

UAV Systems & Control

Lecture 1

Introduction

- UAVs
- Course Topics and Grading
- UAV 6DOF Design Project
- Quadcopter

Vector Geometry

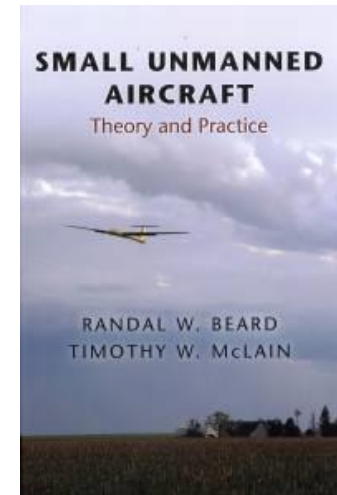
System Modeling

Instructors:

Jeff Barton: jeffrey.barton@jhuapl.edu

Prefer: jbarton629@gmail.com

Jon Castelli: jonathan.castelli@jhuapl.edu



Class generally uses and follows:
Beard & McLain, "Small Unmanned Aircraft",
Princeton University Press, 2012

Numerous images and text courtesy of
book website: <http://uavbook.byu.edu>

What will this class cover?

General Objective of Course

- Gain an understanding of UAV modeling, flight control and state estimation

Course Outline

- System Modeling & Controls Refresher
- Vector Geometry
- Kinematics & Dynamics
- Forces and Moments
 - Gravity, Aerodynamics, Propeller
- Fixed Wing Equations of Motion
- Fixed Wing Simplified Response Models
- Fixed Wing Lateral & Longitudinal Control (akin to PID)
- Sensor Modeling
- State Estimation (via Extended Kalman Filter)
- Quadcopter Modeling & Control
- Intro to Advanced Topics

Prerequisites:

- Some Matrix Math
- Control Systems
 - Laplace
 - Block Diagrams
 - Feedback
- MATLAB

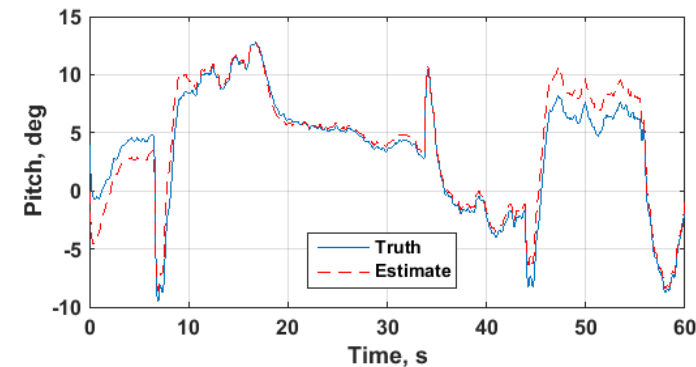
Grading

- **In-class weekly quizzes: 10%**
 - Not intended to be “hard”.
 - Will help us understand what students are learning.
- **Homework, including development of fixed-wing 6DOF: 70%**
- **Final design project: 20%**
 - Convert the fixed-wing 6DOF to a quadcopter 6DOF
- **Homework is expected to reflect individual work and should be turned in *on-time*.**
 - Homework builds off of previous homework, so it is essential that you keep up and always keep a copy of the homework you turn in
 - Late homework will be docked 10% per day.
 - If you are having difficulty, email us. We will do our best to respond within 24 hours. (So don't wait until the day before class)

Homework Expectations:

- Legible, and typed if possible
- Organized
- If we say “explain”, we are looking for a thoughtful proof of understanding
- Figures should always be labeled!

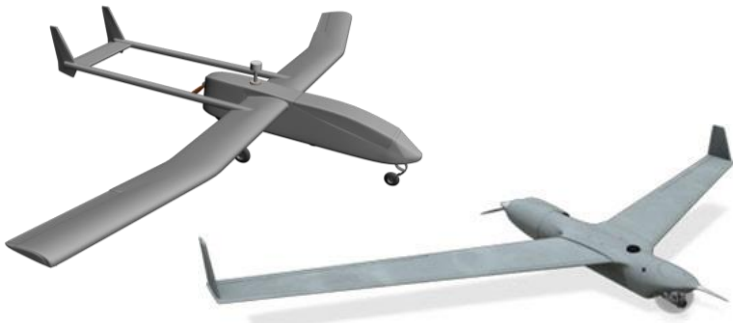
Example:



Note: xlabel, ylabel, legend, grid

What is a UAV? What types are there?

- A UAV is an Unmanned Aerial Vehicle
 - Reusable aircraft
 - Can be
 - Completely autonomous
 - Remotely Piloted (Remote pilot “steers”)
 - Uses on-board sensors and algorithms to:
 - Estimate its position & orientation (yaw, pitch roll)
 - Automatically control and/or stabilize its flight
- An Unmanned Aerial System (UAS) consists of a UAV along with all of its supporting communication and Ground Control Station (GCS) systems.



http://en.wikipedia.org/wiki/Unmanned_aerial_vehicle



Potential Applications for Small UAVs

Civil and Commercial:

- Filming – Sports, concerts, movies, commercials
- Monitoring environment – meteorology, pollution, mapping, mineral exploration
- Monitoring disaster areas – forest fires, avalanches, nuclear contamination
- Communications relays – news broadcasts, disaster relief, sports events
- Law enforcement – road traffic, border patrol, drug control
- Precision agriculture – crop monitoring

Military:

- Special Operations: Situational Awareness
- Intelligence, surveillance, and reconnaissance
- Communication node
- Battle Damage Assessment

Homeland Security:

- Border patrol
- Surveillance
- Rural/Urban Search and Rescue

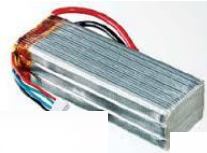


Components of a UAS (Unmanned Aerial System)

- Airframe
 - Main Fuselage
 - Wing
 - Tail
- Propulsion - Electric Motor or Gas Engine
 - Electronic speed controller (Electric motor)
 - Servo Actuator (Gas Engine)
 - Propeller
 - Power source (Battery/Fuel)



- Control Surfaces
 - Ailerons, Elevator, Rudder
 - V-Tail, Elevons
 - Servo actuators

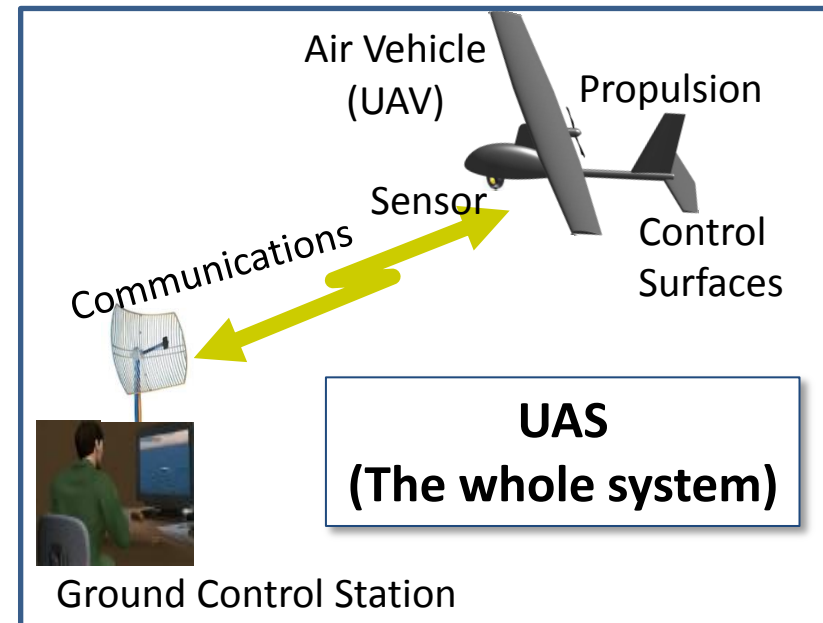
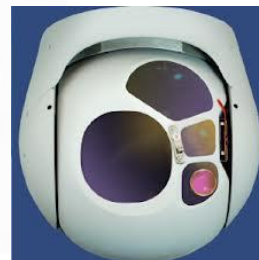


- Control System
 - RC Transmitter
 - Autopilot

- Communication System
 - Transmitter
 - Antenna
 - RC receiver (optional)

