UAV Systems & Control
Lecture 2

Control Systems

Vector Operations

Rotating Coordinates

Vector Differentiation

Controls Refresher

SISO Feedback Control

Control Structures (e.g. PID, etc.)

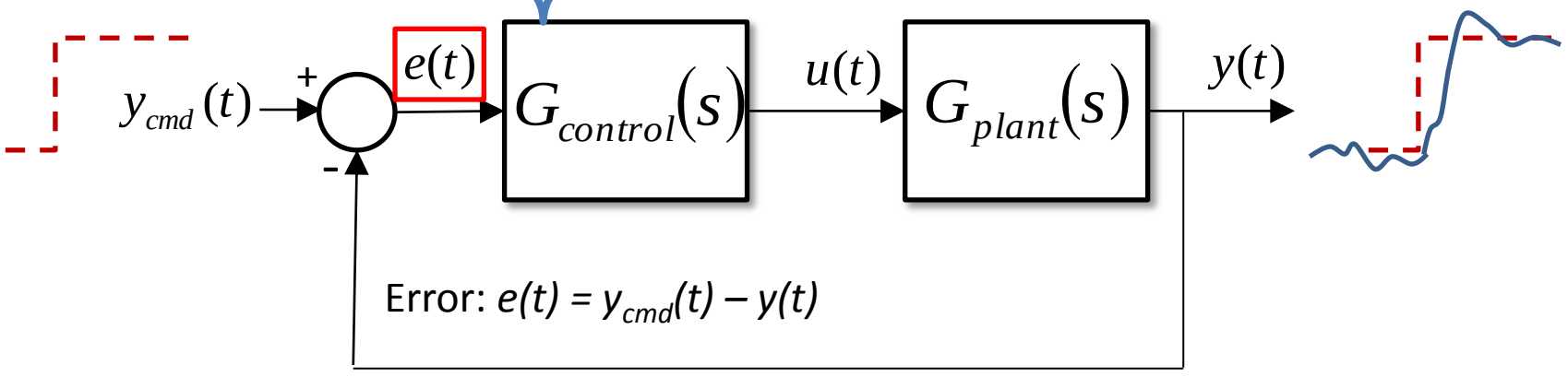
Controller Implementation

Feedback Control

- Controls engineers use feedback control to regulate systems

Feedback Controller:
Device, algorithm or filter designed to drive feedback error toward zero in a desirable way
- e.g. autopilot

Plant: Physical process we're trying to control
- Has 1+ inputs (*u(t)*) & 1+ outputs (*y(t)*)
- Aircraft example:
  u(t)=elevator surface
  y(t)=pitch angle



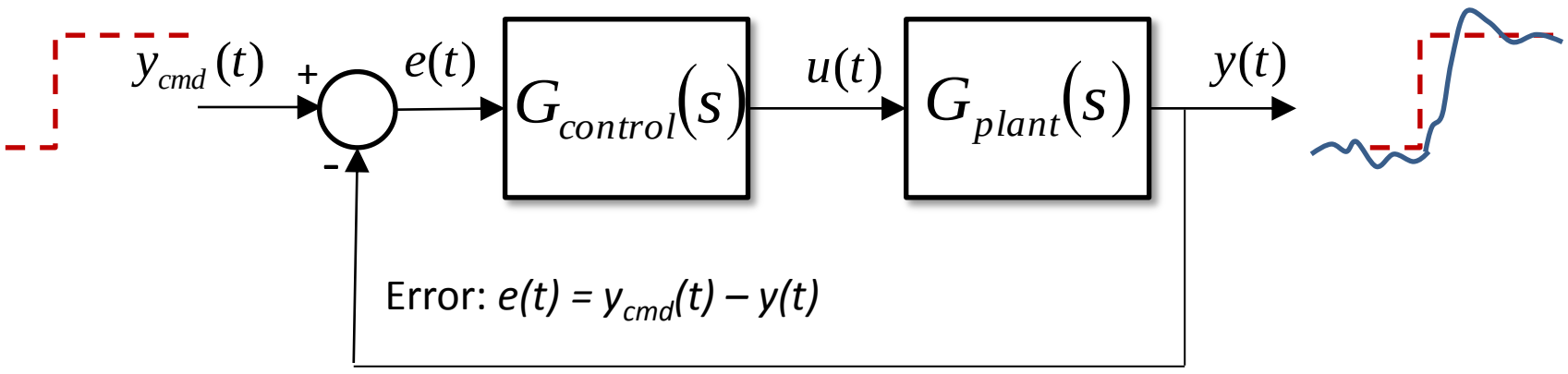
$y(t) \rightarrow y_{cmd}(t)$ as $e(t) \rightarrow 0$

Feedback Control

- Controls engineers use feedback control to regulate systems

Feedback controller is used to "shape" the response from desired to achieved
- e.g. Design $G_{control}(s)$ to achieve a desired closed loop transfer function (hence a desired time response)

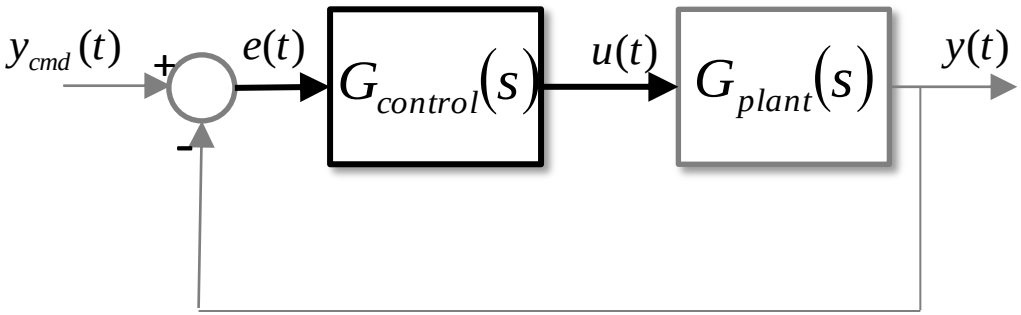
$$\frac{Y(s)}{Y_{cmd}(s)} = \frac{G_{control}(s)G_{plant}(s)}{1+G_{control}(s)G_{plant}(s)}$$

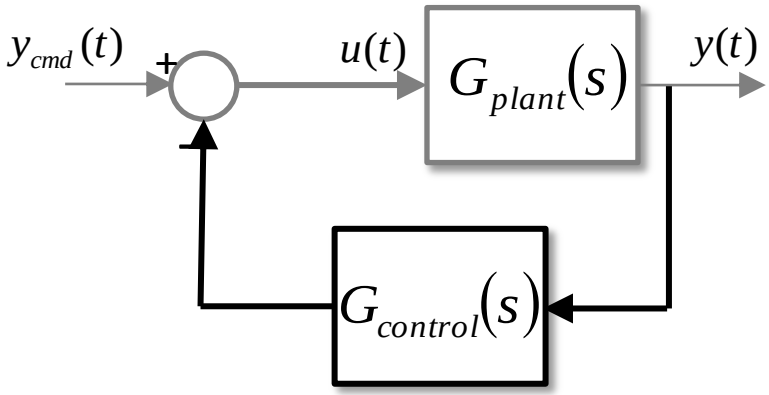


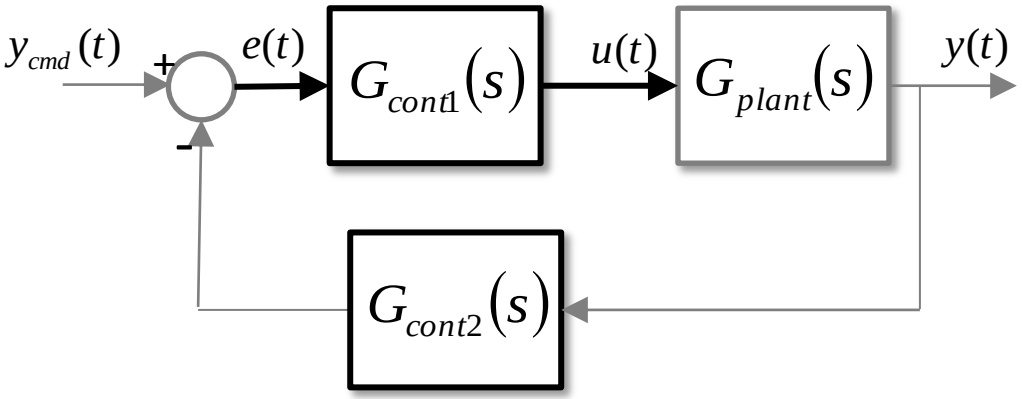
$y(t) \rightarrow y_{cmd}(t)$ as $e(t) \rightarrow 0$

Feedback Control

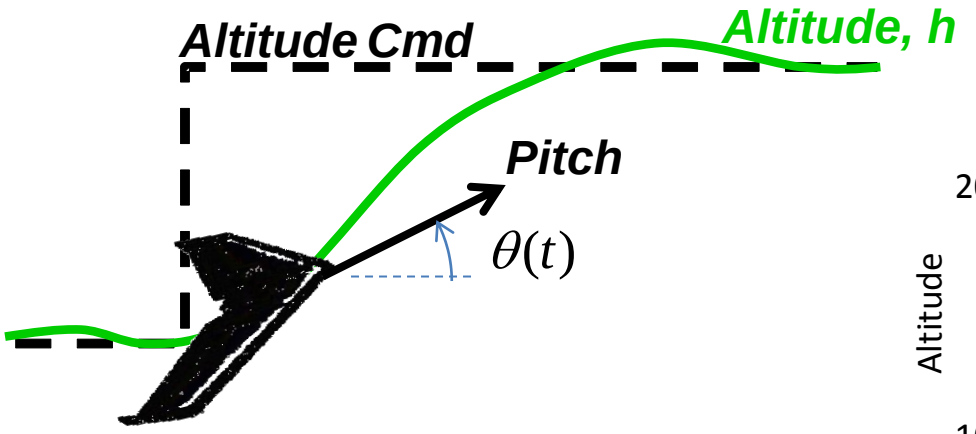
- Controllers/Compensators can be in the forward path or the feedback path, or both

