

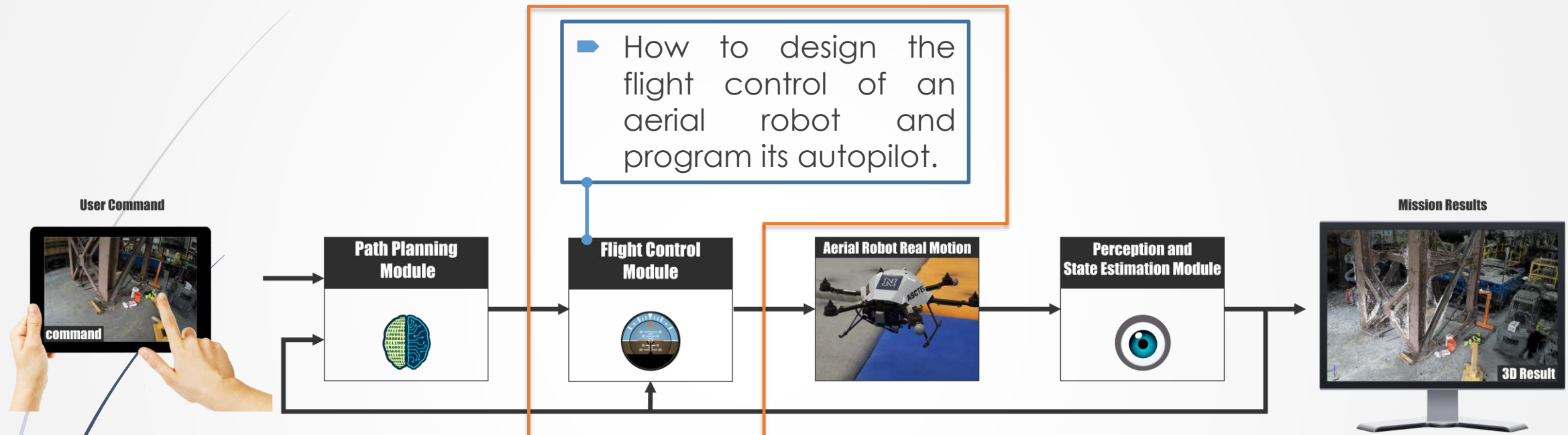


# CS491/691: Introduction to Aerial Robotics

## **Topic: PID Flight Control**

Dr. Kostas Alexis, Dr. Christos Papachristos (CSE)

# The Aerial Robot Loop



Section 3 of our course

# MAV Dynamics

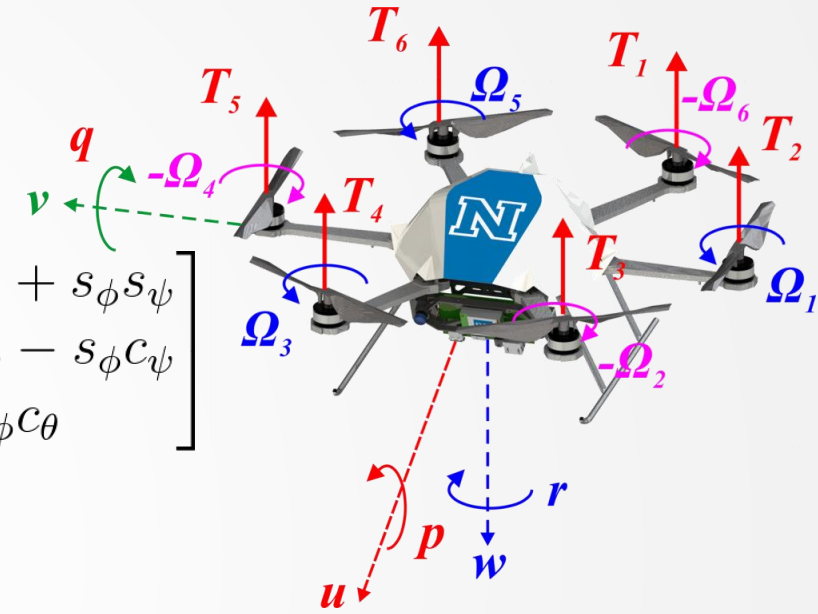
- To append the forces and moments we need to combine their formulation with

$$\begin{bmatrix} \dot{p}_n \\ \dot{p}_e \\ \dot{p}_d \end{bmatrix} = \mathcal{R}_b^v \begin{bmatrix} u \\ v \\ w \end{bmatrix}, \quad \mathcal{R}_b^v = \begin{bmatrix} c_\theta c_\psi & s_\phi s_\theta c_\psi - c_\phi s_\psi & c_\phi s_\theta c_\psi + s_\phi s_\psi \\ c_\theta s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi \\ -s_\theta & s_\phi c_\theta & c_\phi c_\theta \end{bmatrix}$$

$$\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} rv - qw \\ pw - ru \\ qu - pv \end{bmatrix} + \frac{1}{m} \begin{bmatrix} f_x \\ f_y \\ f_z \end{bmatrix}$$

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

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} \frac{J_y - J_z}{J_x} qr \\ \frac{J_z - J_x}{J_y} pr \\ \frac{J_x - J_y}{J_z} r \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} + \begin{bmatrix} \frac{1}{J_x} M_x \\ \frac{1}{J_y} M_y \\ \frac{1}{J_z} M_z \end{bmatrix}$$



- Next step: append the MAV forces and moments

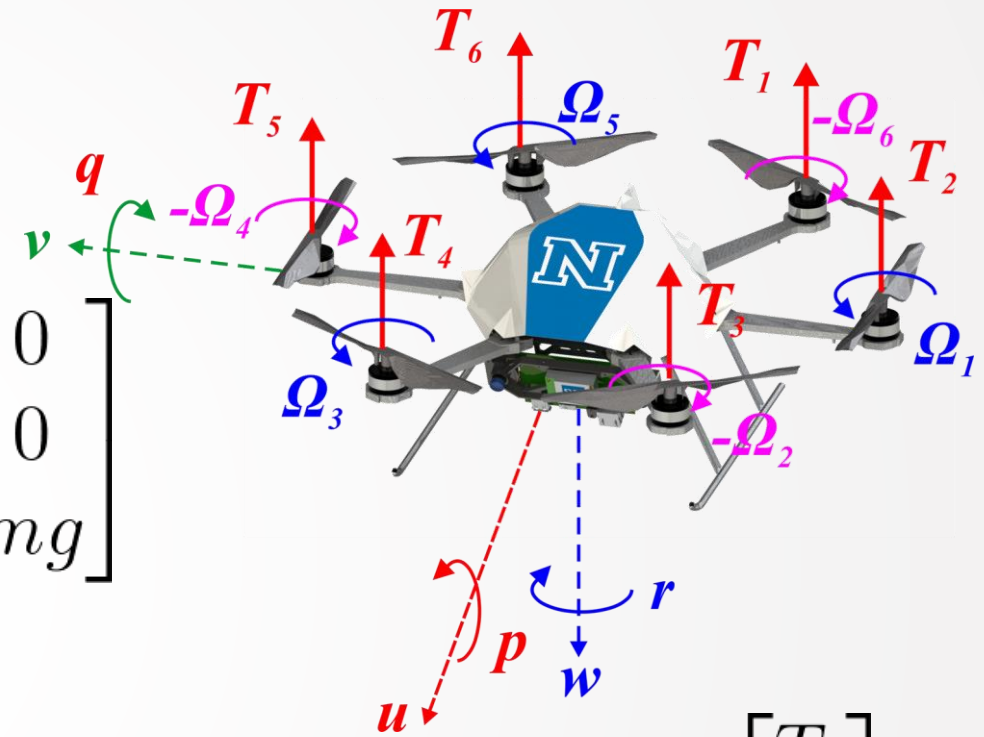
# MAV Dynamics

- MAV forces in the body frame:

$$\mathbf{f}_b = \begin{bmatrix} f_x \\ f_y \\ f_z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \sum_{i=1}^6 T_i \end{bmatrix} - \mathcal{R}_v^b \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix}$$

