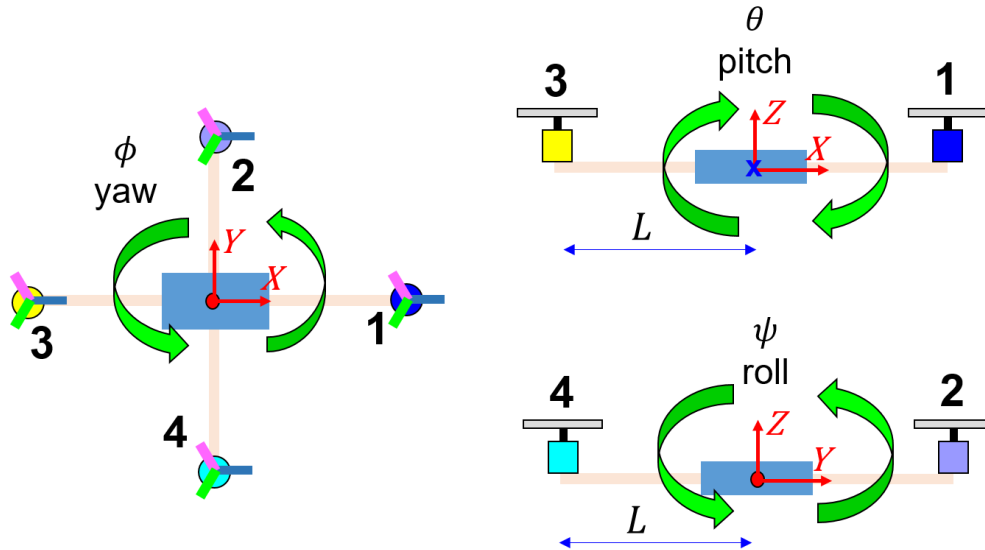


Task: The time derivative of the DCM

In this task, we're going to explore how to compute the time derivative of a Direction Cosine Matrix (DCM). By knowing how to do this, we then have an approach for propagating the angular pose of a rigid body that does NOT suffer from the singularities that afflict the propagation of Euler angles. The fundamental relationship that we'll derive is this one:

$${}^G\dot{R}_B(t) = [S({}^G\omega_B(t))] \cdot {}^G R_B(t)$$

At the end of this task we'll demonstrate the application of this expression using a mathematical model of a quadcopter.



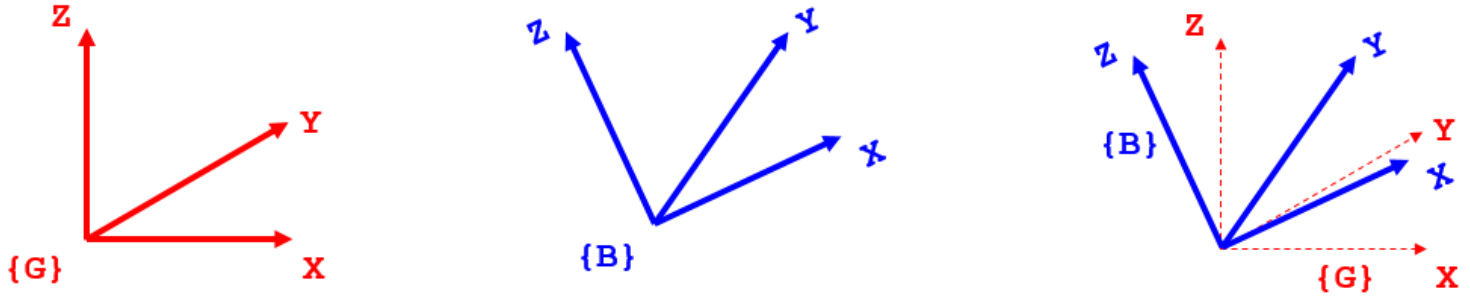
Motivation:

A common approach for integrating the equations of motion (Newton's) of a quadcopter, is to convert the vehicles angular velocity vector ${}^B\omega_B$ into it's corresponding Euler rates. These Euler rates then get integrated to produce the angular orientation of the vehicle. A problem arises however in that the conversion of ${}^B\omega_B$ to Euler rates contains a singularity at specific poses (eg: a ZYX Euler sequence has such a singularity at a pitch angle of 90 degrees). An alternate approach then to propagating the vehicle's angular pose, is to integrate the DCM matrix directly. This approach does NOT suffer from the singularity problem associated with integrating Euler rates.

