Quad dynamics eqn (Position)

$$m\begin{bmatrix}\ddot{x}\\ \ddot{y}\\ \ddot{z}\end{bmatrix} = T\underbrace{\begin{bmatrix}\sin\psi\sin\phi + \cos\psi\cos\phi\sin\theta\\ -\cos\psi\sin\phi + \sin\psi\cos\phi\sin\theta\\ \cos\phi\cos\theta\end{bmatrix}}_{\hat{z}_{be}} + \begin{bmatrix}-F_{xg}\\ -F_{yg}\\ -mg\end{bmatrix}$$

Clearly from $\hat{z}_{be}$ $\longrightarrow$ This ZYX - Euler Angles

Applied in the order: Z-Y-X Rotation sequence

(or)

Yaw, Pitch, Roll

$\psi$ $\phi$ $\theta$

Quad attitude dynamics eqn:

$$I\dot{\omega} = \tau - \omega \times I\omega - \tau_g$$

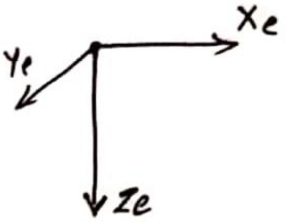
Newton-Euler eqn: for $\tau$ in body frame. $\omega_b = \begin{bmatrix}p\\ q\\ r\end{bmatrix}$

## 1. Plant

To solve these dynamics eqns: Use Aerospace Blockset

$\boxed{\text{6DOF Equations of Motion}}$ – Uses reasonable assumptions

Earth Coordinate Frame.

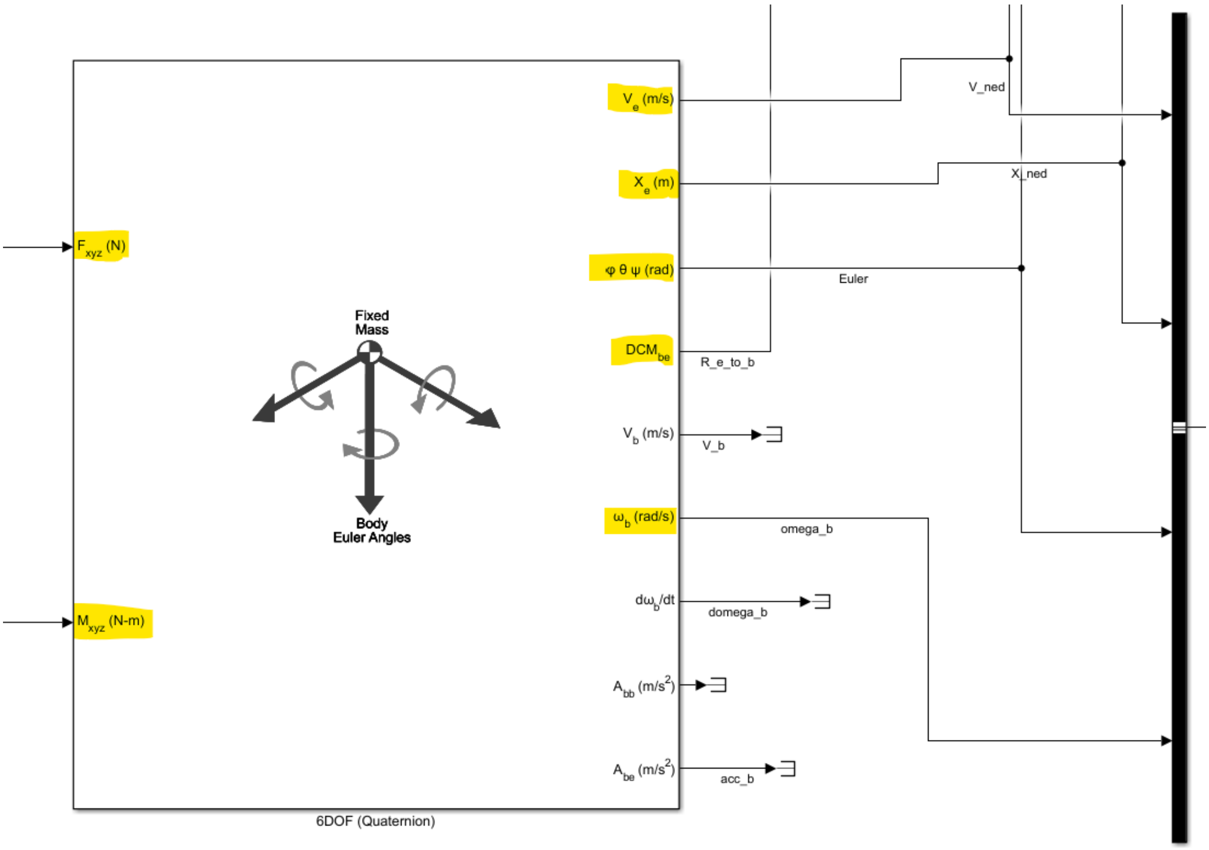



Inputs: Force & Torques in Body frame

Outputs: $\underbrace{\begin{bmatrix}x\\ y\\ z\end{bmatrix}}_{Xe}$, $\underbrace{\begin{bmatrix}dx/dt\\ dy/dt\\ dz/dt\end{bmatrix}}_{Ve}$ in Earth frame

Euler angles $\begin{bmatrix}\theta\\ \phi\\ \psi\end{bmatrix}$, $\omega_b = \begin{bmatrix}p\\ q\\ r\end{bmatrix}$

$DCM_{be}$: Rotation Matrix Earth to Body.




F → input

$mg\,\hat{z}_{earth}$ and dist_Fx $\hat{x}_{earth}$, dist_Fx $\hat{y}_{earth}$ are rotated to body frame.