- Moments in the body frame:

$$\mathbf{m}_b = \begin{bmatrix} M_x \\ M_y \\ M_z \end{bmatrix} = \begin{bmatrix} ls_{30} & l & ls_{30} & -ls_{30} & -l & ls_{30} \\ -lc_{60} & 0 & lc_{60} & lc_{60} & 0 & -lc_{60} \\ -k_m & k_m & -k_m & k_m & -k_m & k_m \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \\ T_6 \end{bmatrix}$$



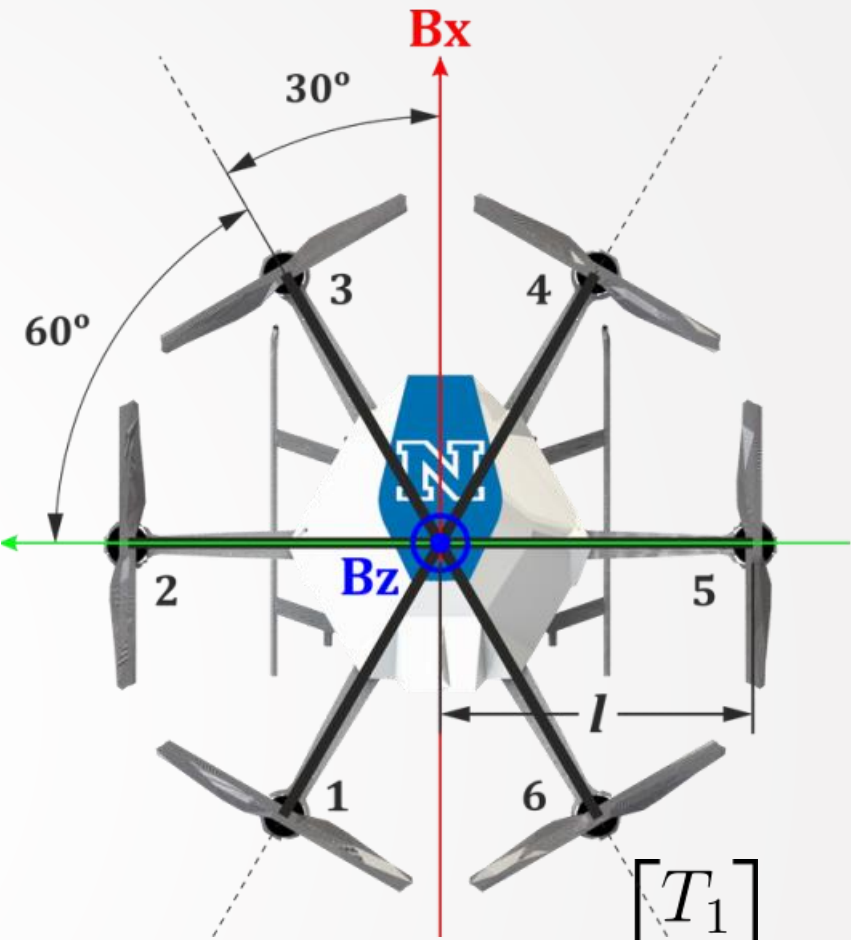
# MAV Dynamics

- MAV forces in the body frame:

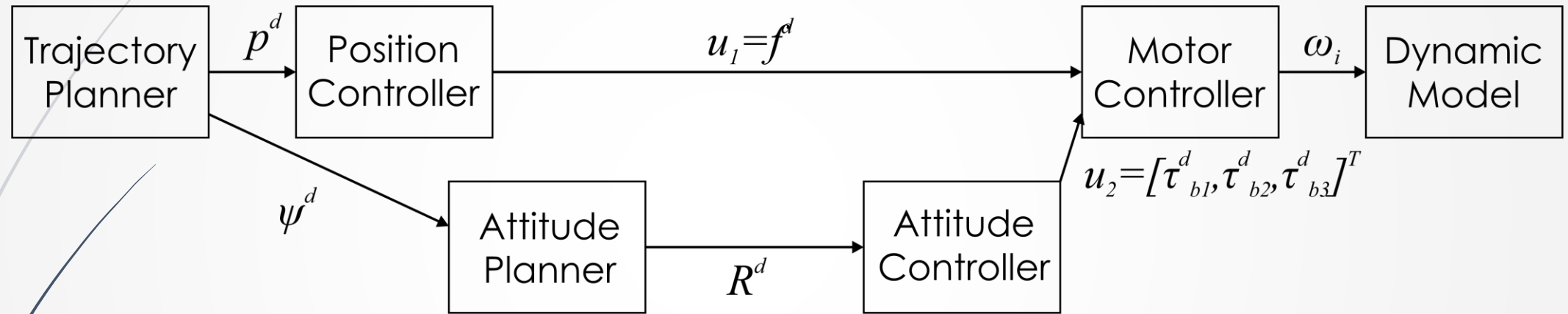
$$\mathbf{f}_b = \begin{bmatrix} f_x \\ f_y \\ f_z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \sum_{i=1}^6 T_i \end{bmatrix} - \mathcal{R}_v^b \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix}$$

- Moments in the body frame:

$$\mathbf{m}_b = \begin{bmatrix} M_x \\ M_y \\ M_z \end{bmatrix} = \begin{bmatrix} ls_{30} & l & ls_{30} & -ls_{30} & -l & ls_{30} \\ -lc_{60} & 0 & lc_{60} & lc_{60} & 0 & -lc_{60} \\ -k_m & k_m & -k_m & k_m & -k_m & k_m \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \\ T_6 \end{bmatrix}$$