Bradley Horton : 08-Aug-2016, bradley.horton@mathworks.com.au

The Background:

Say we have an inertial frame referred to as the G-frame. And we also have a body fixed frame referred to as the B-frame. The B-frame is rotating relative to the G-frame. The origins of the G-frame and B-frame are always co-incident.



We can convert vectors expressed in each frame, into their corresponding components in the other frame, using Direction Cosine Matrices, eg:

- ${}^B\mathbf{v} = {}^B\mathbf{R}_G \cdot {}^G\mathbf{v}$
- ${}^G\mathbf{v} = {}^G\mathbf{R}_B \cdot {}^B\mathbf{v}$

In this tutorial we're going to derive the expression for:

- ${}^G\dot{\mathbf{R}}_B(t)$

Some Nomenclature:

Define some nomenclature for the body B-frame:

- ${}^B\mathbf{p}$: the position of point p , expressed in components of the B-frame.
- ${}^B\mathbf{v}_p$: the **velocity** of point p , relative to the G-frame and expressed in components of the B-frame.
- ${}^B\mathbf{v}$: short hand notation for ${}^B\mathbf{v}_p$
- ${}^B_G\boldsymbol{\omega}_B$: the **angular velocity** of the B-frame relative to the G-frame and expressed in components of the B-frame.
- ${}^B\boldsymbol{\omega}$: short hand notation for ${}^B_G\boldsymbol{\omega}_B$

Define some nomenclature for the inertial G-frame:


- ${}^G\mathbf{p}$: the position of point p , expressed in components of the G-frame.
- ${}^G\mathbf{v}_p$: the **velocity** of point p , relative to the G-frame and expressed in components of the G-frame.
- ${}^G\mathbf{v}$: short hand notation for ${}^G\mathbf{v}_p$
- ${}^G_G\boldsymbol{\omega}_B$: the **angular velocity** of the B-frame relative to the G-frame and expressed in components of the G-frame.
- ${}^G\boldsymbol{\omega}$: short hand notation for ${}^G_G\boldsymbol{\omega}_B$

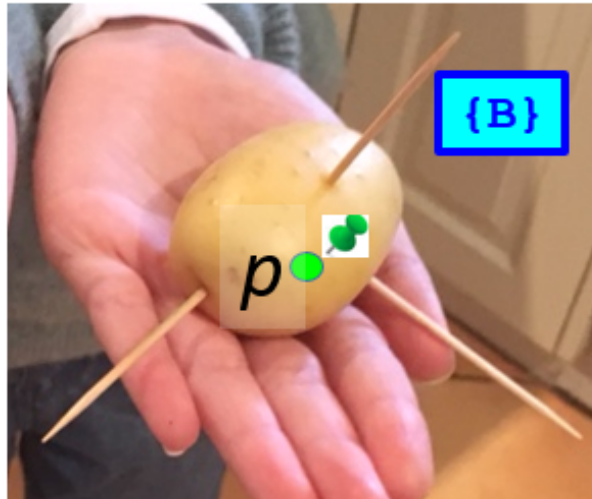
Define some nomenclature for converting vectors between the B and G-frames and vice versa:

- ${}^B\mathbf{v} = {}^B\mathbf{R}_G \cdot {}^G\mathbf{v}$
- ${}^G\mathbf{v} = {}^G\mathbf{R}_B \cdot {}^B\mathbf{v}$

- ${}^G R_B = {}^B R_G^{-1} = {}^B R_G^T$

The point p :

Consider a rigid body such as the humble potatoe. We attach a body fixed frame to this body and call it the B-frame. The B-frame rotates with the body - it is fixed to the body. Consider also a fixed point on the potatoe, and let's call that point, p . Relative to the B-frame, the point p does NOT move, eg: imagine that point p is a "thumb tac"  that has been pushed into the potate.



