Attitude control

Here we look at the steps from choosing attitude representation to attitude control strategy. We focus on the most common methods: Euler angles, rotation matrix, and quaternion.

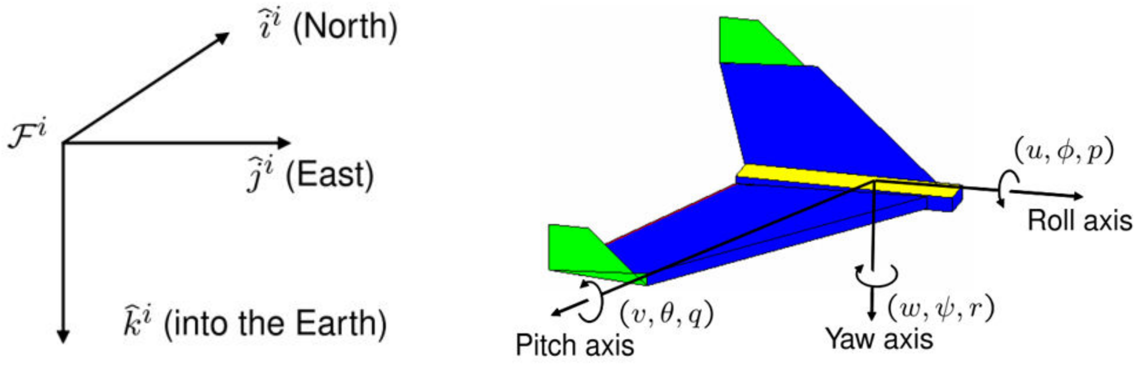
Note: You should read previous chapters on drone equations of motions and attitude representations before preoceeding.

Attitude representation

```
syms t
syms phi(t) theta(t) psi(t) p(t) q(t) r(t) % roll pitch yaw p q r
syms phi_dot(t)  theta_dot(t)  psi_dot(t) % euler rates
syms p_dot(t) q_dot(t) r_dot(t) % body rates
syms phi_t theta_t psi_t phi_dot_t theta_dot_t psi_dot_t % used to remove time dependence
```

Coordinate frames

- Inertial frame: NED. X-axis(roll) pointing forward(North), Y-axis(pitch) pointing right (east),Z-axis pointing down



Let's use the Z - Y - X Euler angle convention

```
R1_z = [
      cos(psi),  -sin(psi),   0;
     sin(psi),  cos(psi),   0;
         0   ,     0    ,   1;

    ];
R2_y = [
        cos(theta) ,   0     , sin(theta);
           0       ,   1     ,     0    ;
       -sin(theta)  ,   0     , cos(theta);

    ];
R3_x = [
        1   ,       0     ,     0      ;
        0   ,  cos(phi)   ,  -sin(phi);
        0   ,  sin(phi)  , cos(phi);
    ];
```

Body to world(inertial) frame rotation

```
wRb = R1_z*R2_y*R3_x % body to world
```

```
wRb(t) =
```

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

World to body frame rotation

```
bRw=transpose(wRb) % world to body
```

```
bRw(t) =
```

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

Euler angles from rotation matrix

$$\mathbf{u}_{123}(R)=\begin{bmatrix}\phi_{123}(R)\\ \theta_{123}(R)\\ \psi_{123}(R)\end{bmatrix}=\begin{bmatrix}\operatorname{atan2}\left(r_{23},r_{33}\right)\\ -\operatorname{asin}\left(r_{13}\right)\\ \operatorname{atan2}\left(r_{12},r_{11}\right)\end{bmatrix}$$

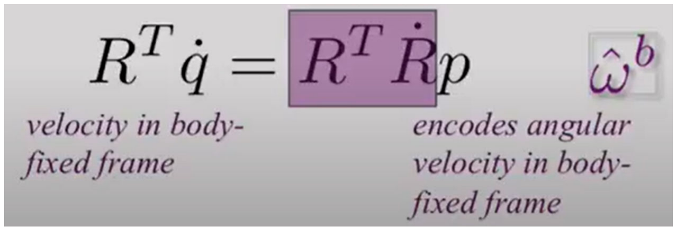
Generate matlab function and simulink block for attitude representation

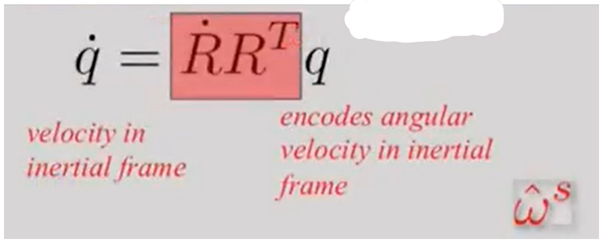
Rotation matrix

```
% remove time dependence form sym variables
cur_labels = [phi(t), theta(t), psi(t)];
new_labels = [phi_t, theta_t, psi_t];
bRw_new         = subs(bRw(t),cur_labels,new_labels);
wRb_new         = subs(wRb(t),cur_labels,new_labels);

INPUT_VAR_ORDER = {'phi_t', 'theta_t','psi_t'};
% matlab functions
matlabFunction(bRw_new,'File','generated/bRw', ...
              'Optimize',false, 'Vars', INPUT_VAR_ORDER);
matlabFunction(wRb_new,'File','generated/wRb', ...
              'Optimize',false, 'Vars', INPUT_VAR_ORDER);
% simulink blocks
SIM_FILENAME        = 'attitude_rep_blocks';
if exist(SIM_FILENAME,'file')==0
    new_system(SIM_FILENAME)
end
```

```
open_system(SIM_FILENAME)
BLOCK_NAME       = [SIM_FILENAME,'/bRw'];
matlabFunctionBlock(BLOCK_NAME, bRw_new, ...
                    'Optimize',false, 'Vars', INPUT_VAR_ORDER);
BLOCK_NAME       = [SIM_FILENAME,'/wRb'];
matlabFunctionBlock(BLOCK_NAME, wRb_new, ...
                    'Optimize',false, 'Vars', INPUT_VAR_ORDER);
```