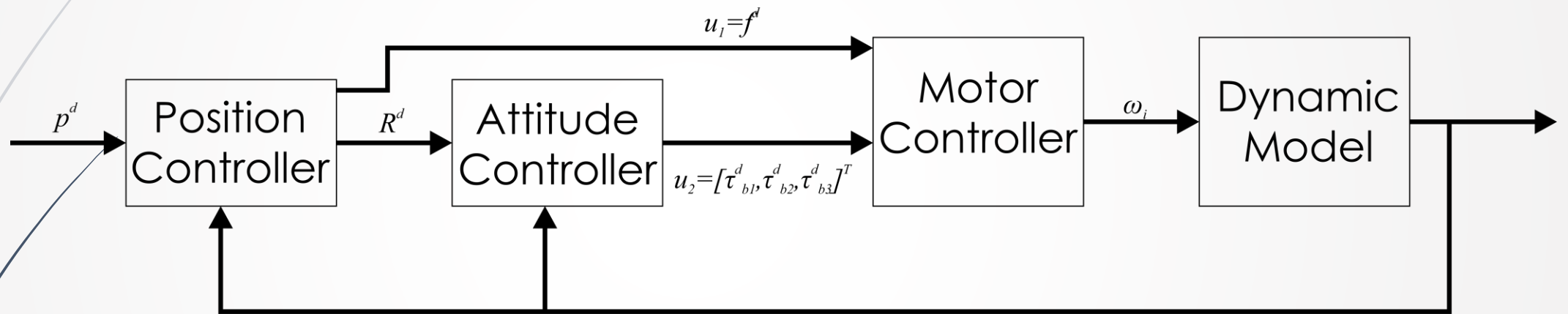


# Control System Block Diagram



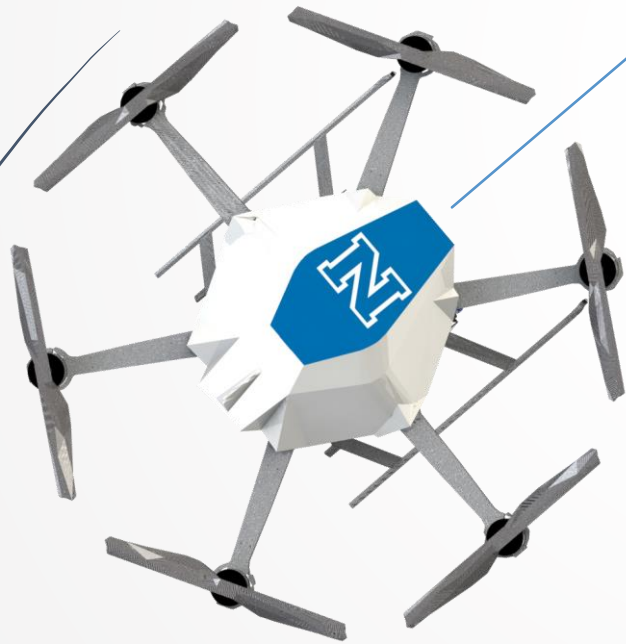
► There are simpler

# Control System Block Diagram



➤ Simplified loop

# Controlling a Multicopter along the x-axis



- ▶ Assume a single-axis multicopter.
  - ▶ The system has to coordinate its pitching motion and thrust to move to the desired point ahead of its axis.
  - ▶ Roll is considered to be zero, yaw is considered to be constant. No initial velocity. No motion is expressed in any other axis.
  - ▶ A system of only two degrees of freedom.



# Controlling a Multicopter along the x-axis

- Simplified linear dynamics

$$\begin{bmatrix} \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 \\ 1/J_y \end{bmatrix} M_y$$

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ -g \end{bmatrix} \theta$$



# Controlling a Multicopter along the x-axis

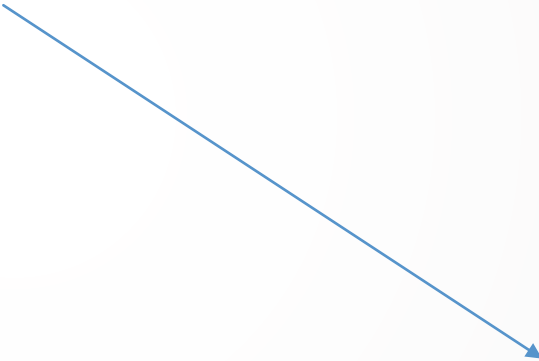
- Explanation of translation model

$$ma = F_x \Rightarrow ma = T_{TOT} \cos(-\theta) \Rightarrow ma \approx -\theta T_{TOT}$$

$$ma = -\theta mg$$

$$\text{s.t. } T_{TOT} = mg$$

$$a = \ddot{x}$$


$$\begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ -g \end{bmatrix} \theta$$

# Controlling a Multicopter along the x-axis

- Simplified linear dynamics

$$\begin{bmatrix} \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 \\ 1/J_y \end{bmatrix} M_y$$

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ -g \end{bmatrix} \theta$$

How does this system behave?



# Controlling a Multicopter along the x-axis

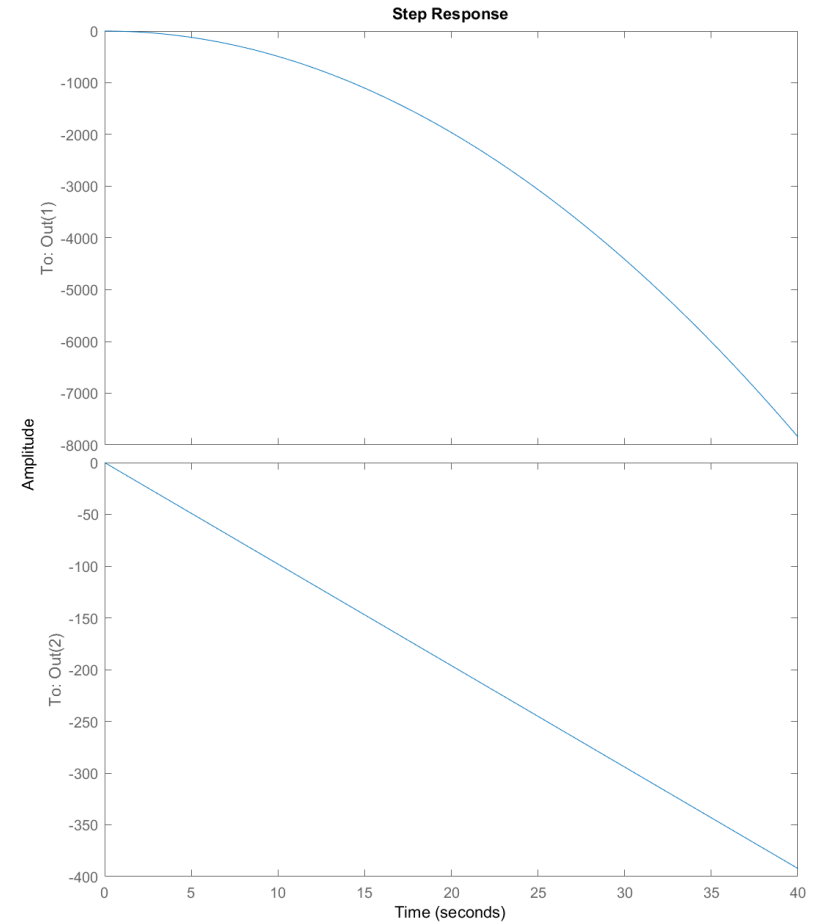
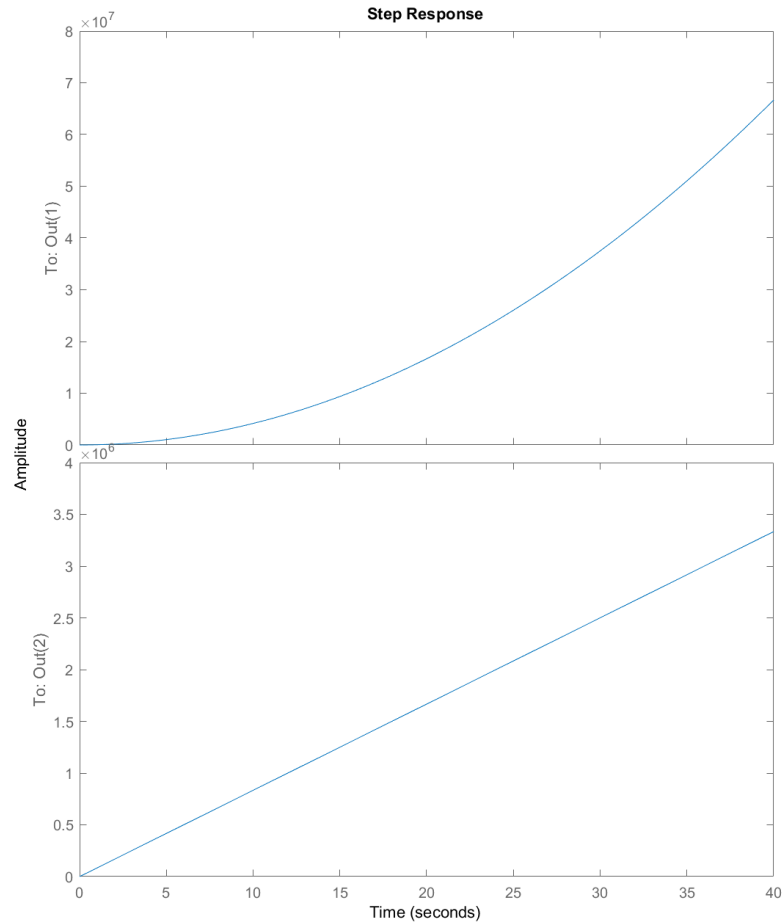
```
%% Simple Modeling and Control study
clear;
J_y = 1.2e-5;
g = 9.806; mass = 1.2;

% Pitch Linear Model
A_p = [0 1; 0 0]; B_p = [0; 1/J_y];
C_p = eye(2); D_p = zeros(2,1);
ss_pitch = ss(A_p,B_p,C_p,D_p);

% x Linear Model
A_x = [0 1; 0 0]; B_x = [0; -g];
C_x = eye(2); D_x = zeros(2,1);
ss_x = ss(A_x,B_x,C_x,D_x);

% Observe the Step responses of the system
subplot(1,2,1); step(ss_pitch);
subplot(1,2,2); step(ss_x);
```