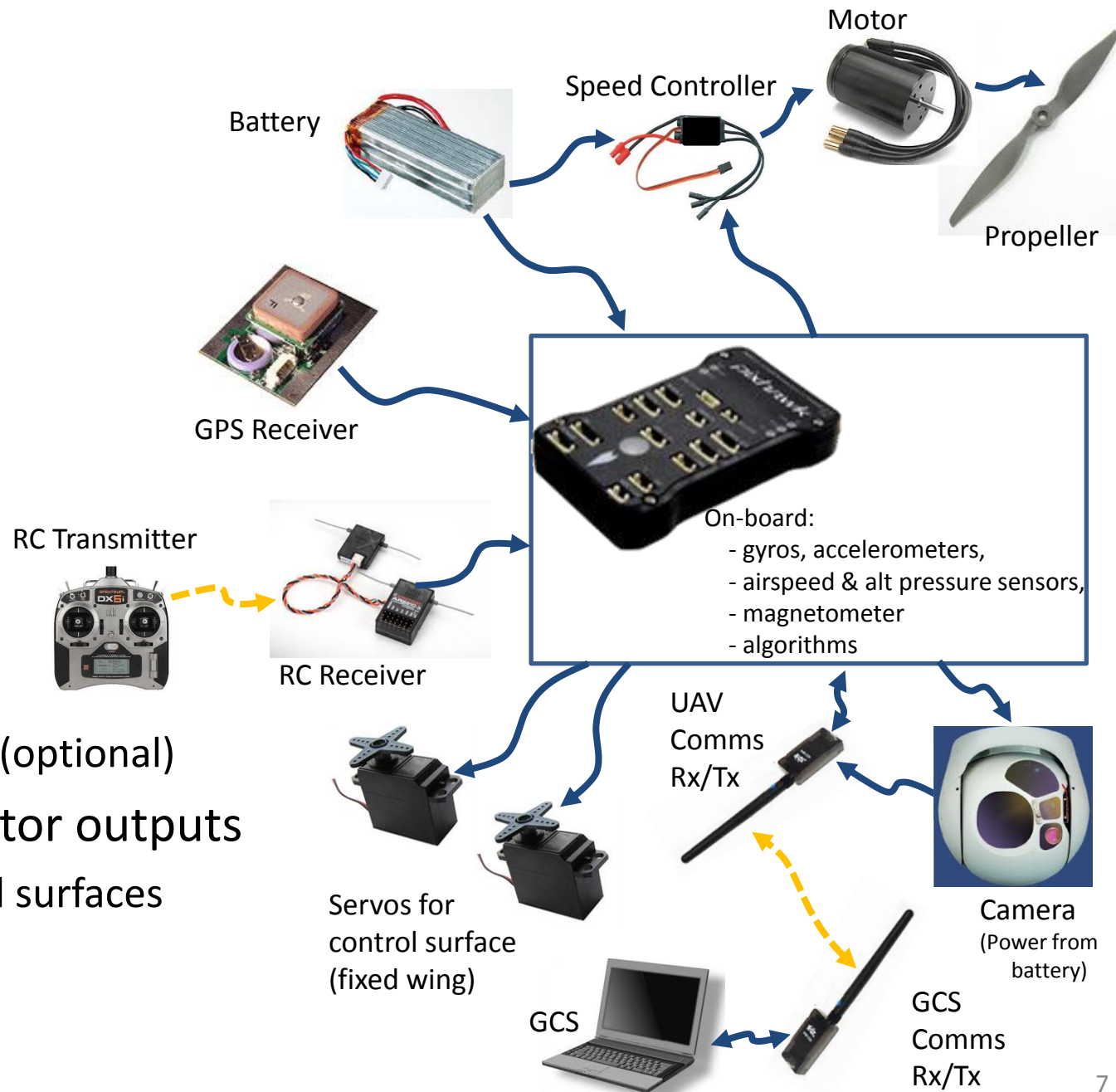


- Payload/Sensor
 - Camera
 - Radar

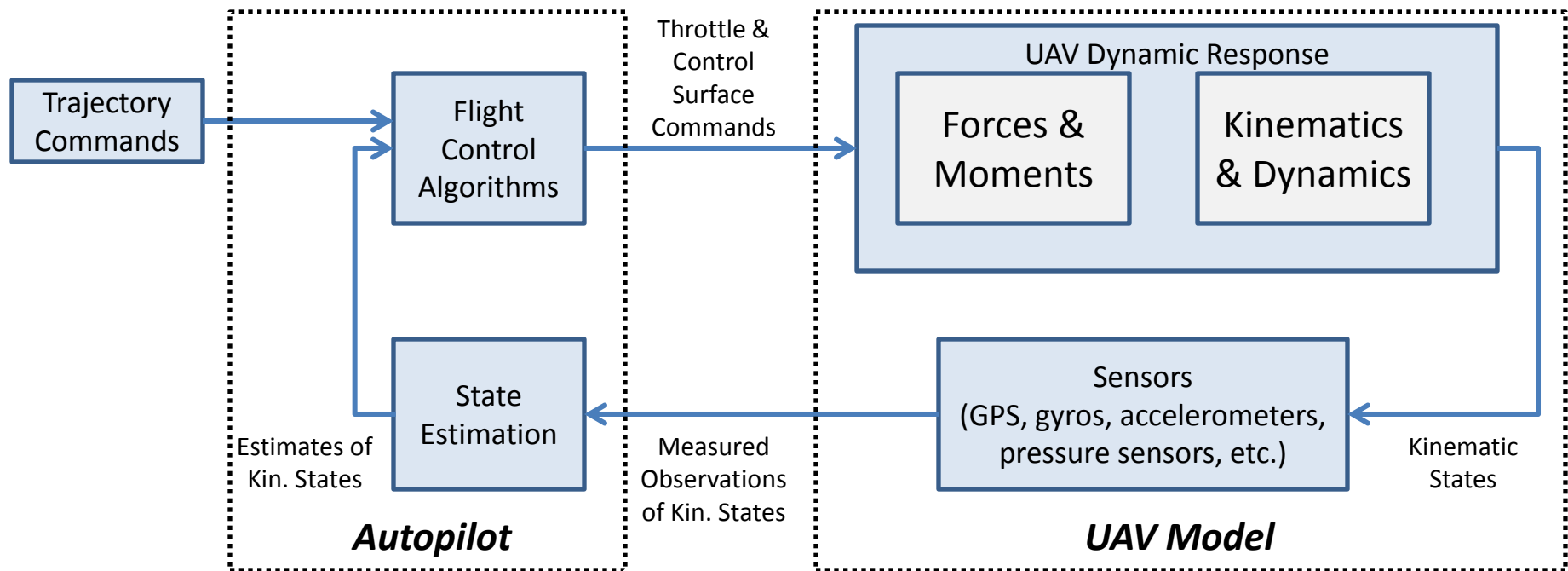


Autopilot

- CPU
 - State Estimator
 - Flight Controller
- Sensors
 - Gyros
 - Accelerometer
 - GPS
 - Magnetometer
 - Pressure
 - Other Altimeter (optional)
- Actuator and Motor outputs
 - Including control surfaces and the motor

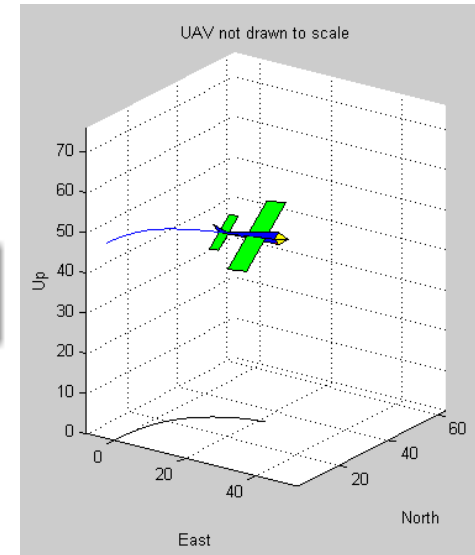
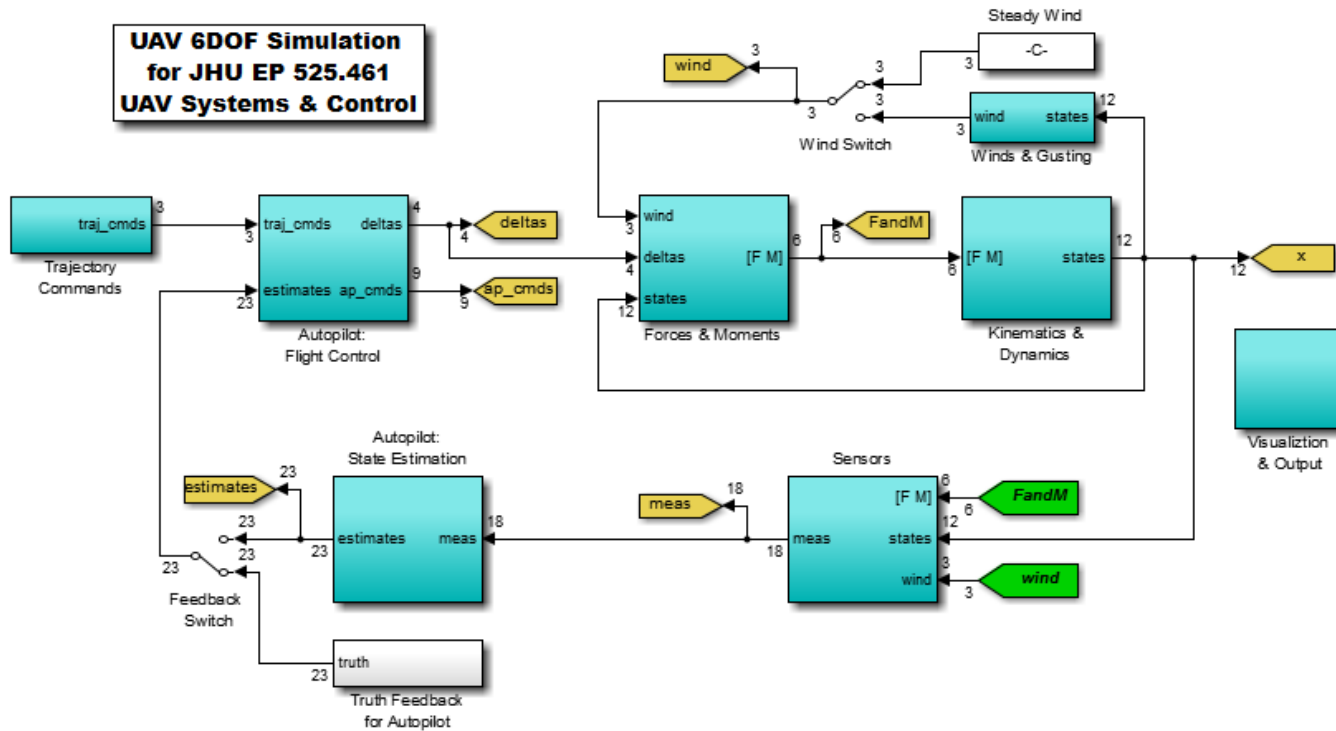


UAV Flight Control System Modeling



- UAV flight control is an interconnection of multiple systems
 - The physical UAV consists of its dynamical response and sensors
 - Autopilot algorithms:
 - Estimate the UAV state
 - Generate flight control commands (i.e. to control surfaces and motors)

UAV 6DOF Simulation



- Students will develop a full 6DOF simulation of a fixed-wing UAV using Simulink and Matlab
 - Simulink architecture is provided
 - Coding is via Matlab functions

IRIS Quadcopter

- We will reinforce applicable topics via hands-on experiences with a quadcopter UAV



- Pixhawk open-source autopilot
- 915 MHz telemetry
- 15-20 min endurance
- 400 g payload
- GoPro compatible mount
- Compatible with Tarot 2D gimbal
- 850 kv motors with 10x4.7 propellers
- Custom 4-in-1 20 amp ESC

Vector Geometry 1/2

Vectors

Matrices

Coordinate Frames

Euler Angles

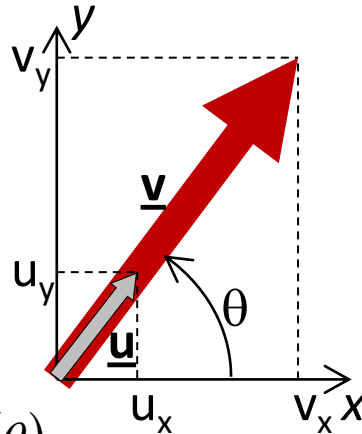
Vectors

Vector are defined by magnitude and direction

$$\underline{v} = \begin{bmatrix} v_x \\ v_y \end{bmatrix}$$

$$v = |\underline{v}| = \sqrt{v_x^2 + v_y^2}$$

$$v_x = v \cos(\theta), v_y = v \sin(\theta)$$

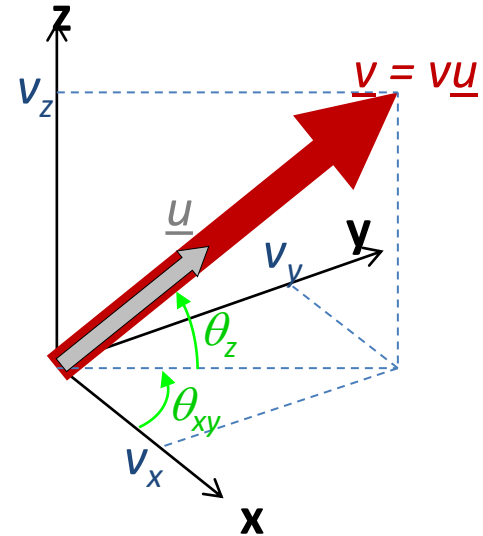


\underline{u} : unit vector ($|\underline{u}| = 1$)

$$\underline{u} = \begin{bmatrix} u_x \\ u_y \end{bmatrix} = \begin{bmatrix} |\underline{u}| \cos(\theta) \\ |\underline{u}| \sin(\theta) \end{bmatrix} = \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix}$$

$$|\underline{u}| = \sqrt{u_x^2 + u_y^2} = \sqrt{\cos^2(\theta) + \sin^2(\theta)} = 1$$

$$\underline{v} = \begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} v \cos(\theta) \\ v \sin(\theta) \end{bmatrix} = v \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix} = v \underline{u}$$



$$\underline{v} = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = \begin{bmatrix} v_x & v_y & v_z \end{bmatrix}^T \quad (T : \text{Transpose})$$