Passed only if (dist_cond != 0)

Obtain
$$\begin{bmatrix}0\\0\\T\end{bmatrix} + R_{eb}\left[\begin{bmatrix}0\\0\\mg\end{bmatrix} + \begin{bmatrix}dist\_Fx\\dist\_Fy\\0\end{bmatrix}\right]$$

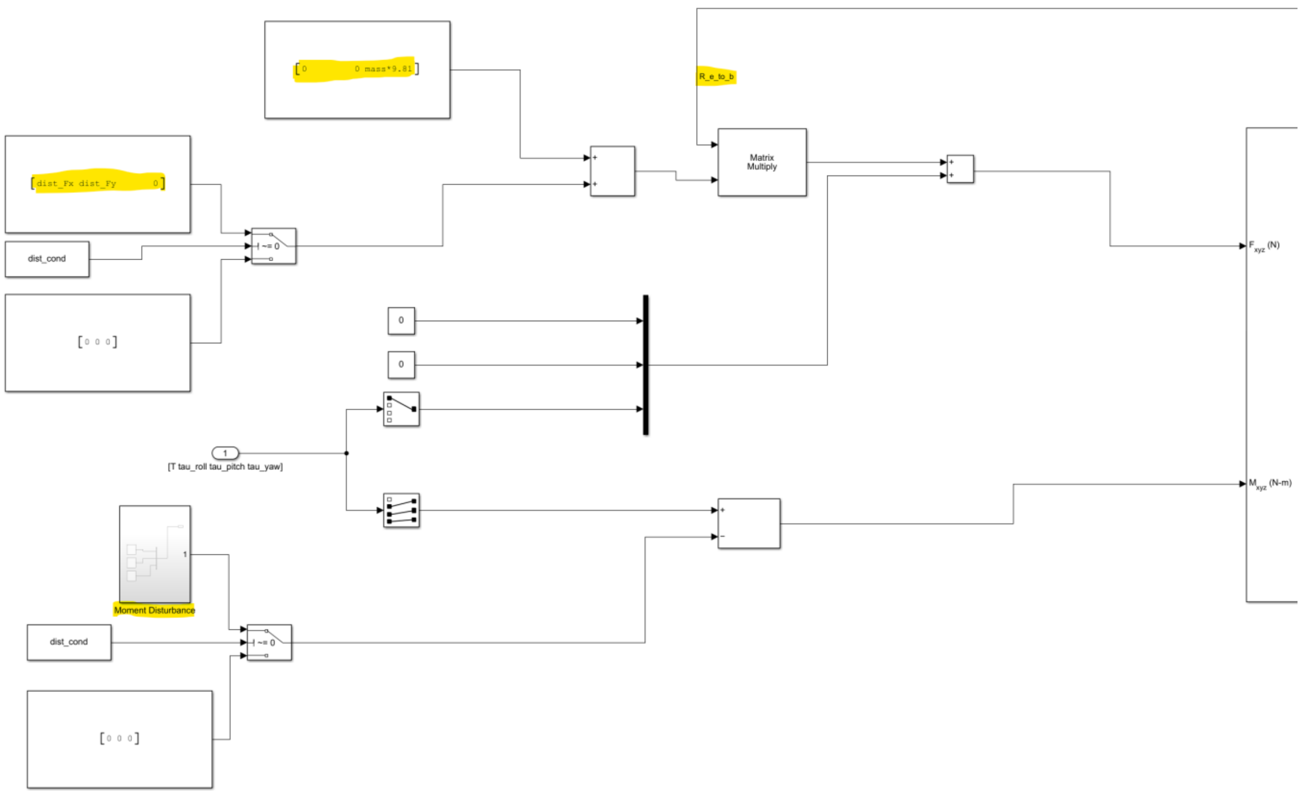
Rotate from e to b

Note: +ve Z-axis in pointing downward.

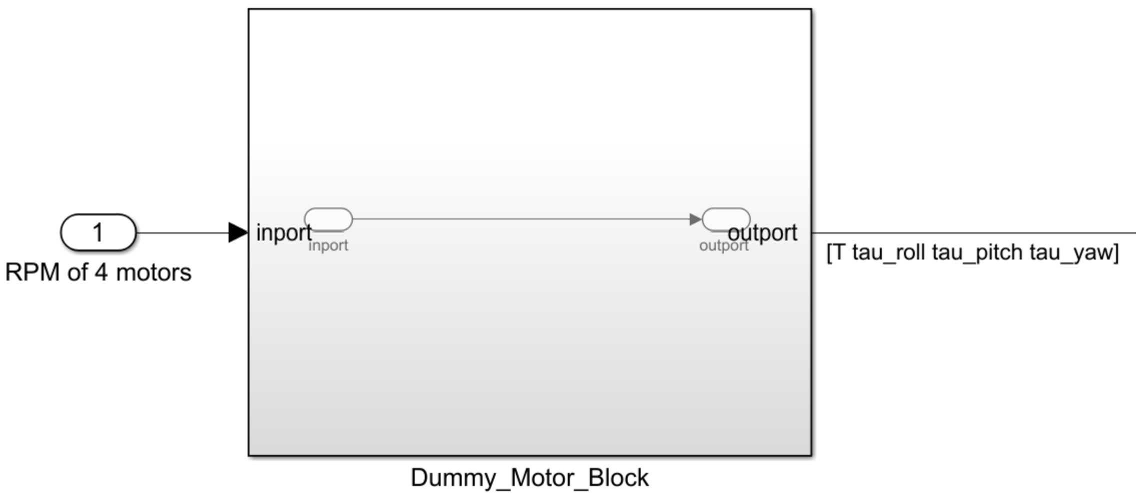
τ - input

Passed only if (dist_cond != 0)