# Controlling a Multicopter along the x-axis



# Controlling a Multicopter along the x-axis

- Simplified linear dynamics

$$\begin{bmatrix} \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 \\ 1/J_y \end{bmatrix} M_y$$

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ -g \end{bmatrix} \theta$$

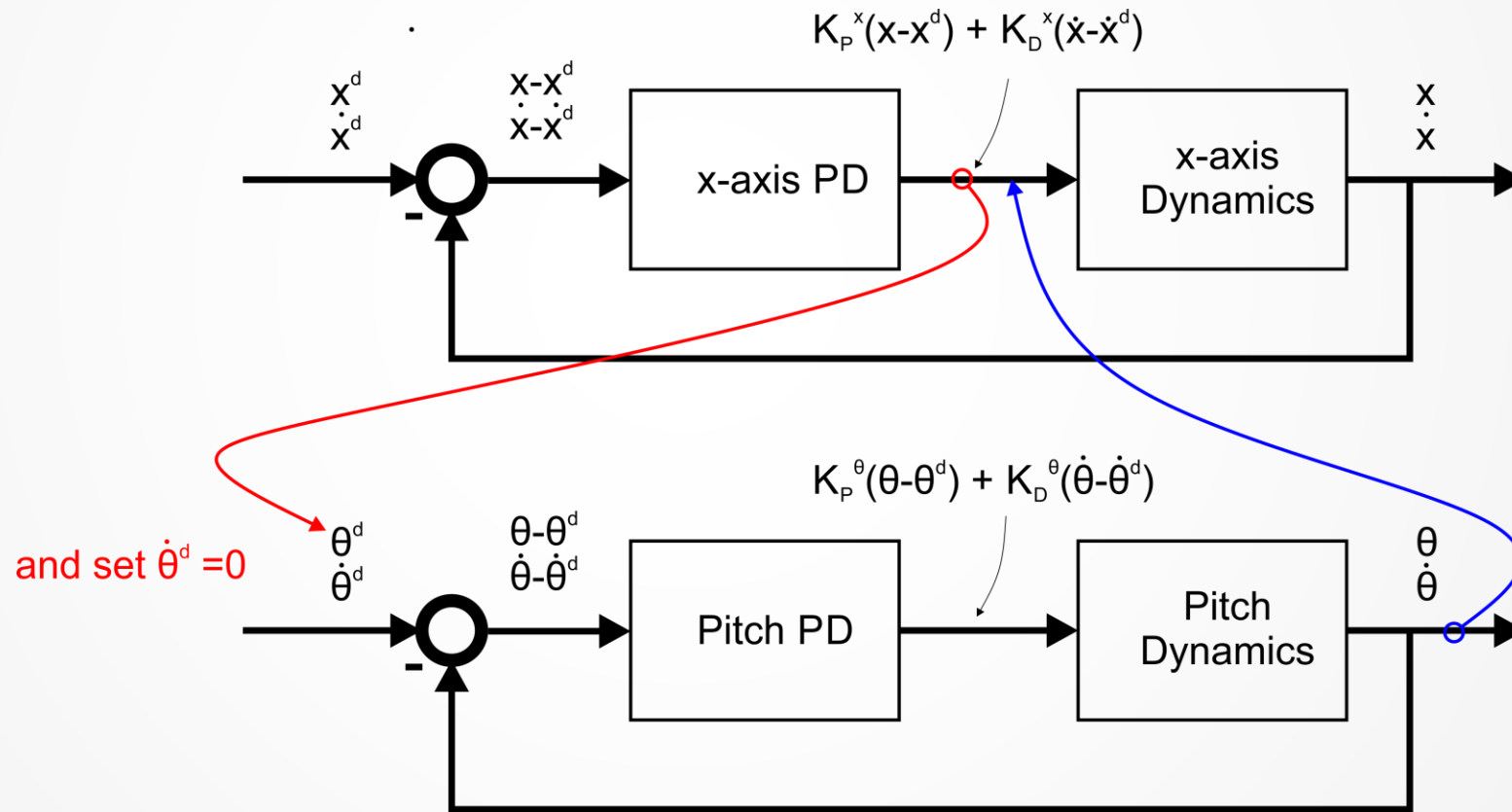
**How to control this system?**

Most common way: use of PD controllers



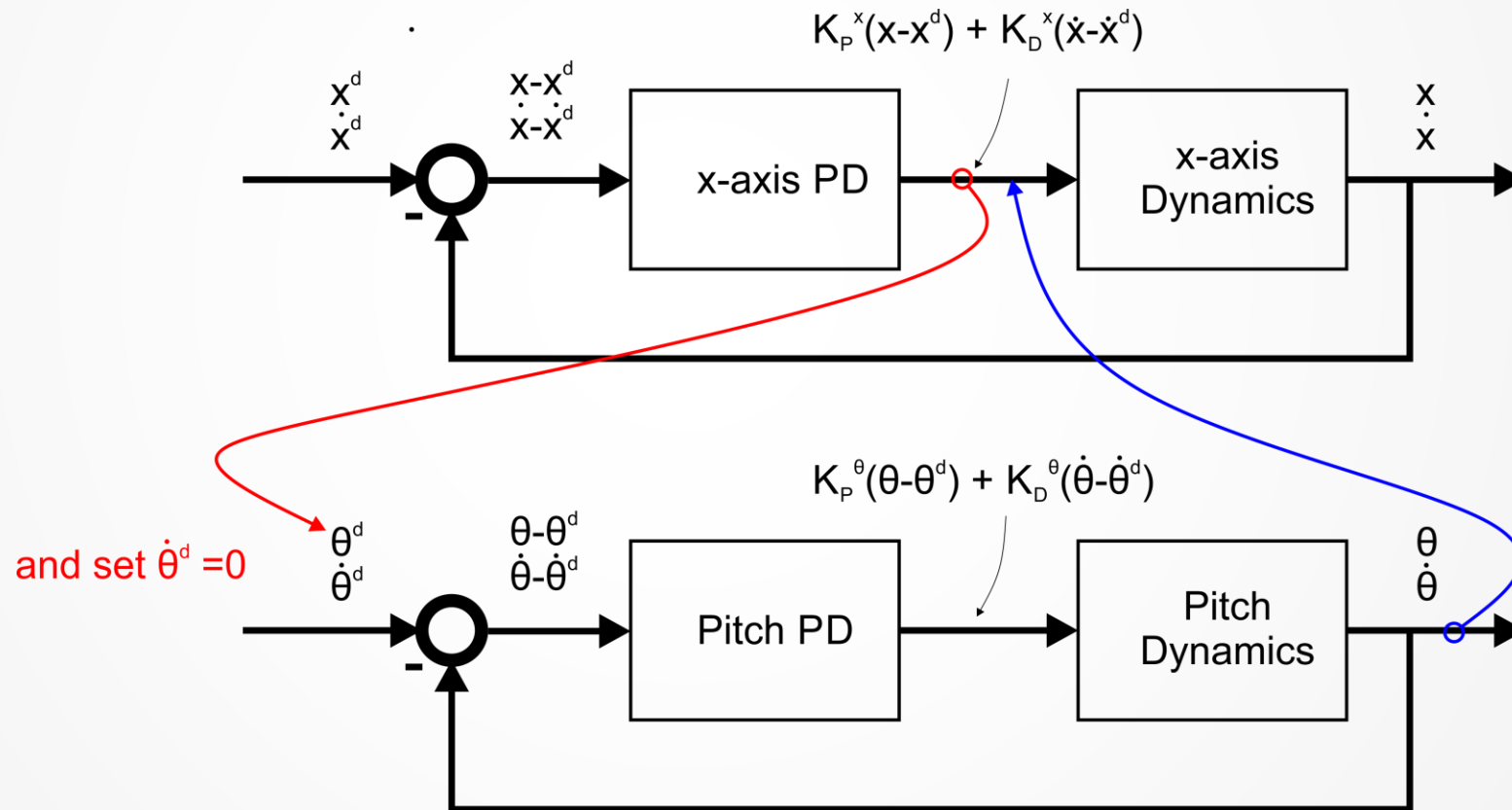
# Controlling a Multicopter along the x-axis

## Decoupled Control Structure



# Controlling a Multicopter along the x-axis

## Decoupled Control Structure

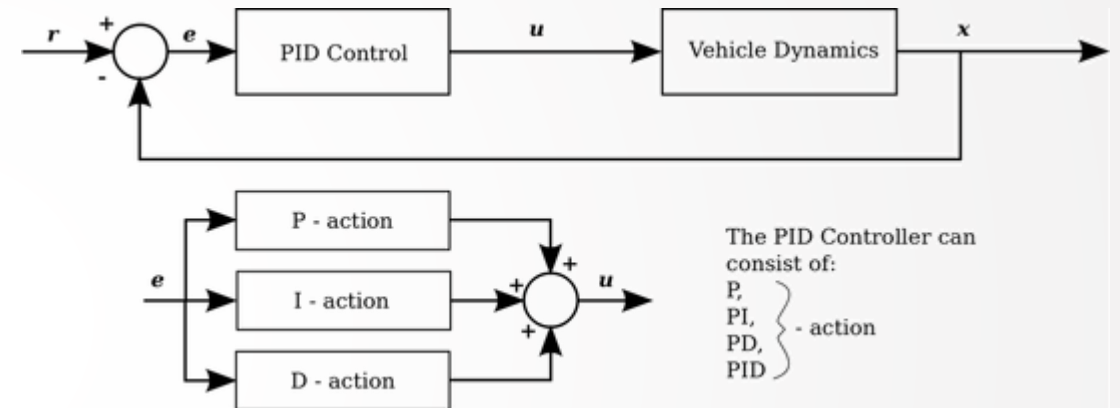