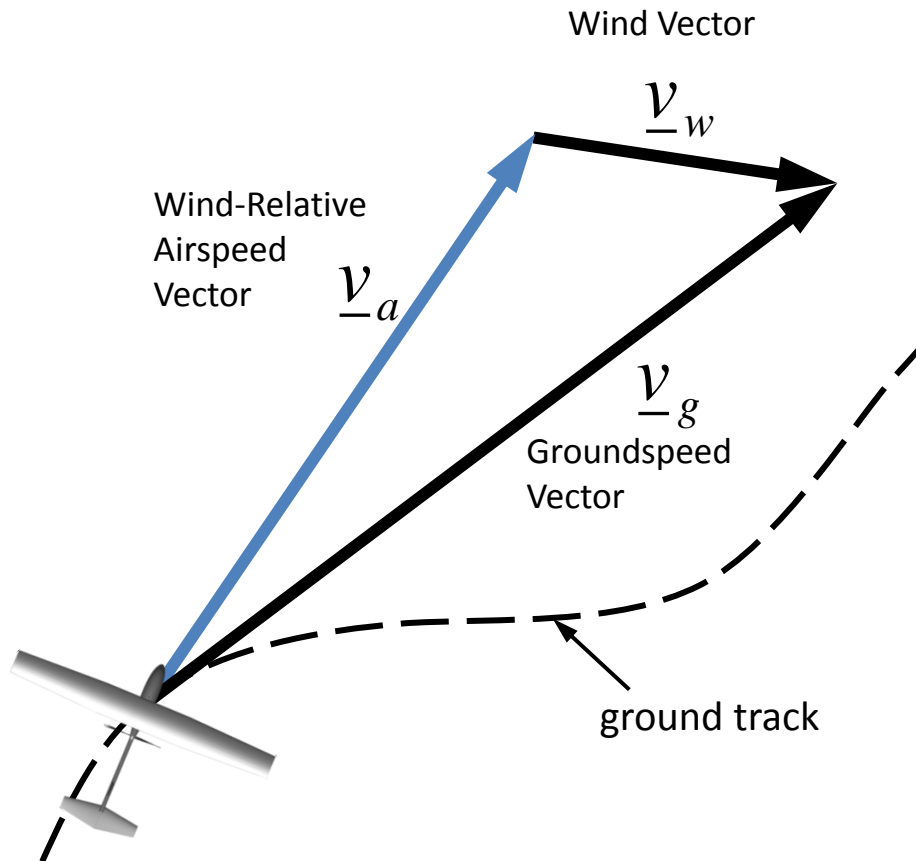
$$v = |\underline{v}| = \sqrt{v_x^2 + v_y^2 + v_z^2}$$

$$\underline{v} = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = v \begin{bmatrix} \cos(\theta_{xy}) \cos(\theta_z) \\ \sin(\theta_{xy}) \cos(\theta_z) \\ \sin(\theta_z) \end{bmatrix} = v \underline{u}$$

v : Magnitude

\underline{u} : Direction (represented as a unit vector)

Vector Example: The Wind Triangle



\underline{v}_g : Velocity vector, relative to ground, m/s

\underline{v}_w : Wind Vector, relative to ground, m/s

\underline{v}_a : Vehicle airspeed vector, aka “Wind-Relative velocity vector”, aka “Velocity relative to moving air”, m/s

$$\underline{v}_a = \underline{v}_g - \underline{v}_w$$

V_a : Airspeed (speed relative to air), m/s

$$V_a = |\underline{v}_a|$$

Don't confuse “airspeed” with “windspeed”

We will find that aerodynamic forces and moments are a function of \underline{v}_a , the vehicle's velocity *relative to* the moving air mass (wind).


Matrices

- Matrices are 2D arrays of numbers
- Vectors are just a special case of matrices

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \quad \underline{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix} \quad \underline{u} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

- Matrix sizes are denoted by the number of rows and columns
 - B is a 3x2 matrix with 3 rows and 2 columns
 - A is a square 3x3 matrix with 3 rows and 3 columns
- Matrix of similar sizes can be added and subtracted
- Matrices can be multiplied if the # of *columns* of the first matrix match the # of *rows* of the second matrix

$$C = AB = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \\ c_{31} & c_{32} \end{bmatrix}$$



 3x3 3x2
 Must match

$$c_{ij} = \sum_{k=1}^{k=\#ofcolumnsOfA} a_{ik} b_{kj}$$

$$c_{32} = \sum_{k=1}^{k=3} a_{3k} b_{k2} = a_{31} b_{12} + a_{32} b_{22} + a_{33} b_{32}$$

c_{32} is the dot product of the 3rd row of A with the 2nd column of B

Matrices

- In general, FG is not equal to GF (where F and G are matrices)
- Transpose means switching rows with columns

$$B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix} \quad B^T = \begin{bmatrix} b_{11} & b_{21} & b_{31} \\ b_{12} & b_{22} & b_{32} \end{bmatrix}$$

MATLAB:

```
A = [1 2; 3 4];
A_transpose = A';
A_det = det(A);
A_inv = inv(A);
I_3x3 = eye(3);
```

- “Rank” of a matrix is the number of independent rows in a matrix
 - Independent rows: rows that are not linear combinations of other rows
 - “Full Rank” means its rank equals its number of rows
- A square matrix has a determinant (a scalar measure of size... *ish*)

$$|A_{2 \times 2}| = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = (ad - bc) \quad |A_{3 \times 3}| = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11} \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} - a_{12} \begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} + a_{13} \begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix}$$

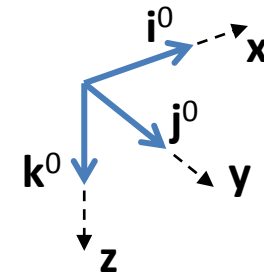
Diagram illustrating the expansion of the 3x3 determinant along the first row. The first row elements are circled in yellow. The signs for the cofactors are +, -, +. The corresponding 2x2 minors are highlighted in yellow.

- A square matrix with full rank can be inverted:

$$A^{-1}A = AA^{-1} = I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Coordinate Frames

- In guidance and control of aircraft, reference frames are used *a lot*
- We define a frame (e.g. Frame 0) as three orthogonal unit vectors
 - e.g. $\mathbf{i}^0, \mathbf{j}^0, \mathbf{k}^0$ (along x, y & z directions)
- Frames used to describe relative position and orientation of objects
 - Aircraft relative to direction of wind
 - Camera relative to aircraft
 - Aircraft relative to inertial frame
- Some things most easily calculated or described in certain reference frames
 - Newton's law (in an *inertial*, or non-moving, frame)
 - Aircraft attitude (relative to *North-East-Down* frame)
 - Aerodynamic forces/torques (in *body-fixed* frame)
 - Accelerometers, rate gyros (in *body-fixed* frame)
 - GPS (in *Earth-fixed* frame)
 - Mission requirements (e.g. in *North-East-Down* frame)



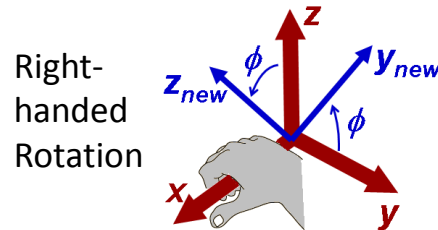
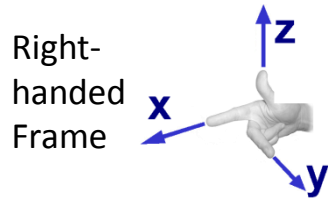
Must know how to transform between different reference frames

Coordinate Frames

Coordinate frame defined by **orientation** and **reference position**

Some coordinate frames rotate and/or translate wrt others

Most coordinate frames follow right-hand-rule



Common Coordinate Frames:

NED: x=North, y=East, z=Down

Referenced to some (lat,lon)

ENU: x=East, y=North, z=Up

Referenced to some (lat,lon)

Body, Forward-Right-Down:

x="Forward" (axially out nose)

y="Right" (out right wing)

z="Down" (out belly of aircraft)

Fixed to body (position & orientation)

ECEF: Earth-Centered-Earth-Fixed

Origin at earth center, rotates with Earth

x: toward (lat,lon)=(0,0)

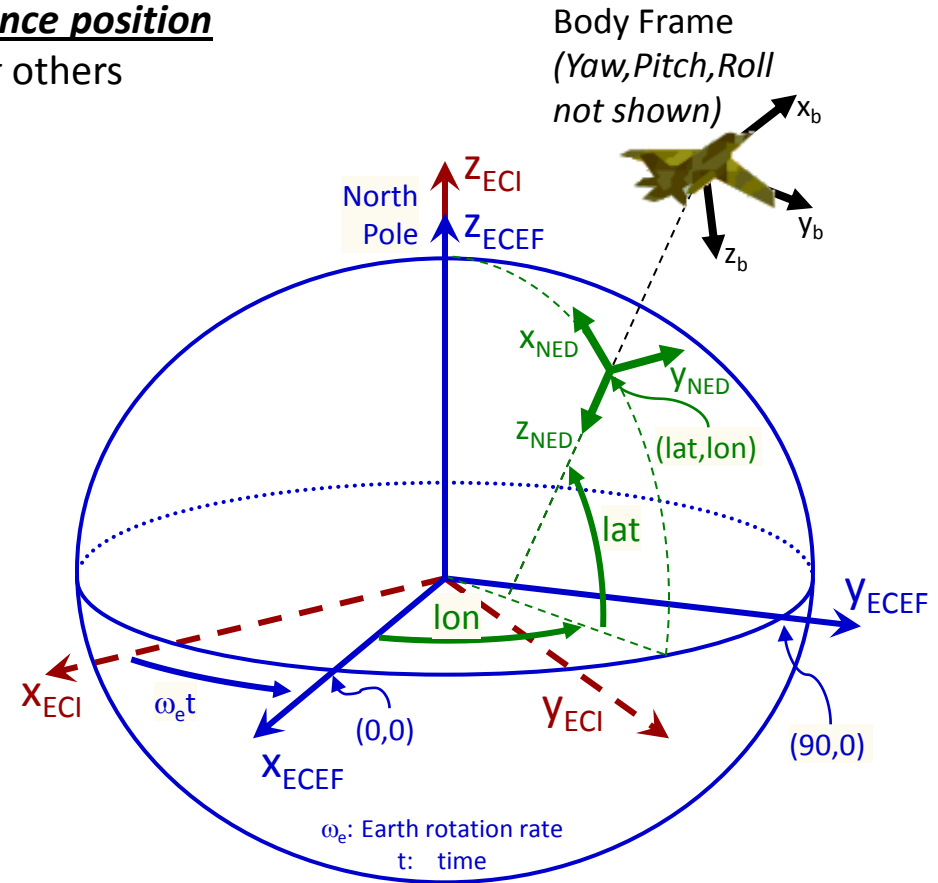
z: toward north pole

y: completes right-hand-rule

ECI: Earth-Centered Inertial

Starts as ECEF at some time reference
(e.g. vernal equinox)

Does not rotate with Earth



"Inertial" frame:

Any frame assumed to be non-moving (rotating or translating)

e.g.: In a flat-earth simulation, NED may be considered inertial

e.g.: In a curved-earth simulation ignoring Earth rotation,
ECEF may be considered inertial

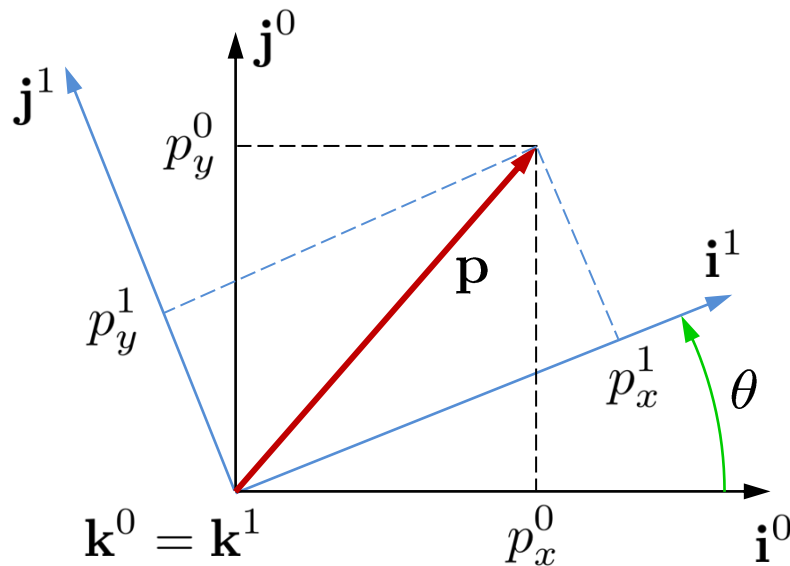
e.g.: In a curved-earth simulation modeling Earth rotation,
ECI may be considered inertial

Rotation of a Coordinate Frame

Consider two coordinate frames.