Angular rates


$R^T \dot{q} = \boxed{R^T \dot{R}} p$
$\hat{\omega}^b$
velocity in body-fixed frame
encodes angular velocity in body-fixed frame


$\dot{q} = \boxed{\dot{R} R^T} q$
velocity in inertial frame
encodes angular velocity in inertial frame
$\hat{\omega}^s$

Angulare velocity in body-fixed frame

```
R = wRb;
R_dot = simplify(diff(R,t));
omega_b = simplify(R.'*R_dot);
omega_b = omega_b(t)
```

```
omega_b =
```

$$\left(\begin{array}{ll} 0 & \\ \cos(\phi(t))\cos(\theta(t))\frac{\partial}{\partial t}\psi(t) - \sin(\phi(t))\frac{\partial}{\partial t}\theta(t) & \\ -\cos(\phi(t))\frac{\partial}{\partial t}\theta(t) - \cos(\theta(t))\sin(\phi(t))\frac{\partial}{\partial t}\psi(t) & (\sin(\phi(t))\sin(\psi(t)) + \cos(\phi(t))\cos(\psi(t))\sin(\theta(t))) \end{array}\right.$$

Another manual way using $\omega_b = (R_z R_x R_y)^T * (\dot{R}_z R_x R_y + R_z \dot{R}_x R_y + R_z R_x \dot{R}_y)$

```
% another way to calculate it manually
w_b2 = simplify( (R1_z*R2_y*R3_x).'*(diff(R1_z,t)*R3_x*R2_y + R1_z*diff(R3_x,t)*R2_y + R1_z*R3_
```

```
w_b2(t) =
```

$$\left(\begin{array}{l} \\ (\cos(\phi(t))\cos(\psi(t)) + \sin(\phi(t))\sin(\psi(t))\sin(\theta(t)))\ \left(\cos(\psi(t))\cos(\theta(t))\frac{\partial}{\partial t}\psi(t) - \sin(\psi(t))\sin(\theta(t)\right. \\ -(\cos(\psi(t))\sin(\phi(t)) - \cos(\phi(t))\sin(\psi(t))\sin(\theta(t)))\ \left(\cos(\psi(t))\cos(\theta(t))\frac{\partial}{\partial t}\psi(t) - \sin(\psi(t))\sin(\theta(t\right. \end{array}\right.$$

Angulare velocity in inertial frame

```
omega_w = simplify(R_dot*R.');
omega_w = omega_w(t)
omega_w =
```

$$\left((\sin(\phi(t))\sin(\psi(t)) + \cos(\phi(t))\cos(\psi(t))\sin(\theta(t)))\ \left(-\cos(\phi(t))\cos(\psi(t))\frac{\partial}{\partial t}\ \phi(t) + \sin(\phi(t))\sin(\psi(\right.\right.$$

Euler rates to body rates conversion

Extract the vector from the skew-symmetric matrix (body-fixed frame)

```
w_x = simplify(omega_b(3,2));
w_y = simplify(omega_b(1,3));
w_z = simplify(omega_b(2,1));
wb = [w_x;w_y;w_z];
```

Substitute with shorter symbols

```
cur_labels = [phi(t), theta(t), psi(t),diff(phi(t), t), diff(theta(t), t),diff(psi(t), t) ];
new_labels = [phi_t, theta_t, psi_t, phi_dot_t, theta_dot_t, psi_dot_t];
wb_new = subs(wb,cur_labels,new_labels);
% simplify the expressions
wb_new = simplify(wb_new)
```

```
wb_new =
```

$$\begin{pmatrix} \dot{\phi}_t - \dot{\psi}_t \sin(\theta_t) \\ \dot{\theta}_t \cos(\phi_t) + \dot{\psi}_t \cos(\theta_t)\sin(\phi_t) \\ \dot{\psi}_t \cos(\phi_t)\cos(\theta_t) - \dot{\theta}_t \sin(\phi_t) \end{pmatrix}$$

Isolate the euler rates

```
syms p q r
x = [phi_dot_t, theta_dot_t, psi_dot_t].';
[A,b] = equationsToMatrix(  wb_new(1)==p, ...
                            wb_new(2)==q, ...
                            wb_new(3)==r, ...
                            x);
A
```

```
A =
```

$$\begin{pmatrix} 1 & 0 & -\sin(\theta_t) \\ 0 & \cos(\phi_t) & \cos(\theta_t)\sin(\phi_t) \\ 0 & -\sin(\phi_t) & \cos(\phi_t)\cos(\theta_t) \end{pmatrix}$$

The relashionship between body rates (pqr) and euler rates is:

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = A * \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix}$$

Euler rates from our body rates

Assuming the drone does NOT have a pitch angle of $\theta = \pm 90$, then we can use the results of the previous section to calculate the Euler rates from our body rates.