How to select the PD gains?



# A mini introduction to PID Control

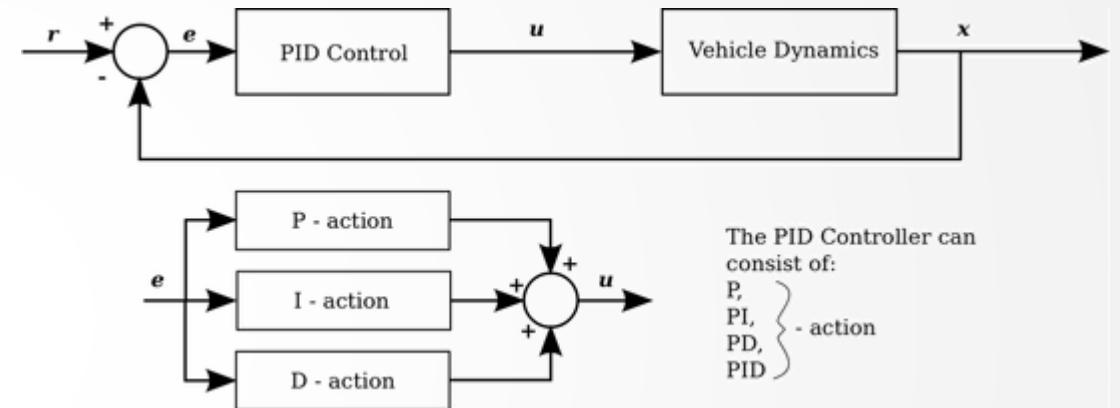
- ▶ PID Control stands for Proportional-Integral-Derivative feedback control and corresponds to one of the most commonly used controllers used in industry.
- ▶ It's success is based on its capacity to efficiently and robustly control a variety of processes and dynamic systems, while having an extremely simple structure and intuitive tuning procedures.
- ▶ Not comparable in performance with modern control strategies, but still the most common starting point