Frame 0: $\mathbf{i}^0, \mathbf{j}^0, \mathbf{k}^0$ Frame 1: $\mathbf{i}^1, \mathbf{j}^1, \mathbf{k}^1$

Frame 1 is rotated θ radians (about \mathbf{k}^0) relative to Frame 0.



\mathbf{p} can be expressed in Frame 0 coordinates:

$$\mathbf{p} = p_x^0 \mathbf{i}^0 + p_y^0 \mathbf{j}^0 + p_z^0 \mathbf{k}^0$$

\mathbf{p} can be expressed in Frame 1 coordinates:

$$\mathbf{p} = p_x^1 \mathbf{i}^1 + p_y^1 \mathbf{j}^1 + p_z^1 \mathbf{k}^1$$

But the vector \mathbf{p} hasn't changed, so:

$$p_x^1 \mathbf{i}^1 + p_y^1 \mathbf{j}^1 + p_z^1 \mathbf{k}^1 = p_x^0 \mathbf{i}^0 + p_y^0 \mathbf{j}^0 + p_z^0 \mathbf{k}^0$$

With manipulation (taking the dot product of both sides with $\mathbf{i}^1, \mathbf{j}^1$, and \mathbf{k}^1) a relationship can be made between the vector expressed in Frame 0 (\mathbf{p}^0) and the same vector expressed in Frame 1 (\mathbf{p}^1):

$$\mathbf{p}^1 \triangleq \begin{pmatrix} p_x^1 \\ p_y^1 \\ p_z^1 \end{pmatrix} = \begin{pmatrix} \mathbf{i}^1 \cdot \mathbf{i}^0 & \mathbf{i}^1 \cdot \mathbf{j}^0 & \mathbf{i}^1 \cdot \mathbf{k}^0 \\ \mathbf{j}^1 \cdot \mathbf{i}^0 & \mathbf{j}^1 \cdot \mathbf{j}^0 & \mathbf{j}^1 \cdot \mathbf{k}^0 \\ \mathbf{k}^1 \cdot \mathbf{i}^0 & \mathbf{k}^1 \cdot \mathbf{j}^0 & \mathbf{k}^1 \cdot \mathbf{k}^0 \end{pmatrix} \begin{pmatrix} p_x^0 \\ p_y^0 \\ p_z^0 \end{pmatrix}$$

or:

$$\mathbf{p}^1 = \mathcal{R}_0^1 \mathbf{p}^0$$

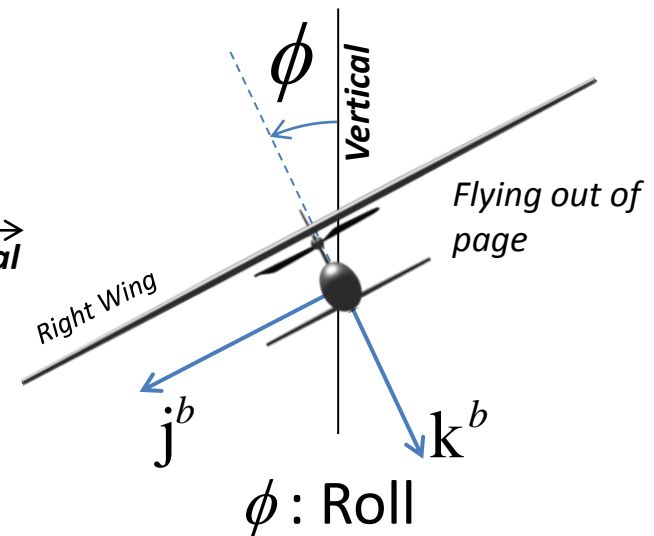
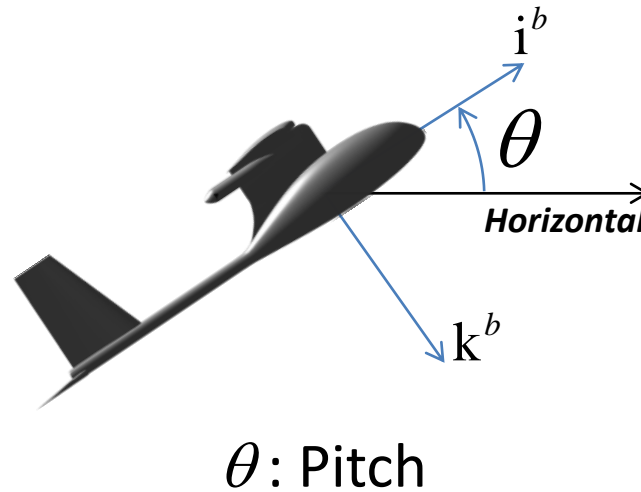
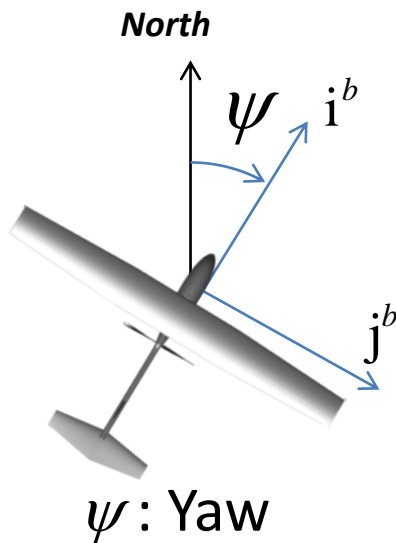
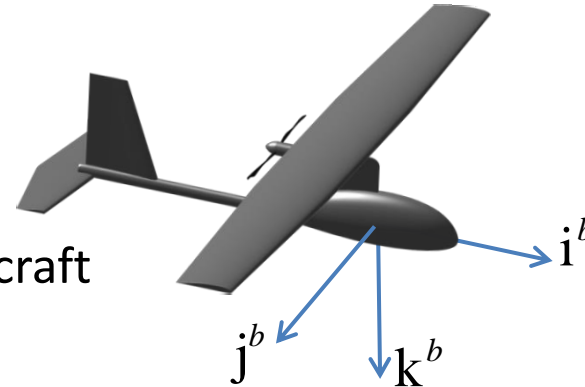
where

$$\mathcal{R}_0^1 \triangleq \begin{pmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

(rotation about \mathbf{k} axis)

Euler Angles

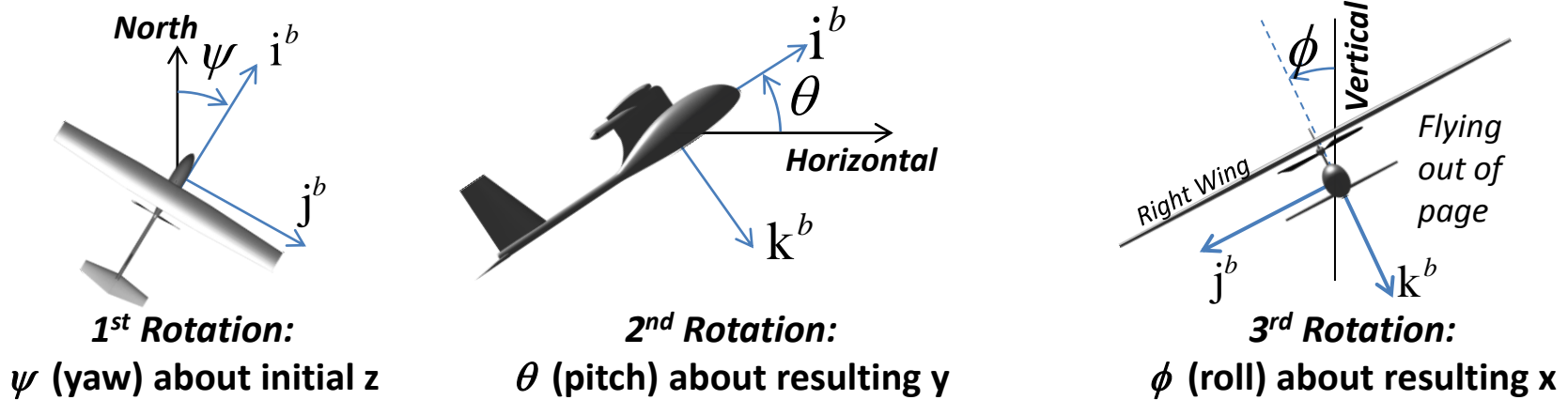
- Define body frame as: $\mathbf{i}^b, \mathbf{j}^b, \mathbf{k}^b$
 - Forward, Right, Down unit vectors
- Need way to describe attitude of aircraft
- Common approach: Euler angles



- Pro: Intuitive
- Con: Mathematical singularity at $\theta = \pm 90^\circ$
 - “Gimbal Lock”: When two of the rotational axes coincide, so a degree of freedom is lost (e.g. ψ and ϕ are indistinguishable at $\theta = \pm 90^\circ$)
 - Quaternions are alternative for overcoming singularity

Rotation Matrix from Euler Angles

Any rotation between two 3D coordinate frames can be expressed through three Euler rotations. Consider the rotation: NED frame to the vehicle body frame:



$$R_{NED}^b = R_x(\phi) R_y(\theta) R_z(\psi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \phi & \sin \phi \\ 0 & -\sin \phi & \cos \phi \end{bmatrix} \begin{bmatrix} \cos \theta & 0 & -\sin \theta \\ 0 & 1 & 0 \\ \sin \theta & 0 & \cos \theta \end{bmatrix} \begin{bmatrix} \cos \psi & \sin \psi & 0 \\ -\sin \psi & \cos \psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

← Rot. Matrices are multiplied right-to-left! →

$$R_{NED}^b = R_x(\phi) R_y(\theta) R_z(\psi) = \begin{bmatrix} \cos \theta \cos \psi & \cos \theta \sin \psi & -\sin \theta \\ \sin \phi \sin \theta \cos \psi - \cos \phi \sin \psi & \sin \phi \sin \theta \sin \psi + \cos \phi \cos \psi & \sin \phi \cos \theta \\ \cos \phi \sin \theta \cos \psi + \sin \phi \sin \psi & \cos \phi \sin \theta \sin \psi - \sin \phi \cos \psi & \cos \phi \cos \theta \end{bmatrix}$$

Rotation Matrices

We can convert between coordinate frames using rotation matrices.

The order of rotation matters! (e.g. $Z \rightarrow Y \rightarrow X$, or $X \rightarrow Y \rightarrow Z$, or $Y \rightarrow X \rightarrow Z$, etc.).