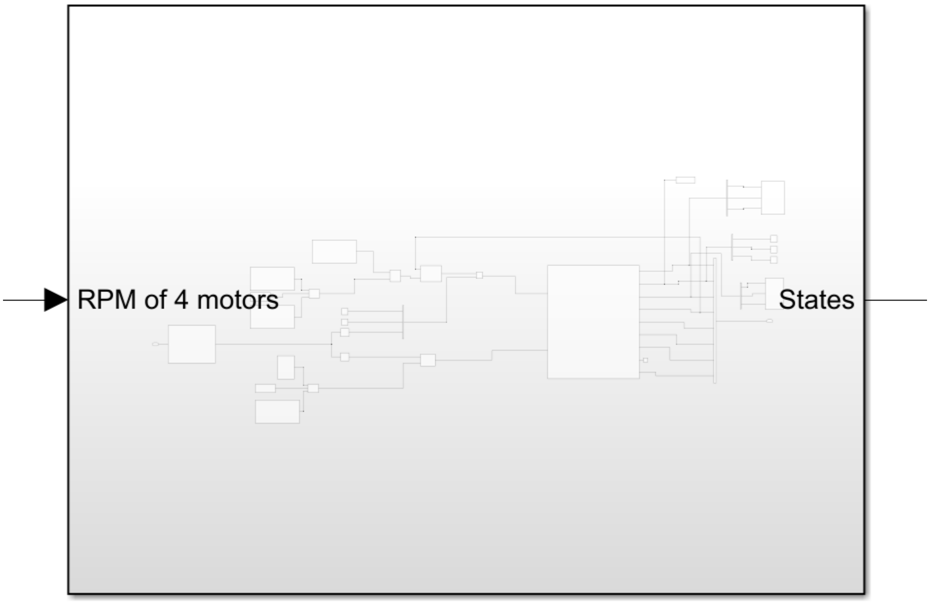
$$\begin{bmatrix}\tau_x\\\tau_y\\\tau_z\end{bmatrix} \begin{array}{l}\rightarrow \text{Roll torque}\\ \rightarrow \text{Pitch torque}\\ \rightarrow \text{Yaw torque}\end{array} + \begin{bmatrix}-\cos t\\-\sin t\\-2\sin t\end{bmatrix}$$




A Dummy Motor Block has been added, which
input : speed of 4 motors
output : T, τ_roll, τ_pitch, τ_yaw
(Depends on Motor, Propeller etc).



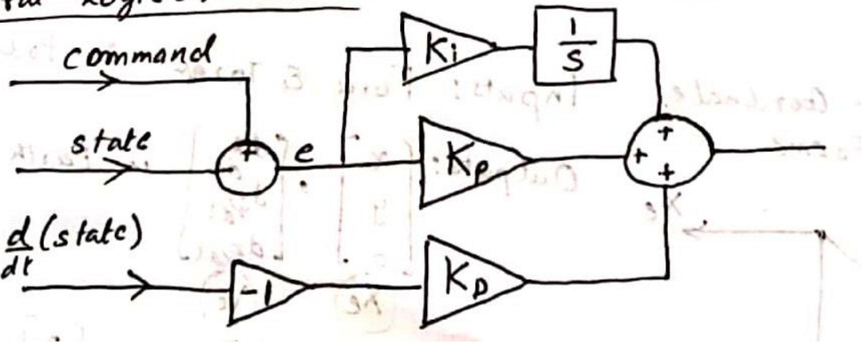
∴ Plant: Input: Speed of 4 motors
Output: State of Quadcopter based on Eqns of Dynamics



Plant

## 2. Flight Control

General Logic Used for PID:

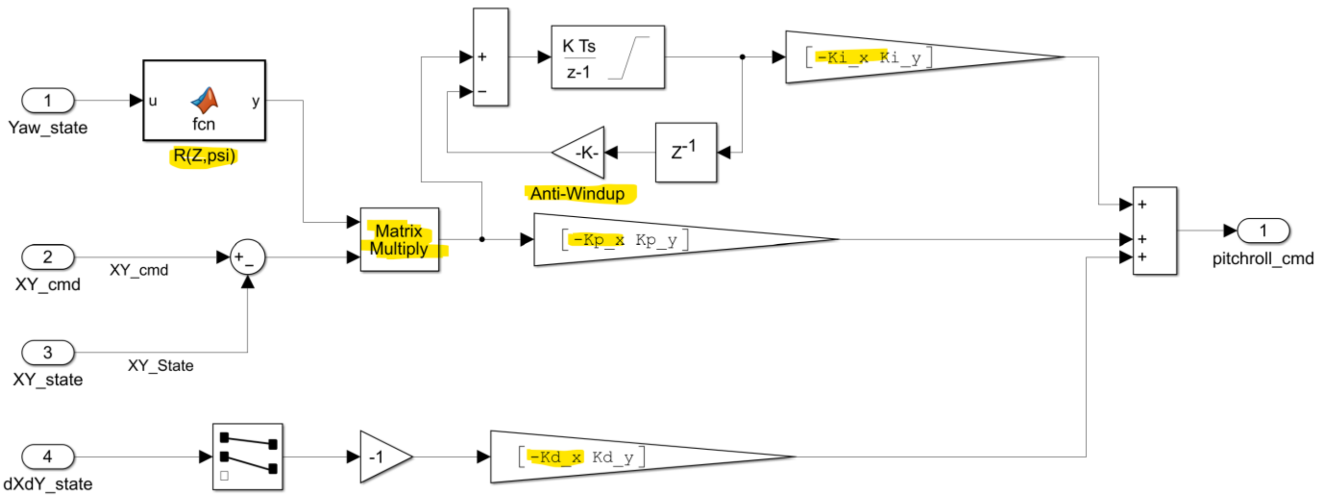


## Cascaded Control : XY, $\theta\phi$;

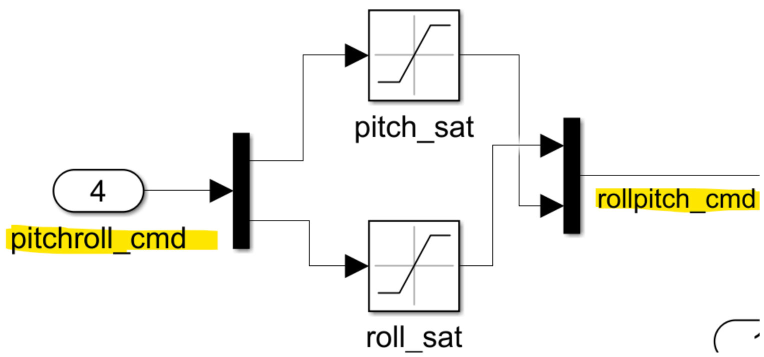
$\begin{bmatrix} X \\ Y \end{bmatrix}$ is in body frame. Since error in $\begin{bmatrix} X \\ Y \end{bmatrix}$ is used to command $\begin{bmatrix} \theta \\ \phi \end{bmatrix}$, we rotate $\begin{bmatrix} X \\ Y \end{bmatrix}_{error}$ by $R(Z, \psi)$, assuming $\theta$ and $\phi$ are small.