**How to select the PD gains?**

# A mini introduction to PID Control

- ▶ PID Control stands for Proportional-Integral-Derivative feedback control and corresponds to one of the most commonly used controllers used in industry.
- ▶ It's success is based on its capacity to efficiently and robustly control a variety of processes and dynamic systems, while having an extremely simple structure and intuitive tuning procedures.
- ▶ Not comparable in performance with modern control strategies, but still the most common starting point



**How to select the PD gains?**

# Controlling a Multicopter along the x-axis

## ➤ MATLAB Implementation

```
%% Simple Modeling and Control study
clear;
J_y = 1.2e-5;
g = 9.806; mass = 1.2;

% Pitch Linear Model
A_p = [0 1; 0 0]; B_p = [0; 1/J_y];
C_p = eye(2); D_p = zeros(2,1);
ss_pitch = ss(A_p,B_p,C_p,D_p);

% x Linear Model
A_x = [0 1; 0 0]; B_x = [0; -g];
C_x = eye(2); D_x = zeros(2,1);
ss_x = ss(A_x,B_x,C_x,D_x);

% Observe the Step responses of the system
subplot(1,2,1); step(ss_pitch);
subplot(1,2,2); step(ss_x);

%% Design the PD Controller for Pitch

ss_pitch_tf = tf(ss_pitch); ss_pitch_tf = ss_pitch_tf(1);
pidTuner(ss_pitch_tf, 'PD')

%% Design the PD Controller for X translational dynamics

ss_x_tf = tf(ss_x); ss_x_tf = ss_x_tf(1);
pidTuner(ss_x_tf, 'PD')

%% Verification
close all;

K_P_pitch = 25.8e-5; K_D_pitch = 9.82e-5;
PD_PITCH_GAINS = [K_P_pitch 0; 0 K_D_pitch];

ss_pitch_cl = feedback(PD_PITCH_GAINS*ss_pitch,[1 1]);

K_P_x = -1.17; K_D_x = -0.823;
PD_X_GAINS = [K_P_x 0; 0 K_D_x];
ss_x_cl = feedback(PD_X_GAINS*ss_x,[1 1]);

subplot(1,2,1); step(ss_pitch_cl);
subplot(1,2,2); step(ss_x_cl);
```

# Controlling a Multicopter along the x-axis

## ➤ MATLAB Implementation

```
%% Simple Modeling and Control study
clear;
J_y = 1.2e-5;
g = 9.806; mass = 1.2;

% Pitch Linear Model
A_p = [0 1; 0 0]; B_p = [0; 1/J_y];
C_p = eye(2); D_p = zeros(2,1);
ss_pitch = ss(A_p,B_p,C_p,D_p);

% x Linear Model
A_x = [0 1; 0 0]; B_x = [0; -g];
C_x = eye(2); D_x = zeros(2,1);
ss_x = ss(A_x,B_x,C_x,D_x);

% Observe the Step responses of the system
subplot(1,2,1); step(ss_pitch);
subplot(1,2,2); step(ss_x);

%% Design the PD Controller for Pitch

ss_pitch_tf = tf(ss_pitch); ss_pitch_tf = ss_pitch_tf(1);
pidTuner(ss_pitch_tf, 'PD')

%% Design the PD Controller for X translational dynamics

ss_x_tf = tf(ss_x); ss_x_tf = ss_x_tf(1);
pidTuner(ss_x_tf, 'PD')

%% Verification
close all;

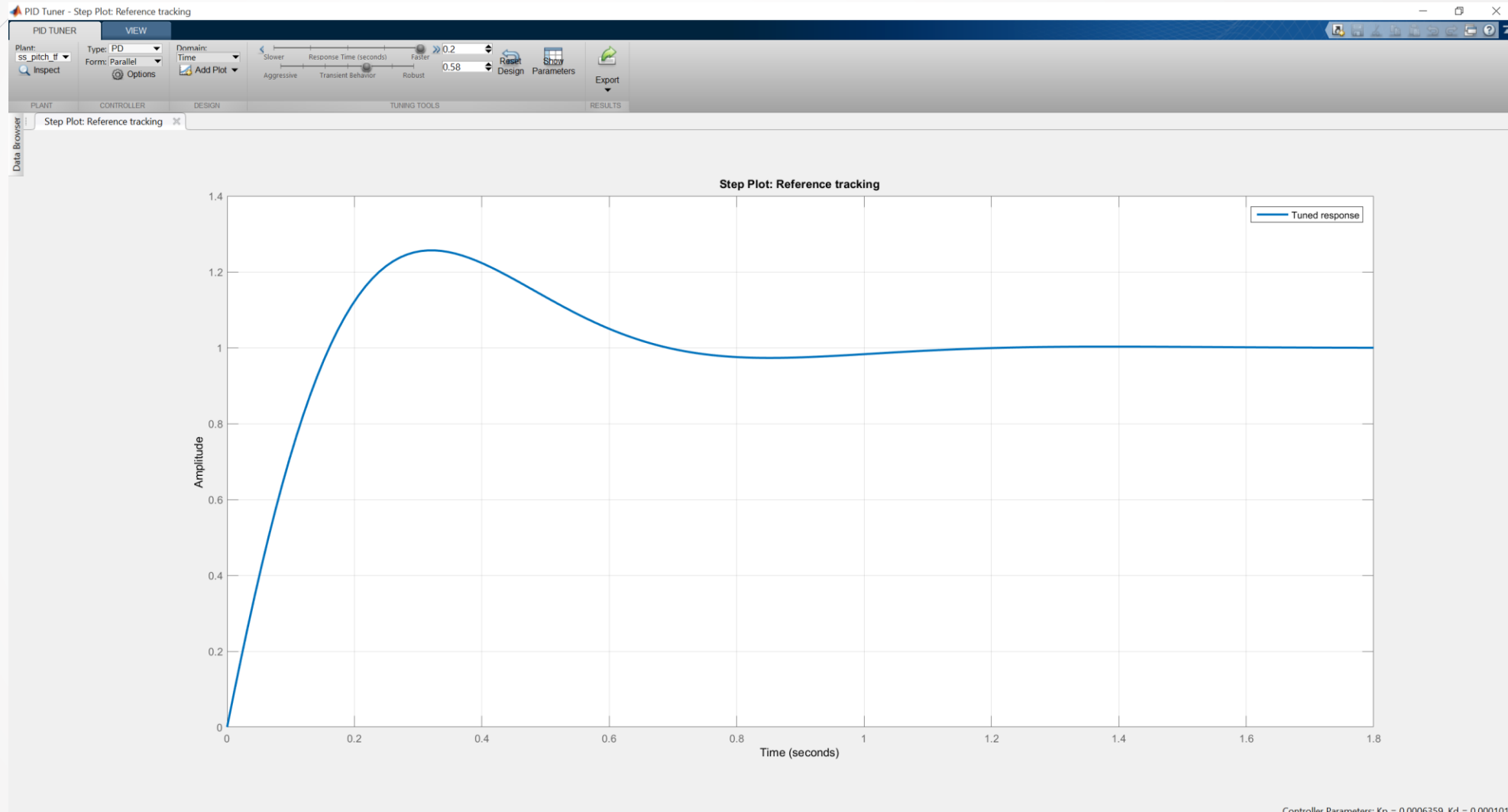
K_P_pitch = 25.8e-5; K_D_pitch = 9.82e-5;
PD_PITCH_GAINS = [K_P_pitch 0; 0 K_D_pitch];

ss_pitch_cl = feedback(PD_PITCH_GAINS*ss_pitch,[1 1]);

K_P_x = -1.17; K_D_x = -0.823;
PD_X_GAINS = [K_P_x 0; 0 K_D_x];
ss_x_cl = feedback(PD_X_GAINS*ss_x,[1 1]);

subplot(1,2,1); step(ss_pitch_cl);
subplot(1,2,2); step(ss_x_cl);
```