We will generally use the $Z \rightarrow Y \rightarrow X$ rotation order:

$$R = R_x(\phi)R_y(\theta)R_z(\psi)$$

$$R = \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \phi & \sin \phi \\ 0 & -\sin \phi & \cos \phi \end{bmatrix}}_{\text{3rd : } \phi \text{ about } x} \underbrace{\begin{bmatrix} \cos \theta & 0 & -\sin \theta \\ 0 & 1 & 0 \\ \sin \theta & 0 & \cos \theta \end{bmatrix}}_{\text{2nd : } \theta \text{ about } y} \underbrace{\begin{bmatrix} \cos \psi & \sin \psi & 0 \\ -\sin \psi & \cos \psi & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{1st : } \psi \text{ about } z} = \begin{bmatrix} \cos \theta \cos \psi & \cos \theta \sin \psi & -\sin \theta \\ \sin \phi \sin \theta \cos \psi - \cos \phi \sin \psi & \sin \phi \sin \theta \sin \psi + \cos \phi \cos \psi & \sin \phi \cos \theta \\ \cos \phi \sin \theta \cos \psi + \sin \phi \sin \psi & \cos \phi \sin \theta \sin \psi - \sin \phi \cos \psi & \cos \phi \cos \theta \end{bmatrix}$$

← Rotations are right to left

Rotation Matrix Properties:

R_A^B : Transformation from Frame A to Frame B

$$R_B^A = \left(R_A^B\right)^{-1} = \left(R_A^B\right)^T \quad (\text{Note: } R^{-1} = R^T \text{ is a property of Rotation Matrices, not matrices in general})$$

$$R_A^C = R_B^C R_A^B \quad \text{Rot. Matrices can be cascaded (right-to-left)}$$

A vector \underline{x}^A in Frame A coordinates can be expressed in Frame B:

$$\underline{x}^B = R_A^B \underline{x}^A$$

Extract Euler Angles from Rot. Matrix:

$ R_{13} \neq 1$	$ R_{13} = 1 \quad \left(\theta = \pm \frac{\pi}{2} = \pm 90^\circ\right)$
$\phi = \tan^{-1}\left(\frac{R_{23}}{R_{33}}\right)$	$\phi = 0$
$\theta = -\sin^{-1}(R_{13})$	$\theta = -\sin^{-1}(R_{13}) = \pm \frac{\pi}{2}$
$\psi = \tan^{-1}\left(\frac{R_{12}}{R_{11}}\right)$	$\psi = \sin^{-1}(R_{21})$

Remember to use `atan2(num,den)`

Frame Example: The Wind Triangle

Groundspeed vector can be expressed in NED or body coordinates:

$$\underline{v}_g^{NED} = [V_{north} \quad V_{east} \quad V_{down}]^T$$

$$\underline{v}_g^b = \begin{bmatrix} u \\ v \\ w \end{bmatrix} = R_{NED}^b \underline{v}_g^{NED}$$

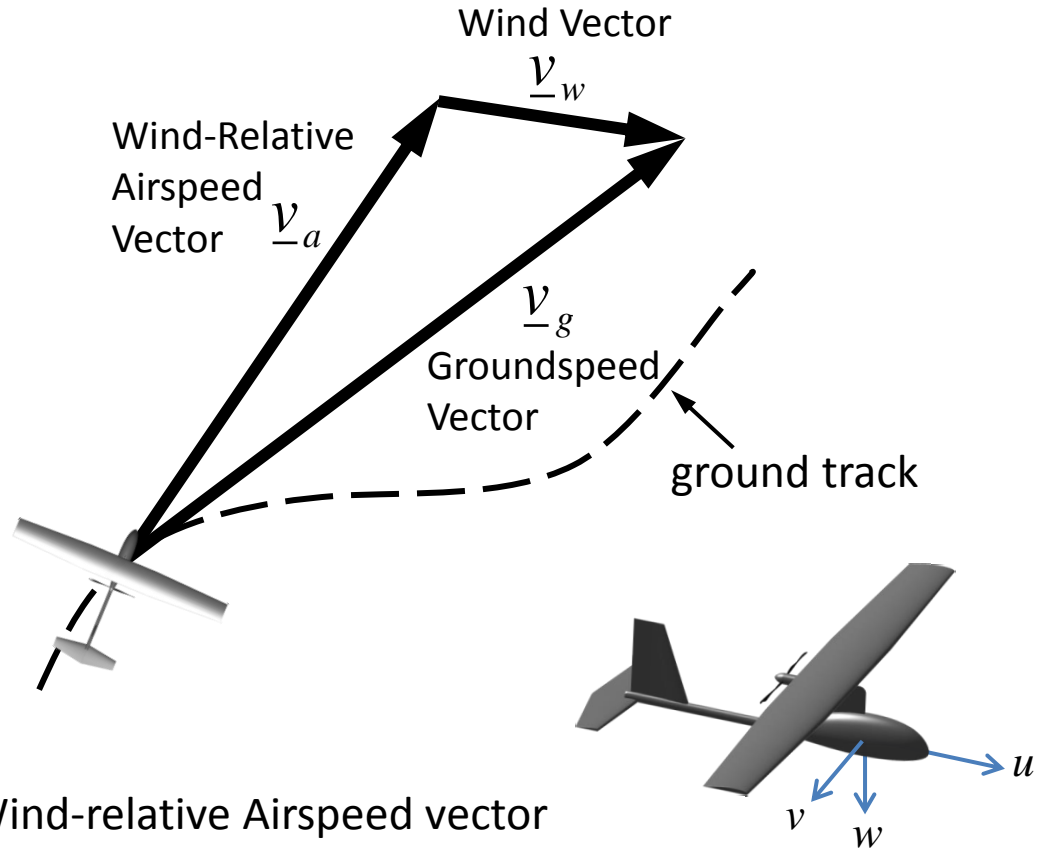
Similarly, Wind vector can be expressed in NED or body coordinates:

$$\underline{v}_w^{NED} = [w_n \quad w_e \quad w_d]^T$$

$$\underline{v}_w^b = \begin{bmatrix} u_w \\ v_w \\ w_w \end{bmatrix} = R_{NED}^b \underline{v}_w^{NED}$$

Wind-relative Airspeed vector expressed in body coordinates:

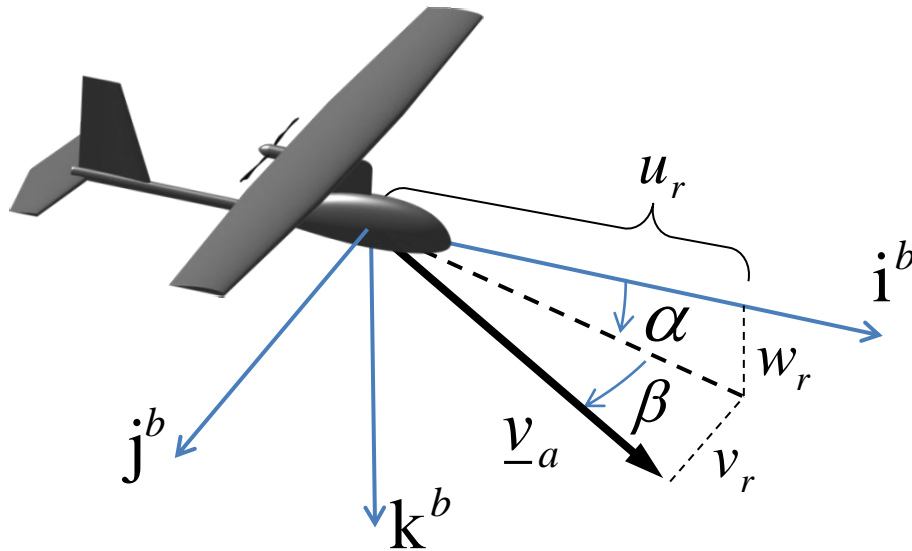
$$\underline{v}_a^b = \begin{bmatrix} u_r \\ v_r \\ w_r \end{bmatrix} = \underline{v}_g^b - \underline{v}_w^b = \begin{bmatrix} u - u_w \\ v - v_w \\ w - w_w \end{bmatrix}$$



Note:

The letters **u , v and w** are used to express various velocity vectors in body frame

Aerodynamic Angles



Wind-relative Airspeed vector expressed in body coordinates:

$$\underline{v}_a^b = \begin{bmatrix} u_r \\ v_r \\ w_r \end{bmatrix} = \underline{v}_g^b - \underline{v}_w^b$$

Airspeed:

$$V_a = |\underline{v}_a^b| = \sqrt{u_r^2 + v_r^2 + w_r^2}$$

We will find that aerodynamic forces are highly dependent on the aerodynamic angles (α and β) which describe the direction of the airspeed vector relative to body frame.

Angle-of-Attack:

$$\alpha = \tan^{-1} \left(\frac{w_r}{u_r} \right)$$

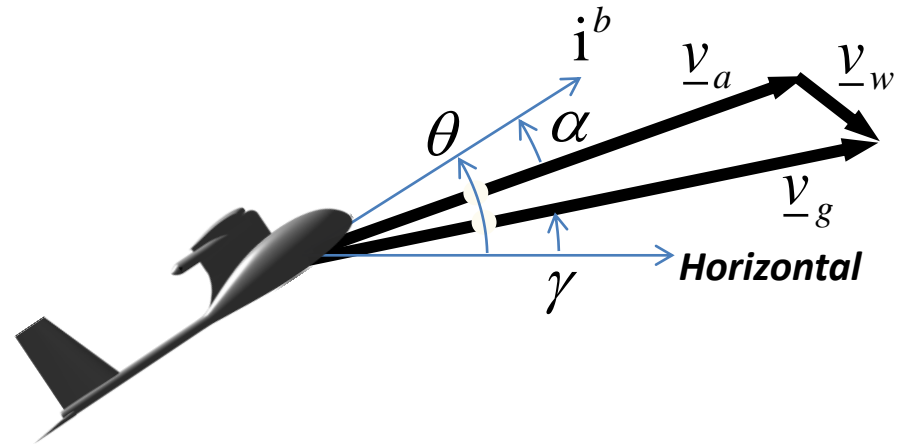
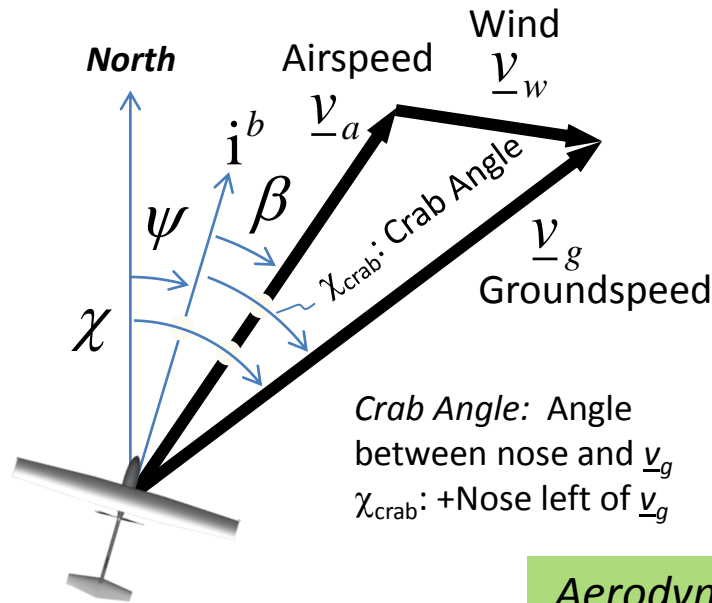
Sideslip:

$$\beta = \sin^{-1} \left(\frac{v_r}{\sqrt{u_r^2 + v_r^2 + w_r^2}} \right)$$

where:

$$\underline{v}_a^b = \begin{bmatrix} u_r \\ v_r \\ w_r \end{bmatrix} = V_a \begin{bmatrix} \cos \alpha \cos \beta \\ \sin \beta \\ \sin \alpha \cos \beta \end{bmatrix}$$

Vehicle Angles from Wind Triangle



Euler Angles describing orientation relative to NED:

ψ : Yaw, +EastOfNorth
 θ : Pitch, +NoseUp
 ϕ : Roll, +RightWingDown
 (ϕ not shown)

$$\mathbf{R}_{NED}^b = \mathbf{R}_x(\phi)\mathbf{R}_y(\theta)\mathbf{R}_z(\psi)$$