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = A^{-1} \begin{bmatrix} p \\ q \\ r \end{bmatrix} = A^{-1}\omega_b$$

```
euler_rates = A \ [p; q; r];
euler_rates = simplify(euler_rates)
```

```
euler_rates =
```

$$\begin{pmatrix} \dfrac{p\cos(\theta_t) + r\cos(\phi_t)\sin(\theta_t) + q\sin(\phi_t)\sin(\theta_t)}{\cos(\theta_t)} \\ q\cos(\phi_t) - r\sin(\phi_t) \\ \dfrac{r\cos(\phi_t) + q\sin(\phi_t)}{\cos(\theta_t)} \end{pmatrix}$$

Generate matlab functions and simulink block for angular rates

```
% matlab function
FILENAME        = 'generated/euler_rates';
INPUT_VAR_ORDER = {'phi_t', 'theta_t', 'p', 'q', 'r'};
matlabFunction(euler_rates,'File',FILENAME, ...
              'Optimize',false, 'Vars', INPUT_VAR_ORDER);
% simulink block
SIM_FILENAME        = 'rates_blocks';
BLOCK_NAME      = [SIM_FILENAME,'/euler_rates'];
if exist(SIM_FILENAME,'file')==0
    new_system(SIM_FILENAME)
end
open_system(SIM_FILENAME)

matlabFunctionBlock(BLOCK_NAME, euler_rates, ...
                    'Optimize',false, 'Vars', INPUT_VAR_ORDER);
```

Euler angles representation

This method has singularities however it is easy to understand and implement.

We discussed this representation in previous chapters and simulated a drone EOMs using this representation. In short, for stort from initial euler angles, conver the current body rates to euler rates using matrix A, then integrate to get euler angles. Rotation matrix from euler angles is construct to transform quantities between body and inertial frames.

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = A^{-1} \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

$$x_b = {}^bR_w \omega_w$$

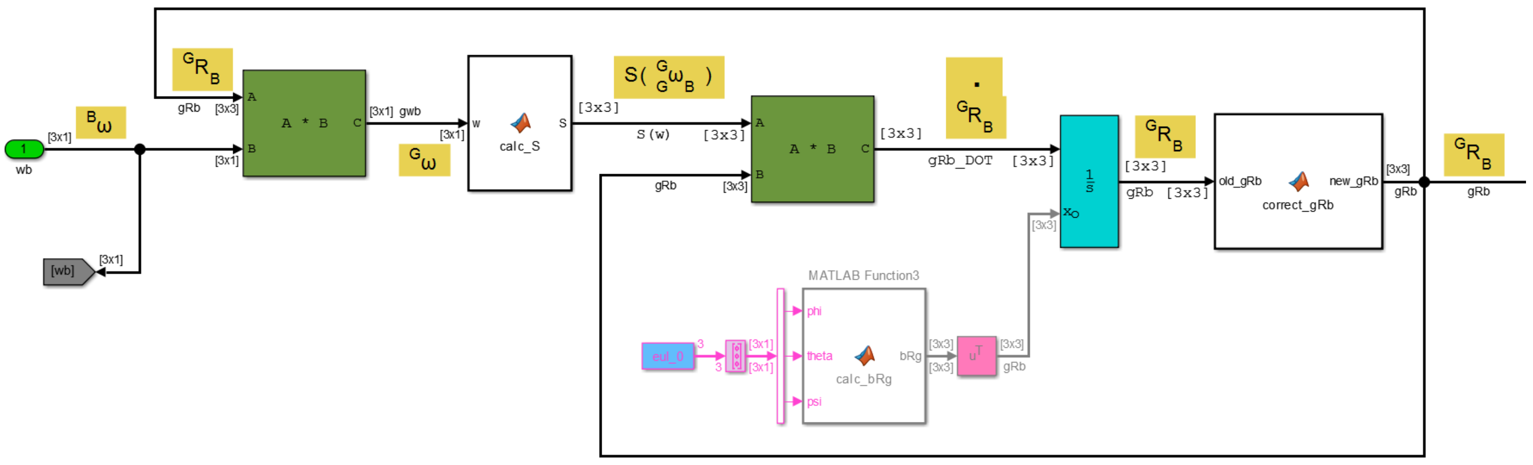
Rotation matrix representation

we can avoid singularities in Euler angles representation by using rotation matrix representation. We will need both a rotation matrix R that represents attitude and a rotation matrix derivative $\dot{R}$ that represents the angular rates.

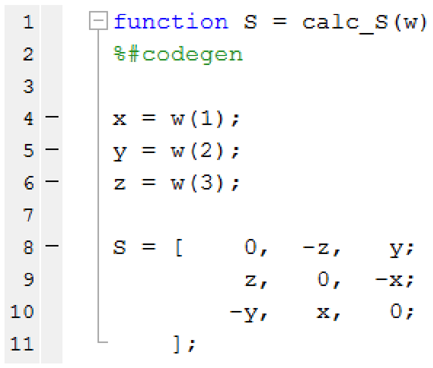
The following is the summary of the steps required to represent attitude (in world frame) using rotation matrix:

1. Starting from body angular rates (pqr) ω_b which comes from sensors and state estimators.
2. Start from intial attitude R_0 calculated from intial euler angles $(\phi_0, \theta_0, \psi_0)$
3. Convert body rates ω_b to angular rates in world frame ω_w using the rotation matrix R. $\omega_w = {}^wR_b \omega_b$
4. Represent the world angular rates using skew-symmetric matrix $[\omega_w]$
5. Calculate the rotation matrix derivative using $\dot{R} = [\omega_w]\, {}^wR_b$
6. Integrate $\dot{R}$ to get ${}^wR_b$ and ensure it is still orthogonal
7. Feed ${}^wR_b$ back to calculate ω_w in step 3.