# Controlling a Multicopter along the x-axis



Controller Parameters:  $K_p = 0.0006359$ ,  $K_d = 0.0001018$

# Controlling a Multicopter along the x-axis

## ➤ MATLAB Implementation

```
%% Simple Modeling and Control study
clear;
J_y = 1.2e-5;
g = 9.806; mass = 1.2;

% Pitch Linear Model
A_p = [0 1; 0 0]; B_p = [0; 1/J_y];
C_p = eye(2); D_p = zeros(2,1);
ss_pitch = ss(A_p,B_p,C_p,D_p);

% x Linear Model
A_x = [0 1; 0 0]; B_x = [0; -g];
C_x = eye(2); D_x = zeros(2,1);
ss_x = ss(A_x,B_x,C_x,D_x);

% Observe the Step responses of the system
subplot(1,2,1); step(ss_pitch);
subplot(1,2,2); step(ss_x);

%% Design the PD Controller for Pitch

ss_pitch_tf = tf(ss_pitch); ss_pitch_tf = ss_pitch_tf(1);
pidTuner(ss_pitch_tf, 'PD')

%% Design the PD Controller for X translational dynamics

ss_x_tf = tf(ss_x); ss_x_tf = ss_x_tf(1);
pidTuner(ss_x_tf, 'PD')

%% Verification
close all;

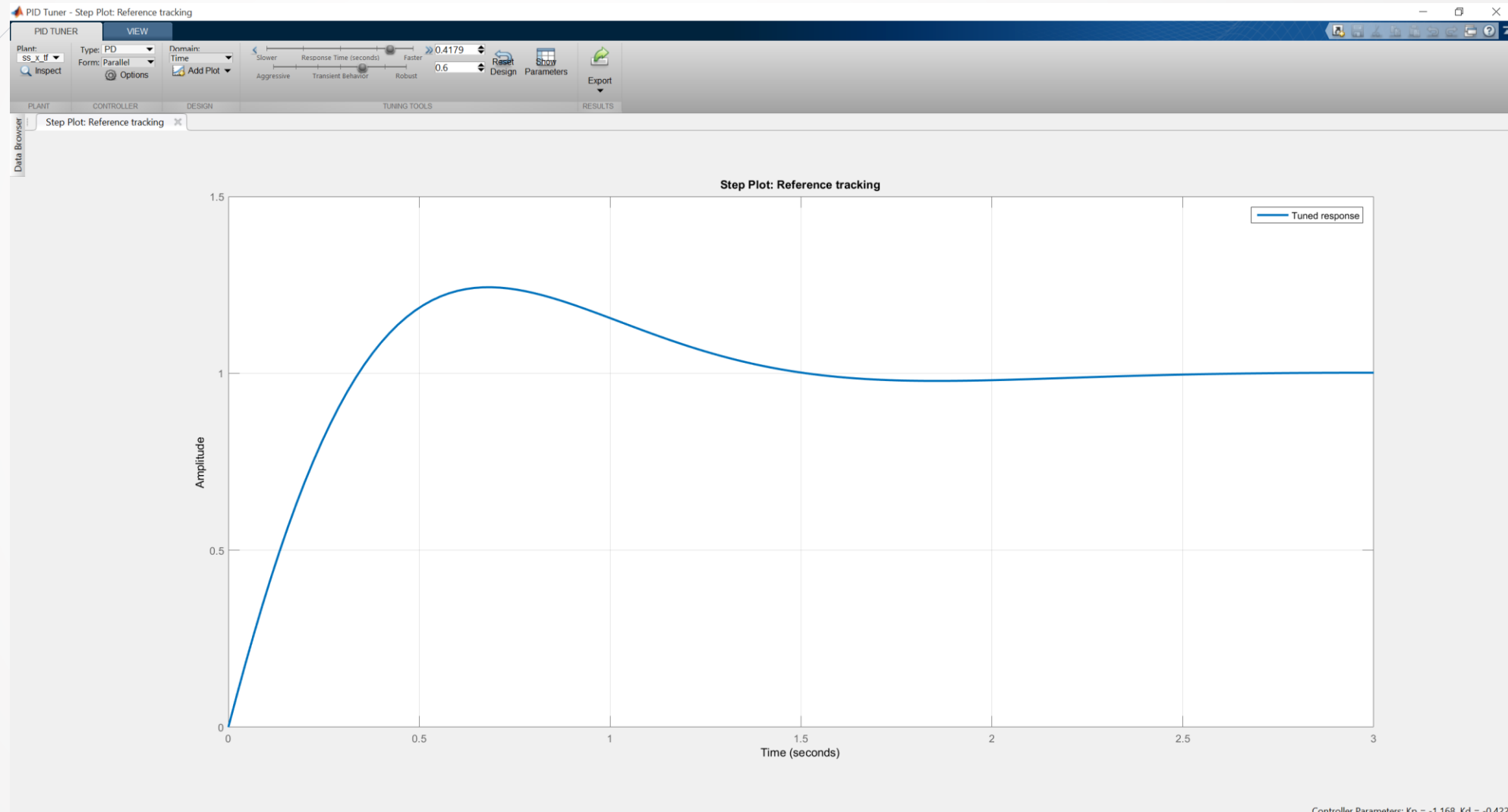
K_P_pitch = 25.8e-5; K_D_pitch = 9.82e-5;
PD_PITCH_GAINS = [K_P_pitch 0; 0 K_D_pitch];

ss_pitch_cl = feedback(PD_PITCH_GAINS*ss_pitch,[1 1]);

K_P_x = -1.17; K_D_x = -0.823;
PD_X_GAINS = [K_P_x 0; 0 K_D_x];
ss_x_cl = feedback(PD_X_GAINS*ss_x,[1 1]);

subplot(1,2,1); step(ss_pitch_cl);
subplot(1,2,2); step(ss_x_cl);
```