Aerodynamic angles describing Airspeed Vector relative to body:

α : Angle-Of-Attack, +Nose above \underline{v}_a
 β : Sideslip, +Nose left of \underline{v}_a

$$\underline{v}_a^b = \begin{bmatrix} u_r \\ v_r \\ w_r \end{bmatrix} = V_a \begin{bmatrix} \cos \alpha \cos \beta \\ \sin \beta \\ \sin \alpha \cos \beta \end{bmatrix}$$

Angles describing Groundspeed Vector relative to NED:

χ : Course (Horizontal), +EastOfNorth
 γ : Flight Path Angle (Vertical), +Up

$$\underline{v}_g^{NED} = \begin{bmatrix} V_{north} \\ V_{east} \\ V_{down} \end{bmatrix} = V_g \begin{bmatrix} \cos \chi \cos \gamma \\ \sin \chi \cos \gamma \\ -\sin \gamma \end{bmatrix}$$

Homework: $V_g = ?$ $\gamma = ?$ $\chi = ?$

System Modeling Refresher

Differential Equations

State Space

Laplace Transfer Functions

Block Diagrams



Further background in Laplace:

<http://www.calpoly.edu/~fowen/me422/Chapter2.pdf>

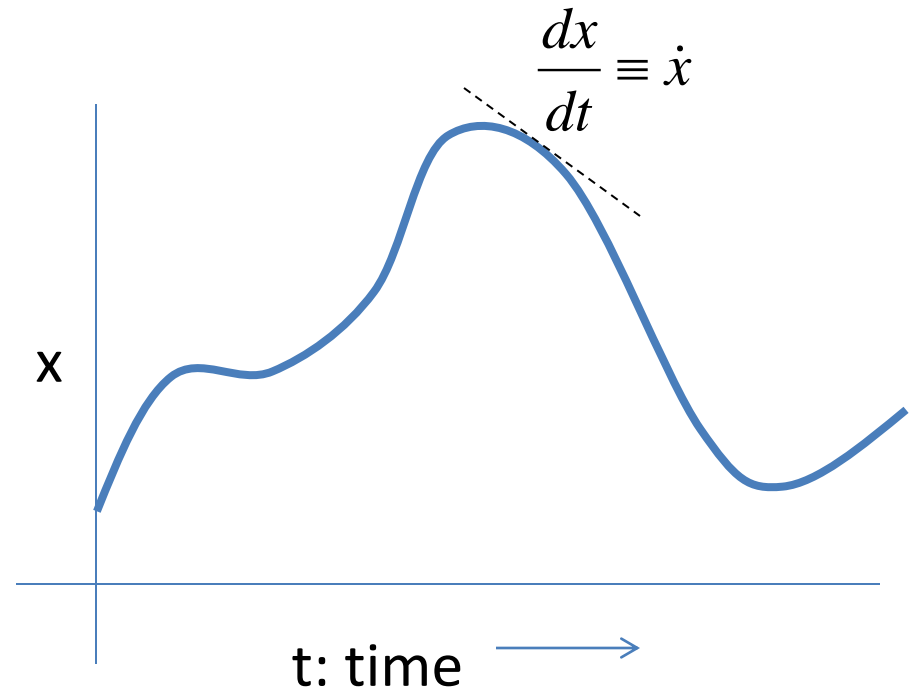
Differential Equations

- We will use differential equations to describe the motion of systems as a function of time

$$\frac{dx}{dt} = f(x, \text{other variables})$$

or

$$\dot{x} = f(x, \text{other variables})$$



Dynamic response example using Matlab

Simple script to emulate: $\dot{x} = -3x$, $x(0) = 5$

```
% Initial state value
x = 5;

% Time starts at zero, increments by dt seconds
t = 0;
dt = 0.001;

% For retaining history
tHistory = [];
xHistory = [];

% Loop through time, stopping after 2 seconds
while t < 2

    xdot = -3*x;           % Define state derivatives

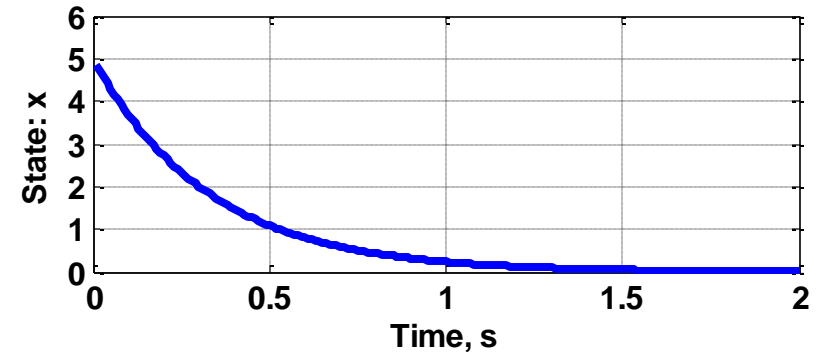
    x = x + xdot*dt;       % Propagate state (simple Euler integration)

    t = t + dt;            % Increment time

    tHistory(end+1) = t;   % Retain history
    xHistory(end+1) = x;

end

% Plot
plot(tHistory, xHistory);
```



Coupled Differential Equations: Characteristics

The dynamical response of a system may be:

- Coupled between 2 or more variable “states”

$$\dot{x}_1 = f_1(x_1, x_2)$$

$$\dot{x}_2 = f_2(x_1, x_2)$$

Example: $\dot{x}_1 = 4x_1 + 10x_2$
 $\dot{x}_2 = -3x_1$

- A function of one or more input stimuli:

$$\dot{x}_1 = f_1(x_1, x_2, u_1, u_2)$$

$$\dot{x}_2 = f_2(x_1, x_2, u_1, u_2)$$

Example: $\dot{x}_1 = 4x_1 + 10x_2 + u_1 - 2u_2$
 $\dot{x}_2 = -3x_1 + 5u_1$

- Linear (i.e. Derivatives are a linear combination of variables & inputs)

Examples: $\dot{x}_1 = 4x_1 + 10x_2 + u_1 - 2u_2$ $\dot{x}_1 = a_{11}x_1 + a_{12}x_2 + b_{11}u_1 + b_{12}u_2$
 $\dot{x}_2 = -3x_1 + 5u_1$ $\dot{x}_2 = a_{21}x_1 + a_{22}x_2 + b_{21}u_1 + b_{22}u_2$

where a_{ij} and b_{ij} are constants

- Non-Linear

Examples:

$\dot{x}_1 = 4x_1 + x_1x_2$ $\dot{x}_2 = -3/x_1$	$\dot{x}_1 = 4x_1 + \tan x_2$ $\dot{x}_2 = -3x_1^2$	$\dot{x}_1 = a_{11}x_1 + a_{12}\sqrt{x_2} + b_{11}u_1 + b_{12}u_2^3$ $\dot{x}_2 = a_{21}x_1 + a_{22}x_2 + b_{21}u_1u_2 + b_{22}x_1u_2$
--	---	--

State Space Models

- Given a set of linear and coupled differential equations, we can build a linear mapping between the *states* and the *state derivatives*:

$$\begin{aligned}\dot{x}_1 &= a_{11}x_1 + a_{12}x_2 + a_{13}x_3 & + & b_{11}u_1 + b_{12}u_2 \\ \dot{x}_2 &= a_{21}x_1 + a_{22}x_2 + a_{23}x_3 & + & b_{21}u_1 + b_{22}u_2 \\ \dot{x}_3 &= a_{31}x_1 + a_{32}x_2 + a_{33}x_3 & + & b_{31}u_1 + b_{32}u_2\end{aligned}$$

where:

x_j is the j th system state

a_{ij} and b_{ij} are constants

u_j is the j th input

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \underbrace{\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}}_A \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \underbrace{\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix}}_B \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

Inputs can be:

- zero
- constant
- A function of time

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = A \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + B \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \quad \Rightarrow \quad \underline{\dot{x}} = A\underline{x} + B\underline{u}$$

State Space Models

- In a state space model, the A and B matrices completely describe the dynamical motion of the states given:

- An initial condition of the states
- Time-varying inputs (stimuli)

$$\dot{\underline{x}} = \underline{A}\underline{x} + \underline{B}\underline{u} \quad \underline{x}(0) = \begin{bmatrix} x_1(0) \\ x_2(0) \\ x_3(0) \end{bmatrix} \quad \underline{u} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} f_{u1}(t) \\ f_{u2}(t) \end{bmatrix}$$

- A state space model can also have one or more outputs, which are a linear combinations of states and inputs

$$y_1 = c_{11}x_1 + c_{12}x_2 + c_{13}x_3 + d_{11}u_1 + d_{12}u_2$$

$$y_2 = c_{21}x_1 + c_{22}x_2 + c_{23}x_3 + d_{21}u_1 + d_{22}u_2$$

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} d_{11} & d_{12} \\ d_{21} & d_{22} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$



$$\underline{y} = \underline{C}\underline{x} + \underline{D}\underline{u}$$

\underline{y} is a combination of states and inputs

- \underline{y} has no bearing on the dynamics of the states
- Think of \underline{y} as an “observation” of states

D is often zero

- Output is purely a linear combination of states

State Space response example using Matlab

Simple script to emulate:

$$\dot{\underline{x}} = \begin{bmatrix} -1 & -2 \\ 1 & -1 \end{bmatrix} \underline{x} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \underline{u}$$

$$y = \begin{bmatrix} 2 & -1 \end{bmatrix} \underline{x} + 0 \underline{u}$$

% System

A = [-1 -2; 1 -1];

B = [0; 1];

C = [2 -1];

D = 0;

% Initial state value

x = [0; 0]; **% x is 2x1**

% Time starts at zero,

% increments by dt sec

t = 0;

dt = 0.001;

% For retaining history

tHistory = [];

xHistory = [];

yHistory = [];

uHistory = [];

% Loop through time, stopping after 5 seconds

while t < 5

u = sin(2*t); **% Input function**

xdot = A*x + B*u; **% Define state derivatives**

x = x + xdot*dt; **% Propagate state (simple Euler integration)**

y = C*x + D*u; **% Output is a combination of x and u**

t = t + dt; **% Increment time**

tHistory(end+1) = t; **% Retain history**

xHistory(end+1,:) = x';

yHistory(end+1) = y;

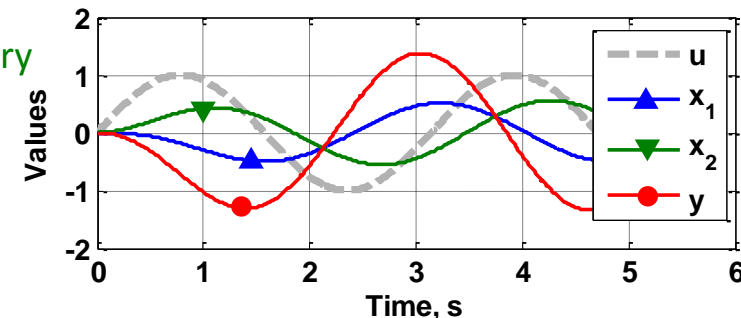
uHistory(end+1) = u;

end

Note the colon:

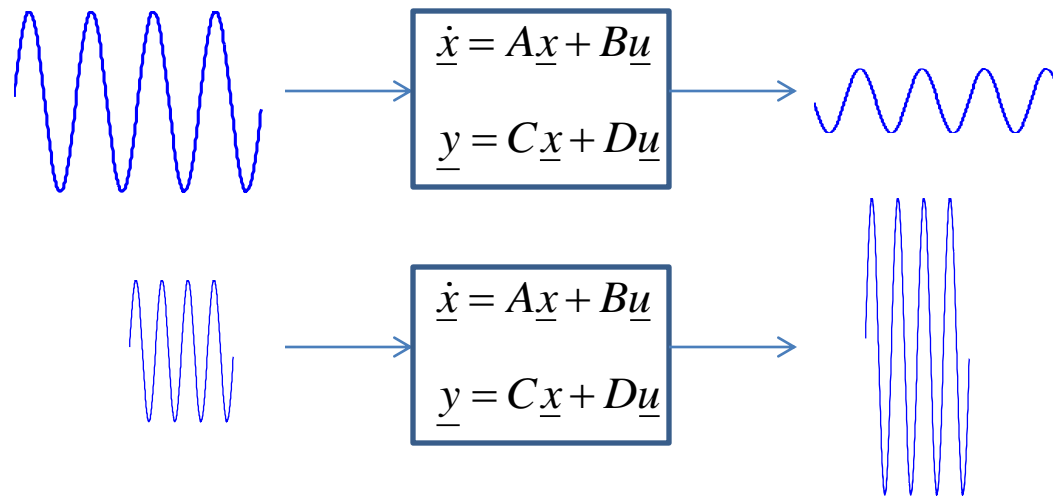
xHistory(end+1,:) = x';

The colon here indicates "all rows". So, if x is 2-by-1, xHistory will be an n-by-2 array.