Also Note $+\tau$_roll(x) results in $+y$ displacement
$+\tau$_pitch(y) results in $-x$ displacement

$\therefore$ Gain in $X$ direction is taken as $-K_{px}$




Since Roll and Pitch are assumed small, we provide saturation:




Since Pitch Roll control is actuated by XY control, the saturation given to Pitch Roll, necessitates an anti-windup scheme (as shown above).

<u>Observation</u>: As $\begin{bmatrix}\theta \\ \phi\end{bmatrix}_{\substack{saturation \\ cmd}}$ values are increased, errors begin to grow.

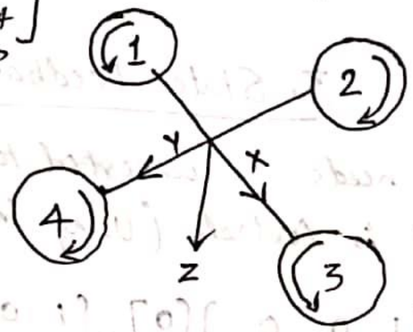
<u>Motor Mixing Algorithm</u>: Uses $\begin{bmatrix} T \\ \tau_{roll} \\ \tau_{pitch} \\ \tau_{yaw} \end{bmatrix}$ to generate rpm commands for the 4 motors

Note:

$$\begin{bmatrix} T \\ \tau_{roll} \\ \tau_{pitch} \\ \tau_{yaw} \end{bmatrix} = \underbrace{\begin{bmatrix} k_f & k_f & k_f & k_f \\ 0 & k_f L & 0 & -k_f L \\ -k_f L & 0 & k_f L & 0 \\ k_m & -k_m & k_m & -k_m \end{bmatrix}}_{J_T} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix}$$

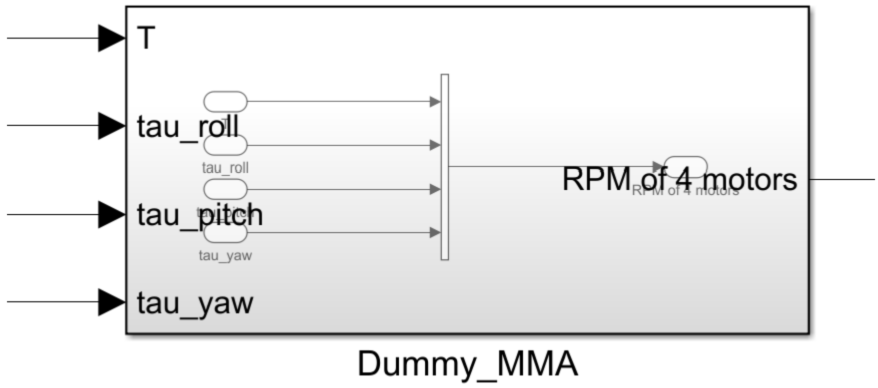
$$\underbrace{\begin{bmatrix} T \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix}}_{\vec{T}} \longrightarrow \underset{\sqrt{J_T^{-1}\vec{T}}}{\boxed{MMA}} \longrightarrow \underbrace{\begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix}}_{\vec{\omega}}$$

MMA assigns direction of rotation appropriately.

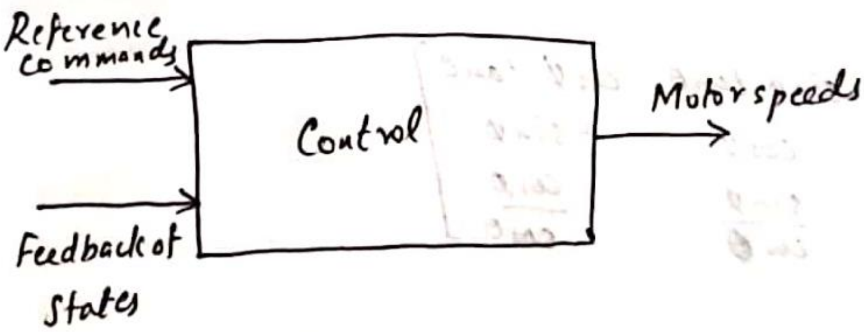



<u>Plus Configuration QuadCopter</u>

<u>Note</u>: Dummy_MMA is used, as the conversion is done for a motor model (which is absent in our model).




Note: Dummy_MMA is used, as the conversion is done for a motor model (which is absent in our model).



grow.

## 3. State Feedback ($\omega_b$, $\begin{bmatrix} x \\ y \\ z \end{bmatrix}$ $\begin{bmatrix} dx/dt \\ dy/dt \\ dz/dt \end{bmatrix}$ $\begin{bmatrix} \theta \\ \phi \\ \psi \end{bmatrix}$)

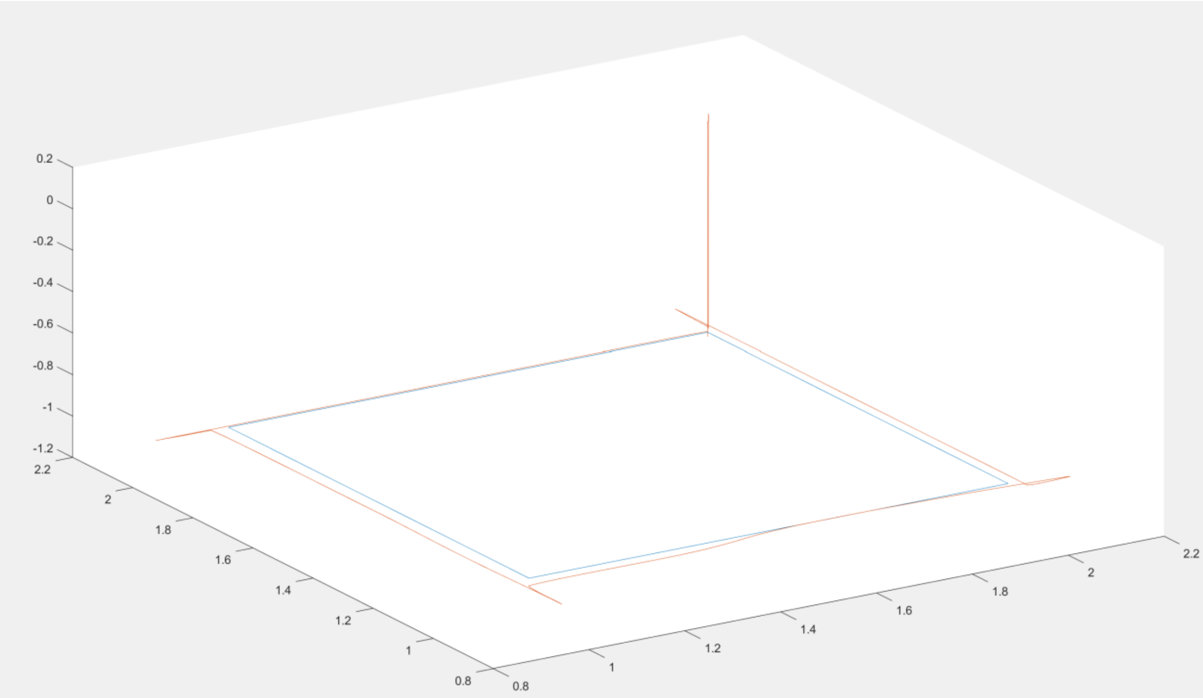
Here $\omega_b$ needs to converted to Euler rates (Body Frame) to be used in Control. (Using intermediate states)