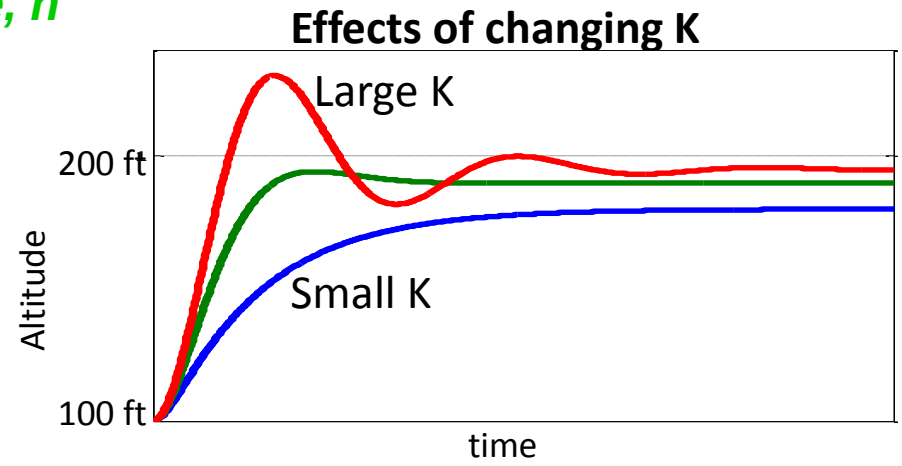




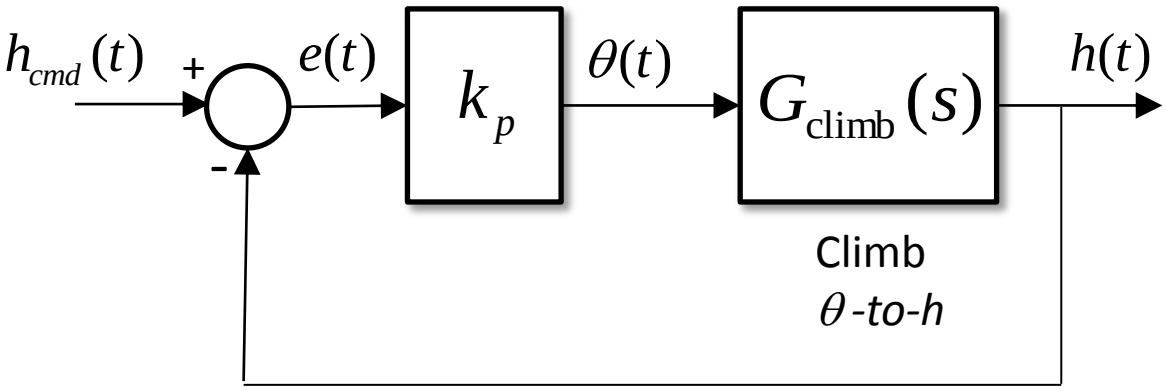


Example Altitude Control Logic





Pitch Cmd = k_p*(AltitudeCmd-Altitude)



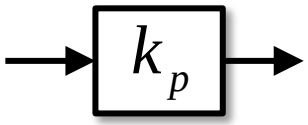
In this example, increasing *K* reduced the rise time and steady-state error, but it also increased overshoot

- We'll need more flexibility to *shape* the desired output

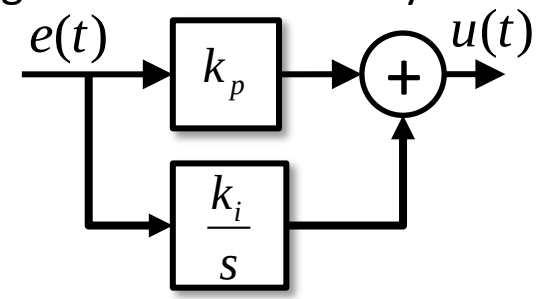
Feedback Control

Controller $G_{control}(s)$ can have many forms:

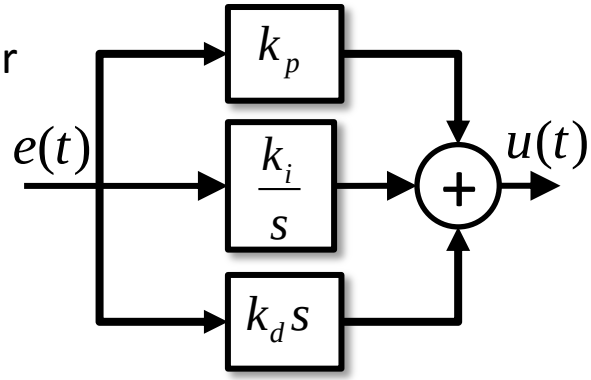
P (Proportional)
Control signal is proportional to error



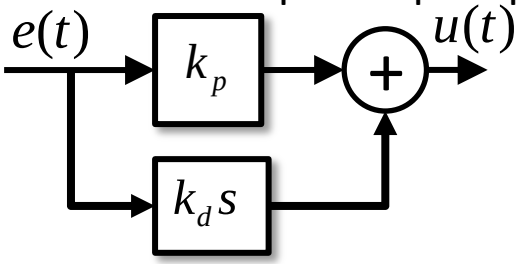
PI (Proportional-Integral)
- Integral to remove steady state error



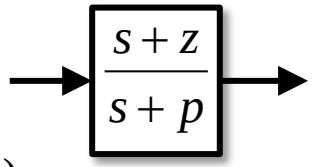
PID (Proportional-Integral-Derivative)

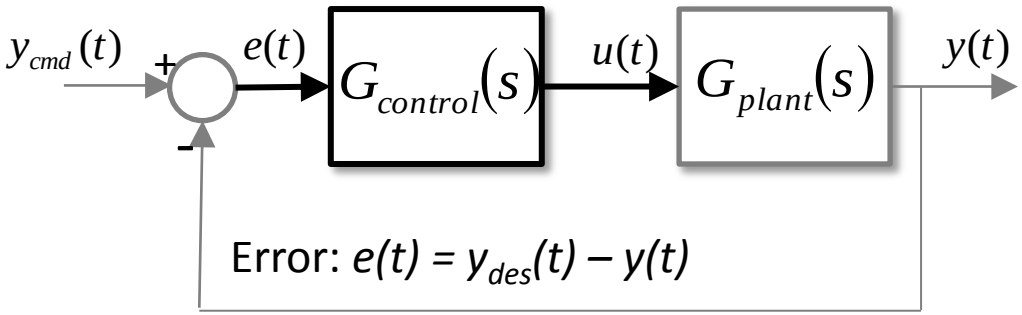


PD (Proportional-Derivative)
- Derivative to speed up response

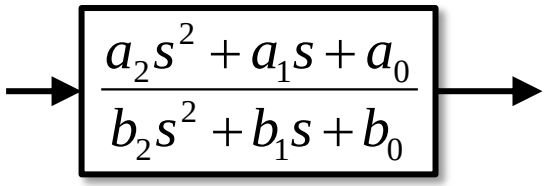


Lead/Lag (or Lag/Lead)
Improve a frequency response or move a pole



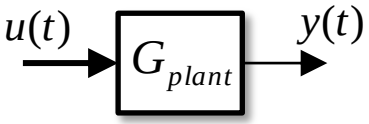


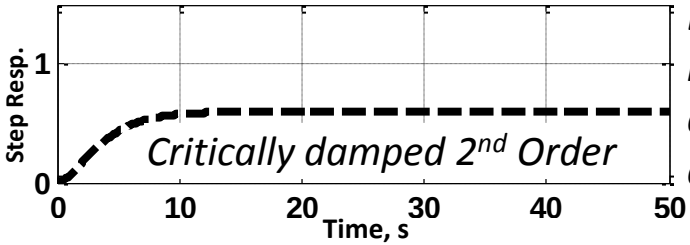
Any Designed Filter



Feedback Control Effects

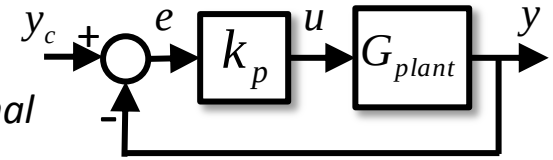
Consider the plant:

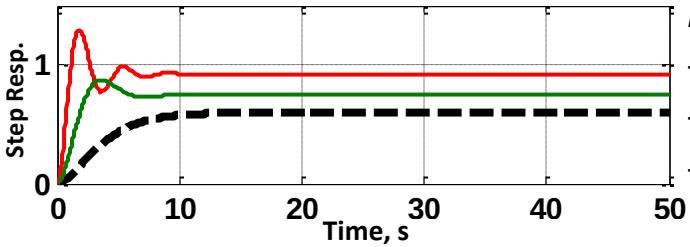




Example Plant response has steady-state error and is slower than desired

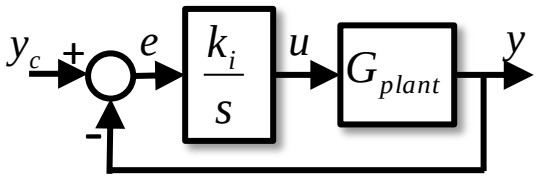
P: *Pure Proportional*

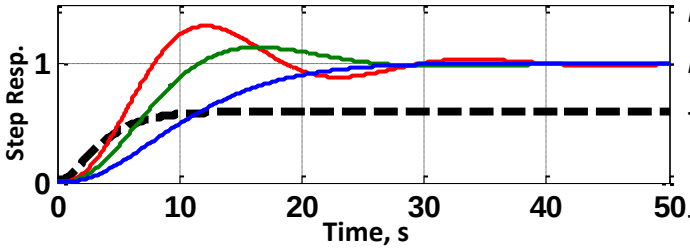




Higher k_p:
- Reduces s.s. error
- Speeds up response
- More oscillatory

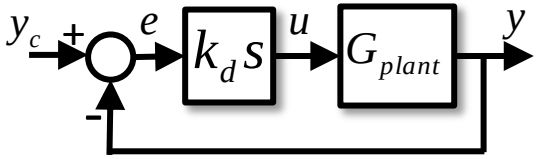
I: *Pure Integral*

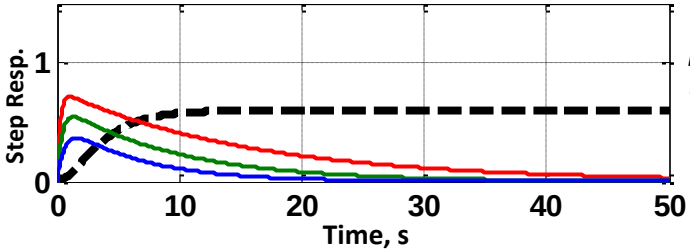




Eliminates s.s. error
Higher k_i:
- Speeds up response (slower than original?)
- More oscillatory

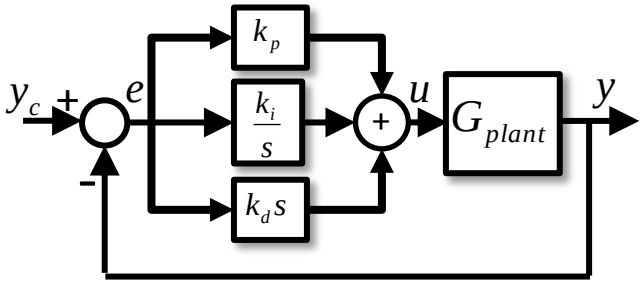
D: *Pure Derivative*

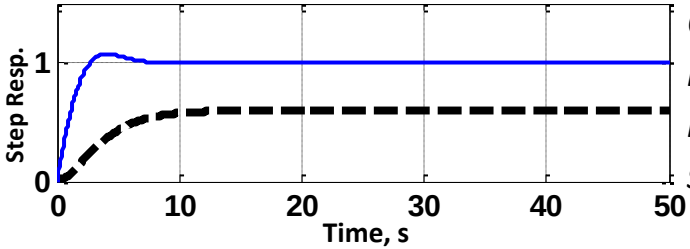




Fast initial response
Tapers to zero

PID:

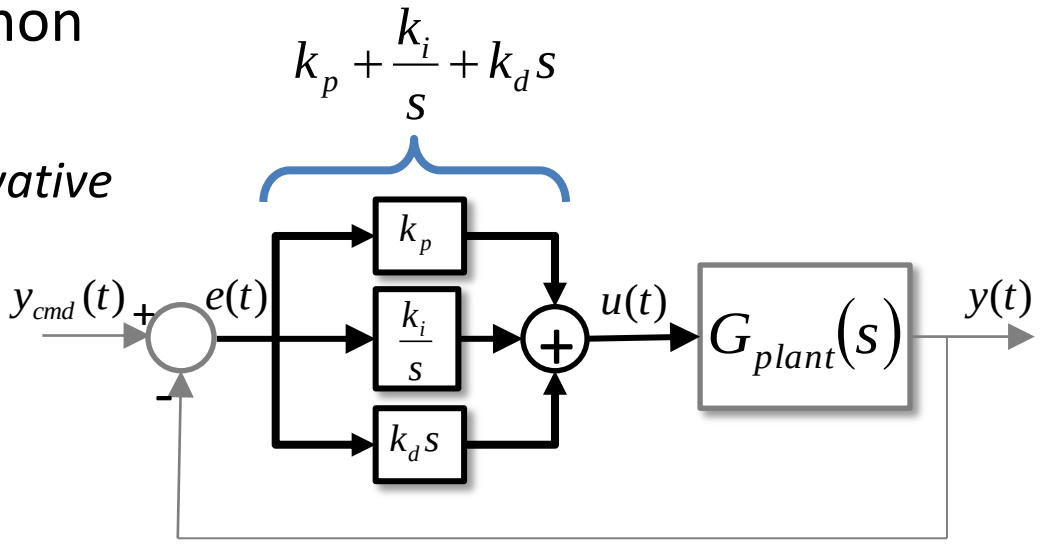




Can achieve desirable response: fast, stable, not oscillatory, & no steady-state error

PID controllers

- PID controllers are *very* common and very useful
  - PID: *Proportional-Integral-Derivative*
- In general:
  - Proportional gain (k_p) largely affects stability
  - Integral gain (k_i) is used to remove steady state error
  - Derivative gain (k_d) is used to speed up response
- There are many PID *tuning* methods



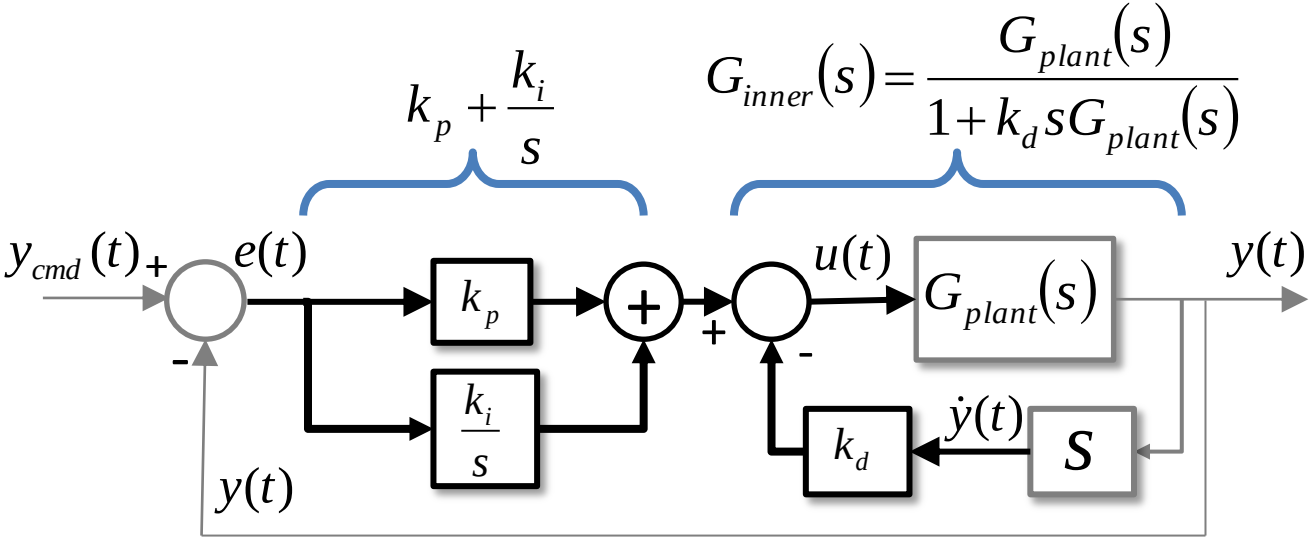
Example differentiator:

```
error_dot = (error - error_prev)/dt;
error_prev = error;
```

- Major Drawback:
  - Derivative term requires differentiating error signal
    - Significantly amplifies any high frequency noise in *e(t)*
      - Can be minimized with a low-pass filter

PI with rate feedback

- In designing our autopilot, we won't use PID because we don't want to differentiate really noisy measurements
- Instead, we'll make heavy use of a PI controller with rate feedback
  - Presumption: Both *y* and *ydot* are available as feedback measurements

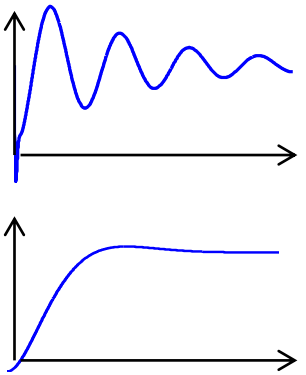


Inner loop (with k_d) provides "rate damping"
- mitigates ringing
- "slows down" plant dynamics

Outer loop (with k_p and k_i) drives desired closed-loop response
- rise time, steady-state error, etc.

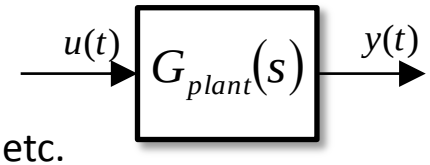
$$\frac{Y(s)}{Y_{cmd}(s)} = \frac{\left(k_p + \frac{1}{s}k_i\right)G_{inner}(s)}{1 + \left(k_p + \frac{1}{s}k_i\right)G_{inner}(s)}$$

A Controls Engineer's Job

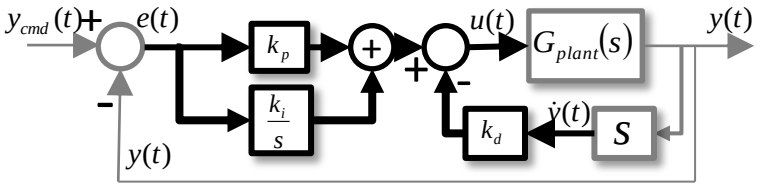


- **We have:** Some input/output system with an undesirable response
- **We want:** Some desired response (e.g. stable, sufficient bandwidth, desired rise time, minimal overshoot, etc.)

- **Steps to designing a controller:**
  1. Model the plant system we are trying to control
     - What are the: states, inputs, outputs, dynamics, input limits, etc.
     - Transfer Functions and/or State Space



  2. Determine a desired controller structure
     - e.g. PID, PI w/ rate feedback, etc.



  3. Determine controller gains/coefficients to meet desired response
     - Many possible techniques
       - Pole placement, root locus, frequency response shaping, etc.

$$k_p, k_i, k_d \; ?$$

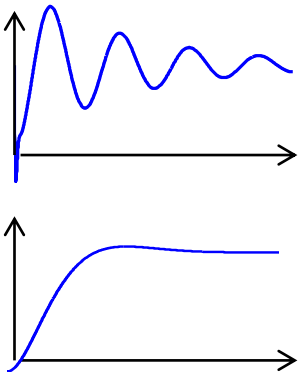
  4. Verify response characteristics and robustness
     - Linear Step Responses, Gain & Phase margins, etc.
  5. Implement controller
     - e.g. software, hardware, etc.

```
u = kp*error + ki*error_int - kd*y_dot;
```

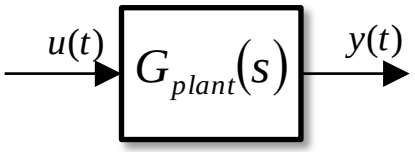
A Controls Engineer's Job

In this course…

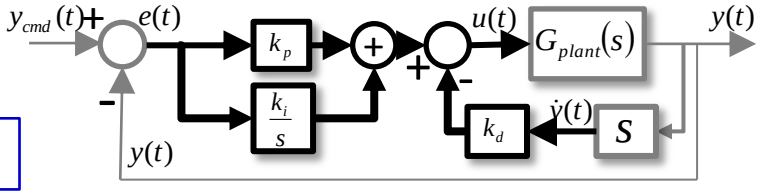
We will develop UAV autopilot algorithms to control:
- pitch, roll, altitude, course, speed



- **Steps to designing a controller:**
  1. Model the plant system we are trying to control
     - We will develop plant response models directly from kinematics & dynamics



  2. Determine a desired controller structure
     - We will use variants of PI with rate feedback



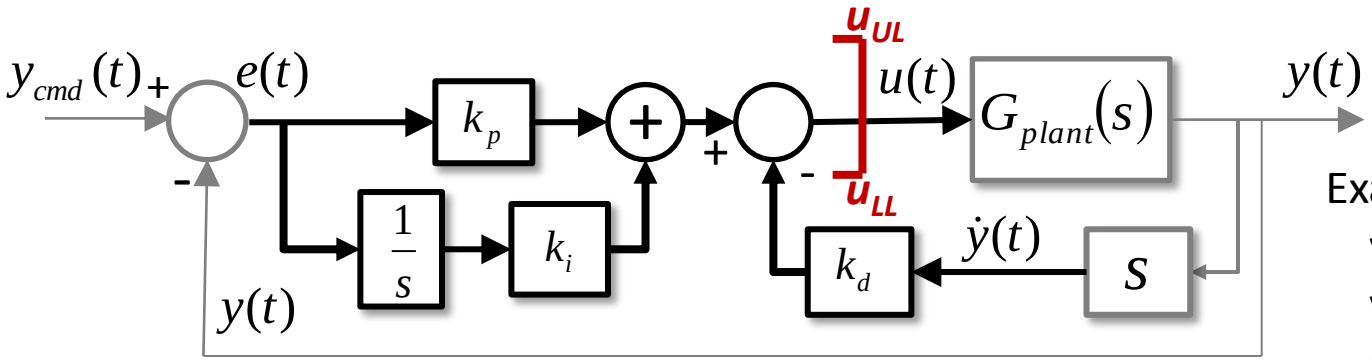
  3. Determine controller gains/coefficients to meet desired response
     - We will select gains generally to match desired 1st and 2nd order responses, accounting for physical system limitations

k_p, k_i, k_d ?