# Controlling a Multicopter along the x-axis



Controller Parameters:  $K_p = -1.168$ ,  $K_d = -0.4227$

# Controlling a Multicopter along the x-axis

## ➤ MATLAB Implementation

```
%% Simple Modeling and Control study
clear;
J_y = 1.2e-5;
g = 9.806; mass = 1.2;

% Pitch Linear Model
A_p = [0 1; 0 0]; B_p = [0; 1/J_y];
C_p = eye(2); D_p = zeros(2,1);
ss_pitch = ss(A_p,B_p,C_p,D_p);

% x Linear Model
A_x = [0 1; 0 0]; B_x = [0; -g];
C_x = eye(2); D_x = zeros(2,1);
ss_x = ss(A_x,B_x,C_x,D_x);

% Observe the Step responses of the system
subplot(1,2,1); step(ss_pitch);
subplot(1,2,2); step(ss_x);

%% Design the PD Controller for Pitch

ss_pitch_tf = tf(ss_pitch); ss_pitch_tf = ss_pitch_tf(1);
pidTuner(ss_pitch_tf, 'PD')

%% Design the PD Controller for X translational dynamics

ss_x_tf = tf(ss_x); ss_x_tf = ss_x_tf(1);
pidTuner(ss_x_tf, 'PD')

%% Verification
close all;

K_P_pitch = 25.8e-5; K_D_pitch = 9.82e-5;
PD_PITCH_GAINS = [K_P_pitch 0; 0 K_D_pitch];

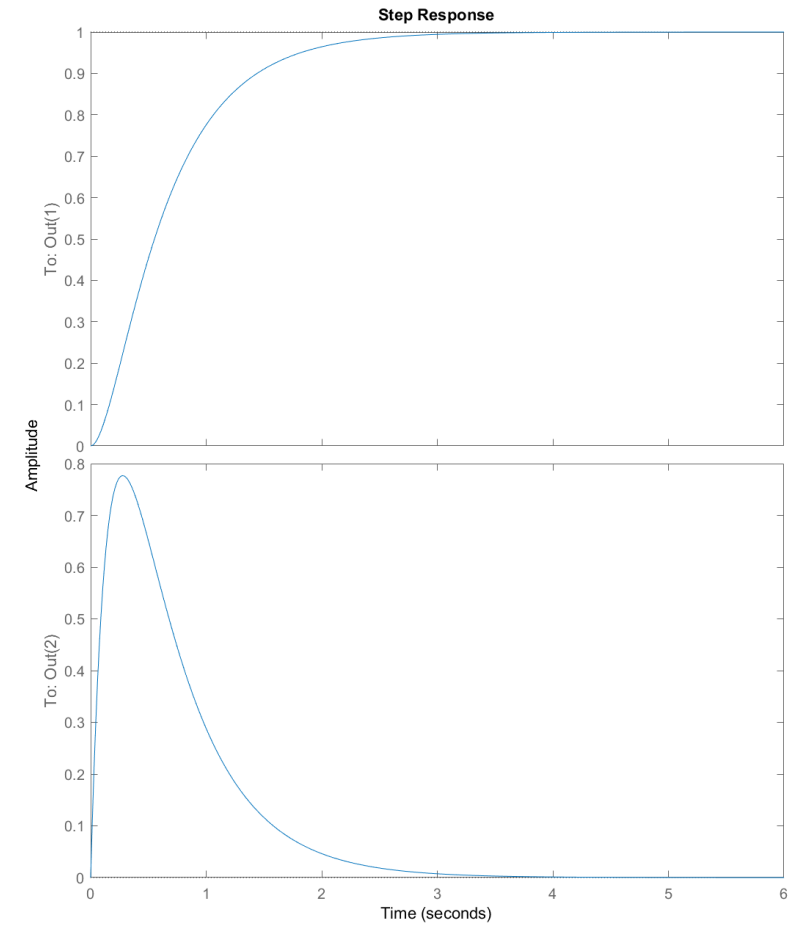
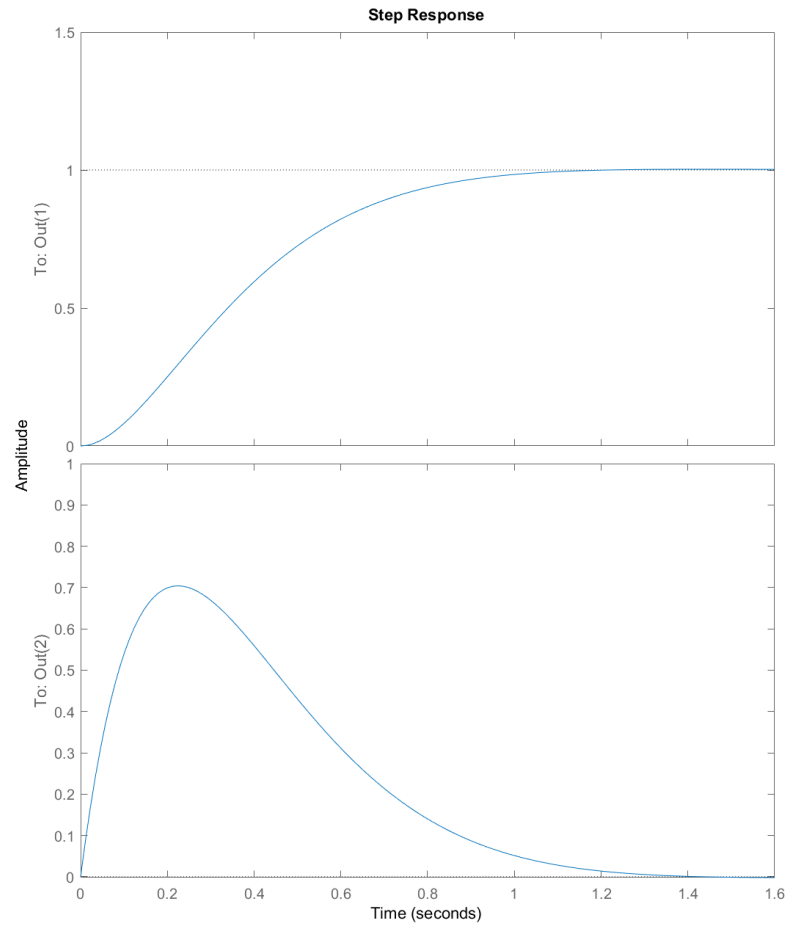
ss_pitch_cl = feedback(PD_PITCH_GAINS*ss_pitch,[1 1]);

K_P_x = -1.17; K_D_x = -0.823;
PD_X_GAINS = [K_P_x 0; 0 K_D_x];
ss_x_cl = feedback(PD_X_GAINS*ss_x,[1 1]);

subplot(1,2,1); step(ss_pitch_cl);
subplot(1,2,2); step(ss_x_cl);
```



# Controlling a Multicopter along the x-axis





# Real-life Limitations

- ▶ The control margins of the aerial vehicle have limits and therefore the PID controller has to be designed account for these constraints.
- ▶ The integral term needs special caution due to the often critically stable or unstable characteristics expressed by unmanned aircraft.
- ▶ With the exception of hover/or trimmed-flight, an aerial vehicle is a nonlinear system. As the PID is controller, it naturally cannot maintain an equally good behavior for the full flight envelope of the system. A variety of techniques such as Gain scheduling are employed to deal with this fact.



# Find out more

- ▶ <http://www.kostasalexis.com/pid-control.html>
- ▶ <http://www.kostasalexis.com/lqr-control.html>
- ▶ <http://www.kostasalexis.com/linear-model-predictive-control.html>
- ▶ <http://ctms.engin.umich.edu/CTMS/index.php?example=InvertedPendulum&section=ControlStateSpace>
- ▶ <http://www.kostasalexis.com/literature-and-links.html>

A black and white photograph of a drone flying in the foreground. The drone is a quadcopter with a white protective cover over its camera. In the background, there is a construction site with several large cranes and a building under construction. The scene is slightly blurred, suggesting motion or a shallow depth of field.

**Thank you!**

Please ask your question!