Let's find ${}^G \dot{R}_B(t)$:

Let's say that at time $t = 0$, the B and G frames are co-incident. Now since point p is "fixed" in the B-frame (ie: point p canNOT move relative to the B-frame), then we can write that:

$${}^G p(0) = {}^B p(t) = {}^B R_G \cdot {}^G p(t) = {}^G R_B^T \cdot {}^G p(t)$$

The above result will prove useful in a moment when we decide to take the derivative of ${}^G p(t)$. Now let's use the above result to write an expression for ${}^G p(t)$:

$${}^G p(t) = {}^G R_B(t) \cdot {}^B p(t)$$

$${}^G p(t) = {}^G R_B(t) \cdot {}^G p(0)$$

Note how all of the p-terms in the above expression are expressed in components of the G-frame. So let's take the time derivative of ${}^G p(t)$, where the derivative is taken with respect to the G-frame.

$${}^G \dot{V}_p = {}^G \frac{d}{dt} ({}^G p(t)) = {}^G \frac{d}{dt} ({}^G R_B(t) \cdot {}^G p(0)) = {}^G \frac{d}{dt} ({}^G R_B(t)) \cdot {}^G p(0) = {}^G \dot{R}_B \cdot {}^G p(0)$$

which we can now write as:

$$\begin{aligned}\dot{{}^G_V p} &= \dot{{}^G_R_B} \cdot {}^G p(0) \\ \dot{{}^G_V p} &= \dot{{}^G_R_B} \cdot {}^G R_B^T \cdot {}^G p(t)\end{aligned}$$

Let's define:

$$S({}^G\omega_B) = \dot{{}^G_R_B} \cdot {}^G R_B^T \text{ where } {}^G\omega_B = \begin{pmatrix} {}^G\omega_{Bx} \\ {}^G\omega_{By} \\ {}^G\omega_{Bz} \end{pmatrix} = \begin{pmatrix} {}^G\omega_x \\ {}^G\omega_y \\ {}^G\omega_z \end{pmatrix}$$

Note that the above equation, can be restated as: $\dot{{}^G_R_B}(t) = [S({}^G\omega_B(t))] \cdot {}^G_R_B(t)$, which is a BIG result !!

Let's continue exploring our $\dot{{}^G_V p}$ expression:

$$\dot{{}^G_V p} = S({}^G\omega_B) \cdot {}^G p(t)$$

Note that $S({}^G\omega_B)$ is actually a SKEW symmetric matrix, ie: $S({}^G\omega_B)^T = -1 * S({}^G\omega_B)$ - this is proven in **Appendix A**. And we can use this fact to rewrite the matrix equation into a vector CROSS product. To do this, we'll make the following definition:

$$S({}^G\omega_B) = \begin{pmatrix} 0 & , & -{}^G\omega_z & , & {}^G\omega_y \\ {}^G\omega_z & , & 0 & , & -{}^G\omega_x \\ -{}^G\omega_y & , & {}^G\omega_x & , & 0 \end{pmatrix}$$

And with this definition of $S({}^G\omega_B)$ we can now write:

$$\dot{{}^G_V p} = {}^G\omega_B \times {}^G p(t) \text{ where } " \times " \text{ denotes the vector cross product}$$

Let's confirm this last step with MATLAB:

```
syms omega_x omega_y omega_z
syms a b c
```

```
S = [      0,      -1*omega_z,      omega_y;
      omega_z,      0,      -1*omega_x;
     -1*omega_y,      omega_x,      0 ]
```

S =

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

```
p = [a,b,c].';
```

```
omega = [omega_x omega_y omega_z].';
```