  4. Verify response characteristics and robustness
     - We will verify transient responses (we won't assess with Gain & Phase margins)
  5. Implement controller
     - We will implement in 6DOF Simulation (Matlab/Simulink)

```
u = kp*error + ki*error_int - kd*y_dot;
```

Controller Implementation



Example:
y = pitch
y_dot = pitch rate
u = elevator deflection

- After designing our controllers, we will need to implement them in software

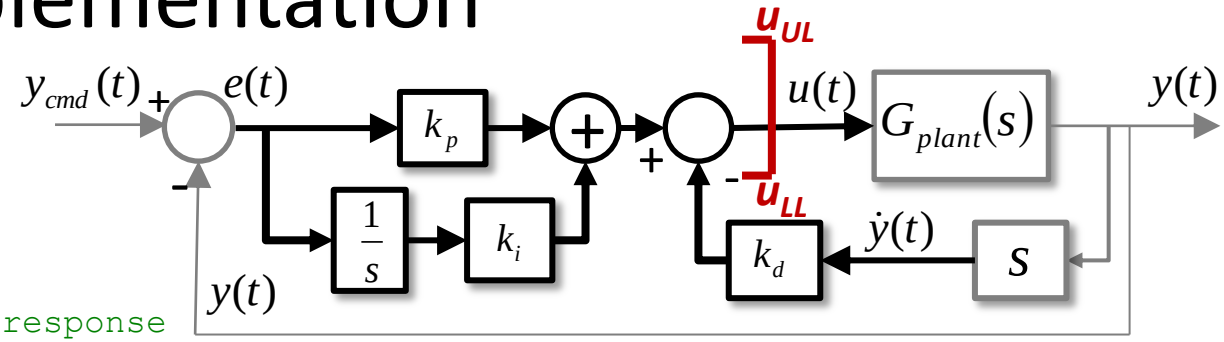
```
% Error between command and response
error = y_c - y;

% Update the integrator (Euler integration)
error_int = error_int + dt*error;

% Perform "PI with rate feedback"
u = kp*error + ki*error_int - kd*y_dot;
```

Note:
`error_int` must be pre-initialized to zero.

But… what if *u(t)* is limited????

Controller Implementation

- Controller implementation accounting for controller output limiting

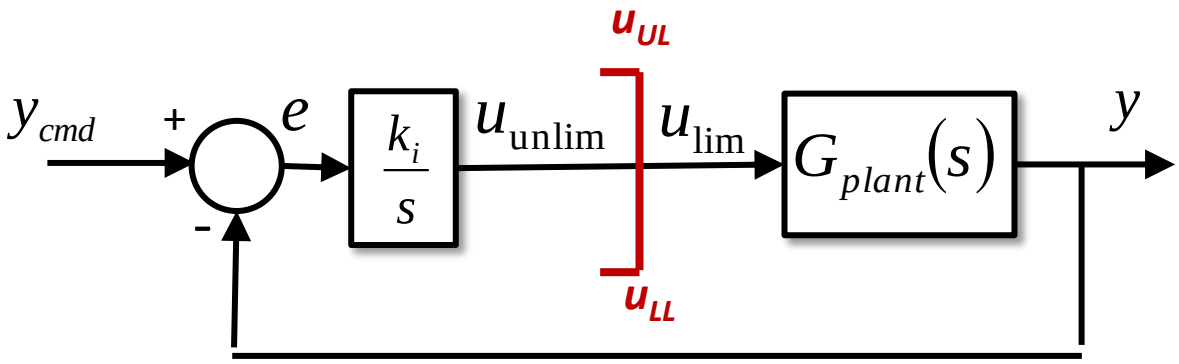


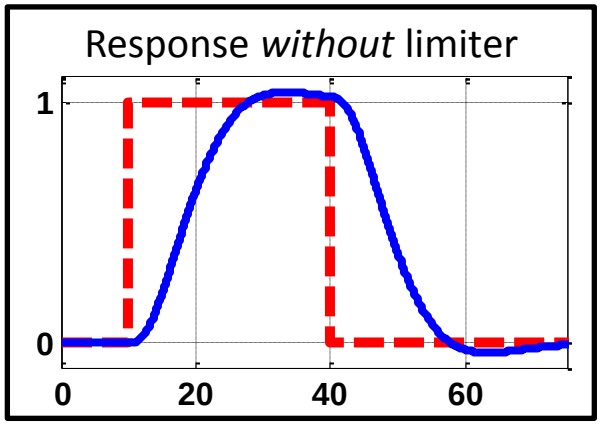

```
% Error between command and response
error = y_c - y;

% Update the integrator (Euler integration)
error_int = error_int + dt*error;

% Perform "PI with rate feedback"
u = kp*error + ki*error_int - kd*y_dot;

% Output saturation & integrator clamping
%   - Limit u to u_upper_limit & u_lower_limit
%   - Clamp if error is driving u past limit
if u > u_upper_limit
    u = u_upper_limit;
    if ki*error>0
        error_int = error_int - dt*error;
    end
elseif u < u_lower_limit
    u = u_lower_limit;
    if ki*error<0
        error_int = error_int - dt*error;
    end
end
```

- Controller output is often limited
  - servo limits
  - avoiding damage to plant
- If we limit the output, we need to avoid integrator wind-up
  - Wind-up: Integrator continuing to accumulate while output is limited **(see next slide)**
- Preferred approach:
  - "Clamp" integrator (hold previous value) if on limit and *e(t)* is driving *u(t)* even further past limit
  - Differs from book advice

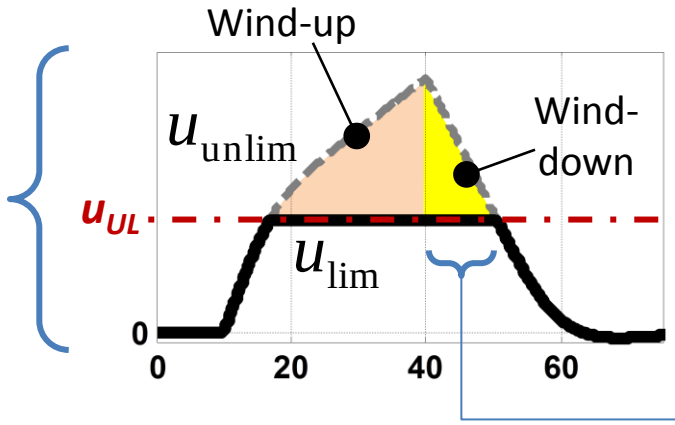
Integrator Wind-up and Clamping

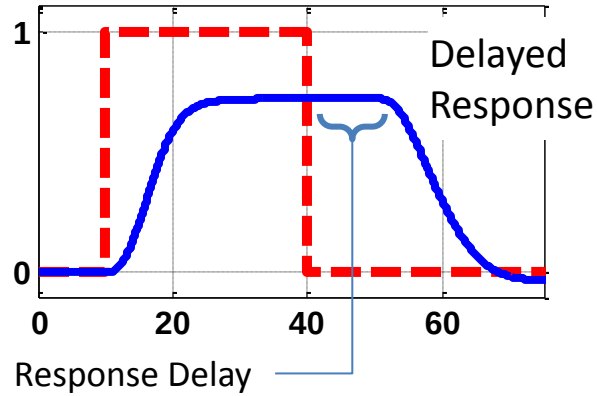
Integrator wind-up can occur with any limited controller containing an integrator, but it is most easily demonstrated with a pure integral controller.



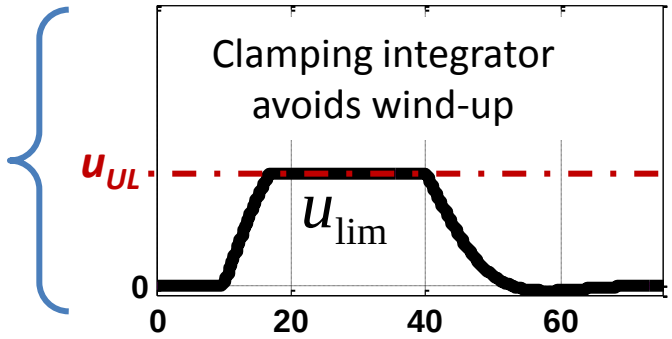


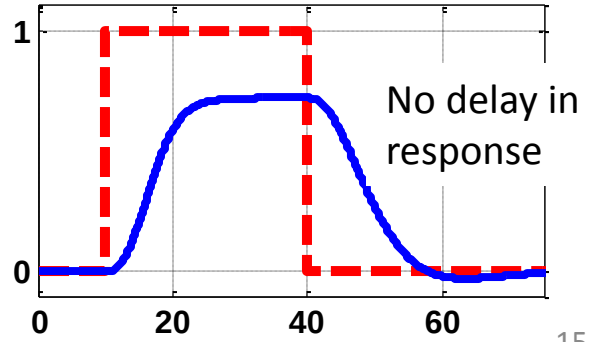
When control signal is limited, the integrator continues to accumulate (wind-up). Causes a delay exiting the limit.





Integrator wind-up can be avoided by holding the previous integrator value (i.e. *clamping*) when control signal is limited *and* the integrator input would drive the control signal *further* past limit.





Vector Geometry 2/2

Vector Operations

Rotating Coordinate Frames

Vector Differentiation

Vectors and their Operations

- Vectors define a direction and a magnitude
- Two vectors u and v are *parallel* if they express the same direction, hence can be related by a non-zero scale factor

$$\text{Parallel}: \quad \underline{u} = k\underline{v}, \quad k \neq 0, \ |\underline{v}| \neq 0$$

- Two vectors u and v are *orthogonal* if they are perpendicular to each other
- A *unit vector* is a vector with a magnitude of 1
- Vectors can be written multiple ways:

$$\underline{u} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} u_1 & u_2 & u_3 \end{bmatrix}^T = u_1\hat{i} + u_2\hat{j} + u_3\hat{k}$$

- Vectors have two multiplication operations:
  - Dot Product & Cross Product

```
MATLAB:
u = [1; 2; 3];
v = [4 5 6]';
a = dot(u,v);
w = cross(u,v);
mag = norm(u);
```

Dot Product

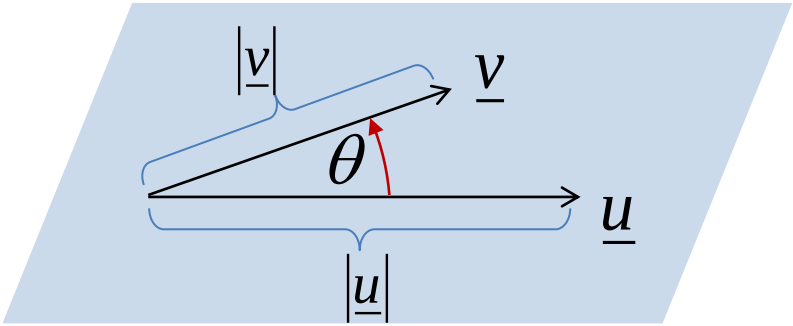
The Dot Product (aka scalar product) is a vector multiplication producing a scalar value