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix}$$

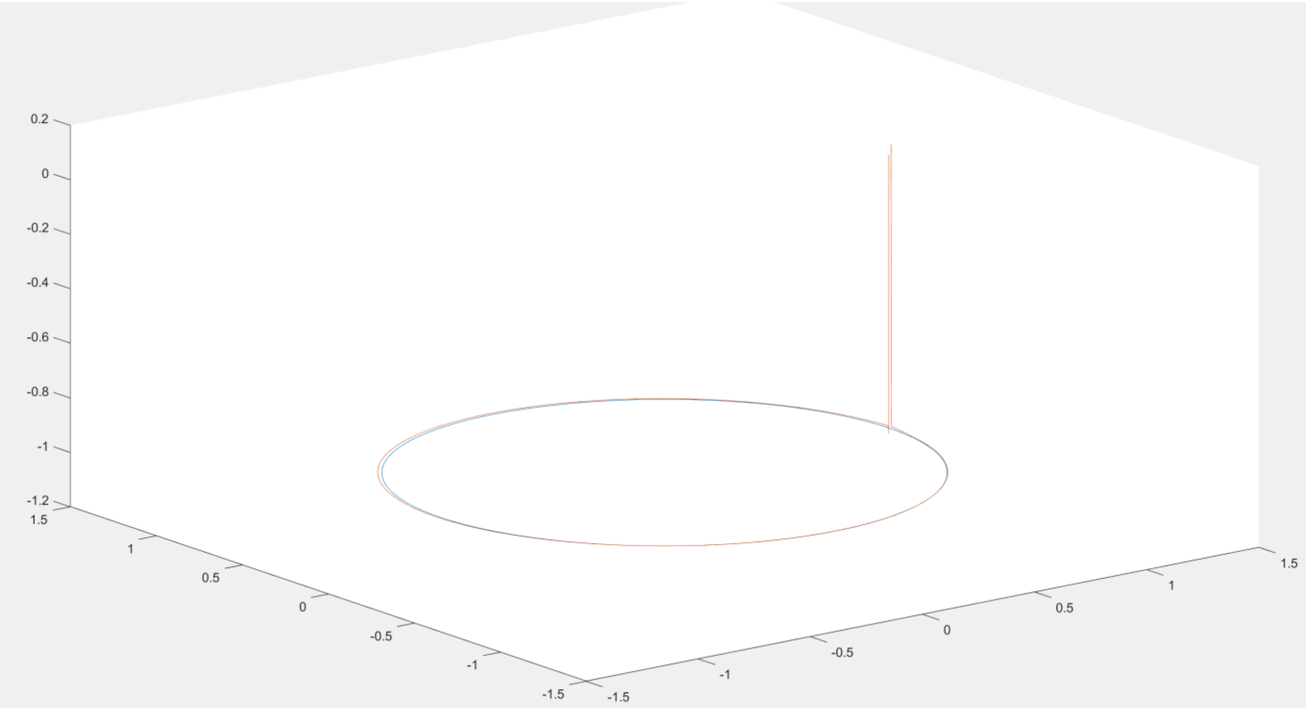
$$\therefore \begin{bmatrix} p \\ q \\ r \end{bmatrix} = J \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \Rightarrow \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = J^{-1} \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

$$J^{-1} = \begin{bmatrix} 1 & \sin\phi \tan\theta & \cos\phi \tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \dfrac{\sin\phi}{\cos\theta} & \dfrac{\cos\phi}{\cos\theta} \end{bmatrix}$$

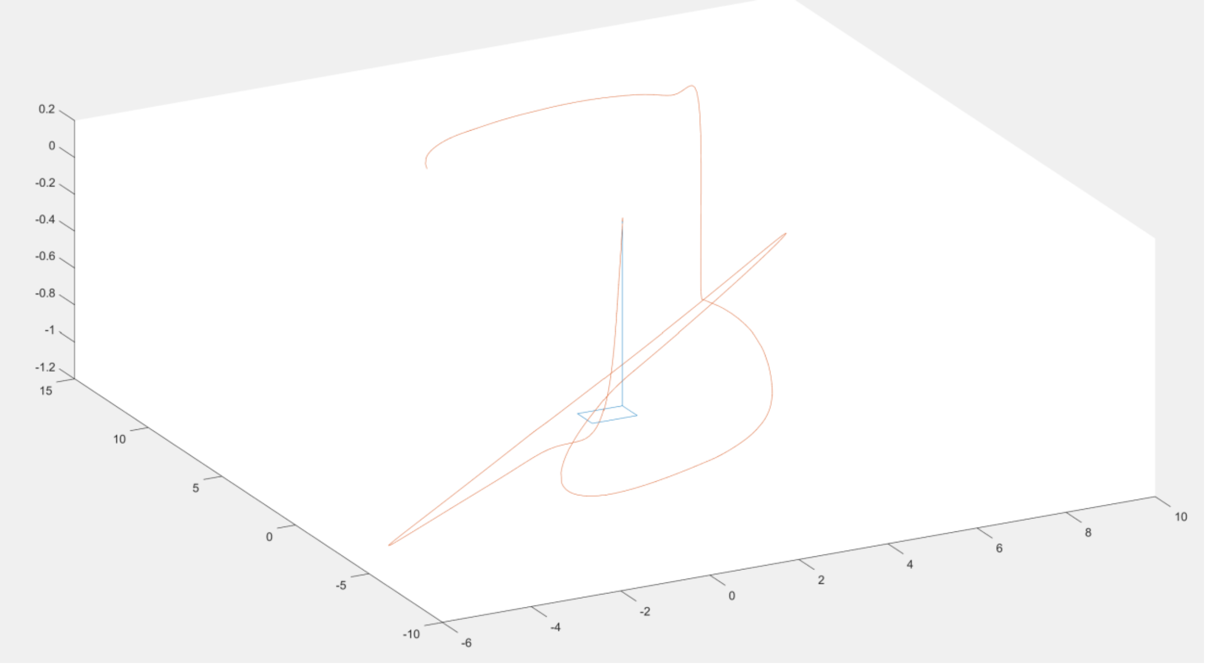
## Performance in the rectangular trajectory (Note: +Z is pointed downwards)- No disturbance (Total time= 14sec)