So here's the result of doing ${}^G_V p = S({}^G\omega_B) \cdot {}^G p(t)$

```
mat_mult_result = S * p
```

```
mat_mult_result =
```

$$\begin{pmatrix} c \omega_y - b \omega_z \\ a \omega_z - c \omega_x \\ b \omega_x - a \omega_y \end{pmatrix}$$

And here's the result of doing ${}^G_V p = {}^G\omega_B \times {}^G p(t)$

```
vec_cross_result = cross(omega, p)
```

```
vec_cross_result =
```

$$\begin{pmatrix} c \omega_y - b \omega_z \\ a \omega_z - c \omega_x \\ b \omega_x - a \omega_y \end{pmatrix}$$

The BIG result:

One of the BIG results that we derived in the previous section was this one:

$$S({}^G\omega_B) = \dot{{}^G R_B} \cdot {}^G R_B^T \text{ where } {}^G\omega_B = \begin{pmatrix} {}^G\omega_{Bx} \\ {}^G\omega_{By} \\ {}^G\omega_{Bz} \end{pmatrix} = \begin{pmatrix} {}^G\omega_x \\ {}^G\omega_y \\ {}^G\omega_z \end{pmatrix}$$

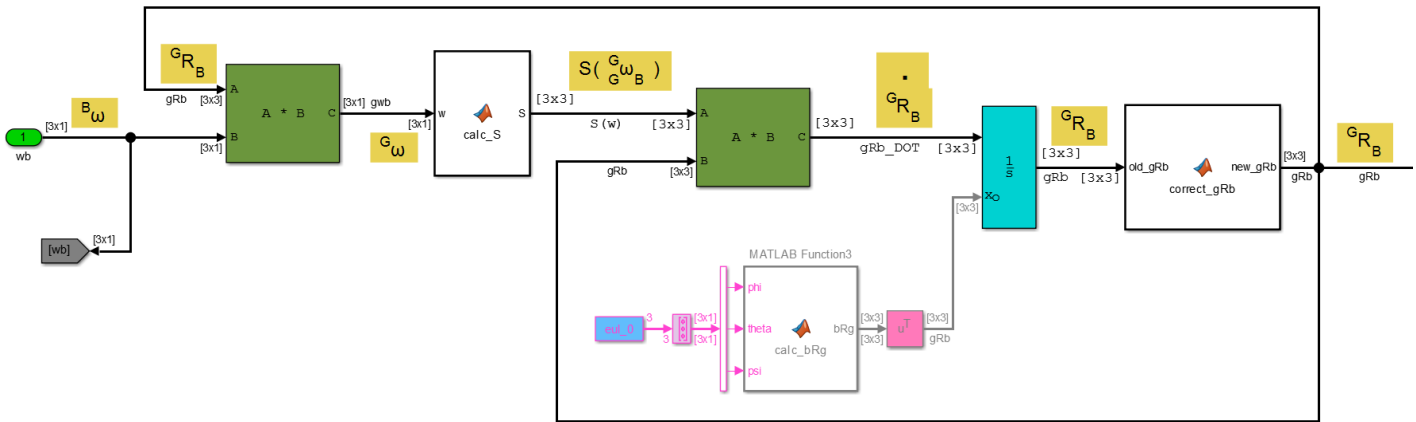
Or, after some simple rearrangement:

$$\dot{{}^G R_B}(t) = [S({}^G\omega_B(t))] \cdot {}^G R_B(t)$$

This equation tells us how to compute the derivative of the Direction Cosine Matrix ${}^G R_B$ if we know the angular velocity of the B-frame relative to the G-frame and expressed in components of the G-frame, ie: ${}^G\omega_B$.