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:

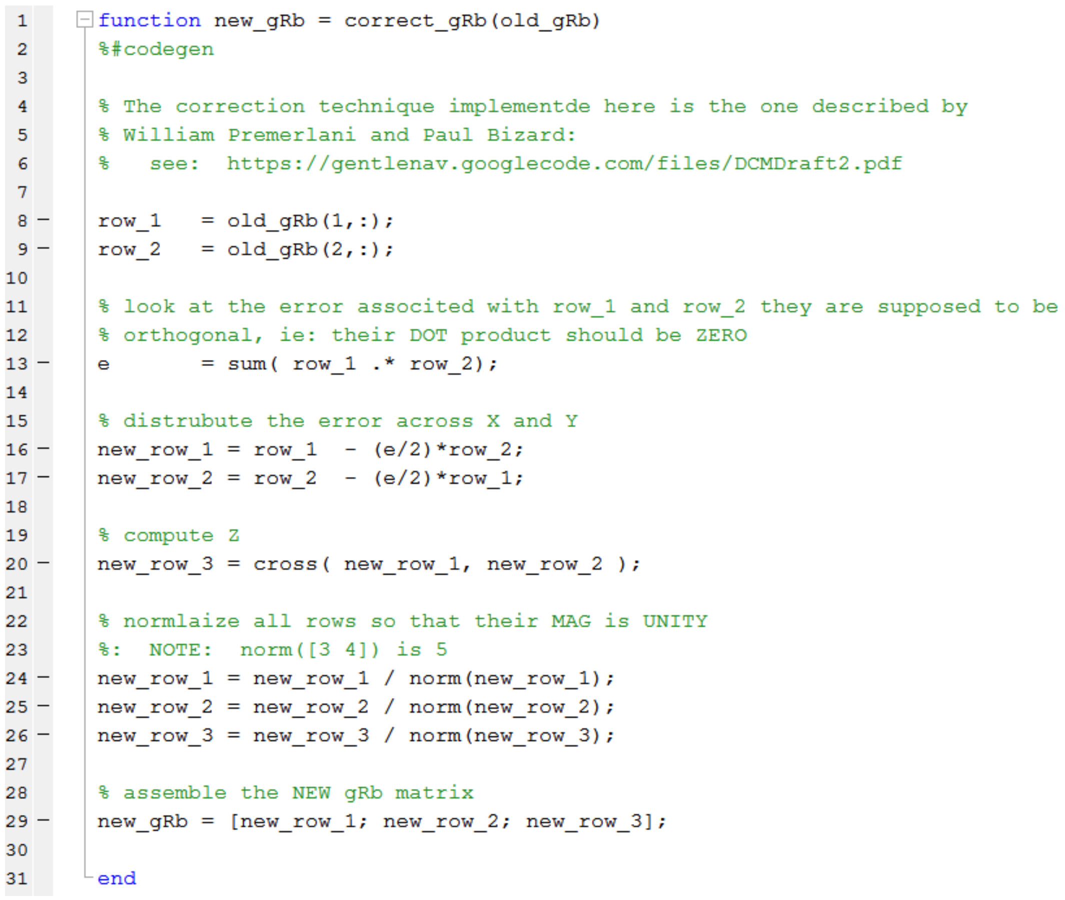


```
function S = calc_S(w)
%#codegen

x = w(1);
y = w(2);
z = w(3);

S = [     0,   -z,    y;
          z,    0,   -x;
         -y,    x,    0;
     ];
```

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

$${}^{w}R_{b}(t)\ .\ {}^{w}R_{b}(t)^{T}=I+\delta e \text{ ,where } \quad \delta e \quad \text{ is a matrix of small terms}$$

We can reduce this δe term using a correction technique such as the one proposed by William Premerlani and Paul Bizard (https://wiki.paparazziuav.org/w/images/e/e5/DCMDraft2.pdf).



```
function new_gRb = correct_gRb(old_gRb)
%#codegen

% The correction technique implementde here is the one described by
% William Premerlani and Paul Bizard:
%   see:  https://gentlenav.googlecode.com/files/DCMDraft2.pdf

row_1   = old_gRb(1,:);
row_2   = old_gRb(2,:);

% look at the error associted with row_1 and row_2 they are supposed to be
% orthogonal, ie: their DOT product should be ZERO
e       = sum( row_1 .* row_2);

% distrubute the error across X and Y
new_row_1 = row_1  - (e/2)*row_2;
new_row_2 = row_2  - (e/2)*row_1;

% compute Z
new_row_3 = cross( new_row_1, new_row_2 );

% normlaize all rows so that their MAG is UNITY
%:  NOTE:  norm([3 4]) is 5
new_row_1 = new_row_1 / norm(new_row_1);
new_row_2 = new_row_2 / norm(new_row_2);
new_row_3 = new_row_3 / norm(new_row_3);

% assemble the NEW gRb matrix
new_gRb = [new_row_1; new_row_2; new_row_3];

end
```