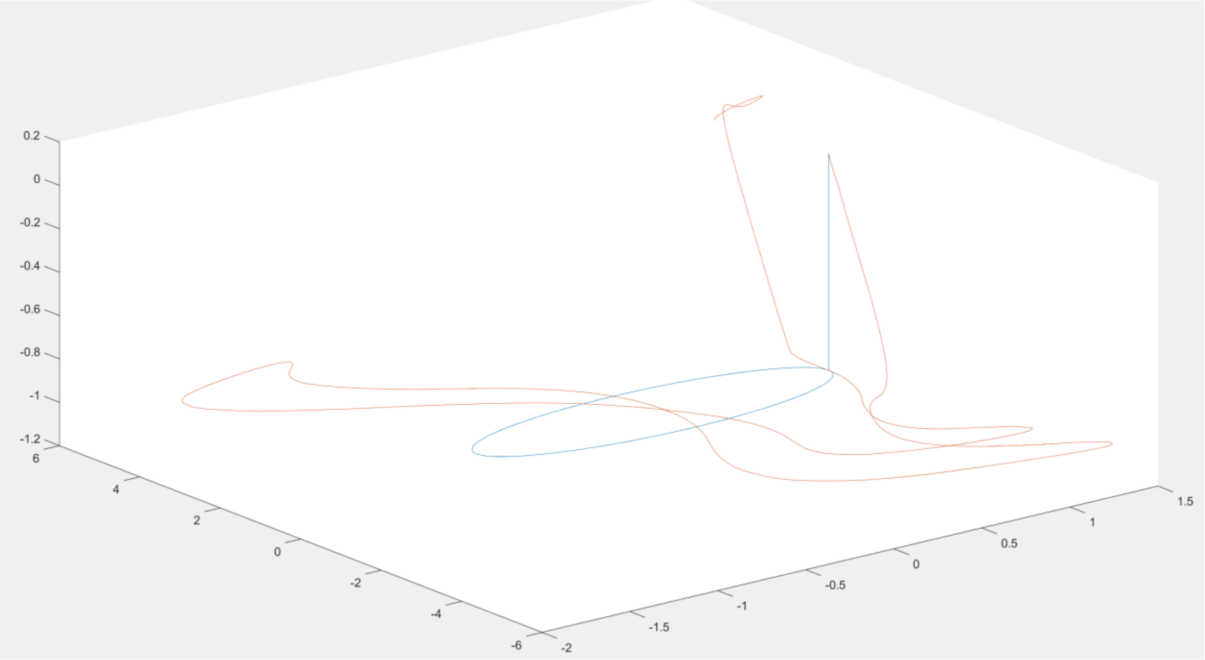

## Performance in the circular trajectory (Note: +Z is pointed downwards) )- No disturbance (Total time= (3+2*pi)sec)



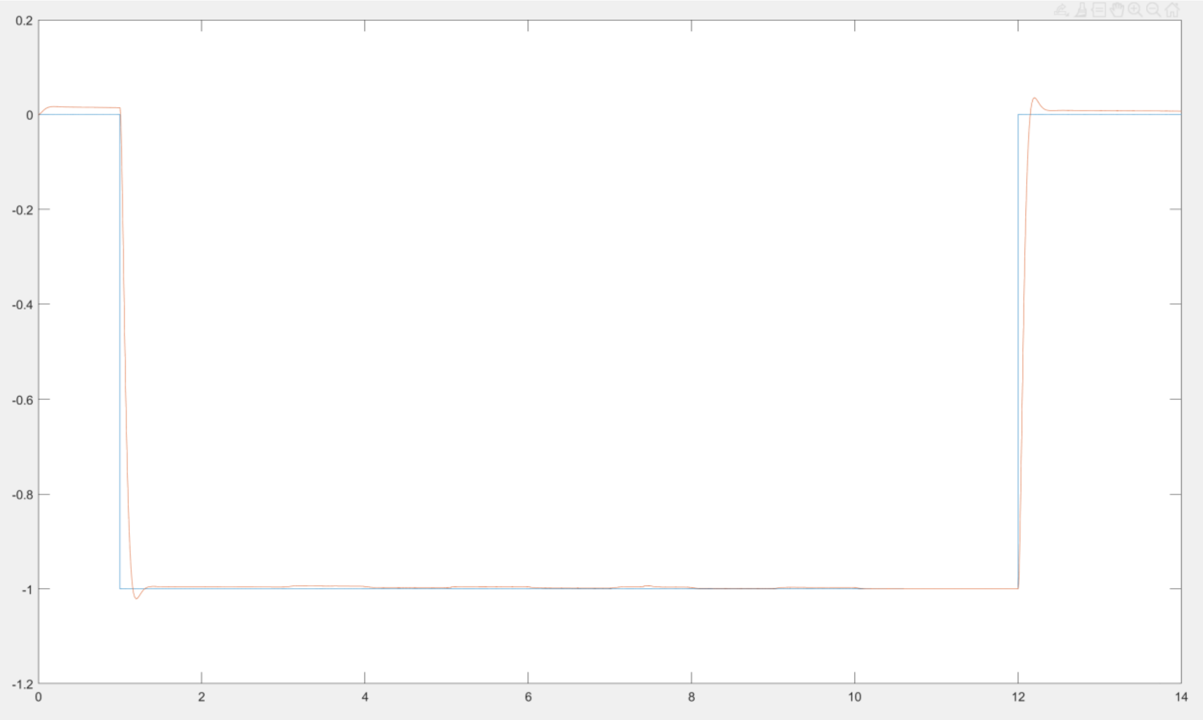
Performance in the rectangular trajectory (Note: +Z is pointed downwards)- Fx=0.05N, Fy=0.05N, tau_x=sin(t), tau_x=cos(t), tau_z=2sin(t)