Given: $\underline{u} = \begin{bmatrix} u_1 & u_2 & u_3 \end{bmatrix}^T \qquad \underline{v} = \begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix}^T$

Definition: $\underline{u} \cdot \underline{v} = u_1 v_1 + u_2 v_2 + u_3 v_3$

$$\underline{u} \cdot \underline{v} = |\underline{u}||\underline{v}| \cos\theta$$

$$\underline{u} \cdot \underline{v} = |\underline{u}||\underline{v}| \cos\theta$$



Properties:

$$\underline{u} \cdot \underline{v} = \underline{v} \cdot \underline{u}$$

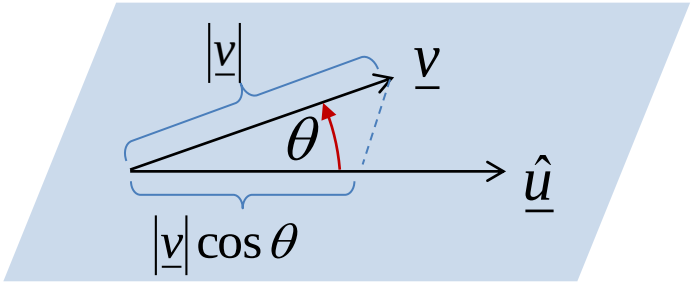
$$\underline{u} \cdot \underline{u} = |\underline{u}|^2$$

$\underline{u}$ is orthogonal to $\underline{v}$

$$\Leftrightarrow \underline{u} \cdot \underline{v} = 0$$

Special Case:

Let $\hat{u}$ be a unit vector: $|\hat{u}| = 1$



$\underline{v} \cdot \hat{u}$ is the *projection* of $\underline{v}$ onto $\hat{u}$

Cross Product

The Cross Product (aka vector product) is a vector multiplication producing a vector perpendicular to the other two

Given: $\underline{u} = \begin{bmatrix} u_1 & u_2 & u_3 \end{bmatrix}^T \qquad \underline{v} = \begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix}^T$

Definition: $$\underline{u} \times \underline{v} = \begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ u_1 & u_2 & u_3 \\ v_1 & v_2 & v_3 \end{vmatrix} = (u_2 v_3 - u_3 v_2)\hat{i} + (u_3 v_1 - u_1 v_3)\hat{j} + (u_1 v_2 - u_2 v_1)\hat{k}$$

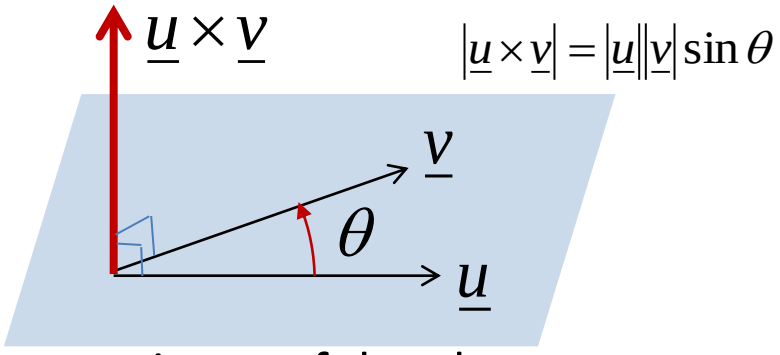
Properties:

$\underline{u} \times \underline{v}$ is orthogonal to both $\underline{u}$ and $\underline{v}$

$\underline{u} \times \underline{v} = -(\underline{v} \times \underline{u})$

$|\underline{u} \times \underline{v}| = |\underline{u}||\underline{v}| \sin\theta$

$\underline{u}$ is parallel to $\underline{v} \Leftrightarrow |\underline{u} \times \underline{v}| = 0$



$\underline{u} \times \underline{v}$ is out of the plane containing $\underline{u}$ and $\underline{v}$

Special Case:

Let $\hat{u}$ and $\hat{v}$ be unit vectors,

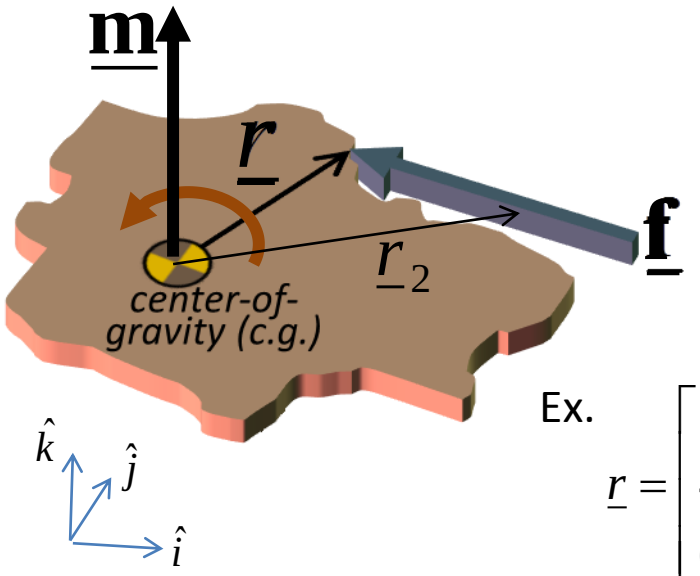
then : $|\hat{u} \times \hat{v}| = \sin\theta$

Right-Hand Rule: Curl fingers from first vector toward second. Thumb points toward cross product.

Cross Product Example: Forces & Moments

- **Force:** An action that causes a change in translational motion
- **Moment (Torque):** An action that causes a change in rotational motion

- A force acting through the center-of-gravity translates without rotating
- A force not acting through the center-of-gravity also induces a rotation
  - A force induces a moment about the center-of-gravity



$\underline{\mathbf{f}}$ = Force vector, N *(Newtons)*

$\underline{r}$ = Vector from c.g. to line of force, m

$\underline{\mathbf{m}}$ = Induced Moment about c.g., N-m

$$\underline{\mathbf{m}} = \underline{r} \times \underline{\mathbf{f}} \quad = \underline{r}_2 \times \underline{\mathbf{f}}$$

Any point along line of force can be used

Ex.

$$\underline{r} = \begin{bmatrix} 3 \\ 4 \\ 0 \end{bmatrix}, \quad \underline{\mathbf{f}} = \begin{bmatrix} -8 \\ 2 \\ 0 \end{bmatrix}, \quad \underline{\mathbf{m}} = \underline{r} \times \underline{\mathbf{f}} = \begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ 3 & 4 & 0 \\ -8 & 2 & 0 \end{vmatrix} = 38\hat{k} = \begin{bmatrix} 0 \\ 0 \\ 38 \end{bmatrix} N \cdot m$$

Rotating Coordinate Frame

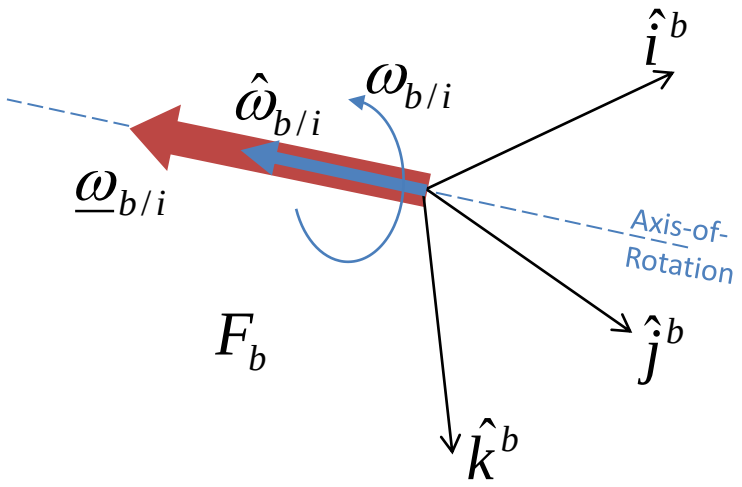
- Consider a frame, F_b , that is rotating relative to an inertial (non-rotating) frame F_i
  - Any instantaneous rotation of a rigid body can be expressed as a rotation rate, $\omega_{b/i}$, about an axis of rotation
  - Further, the axis of rotation can be defined as a unit vector: $\hat{\omega}_{b/i}$
  - Thus, the rotation rate can also be expressed as a vector:

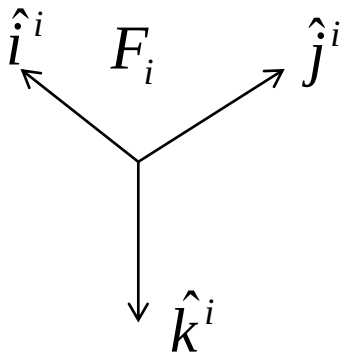
$$\underline{\omega}_{b/i} = \omega_{b/i}\hat{\omega}_{b/i}$$

$\omega_{b/i}$: Scalar rotation rate, rad/s

$\hat{\omega}_{b/i}$: Axis - of - rotation unit vector

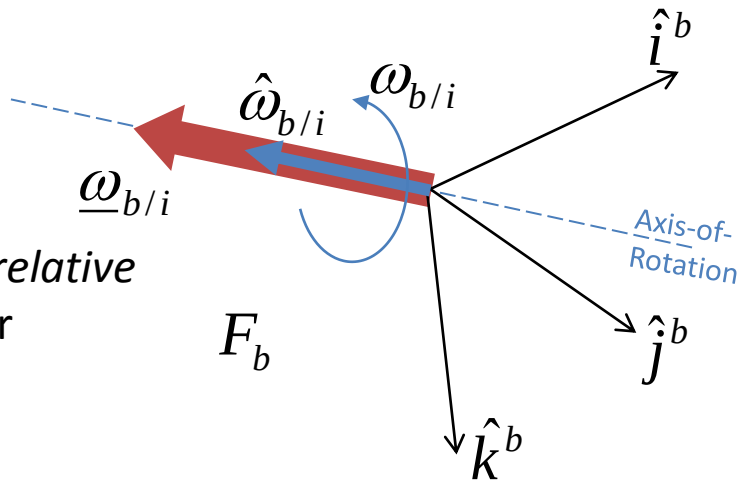
$\underline{\omega}_{b/i}$: Rotation rate vector





Rotating Coordinate Frame

- Consider a frame, F_b , that is rotating relative to an inertial (non-rotating) frame F_i