Quaternion representation

$$\mathbf{q}_{123}(\phi,\theta,\psi) = \begin{bmatrix} c_{\phi/2}c_{\theta/2}c_{\psi/2} + s_{\phi/2}s_{\theta/2}s_{\psi/2} \\ -c_{\phi/2}s_{\theta/2}s_{\psi/2} + c_{\theta/2}c_{\psi/2}s_{\phi/2} \\ c_{\phi/2}c_{\psi/2}s_{\theta/2} + s_{\phi/2}c_{\theta/2}s_{\psi/2} \\ c_{\phi/2}c_{\theta/2}s_{\psi/2} - s_{\phi/2}c_{\psi/2}s_{\theta/2} \end{bmatrix}$$

Quaternion rates from body rates (body-fixed frame):

$\dot{q} = \frac{1}{2} W(q)^T * \omega_b$

$$W(q) = \begin{bmatrix} -q_1 & q_0 & q_3 & -q_2 \\ -q_2 & -q_3 & q_0 & q_1 \\ -q_3 & q_2 & -q_1 & q_0 \end{bmatrix}$$

```
syms q0 q1 q2 q3 w1 w2 w3
w = [w1;w2;w3];
q = [q0;q1;q2;q3];
W = [-q1 q0 q3 -q2;
     -q2 -q3 q0 q1;
     -q3 q2 -q1 q0];
W_transpose = W.'
```

```
W_transpose =
```

$$\begin{pmatrix} -q_1 & -q_2 & -q_3 \\ q_0 & -q_3 & q_2 \\ q_3 & q_0 & -q_1 \\ -q_2 & q_1 & q_0 \end{pmatrix}$$

```
q_dot = 0.5*W_transpose*w
```

```
q_dot =
```

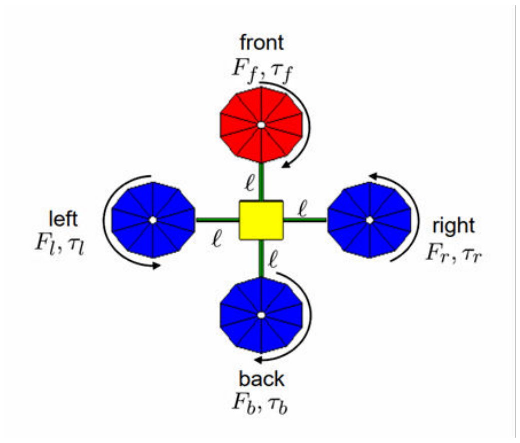
$$\begin{pmatrix} -\frac{q_1 w_1}{2} - \frac{q_2 w_2}{2} - \frac{q_3 w_3}{2} \\ \frac{q_0 w_1}{2} + \frac{q_2 w_3}{2} - \frac{q_3 w_2}{2} \\ \frac{q_0 w_2}{2} - \frac{q_1 w_3}{2} + \frac{q_3 w_1}{2} \\ \frac{q_0 w_3}{2} + \frac{q_1 w_2}{2} - \frac{q_2 w_1}{2} \end{pmatrix}$$

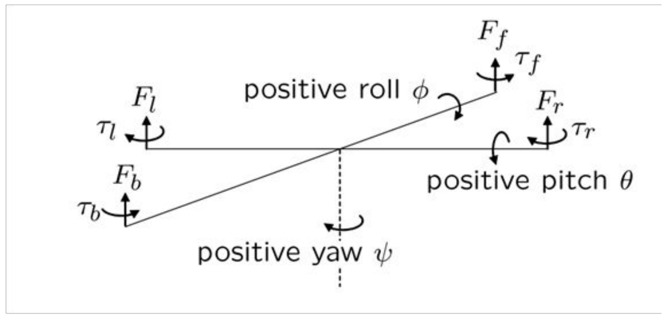
Modelling actuators and mixing

Rotation convention:

the following convention/propeller arrangement is assumed:

The motors are numbered clockwise: **front(1),right(2), back(3), left(4)**





Actuator modelling

The most common model of the propellers-motor pair is a proportional term to the angular velocity .

- Force/torque from each propeller is propertional to propeller rotation speed $f_i = k_t\omega_i^2$, $m_i = k_m\omega_i^2$
- Total thrust is summation of forces from all propellers (negative since Z-axis is pointing down) $F = -\sum_i k_t\omega_i^2$
- Moment around x-axis (roll):$M_x = L(f_l - f_r)$
- Moment around y-axis (pitch): $M_y = L(f_f - f_b)$
- Moment around z-axis (yaw): from reactive torques, $M_z = m_r + m_l - m_f - m_b = k_m(\omega_r^2\omega_l^2 - \omega_f^2 - \omega_b^2)$

The mapping from angular velocities to twists(forces/moments) using allocation matrix C (this is called allocation or mixing):

$$\begin{bmatrix} F \\ M \end{bmatrix} = C * \omega^2$$

$$\begin{bmatrix} F \\ M_x \\ M_y \\ M_z \end{bmatrix} = \begin{bmatrix} -k_t & -k_t & -k_t & -k_t \\ 0 & -Lk_t & 0 & \mathrm{L}k_t \\ Lk_t & 0 & -Lk_t & 0 \\ -k_m & k_m & -k_m & k_m \end{bmatrix} * \begin{bmatrix} \omega_f^2 \\ \omega_r^2 \\ \omega_b^2 \\ \omega_l^2 \end{bmatrix}$$