Performance in the circular trajectory (Note: +Z is pointed downwards)- Fx=0.5N, Fy=0.5N, tau_x=sin(t), tau_x=cos(t), tau_z=2sin(t)

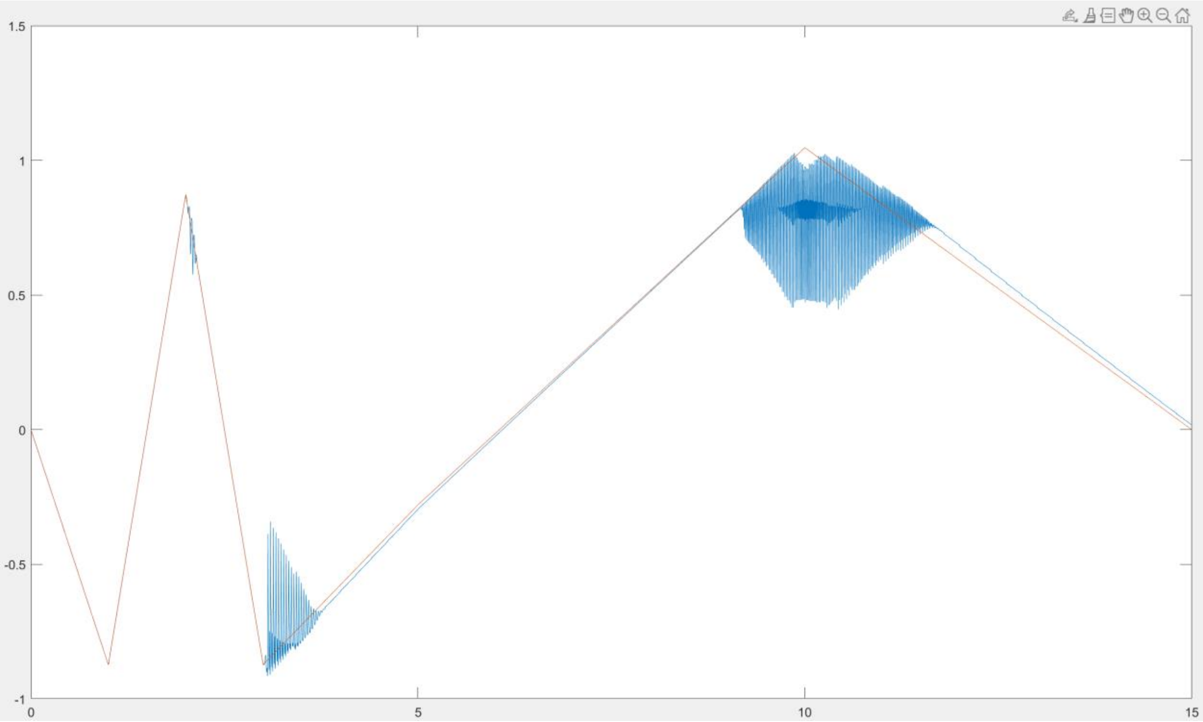


## Altitude Response(Rectangular case): (Sufficiently Stable in all cases)

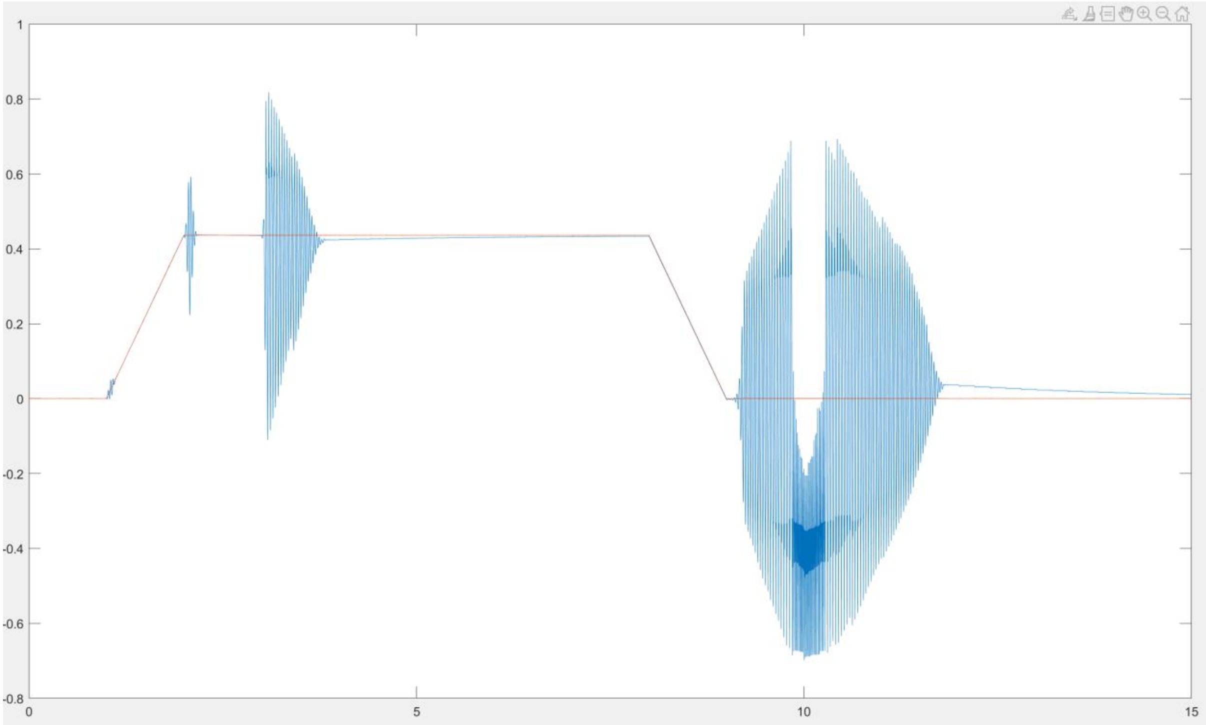



## Attitude Tuning Response:

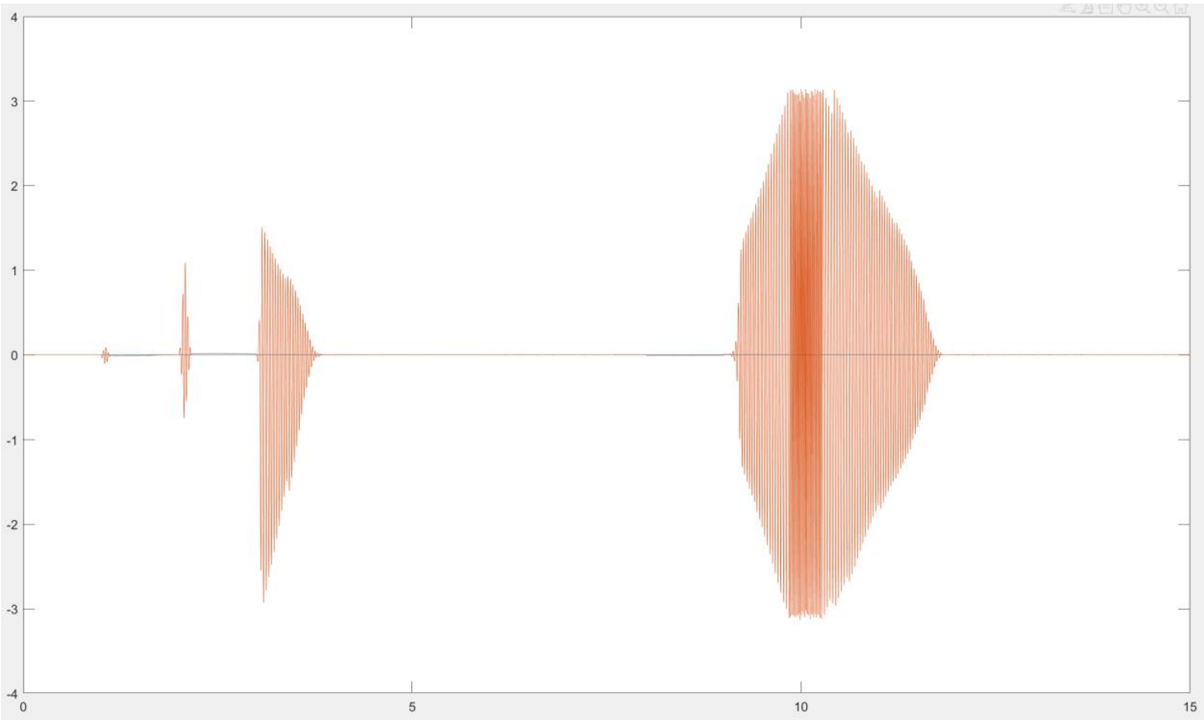
### Roll:



Pitch:



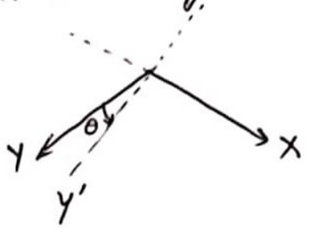

Yaw:



Observation: The Yaw is unaffected, when only one of Roll, Yaw changes. But a simultaneous change in both makes it difficult for the Yaw to control.

Note: The torques are applied in the Body frame.

If a roll $\theta$ already exists, a pitch $\phi$ would now be applied about $Y'$, not $Y$.




But XY controller command is along X Y directions in world frame.

Note: This can be handled to some extent by Yaw controller, but in some cases go out of hand.

## Position Control:

The effect of the inner loop is seen on the outer loop. Disturbances ask for roll and pitch commands simultaneously, leading to growing disturbance in yaw, finally throwing XY controller out of order.

Gradient Descent Algo for Optimal Gains

No: of parameters to optimize, is 18

3 each for $X, Y, Z, \theta, \phi, \psi$.

$$param = \begin{bmatrix} K_{px} \\ K_{py} \\ \vdots \\ K_{dx} \\ K_{dy} \\ \vdots \\ K_{ix} \\ K_{iy} \\ \vdots \end{bmatrix}_{18\times1}$$