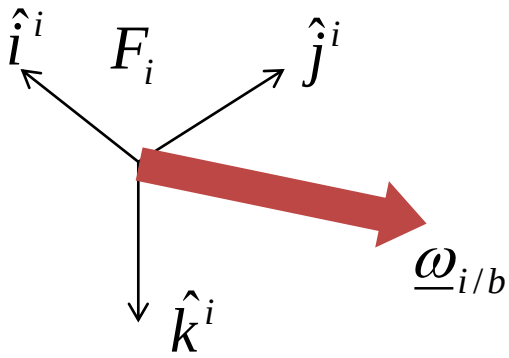


  - Every rotation must be *relative to* something else
    - Note that a frame could be rotating *relative to* an inertial (non-rotating) frame, or *relative to* another rotating frame
  - A rotation vector can be expressed in any coordinate frame

$$\underline{\omega}^b_{b/i}: \ \underline{\omega}_{b/i} \text{ expressed in } F_b \text{ coordinates}$$

$$\underline{\omega}^i_{b/i}: \ \underline{\omega}_{b/i} \text{ expressed in in } F_i \text{ coordinates}$$

$$\underline{\omega}^i_{b/i} = R^i_b \underline{\omega}^b_{b/i}$$



  - The rotation of F_i wrt F_b is simply:

$$\underline{\omega}_{i/b} = -\underline{\omega}_{b/i}$$

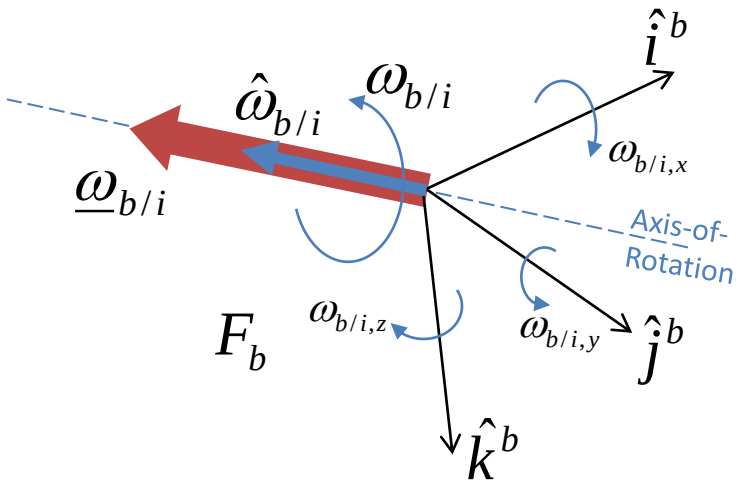
Rotating Coordinate Frame

- Consider a frame, F_b , that is rotating relative to an inertial (non-rotating) frame F_i
  - A rotation rate vector expressed in F_b has components along each axis of F_b
    - Found via dot product

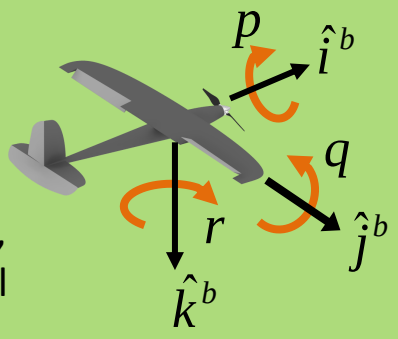
$$\underline{\omega}_{b/i} \cdot \hat{i}^b = \omega^b_{b/i,x} \quad \text{rad/s}$$

$$\underline{\omega}_{b/i} \cdot \hat{j}^b = \omega^b_{b/i,y} \quad \text{rad/s}$$

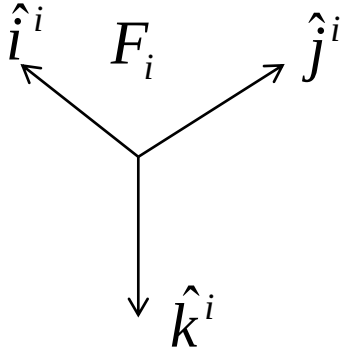
$$\underline{\omega}_{b/i} \cdot \hat{k}^b = \omega^b_{b/i,z} \quad \text{rad/s}$$



$$\underline{\omega}^b_{b/i} = \begin{bmatrix} \omega^b_{b/i,x} \\ \omega^b_{b/i,y} \\ \omega^b_{b/i,z} \end{bmatrix} = \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$



For aircraft, the rotation rates of the body wrt inertial, expressed in body coordinates, are referred to as the "body rates". We will use these often enough that we will more conveniently call them: *[p q r]^T*



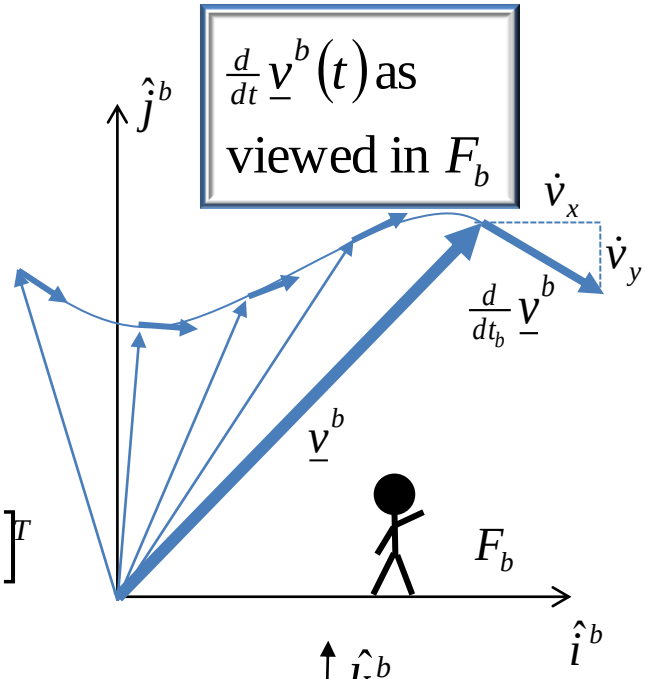
Vector Differentiation

- Consider a coordinate frame, F_b
  - The time derivative (rate-of-change) of a vector $\underline{v}^b$ wrt F_b (i.e. as viewed by an observer attached to F_b) is simply the derivative of each component

$$Let: \underline{v}^b = v_x \hat{i}^b + v_y \hat{j}^b + v_z \hat{k}^b = \begin{bmatrix} v_x & v_y & v_z \end{bmatrix}^T$$

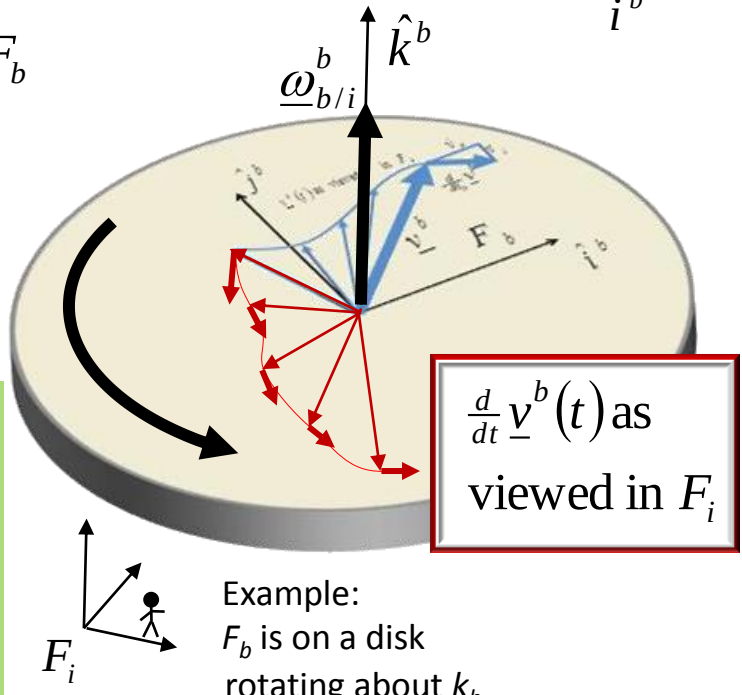
$$Then: \frac{d}{dt_b} \underline{v}^b = \dot{v}_x \hat{i}^b + \dot{v}_y \hat{j}^b + \dot{v}_z \hat{k}^b = \begin{bmatrix} \dot{v}_x & \dot{v}_y & \dot{v}_z \end{bmatrix}^T$$

$$\frac{d}{dt_b}(\ ) : \text{Derivative wrt time, as viewed in } F_b$$



- However, if F_b is rotating relative to F_i
  - The time derivative of $\underline{v}^b$ *as viewed in* F_i can be constructed by combining the derivative as-viewed-in F_b with the rotation rate of F_b wrt F_i:

$$\underbrace{\frac{d}{dt_i}\underline{v}^b}_{\text{Deriv. of } \underline{v}^b \text{ as viewed from non-rotating } F_i} = \underbrace{\frac{d}{dt_b}\underline{v}^b}_{\text{Deriv. of } \underline{v}^b \text{ as viewed from rotating } F_b} + \underbrace{\underline{\omega}^b_{b/i} \times \underline{v}^b}_{\text{Coriolis Effect: Rate of change of } \underline{v}^b \text{ as viewed in } F_i \text{ due to rotation of } F_b \text{ relative to } F_i}$$

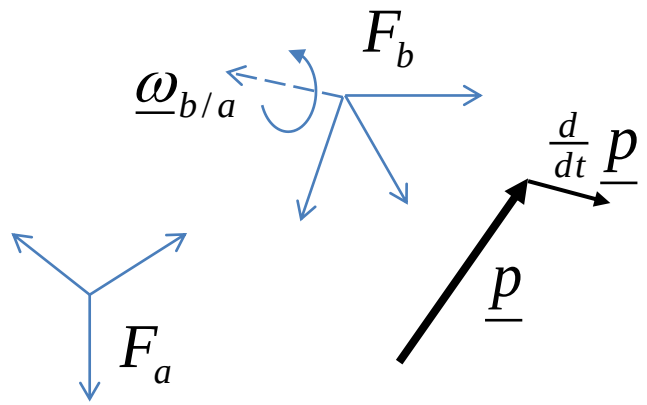


Example:
F_b is on a disk rotating about k_b

Vector Differentiation

- Stated another way:
  - Consider a frame F_a and a frame F_b that is rotating (but not translating) relative to F_a
  - The rate-of-change of any vector $\underline{p}$ as observed in frame F_a can be related to its rate of change as observed in frame F_b by the rotation rate of F_b relative to F_a.

$$\frac{d}{dt_a}\underline{p} = \frac{d}{dt_b}\underline{p} + \underline{\omega}_{b/a} \times \underline{p}$$



Notes:

- d/dt_b : Rate-of-change with respect to frame F_b
- $\underline{\omega}_{a/b} = -\underline{\omega}_{b/a}$

- Further, $\underline{p}$ can be *any* vector expressed in *any* non-translating coordinate frame, e.g. Frame *f*, but $\underline{\omega}_{b/a}$ must be expressed in the same coordinate frame, e.g.