The implementation of the BIG result: - part 1

The significance of this expression for $\dot{{}^G R_B}$, is that we can use it as a mechanism to propagate the orientation of a rigid body. Using this approach there are NO singularities that would prevent us from computing ${}^G R_B(t)$. The implementation of this algorithm would look something like this:



Note that to compute ${}^G R_B(t = 0)$, we could let the user specify an initial Euler angle configuration. Also, the `calc_s` function would be:

```

1  function S = calc_S(w)
2  %#codegen
3
4  x = w(1);
5  y = w(2);
6  z = w(3);
7
8  S = [ 0, -z, y;
9        z, 0, -x;
10       -y, x, 0;
11       ];

```

When we integrate $\dot{{}^G R}_B$, we would expect that the orthogonality relationship of ${}^G R_B(t) \cdot {}^G R_B(t)^T = I$ would still be true. And it almost is ! The combination of "numerically" integrating a derivative and the finite word size of computer calculations, means that we end up with a relationship that is more like this:

$${}^G R_B(t) \cdot {}^G R_B(t)^T = I + \delta e, \text{ where } \delta e \text{ is a matrix of small terms}$$

We can reduce this δe term using a correction technique such as the one proposed by William Premerlani and Paul Bizard (see: [HERE](#)).

```

1  function new_gRb = correct_gRb(old_gRb)
2      %#codegen
3
4      % The correction technique implementde here is the one described by
5      % William Premerlani and Paul Bizard:
6      % see: https://gentlenav.googlecode.com/files/DCMDraft2.pdf
7
8      row_1 = old_gRb(1,:);
9      row_2 = old_gRb(2,:);
10
11     % look at the error associated with row_1 and row_2 they are supposed to be
12     % orthogonal, ie: their DOT product should be ZERO
13     e = sum( row_1 .* row_2 );
14
15     % distrubute the error across X and Y
16     new_row_1 = row_1 - (e/2)*row_2;
17     new_row_2 = row_2 - (e/2)*row_1;
18
19     % compute Z
20     new_row_3 = cross( new_row_1, new_row_2 );
21
22     % normlaize all rows so that their MAG is UNITY
23     %: NOTE: norm([3 4]) is 5
24     new_row_1 = new_row_1 / norm(new_row_1);
25     new_row_2 = new_row_2 / norm(new_row_2);
26     new_row_3 = new_row_3 / norm(new_row_3);
27
28     % assemble the NEW gRb matrix
29     new_gRb = [new_row_1; new_row_2; new_row_3];
30
31 end

```

The implementation of the BIG result: - part 2

If you'd like to see how we actually use the relationship:

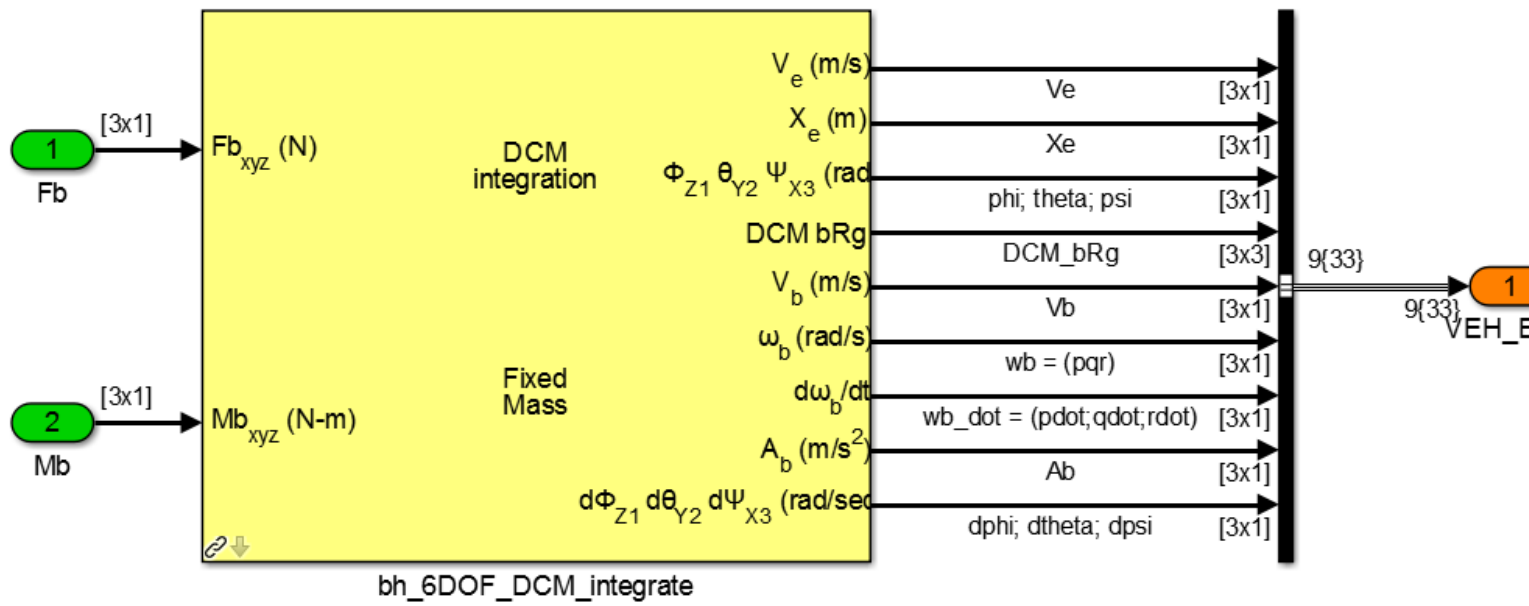
$${}^G\dot{R}_B(t) = [S({}^G\omega_B(t))] \cdot {}^G R_B(t)$$

then have a look at the Simulink model called `<bh_6dof_DCM_integrate.slx>`. In this model we integrate ${}^G\dot{R}_B$ to determine the angular orientation of the quadcopter.

```

clear; clc; bh_quad_params
open_system('bh_6dof_DCM_integrate')

```



Appendix A: The **skew symmetry** of ${}^G\dot{R}_B \cdot {}^G R_B^T$

Earlier in this tutorial we defined the following:

$$S({}_G^G\omega_B) = {}^G\dot{R}_B \cdot {}^G R_B^T \text{ where } {}_G^G\omega_B = \begin{pmatrix} {}^G\omega_{Bx} \\ {}^G\omega_{By} \\ {}^G\omega_{Bz} \end{pmatrix} = \begin{pmatrix} {}^G\omega_x \\ {}^G\omega_y \\ {}^G\omega_z \end{pmatrix}$$

and we made the statement that $S({}_G^G\omega_B)$ was a SKEW symmetric matrix, ie: $S({}_G^G\omega_B)^T = -1 * S({}_G^G\omega_B)$. So let's prove that now. What we want to show is that:

$$\bullet \quad ({}^G\dot{R}_B \cdot {}^G R_B^T)^T = -1 * ({}^G\dot{R}_B \cdot {}^G R_B^T)$$

We start with the identity:

$${}^G R_B(t) \cdot {}^G R_B(t)^T = I$$

and then take the derivative w.r.t t on both sides:

$${}^G\dot{R}_B \cdot {}^G R_B^T + {}^G R_B \cdot {}^G\dot{R}_B^T = 0$$

which then gives us:

$${}^G R_B \cdot {}^G\dot{R}_B^T = -1 * {}^G\dot{R}_B \cdot {}^G R_B^T$$

And we note that the LHS could be written as: ${}^G R_B \cdot {}^G \dot{R}_B^T \equiv ({}^G \dot{R}_B \cdot {}^G R_B^T)^T$ using the standard matrix algebra relationship that $(A \cdot B)^T = B^T \cdot A^T$. So we can finally write:

$$\bullet \quad ({}^G \dot{R}_B \cdot {}^G R_B^T)^T = -1 * ({}^G \dot{R}_B \cdot {}^G R_B^T)$$

Which is our proof that $S({}_G^G \omega_B)$ is SKEW symmetric.