Control

The control strategies will specify forces and torques. The actual motors commands can be found by inversing the allocation matrix

$$\omega^2 = C^{-1}\begin{bmatrix} F \\ M \end{bmatrix}$$

```
syms L kt km w1 w2 w3 w4 thrust mx my mz
w_2 = [w1; w2; w3; w4]; % omega square
T = [thrust; mx; my;mz];

% construct the co-efficient matrix:
C = [
    -kt   ,   -kt    ,   -kt   ,   -kt   ;
    0     ,  -L*kt   ,   0     ,  L*kt   ;
   L*kt   ,   0      ,  -L*kt  ,    0    ;
    -km   ,   km     ,  -km    ,    km   ;
];

T == C *w_2
```

ans =

$$\begin{pmatrix} \mathrm{thrust} = -\mathrm{kt}\,w_1 - \mathrm{kt}\,w_2 - \mathrm{kt}\,w_3 - \mathrm{kt}\,w_4 \\ \mathrm{mx} = L\,\mathrm{kt}\,w_4 - L\,\mathrm{kt}\,w_2 \\ \mathrm{my} = L\,\mathrm{kt}\,w_1 - L\,\mathrm{kt}\,w_3 \\ \mathrm{mz} = \mathrm{km}\,w_2 - \mathrm{km}\,w_1 - \mathrm{km}\,w_3 + \mathrm{km}\,w_4 \end{pmatrix}$$

Using the inverse of allocation matrix to solve for motor commands

```
C_inv = inv(C)
```

C_inv =

$$\begin{pmatrix} -\frac{1}{4\,\mathrm{kt}} & 0 & \frac{1}{2\,L\,\mathrm{kt}} & -\frac{1}{4\,\mathrm{km}} \\ -\frac{1}{4\,\mathrm{kt}} & -\frac{1}{2\,L\,\mathrm{kt}} & 0 & \frac{1}{4\,\mathrm{km}} \\ -\frac{1}{4\,\mathrm{kt}} & 0 & -\frac{1}{2\,L\,\mathrm{kt}} & -\frac{1}{4\,\mathrm{km}} \\ -\frac{1}{4\,\mathrm{kt}} & \frac{1}{2\,L\,\mathrm{kt}} & 0 & \frac{1}{4\,\mathrm{km}} \end{pmatrix}$$

```
w_2 == C_inv*T
```

ans =

$$\begin{pmatrix} w_1 = \frac{\mathrm{my}}{2\,L\,\mathrm{kt}} - \frac{\mathrm{thrust}}{4\,\mathrm{kt}} - \frac{\mathrm{mz}}{4\,\mathrm{km}} \\ w_2 = \frac{\mathrm{mz}}{4\,\mathrm{km}} - \frac{\mathrm{thrust}}{4\,\mathrm{kt}} - \frac{\mathrm{mx}}{2\,L\,\mathrm{kt}} \\ w_3 = -\frac{\mathrm{mz}}{4\,\mathrm{km}} - \frac{\mathrm{thrust}}{4\,\mathrm{kt}} - \frac{\mathrm{my}}{2\,L\,\mathrm{kt}} \\ w_4 = \frac{\mathrm{mz}}{4\,\mathrm{km}} - \frac{\mathrm{thrust}}{4\,\mathrm{kt}} + \frac{\mathrm{mx}}{2\,L\,\mathrm{kt}} \end{pmatrix}$$

```
matlabFunction(C_inv*T,'File','generated/control_allocation', ...
               'Optimize',false, 'Vars', [L,kt,km,thrust,mx,my,mz],'Outputs',{'w2_cmd'});
```

Attitude control

The goal of the attitude control is to to ensure the drone attitude follows the desired one. Attitude control is highly dependent on how the attitude is represented, hence we will discuss three controllers that are used for the three common attitude representations discussed above (Euler angles, Rotation Matrix, Quaternion)

First, we need to formulate the error given the current and desired attitude. Then, we will formulate the command to drive the current attitude to the desired one as fast as possible. The output command represents the desired angular rates ω_d which are fed to the rate controller to regulate actuation torques.

Control hierarchy

The common control hierarchies for attitude+rates are cascaded or single control approaches as follows:

cascaded control

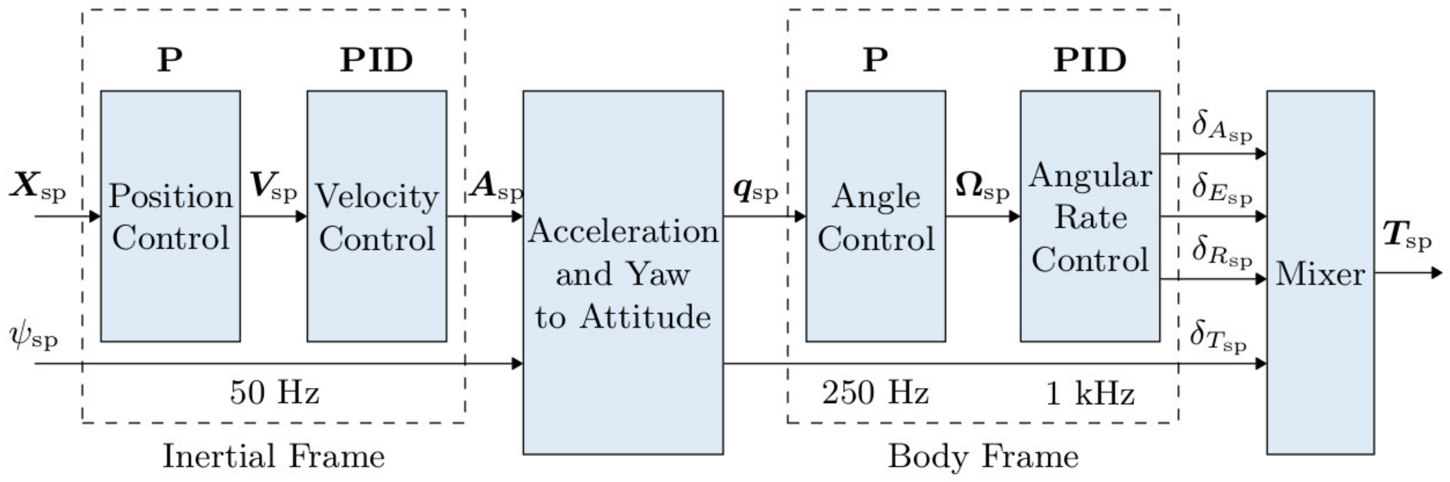
The attitude controller produces the commanded angular rates ω_d to the rate controller which in turn regulates the actuation torques. The actuation torques are combined with altitude controller that regulates altitude (height) through thrust F_d. The combined torques and thrust are fed to the motor controller and control allocation.

Single control (attitude+angular rates)

The attitude controller operates directly on the drone torques by adding the attitude controller output to the rate controller to regulate the actuation torques. The actuation torques are combined with altitude controller that regulates altitude (height) through thrust F_d. The combined torques and thrust and fed to the motor controller and control allocation.

In our simulations we will use cascaded control similar to popular flight control systems such as Ardupilot and PX4.

An example of complete control architecture from PX4 flight stack is below. Note the use of quaternion to represent the attitude and cascading with angular rate controller



There are different approaches for realizing the error calculation of the attitude controller and hence the attitude controller output is also different. Each approach is suitable for different attitude represntation.

Euler angles representation

Error calculation

The error is easily the difference between desired angle and current angle

$$e_{\text{rpy}} = \begin{bmatrix} e_\phi \\ e_\theta \\ e_\psi \end{bmatrix} = \begin{bmatrix} \phi_c \\ \theta_c \\ \psi_c \end{bmatrix} - \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix}$$

control command

The attitude error can be used in a P-controller or P+D-controller to define the desired angular rates which is then fed to the rate controller. or The controller can command the the acuation torques directly

$$\overrightarrow{\omega}_d = k_{\text{rpy}} \overrightarrow{e}_{\text{rpy}} - k_{\text{dot}} \begin{bmatrix} \dot\phi \\ \dot\theta \\ \dot\psi \end{bmatrix}$$

Rotation matrix representation

Note: the rotation matrix refered below is the body to world rotation matrix

Error calculation

The rotation matrix deviationR_e is calculated by multiplying the transpose of desired rotation matrix $R_d{}^T$ by the current (estimated) rotation matrix $\widehat{R}$.

$$R_e = R_d{}^T \widehat{R}$$

The attitude error $\vec{e}_R$ can be found by calculating the axis around which to rotate the error vector that grows with the error angle

$$\vec{e}_R = \frac{1}{2}\left(R_e - R_e^T\right) = \frac{1}{2}\left({R_d}^T \hat{R} - \hat{R}^T R_d\right)$$

control command

The attitude error can be used in a **P-controller** to define the desired angular rates which is then fed to the rate controller

$$\vec{\omega}_d = k_R \vec{e}_R$$

Quaternion representation

Error calculation

The rotation error q_e is calculated by multiplying the inverse of current (estimated) quaternion $\hat{q}^{-1}$ by the desired quaternion q_d .

$$q_e = \hat{q}^{-1} \otimes q_d$$

The attitude error q can be found by extracting the axis around which to rotate for correcting the attitude error and have this error vector grow with the error angle. This is achieve using hte vector componet of the quaternion $\vec{q}_e$ (i.e. $q = \begin{bmatrix} q_0 & \vec{q} \end{bmatrix}$)

$$\vec{e}_q = \text{sign}(q_{e,0})\, \vec{q}_e \quad \text{with}\ \text{sign}(x) = \begin{cases} 1 & \text{if}\ x \geq 0 \\ -1 & \text{if}\ x < 0 \end{cases}$$

Note: We needed to introduce the sign function to avoid the quaternion ambiguity (unwinding behavior) by using the anipodal quaternion such that the first element is always positive.

control command

The attitude error can be used in a P-controller to define the desired angular rates which is then fed to the rate controller

$$\vec{\omega}_d = k_q \vec{e}_q$$

Rate control

The output from attitude control is the desired angular rates (pqr) which are regulated by rate controller. The controller is typically a PD controller unless the drone is operated in acro mode (no attitude controller), then an I-term is required.