$$\text{Let } grad = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ \vdots \\ \vdots \end{bmatrix}_{18\times1} \quad basis = I_{18\times18}$$

```
function  calcGrad
{ for  i = 1: 18

    grad (i) = Cost (param + h * basis(:, i)) - Cos (param - h * basis(:, i))
               ----------------------------------------------------------------
                                         2h

   return(grad);
}
function  Cost (x)
  { Assign  Kp_x = x(1)
            Kp_y = x(2)
              :
            Ki_psi = x(18)
    simout = sim ("Model_name.slx");
    CTE_sq = calc_CTE_sq (simout);
      return (sum (CTE_sq))
                      Array
               number
  }

function  calc_CTE_sq (simout)
  {   Xcmd = simout. X_cmd
      Ycmd = simout. Y_cmd
      Zcmd = simout. Z_cmd
      Xstate = simout. X_state
      Ystate = simout. Y_state
      Zstate = simout. Z_state

    CTE_sq = (X_cmd - X_state)^2 + (Ycmd - Ystate)^2
                + (Zcmd - Zstate)^2

    return (CTE_sq);
  }
```

```
for i=1 to (Max no: of iterations)
  { Check all param values are in desirable range
                                    (non-negative etc)

  del = CalcGrad(param);
  param = param - del * LearningRate;
  }
```

The set of param obtained may be optimal;