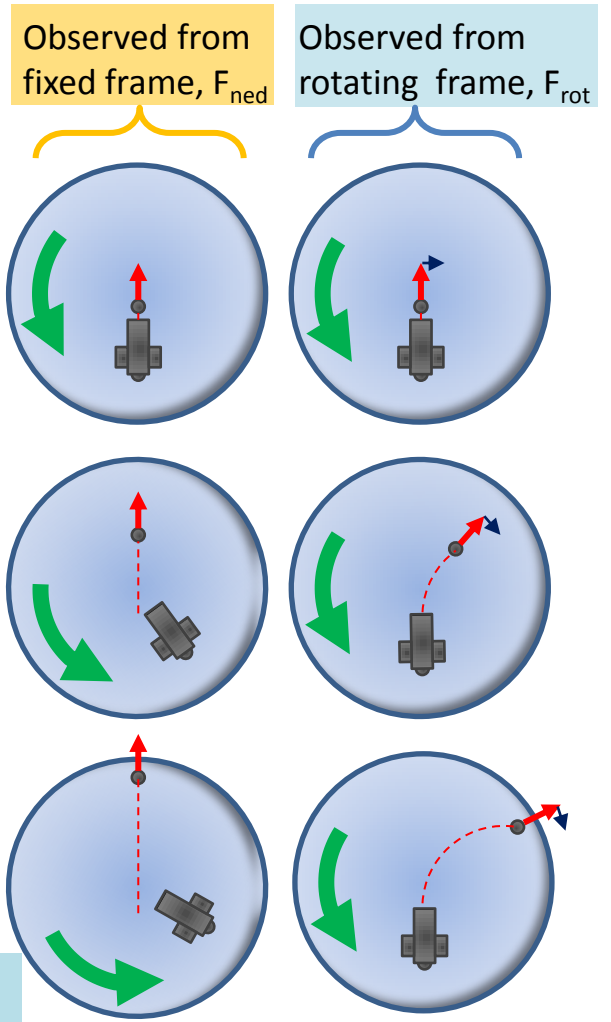
$$\frac{d}{dt_a}\underline{p}^f = \frac{d}{dt_b}\underline{p}^f + \underline{\omega}^f_{b/a} \times \underline{p}^f$$

Proof :

(1)	$\frac{d}{dt_a}\underline{p}^f = \frac{d}{dt_f}\underline{p}^f + \underline{\omega}^f_{f/a} \times \underline{p}^f$	Deriv. of $\underline{p}^f$ wrt F_a
(2)	$\frac{d}{dt_b}\underline{p}^f = \frac{d}{dt_f}\underline{p}^f + \underline{\omega}^f_{f/b} \times \underline{p}^f$	Deriv. of $\underline{p}^f$ wrt F_b
(3)	$\frac{d}{dt_a}\underline{p}^f - \frac{d}{dt_b}\underline{p}^f = \left(\underline{\omega}^f_{f/a} - \underline{\omega}^f_{f/b}\right) \times \underline{p}^f$	(1) minus (2)
(4)	$\frac{d}{dt_a}\underline{p}^f = \frac{d}{dt_b}\underline{p}^f + \left(\underline{\omega}^f_{f/a} + \underline{\omega}^f_{b/f}\right) \times \underline{p}^f$	$\underline{\omega}^f_{b/f} = -\underline{\omega}^f_{f/b}$
(5)	$\frac{d}{dt_a}\underline{p}^f = \frac{d}{dt_b}\underline{p}^f + \underline{\omega}^f_{b/a} \times \underline{p}^f$	$\underline{\omega}^f_{b/a} = \underline{\omega}^f_{f/a} + \underline{\omega}^f_{b/f}$

Vector Differentiation Example

- Consider a cannon fired on a rotating merry-go-round
  - For simplicity, we'll make the following assumptions
    - Cannon fired north with a velocity of v_0
    - No drag, and ignoring gravity, so the NED velocity of the cannonball is $[v_0; 0; 0]$ forever
    - Merry-go-round is rotating at $-\omega$ rad/s about the down direction (i.e. counter-clockwise)



Velocity of the cannonball, in NED coordinates: $\underline{v}_{ball}^{ned} = \begin{bmatrix} v_0 \\ 0 \\ 0 \end{bmatrix}$

Rotation rate of the rotating frame relative to the NED frame, in NED coordinates: $\underline{\omega}_{rot/ned}^{ned} = \begin{bmatrix} 0 \\ 0 \\ -\omega \end{bmatrix}$

$$\underbrace{\frac{d}{dt_{ned}}\underline{v}_{ball}^{ned}}_{\text{Deriv. of } \underline{v}_{ball} \text{ in NED coords as viewed from fixed } F_{ned}} = \underbrace{\frac{d}{dt_{rot}}\underline{v}_{ball}^{ned}}_{\text{Deriv. of } \underline{v}_{ball} \text{ in NED coords as viewed from rotating } F_{rot}} + \underbrace{\underline{\omega}_{rot/ned}^{ned} \times \underline{v}_{ball}^{ned}}_{\text{Coriolis Effect due to rotation of } F_{rot} \text{ relative to } NED}$$

As observed from rotating frame, the cannonball seems to curve!!!

$\psi = -\omega t$: Rotation Angle

Deriv. of $\underline{v}_{ball}$ in rotating frame coordinates (F_{rot}) as viewed from rotating F_{rot}:

$$\frac{d}{dt_{rot}}\underline{v}_{ball}^{rot} = R_{ned}^{rot}\left(\frac{d}{dt_{rot}}\underline{v}_{ball}^{ned}\right) = R_{ned}^{rot}\left(\frac{d}{dt_{ned}}\underline{v}_{ball}^{ned} - \underline{\omega}_{rot/ned}^{ned} \times \underline{v}_{ball}^{ned}\right) = R_{ned}^{rot}\left(\begin{bmatrix} 0 \\ v_0\omega \\ 0 \end{bmatrix}\right) = R_z(\psi)\left(\begin{bmatrix} 0 \\ v_0\omega \\ 0 \end{bmatrix}\right) = \begin{bmatrix} v_0\omega\sin\psi \\ v_0\omega\cos\psi \\ 0 \end{bmatrix}$$

See animation: https://www.youtube.com/watch?v=49JwbrXcPjc

Vector Notation

- Vector and Matrix notation differs throughout literature
- Lectures and book are *mostly* consistent, except
  - Book uses **bold** for vectors & matrices: **r**, **R**
  - Lectures use underlined for vectors ($\underline{r}$) and capitalized for matrices (R)
- For vectors, the subscript is informational, and the superscript indicates the coordinate frame the vector components are expressed in:

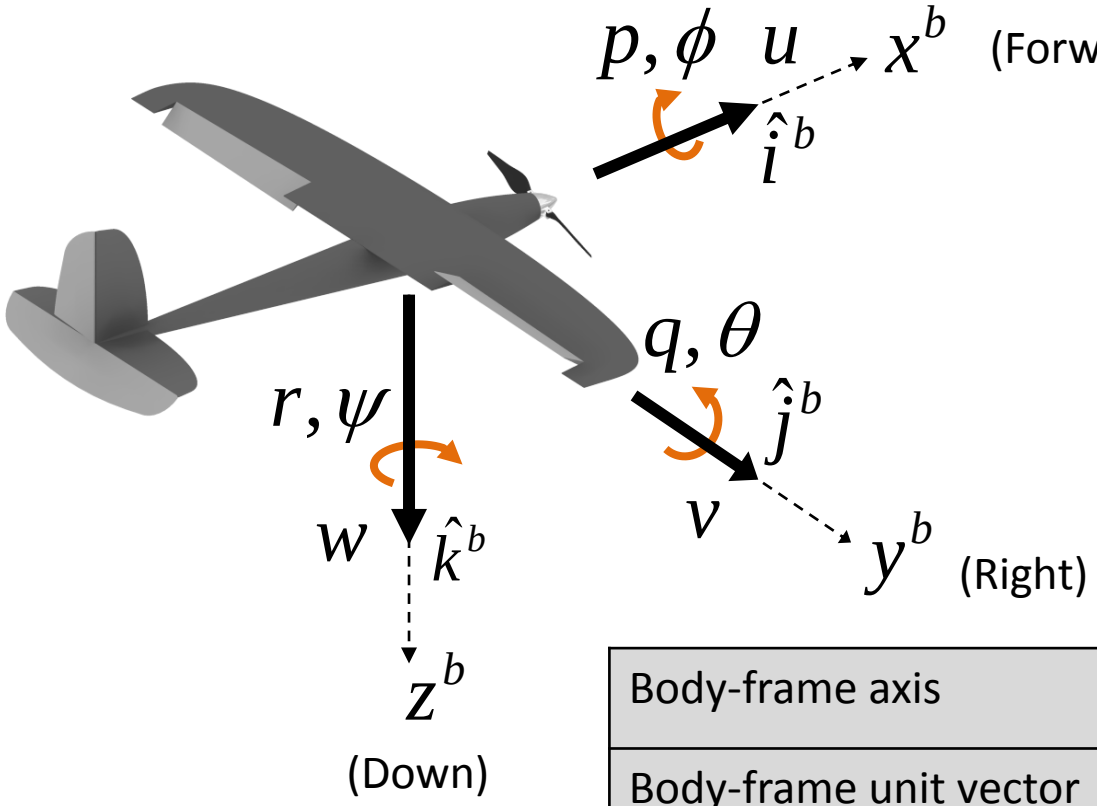
$\underline{v}_g^{NED}$: Velocity relative to ground, expressed in NED coordinates

$\underline{r}_{info}^{frame}$ $\underline{\omega}_{b/a}^{f}$: Rotation rate of Frame *b* relative to Frame *a*, expressed in Frame *f* coordinates

$\underline{f}_{aero}^{b}$: Aerodynamic forces, expressed in body coordinates

- In lectures, unit vectors are expressed with a "hat": $\hat{k}^b$ (Book uses bold: $\mathbf{k}^b$)
- Rotation matrices: R_a^b : Rotation from Frame *a* to Frame *b* (`R_a2b`)
- Interchangeable wording:
  - "relative to" = "with regard to" = "with respect to" = "wrt"
  - "expressed in Frame F" = "expressed in F coordinates" = "in F frame" = "in F coords"