The output of the rate controller is the desired torques which are fed to control allocation/mixer and motor controllers.

control command

$$\overrightarrow{\tau}_d = \begin{bmatrix} \tau_x \\ \tau_x \\ \tau_x \end{bmatrix} = k_p \overrightarrow{e}_p - k_d \overrightarrow{\dot{\omega}}$$

$$\overrightarrow{e}_p = \overrightarrow{\omega}_d - \overrightarrow{\omega}$$

where $\overrightarrow{\dot{\omega}}$ is estimated angular accelerations, $\overrightarrow{\omega}$ is current estimated angular velocit, k_p, k_d are control gains

Steps:

1- Control allocation:

- Force/moment from each propeller is propertional to propeller rotation speed $f_i = k_t\omega_i^2$, $m_i = k_m\omega_i^2$
- Total thrust and moments are calculated according to the allocation matrix C in previous section
- The inverse of allocation matrix (C^{-1}) is used for control since it maps the commanded thrust and moments to the commanded motor speeds which is fed to the motor control block
- for simulation purposes: the allocation matrix is used to simulate the mapping from motor speeds to generated thrusts and moments (mixing)
- **Inputs:** Commanded thrust and moments
- **Output**: commanded motor angular speeds
- **Parameters:** arm length (L), thrust and moment constants (kt and km)

2- Linear motion (Netwon equation):

- Convert from inertial motion $m * a = F$ to body-fixed motion $\mathrm{F}_b = \mathrm{m}.\left(\dot{V}_b + \omega \times \mathrm{V}_b\right)$
- Rearrange equation to be have body linear acceleration $\dot{\boldsymbol{V}}_{\boldsymbol{b}}$ as output: $\dot{\boldsymbol{V}}_{\boldsymbol{b}} = \dfrac{\boldsymbol{F}_{\boldsymbol{b}}}{\boldsymbol{m}} - (\omega \times \mathbf{V}_{\boldsymbol{b}})$
- The force vector consists of thrust from actuators and gravity force >>

$$\dot{\boldsymbol{V}}_{\boldsymbol{b}} = \frac{\begin{bmatrix} 0 \\ 0 \\ -F_{\text{thrust}} \end{bmatrix} + {}^bR_w \begin{bmatrix} 0 \\ 0 \\ \text{mg} \end{bmatrix}}{\boldsymbol{m}} - (\omega \times \mathbf{V}_{\boldsymbol{b}})$$

- Integrate the linear acceleration to get linear velocity
- **Inputs:** the commanded thrust (from attitude control), body angular velocities (pqr) ω (step 3), ${}^bR_w$(step5)
- **Output**: $\mathbf{V}_{\boldsymbol{b}}$

- **Parameters:** mass (m), intital body-fixed velocity (default: $V_{b0} = [0, 0, 0]$)

3- Rotational motion (Euler equation):

- Represented in body-fixed frame: $I\,\dot{\omega} = M_b - \omega \times (I\omega)$
- Rearrange equation to get body angular acceleration($\dot{\omega}$) as output: $\dot{\omega} = \boldsymbol{I}^{-1}(\boldsymbol{M_b} - \omega \times (\boldsymbol{I}\omega))$
- Integrate the body angular acceleration to get body-angular velocity (pqr)
- **Inputs:** the commanded MomentsM_b (from attitude control)
- **output**: ω
- **Parameters:** inertia (I), intital body rates (default: $\omega_0 = [0, 0, 0]$)

4- Caculate Euler rates (Body to Inertial rates):

- Use the convertion matrix A to conver body rates $\omega = \begin{bmatrix} p \\ q \\ r \end{bmatrix}$ to Euler rates
- $\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = A^{-1} \begin{bmatrix} p \\ q \\ r \end{bmatrix}$
- Integrate to get Euler angles
- **Inputs:** body rates (ω) (step 3), Euler angles (this step output)
- **Output**: Euler angles (ϕ, θ, ψ)
- **Parameters:** intitial attitude default:$(\phi_0, \theta_0, \psi_0) = (0, 0, \psi_0)$

5- Caculate rotation matrix from euler angles:

- Use bRw formulated in the first section above to convert Euler angles to DCM
- **Inputs:** Euler angles (ϕ, θ, ψ)(step 4)
- **Output:** rotation matrix ${}^{b}R_{w}$
- **Parameters:**

6- Caculate inertial position and velocity:

- Use wRb formulated in the first section to convert body-fixed velocity V_b in step 2 to inertial velocity V
- Integrate to get inertial position p (NED xyz)
- **Inputs:** ${}^{w}R_{b} = {}^{b}R_{w}^{T}$(step 5), body-fixed velocity V_b(step 2)
- **Output:** inertial positon and velocity V, p
- **Parameters:** initial position (default $p[0, 0, 0]$)

7- Attitude control:

- **Inputs:** commanded attitude, Euler angles $(\phi_c, \theta_c, \psi_c)$
- **Output:** commanded thrust and moments (goes to allocation matrix in step 1)
- **Parameters: None**

Simulations

Drone parameters

```
clear all;
run("quad_params.m")
```

Euler angles attitude control

```
open_system("three_a_attitude_control_euler.slx")
```

DCM attitude control

```
open_system("three_b_attitude_control_rotationMatrix.slx")
```

Quaternion attitude control

```
open_system("three_c_attitude_control_quaternion.slx")
```

References

Randal Beard, Quadrotor Dynamics and Control Rev 0.1, 2008

Quaternion based Estimation and Control for Attitude Tracking of a Quadcopter using IMU sensors, Matthias Grob 2016