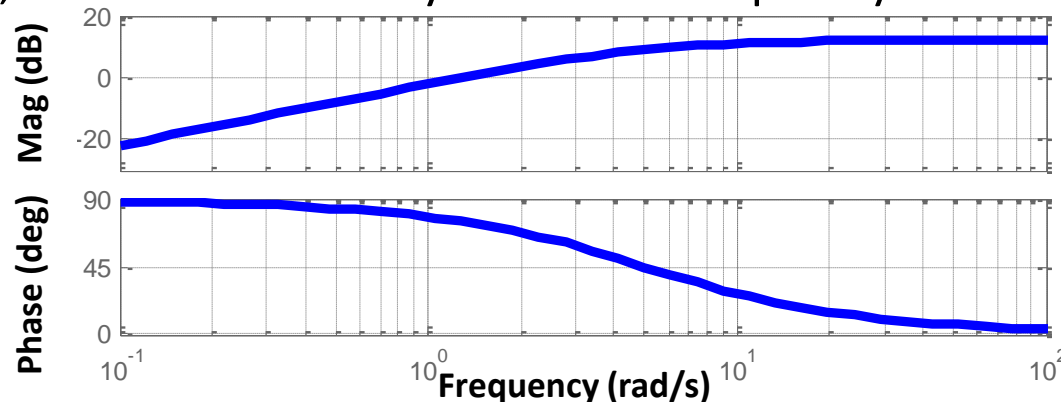


Frequency Domain

- An important property of linear, time-invariant systems is that if the input is sinusoidal, the output will be sinusoidal at the same frequency (with a generally different magnitude and phase)



- Thus, we can also view systems in a frequency domain:



Laplace Transform

- We use the Laplace Transform to convert a time-domain system into the frequency domain

– The Laplace Transform of a time domain function is:

$$F(s) = \mathcal{L}\{f(t)\} = \int_0^{\infty} e^{-st} f(t) dt \quad \leftarrow \begin{array}{l} \text{Just a definition.} \\ \text{We don't actually do this.} \end{array}$$

- In Laplace Domain “s” signifies frequency: $s = j\omega$, radians/sec
- The Laplace of the time derivative of a function is useful:

$$\mathcal{L}\{f(t)\} = F(s) \Rightarrow \mathcal{L}\{\dot{f}(t)\} = sF(s) - f(0)$$

- As a result, we can express differential equations as algebraic equations:

Time Domain		Laplace Domain
\dot{x}	\Leftrightarrow	$sX(s)$

$$\begin{array}{l} \dot{x}_1 = 4x_1 + 10x_2 + 2u \\ \dot{x}_2 = -3x_1 \end{array} \quad \begin{array}{l} \nearrow \\ \searrow \end{array} \quad \begin{array}{l} sX_1(s) = 4X_1(s) + 10\left\{\frac{-3X_1(s)}{s}\right\} + 2U(s) \\ sX_2(s) = -3X_1(s) \end{array} \quad \begin{array}{l} \nearrow \\ \searrow \end{array} \quad \begin{array}{l} \frac{X_1(s)}{U(s)} = \frac{2s}{(s^2 - 4s - 30)} \\ \frac{X_2(s)}{U(s)} = \frac{-6}{(s^2 - 4s - 30)} \end{array}$$

Transfer Functions

- In Laplace, we can conveniently view a SISO (Single-Input-Single-Output) system as a transfer function, defined as the ratio of output to input:

$$G(s) = \frac{Y(s)}{U(s)} = \frac{b_m s^m + b_{m-1} s^{m-1} + \dots + b_1 s + b_0}{a_n s^n + a_{n-1} s^{n-1} + \dots + a_1 s + a_0}$$

- We can re-arrange the transfer function into zero-pole-gain form:

$$G(s) = \frac{Y(s)}{U(s)} = K \frac{(s - z_1)(s - z_2) \dots (s - z_{m-1})(s - z_m)}{(s - p_1)(s - p_2) \dots (s - p_{n-1})(s - p_n)}$$

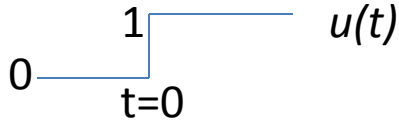
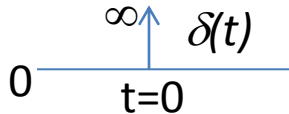
- “Zeros” are the roots of the numerator polynomial
 - “Poles” are the roots of the denominator polynomial (aka eigenvalues)
 - Both zeros and poles may be real or complex!
- A State-Space model (A,B,C,D) can be converted to a Transfer Function:

$$G(s) = \frac{Y(s)}{U(s)} = C(sI - A)^{-1} B + D$$

Note: a MIMO (Multi-Input-Multi-Output) system will simply become an array of transfer functions from each input to each output, e.g.:

$$\begin{bmatrix} G_{11}(s) & G_{12}(s) \\ G_{21}(s) & G_{22}(s) \\ G_{31}(s) & G_{32}(s) \end{bmatrix}$$

Laplace & Transfer Function Properties

- Linearity: $ax(t) + bv(t) \leftrightarrow aX(s) + bV(s)$
- Differentiation: $\dot{x}(t) \leftrightarrow sX(s) - x(0)$
- Integration: $\int_0^t x(\tau) d\tau \leftrightarrow \frac{1}{s} X(s)$
- Unit Step: $u(t) \leftrightarrow \frac{1}{s}$

- Impulse: $\delta(t) \leftrightarrow 1$

- Exciting a system: $\text{Given: } G(s) = \frac{Y(s)}{U(s)} \rightarrow Y(s) = G(s)U(s) \rightarrow y(t) = \mathcal{L}^{-1}(G(s)U(s))$
- Commutativity: $H(s)G(s) = G(s)H(s)$
- Initial Value Theorem: $x(0) = \lim_{s \rightarrow \infty} sX(s)$
- Final Value Theorem: $\lim_{t \rightarrow \infty} x(t) = \lim_{s \rightarrow 0} sX(s)$

Provided that $X(s)$ is rational and its poles are strictly negative

s : *Derivative*

$\frac{1}{s}$: *Integration*

Stability

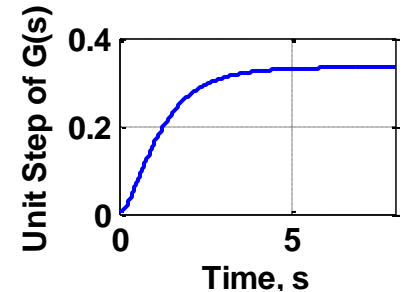
- A system is considered “stable” if a bounded (i.e. finite) input results in a bounded (i.e. finite) output
- For a Laplace Transfer Function, a system is stable if all the poles (roots of the denominator) are negative

$$G(s) = \frac{Y(s)}{U(s)} = K \frac{(s - z_1)(s - z_2) \dots (s - z_{m-1})(s - z_m)}{(s - p_1)(s - p_2) \dots (s - p_{n-1})(s - p_n)}$$

Stable only if all: $p_1 < 0$, $p_2 < 0$, \dots $p_n < 0$

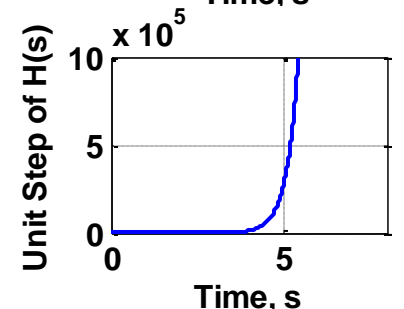
- Stable example:

$$\begin{array}{l} p_1 = -1 \\ p_2 = -3 \end{array} \rightarrow G(s) = \frac{1}{(s - \{-1\})(s - \{-3\})} = \frac{1}{(s + 1)(s + 3)}$$



- Unstable example:

$$\begin{array}{l} p_1 = -1 \\ p_2 = +3 \end{array} \rightarrow H(s) = \frac{1}{(s - \{-1\})(s - \{+3\})} = \frac{1}{(s + 1)(s - 3)}$$

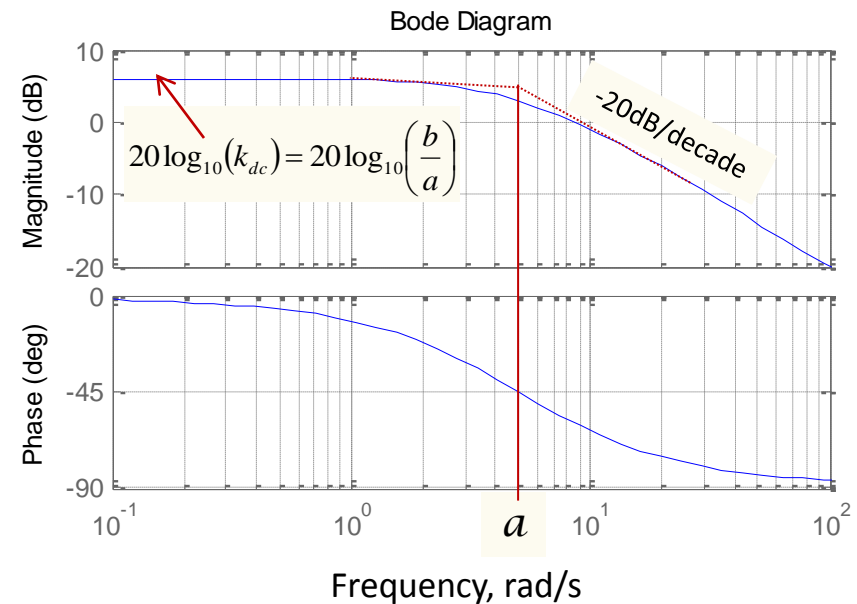
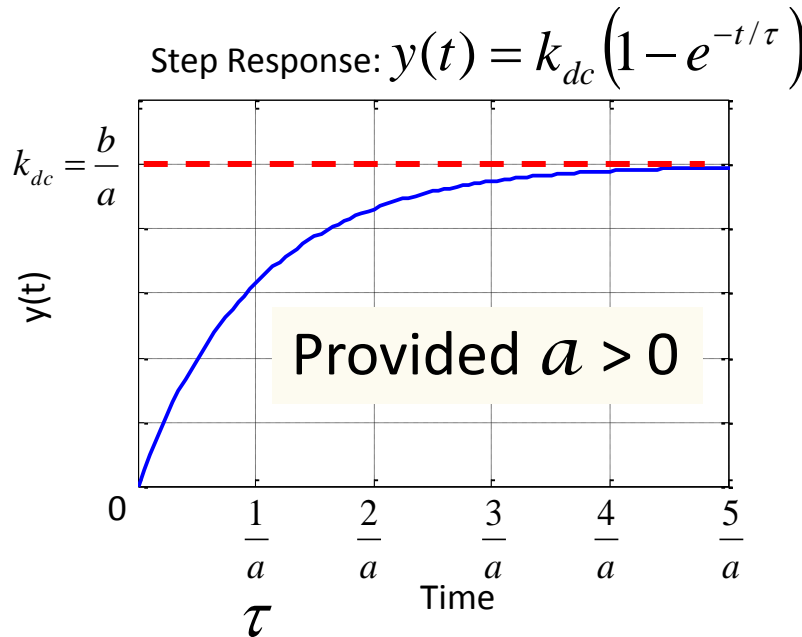