Aircraft Body-Frame Variables

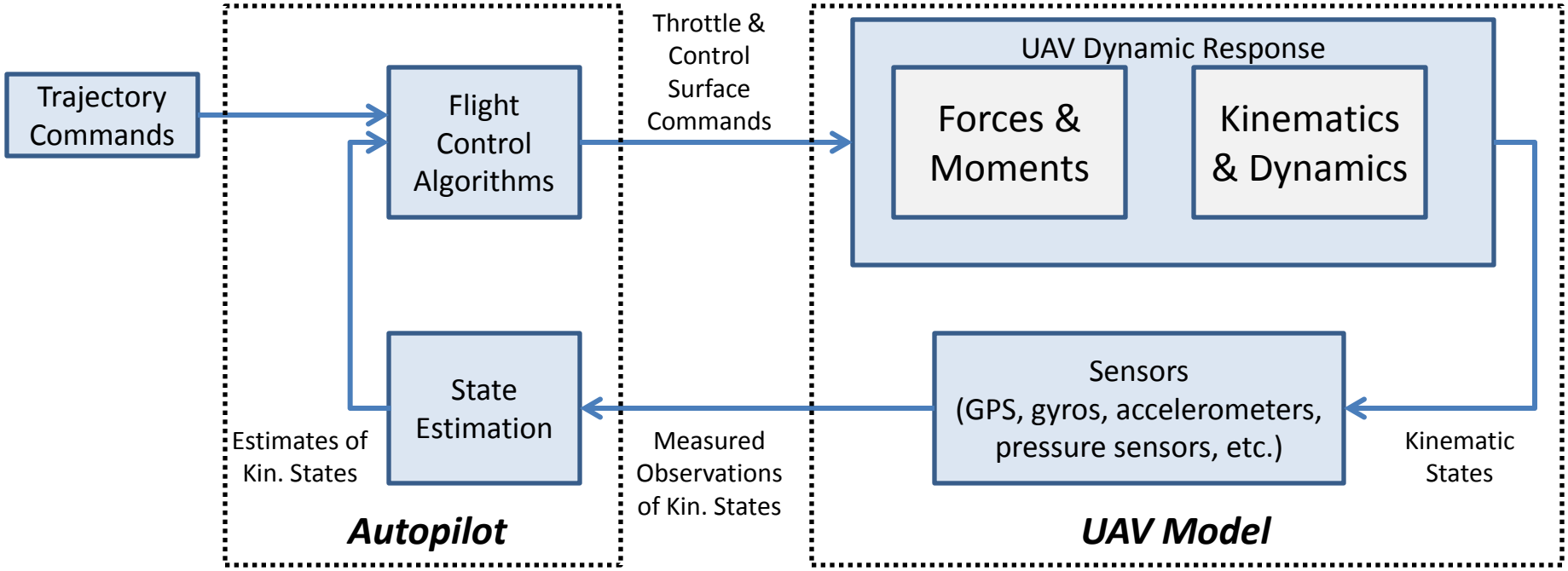


$$\underline{v}_g^b = \begin{bmatrix} u \\ v \\ w \end{bmatrix} \qquad \underline{\omega}_{b/i}^b = \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

Body-frame axis	x^b	y^b	z^b
Body-frame unit vector	$\hat{i}^b$	$\hat{j}^b$	$\hat{k}^b$
Body components of velocity wrt inertial (e.g. ground)	u	v	w
Body rates (components of rotation rate wrt inertial)	p	q	r
Euler angle rotations	ϕ	θ	ψ

Remember right-hand rule!

UAV System



- Next lecture…
  - We'll use the vector geometry we've learned, in conjunction with Newton's 2nd Law, to develop mathematical models for the kinematics and dynamics that govern moving bodies

Lecture 2 Homework, 1/4

1) **In later classes, we will be designing autopilot controller algorithms to control aircraft pitch, roll, altitude, course and speed. All of the controllers we design will use variants of the "PI with rate feedback" structure.**

   a. **The form of the resulting closed-loop system depends on the plant. Consider the simple first order plant: $G_{plant}(s) = b/(s+a)$. What would the closed-loop transfer function be if controlling the plant output with a "PI with rate feedback" controller? Your answer should be a transfer function with coefficients based on a, b, k_p, k_i, and k_d, in the form:**

$$G_{closedLoop}(s) = \frac{b_1 s + b_0}{a_2 s^2 + a_1 s + a_0}$$

   b. **We will find it useful to have a Matlab routine that generates the closed-loop response of a system with this controller in the forward path. Modify the provided skeleton function `PI_rateFeedback_TF.m` to generate the closed-loop transfer function (TF) given a plant model, $G_{plant}(s)$, and the gains k_p, k_i, and k_d:**

```
Gcl = PI_rateFeedback_TF(Gplant, kp, ki, kd)
```

Look in "Later Use Files" Directory

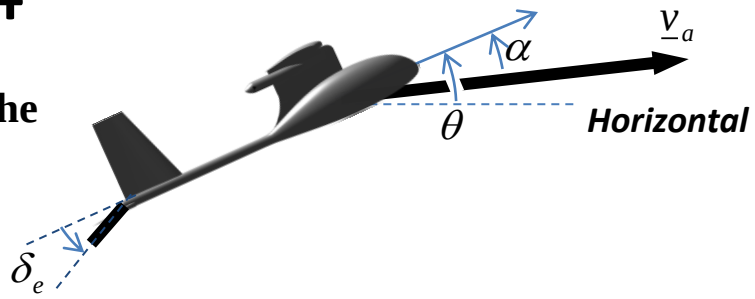
Use the routine to generate the closed loop response using the plant $G_{plant}(s) = 1/(s+2)$, and the controller gains: k_p=3, k_i=4, k_d=5. Does the result compare with your answer in 1(a)?

   c. **The closed-loop step response from 1(b) has a peak time of about 3.5 seconds and a large overshoot, which doesn't meet specifications. "Tune" the gains (via trial and error) to meet the desired specifications: Peak time <= 1.5 seconds, Overshoot: 4%-10%, k_p<=5. Show the resulting gains and the step response, comparing with the response from 1(b).**

Lecture 2 Homework, 2/4

2) Consider the airplane pitch dynamics model from the previous homework:

$$\frac{d}{dt}\begin{bmatrix}\alpha\\ \dot{\theta}\\ \theta\end{bmatrix}=\begin{bmatrix}-3 & 2 & 0\\ -10 & 0 & -15\\ 0 & 1 & 0\end{bmatrix}\begin{bmatrix}\alpha\\ \dot{\theta}\\ \theta\end{bmatrix}+\begin{bmatrix}0\\ -5\\ 0\end{bmatrix}\delta_e$$



a) Similar to last week, consider the response from δ_e to θ. We want to make a controller that automatically adjusts δ_e to control θ. Generate the closed loop transfer function (`Gcl`) if we use a "PI with rate feedback" controller with gains k_p = -12, k_i = -15, and k_d = -3.

b) We want to implement this controller using the same Matlab time-loop script we generated <u>last week</u> in problem 1(d). Use the lecture material to complete the function `PI_rateFeedback_controller.m`. Use the gains specified above and presume the elevator servo is limited to ±20 degrees. In the script you wrote for 1(d) <u>last week</u>, the plant input (u) represents the elevator. Inside the time-loop, replace the plant input (e.g. `u=1;`) with the following code that will implement the controller in 2(a) above.

```
% Use closed loop control to generate elevator deflection (u, degrees)
theta_c   = 1;      % Pitch command (degrees)
theta     = x(3);   % Current pitch response, deg (feedback)
theta_dot = x(2);   % Pitch rate, deg/s (rate feedback)
u = PI_rateFeedback_controller(theta_c,theta,theta_dot,t==0,dt);
```

1 if t==0
0 otherwise

After running the time-loop script with the described controller, plot the resulting pitch (yHistory, presuming C=[0 0 1]) and compare it with the step response from problem 2(a). Also plot the time-history of the elevator angle (uHistory).

Continued on next slide…

Lecture 2 Homework, 3/4

2) Continued…

c) With a pitch command of 1 degree, you should not have hit the 20 degree servo limit. Increase the pitch command to 6 degrees and re-run the time-loop script. You should notice that the elevator servo was limited to 20 degrees during the response. Compare the result with a 6 degree linear step (e.g. `step(6*Gcl)`). Also plot the elevator response.

3) Consider a "plant" with a scalar transfer function *G*. (i.e. *G* is a scalar value.)

a) Determine the closed-loop transfer function, $G_{cl}(s)$, if using a PID controller (with gains k_p, k_i, and k_d) in the forward path.

b) Determine the closed-loop transfer function, $G_{cl}(s)$, if using a "PI with rate feedback" controller (with gains k_p, k_i, and k_d) in the forward path.

c) What is similar and/or different between the two closed-loop transfer functions you generated in (a) and (b)? What would be the difference in response to a step command ($Y(s)=G_{cl}(s)*Y_{cmd}(s)$, with a step command: $Y_{cmd}(s)=1/s$)? (Think about the initial value theorem.)

4) Describe (concisely) the following 6 vectors and provide the scalar component variables.

$$\underline{v}_g^{ned} = \begin{bmatrix} ? \\ ? \\ ? \end{bmatrix},\ \underline{v}_w^{ned} = \begin{bmatrix} ? \\ ? \\ ? \end{bmatrix},\ \underline{v}_g^{b} = \begin{bmatrix} ? \\ ? \\ ? \end{bmatrix},\ \underline{v}_w^{b} = \begin{bmatrix} ? \\ ? \\ ? \end{bmatrix},\ \underline{v}_a^{b} = \begin{bmatrix} ? \\ ? \\ ? \end{bmatrix},\ \underline{\omega}_{b/i}^{b} = \begin{bmatrix} ? \\ ? \\ ? \end{bmatrix}$$

Looking for phrases such as: _____ relative to _____ expressed in _____ coordinates

Lecture 2 Homework, 4/4

5) Newton's 2nd Law states that force equals mass times acceleration (in an inertial frame).

More specifically:

$$\underline{f}^{\,i} = (mass)\cdot\left(\frac{d}{dt_i}\underline{v}_i^{\,i}\right)$$

- $\underline{f}^{\,i}$: Sum of all external forces, expressed in an inertial frame
- *mass*: mass of the vehicle (assumed constant)
- $\underline{v}^{\,i}_{\,i}$: Velocity relative to an inertial frame, expressed in inertial frame
- d/dt_i : Time derivative taken wrt an inertial frame

We will find it convenient to express these relationships in body frame:

$$\underline{f}^{\,b} = (mass)\cdot\left(\frac{d}{dt_i}\underline{v}_i^{\,b}\right)$$

- $\underline{v}_i^{\,b}$: Velocity relative to inertial, expressed in body frame
- $\underline{f}^{\,b}$: Sum of all external forces, expressed in a body frame

a) Using vector differentiation, express $\underline{f}_b$ as a function of the *body-frame-viewed* rate-of-change of $\underline{v}_i^{\,b}$:

$$\underline{f}^{\,b} = (mass)\left(\quad ?\quad\right)$$

- $\underline{v}_i^{\,b}$: Velocity relative to inertial, expressed in body frame
- d/dt_b : Time derivative taken wrt a body frame
- $\underline{\omega}^{\,b}_{\,b/i}$: Rate of change of body frame wrt inertial (*b/i*) expressed in body frame (superscript b)

b) Using the following relationships, express the above vector equation as three separate scalar equations.

$$\underline{f}^{\,b} = f_x\hat{i}^{\,b} + f_y\hat{j}^{\,b} + f_z\hat{k}^{\,b} \qquad \underline{v}_i^{\,b} = u\,\hat{i}^{\,b} + v\,\hat{j}^{\,b} + w\,\hat{k}^{\,b}$$

$$\underline{\omega}^{\,b}_{b/i} = p\,\hat{i}^{\,b} + q\,\hat{j}^{\,b} + r\,\hat{k}^{\,b} \qquad \frac{d}{dt_b}\underline{v}_i^{\,b} = \dot{u}\,\hat{i}^{\,b} + \dot{v}\,\hat{j}^{\,b} + \dot{w}\,\hat{k}^{\,b}$$

Check:

$$f_x = (mass)(\dot{u} + qw - ?)$$

$$f_y = (mass)(?)$$

$$f_z = (mass)(?)$$