatrix}. $$

Physically, we can interpret the vector above as describing a rigid body that has simultaneously experienced a rotation of α units about vector ${}^{B}\widehat{\mathbf{n}}$ and a displacement of the body-frame origin of ${}^{B}\Delta\mathbf{p}$ units.

The usefulness of this formulation is that if the robot has some initial pose represented by ${}^{W}\mathbf{T}_{B_0}$, and then subsequently experiences incremental motion as described in the equation above, we can then (exactly) update the pose representation of our robot using

$$ {}^{W}\mathbf{T}_{B} = {}^{W}\mathbf{T}_{B_0}\mathbf{e}^{\lfloor {}^{B_0}\boldsymbol{\xi} \rfloor}. $$

Note that this approach is analogous to how we incrementally updated the rotation matrix as shown in Section 3.6.1.

Kinematically, the exponential coordinates are derived by obtaining the body-frame velocity information and then integrating the (assumed) constant velocity over the particular time frame. The 6-vector describing the body-frame velocity is commonly known as the **body twist** and is given by

$$ {}^{B}\mathscr{V} = \begin{bmatrix} {}^{B}\boldsymbol{\omega} \\ {}^{B}\mathbf{v} \end{bmatrix}, $$

where ${}^{B}\boldsymbol{\omega}$ is the body-frame angular velocity of the rigid body, and ${}^{B}\mathbf{v}$ is the body-frame translational velocity. The body twist can then by related to the exponential coordinates in SE(3) using

$$ {}^{B}\boldsymbol{\xi} = {}^{B}\mathscr{V}\Delta t = \begin{bmatrix} {}^{B}\boldsymbol{\omega} \\ {}^{B}\mathbf{v} \end{bmatrix}\Delta t, $$

where Δt is the duration of time in which the body twist was experienced.

Example: Transformation matrix from body twist

Using MATLAB, determine the transformation matrix representing the pose of $\{B\}$ with respect to $\{W\}$ when a rigid body initial aligned with $\{W\}$ experiences a translational velocity of ${}^{B}\mathbf{v}$ and a rotational velocity of ${}^{B}\boldsymbol{\omega}$ over a one second period.

```
vx = 0;
vy = 0;
vz = 0;
v = [vx vy vz]'; %body translational velocity vector

wx = -1;
wy = 0;
wz = 0;
```

```
w = [wx wy wz]'; %body rotational velocity vector

dT = 1; %sample time
twist = [w; v]; %body twist
expCoords = twist*dT; %exponential coordinates

S = [0           -expCoords(3)  expCoords(2) expCoords(4);    %tangent space
representation of exponential coordinates
     expCoords(3) 0            -expCoords(1) expCoords(5);
    -expCoords(2) expCoords(1)  0            expCoords(6)
     0             0            0            0];

T = expm(S)
```

```
T = 4×4
    1.0000         0         0         0
         0    0.5403    0.8415         0
         0   -0.8415    0.5403         0
         0         0         0    1.0000
```

```
%plot figures
figure
subplot(1,2,1),hold on
plotTransforms([0 0 0],[1 0 0 0],"MeshFilePath",'multirotor.stl',"MeshColor","red")
axis equal,title('Before rotation and translation')

subplot(1,2,2)
plotTransforms(tform2trvec(T),tform2quat(T),"MeshFilePath",'multirotor.stl',"MeshCol
or","blue")
axis equal,title('After rotation and translation')

sgtitle('Visualising a rotated and translated reference frame using exponential
coordinates')
```

ng a rotated and translated reference frame using exponential coc