First Order Systems

First Order Systems are the simplest dynamic systems:

$$\dot{y} = -ay + bu \rightarrow \frac{Y(s)}{U(s)} = \frac{b}{s+a} = \frac{k_{dc}}{\tau s + 1}$$

$k_{dc} = \frac{b}{a}$ Steady - State Gain
 $\tau = \frac{1}{a}$ Time Constant (63% Rise Time)



Matlab: `s=tf('s'); % Make s`
`step(b/(s+a))`

`bode(b/(s+a))`

Second Order Systems

Second Order Systems exhibit oscillations:

$$\ddot{y} + a_1 \dot{y} + a_0 = bu \quad \rightarrow \quad \frac{Y(s)}{U(s)} = \frac{b}{s^2 + a_1 s + a_0} = \frac{b}{(s - p_1)(s - p_2)}$$

$$p_1 = \frac{1}{2} \left(-a_1 + \sqrt{a_1^2 - 4a_0} \right)$$

$$p_2 = \frac{1}{2} \left(-a_1 - \sqrt{a_1^2 - 4a_0} \right)$$

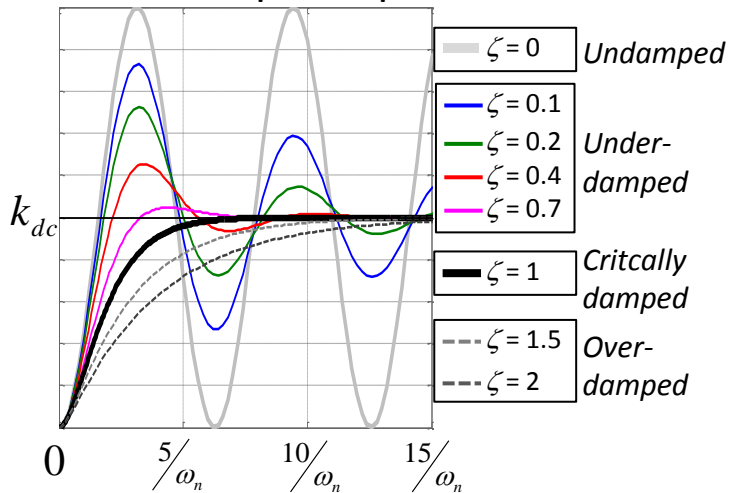
Conveniently re-written using natural frequency (ω_n) and damping (ζ)

$$\ddot{y} + 2\zeta\omega_n \dot{y} + \omega_n^2 = k_{dc}\omega_n^2 u \quad \rightarrow \quad \frac{Y(s)}{U(s)} = \frac{k_{dc}\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}$$

$$p_1 = -\zeta\omega_n + j\omega_n\sqrt{1-\zeta^2}$$

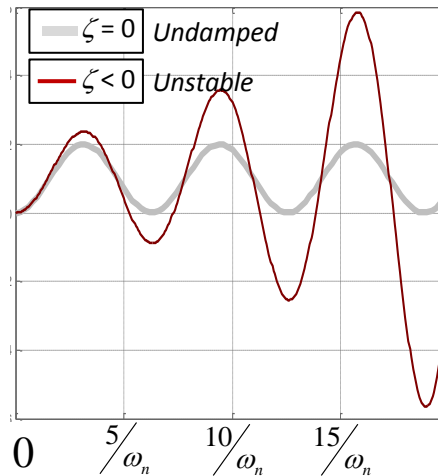
$$p_2 = -\zeta\omega_n - j\omega_n\sqrt{1-\zeta^2}$$

Stable Steps Responses

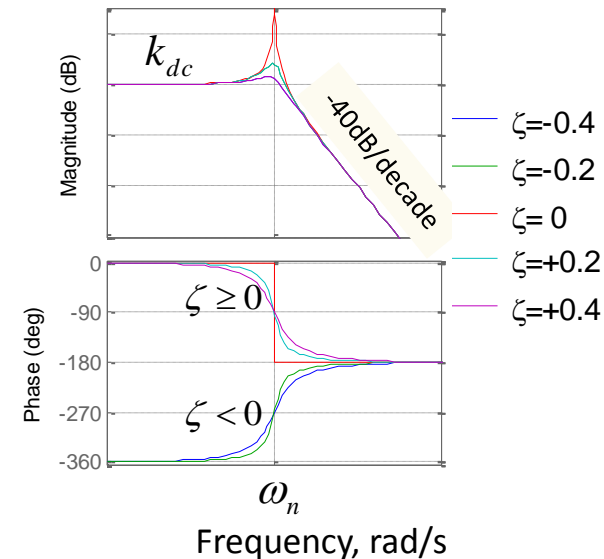


Time

Unstable Steps Response



Time

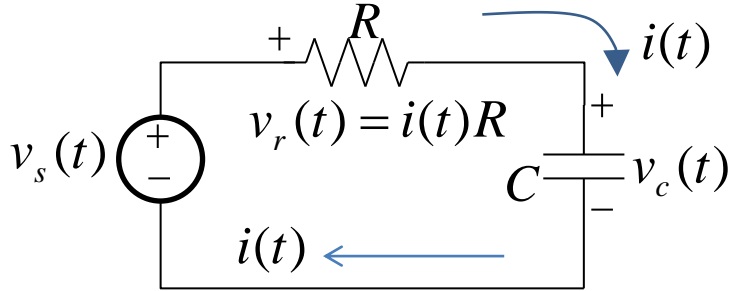


Frequency, rad/s

```
s=tf('s'); % Make s
step(wn^2/(s^2+2*zeta*wn*s+wn^2))
bode(wn^2/(s^2+2*zeta*wn*s+wn^2))
```

1st and 2nd Order Examples

First Order Example: RC Circuit



Given: R , C , and $v_s(t)$

Want: $v_c(t)$ (Voltage across cap.)

Capacitor: $i(t) = C \frac{dv_c(t)}{dt}$

$$v_s(t) = v_r(t) + v_c(t)$$

$$v_s(t) = R \cdot i(t) + v_c(t)$$

$$v_s(t) = RC \frac{d}{dt} v_c(t) + v_c(t)$$

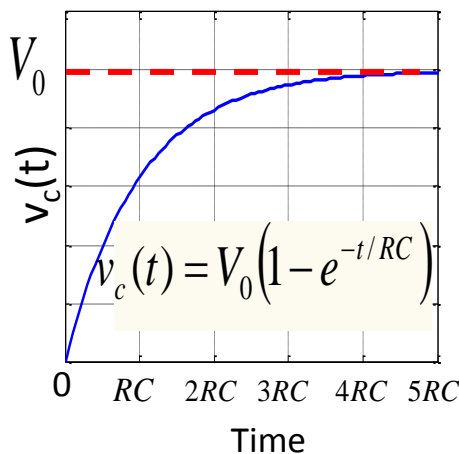
$$V_s(s) = RCsV_c(s) + V_c(s)$$

$$V_s(s) = (RCs + 1)V_c(s)$$

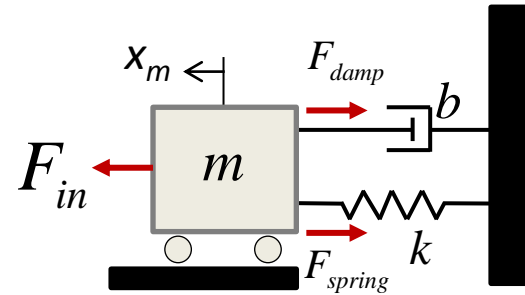
$$\Rightarrow \frac{V_c(s)}{V_s(s)} = \frac{1}{RCs + 1}$$

$$\tau = RC \text{ seconds}$$

Capacitor voltage
due to a step source:
 $v_s(t) = V_0$ for $t \geq 0$



Second Order Example: Spring Damper System



x_m : Mass Pos.

m : mass

$$F_{damp} = b\dot{x}_m$$

$$F_{spring} = kx_m$$

F_{in} = Input Force

Newton's Law: Sum of Forces = mass * accel

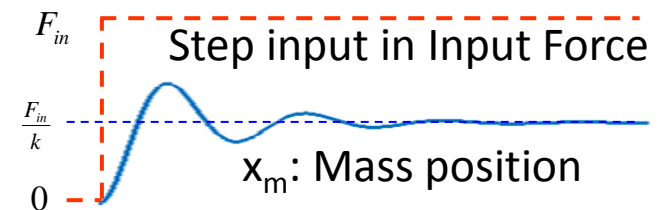
$$F_{in}(t) - F_{damp} - F_{spring} = m\ddot{x}_m$$

$$F_{in}(t) - b\dot{x}_m - kx_m = m\ddot{x}_m$$

$$m\ddot{x}_m + b\dot{x}_m + kx_m = F_{in}(t)$$

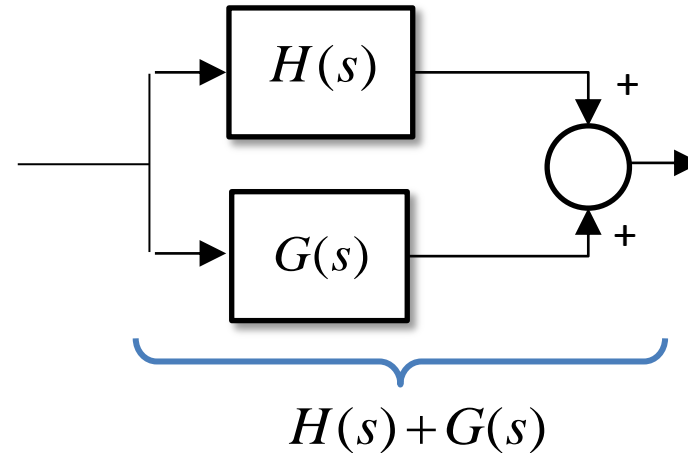
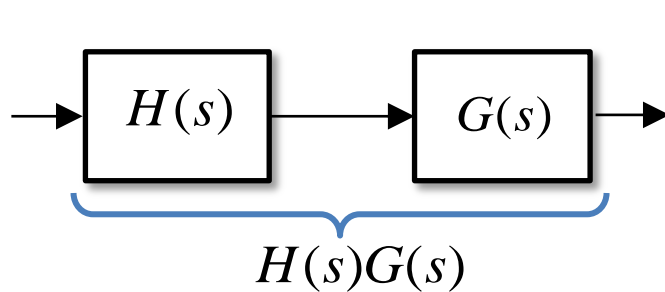
$$ms^2 X_m(s) + bsX_m(s) + kX_m(s) = F_{in}(s)$$

$$\frac{X_m(s)}{F_{in}(s)} = \frac{\frac{1}{m}}{s^2 + \frac{b}{m}s + \frac{k}{m}}$$

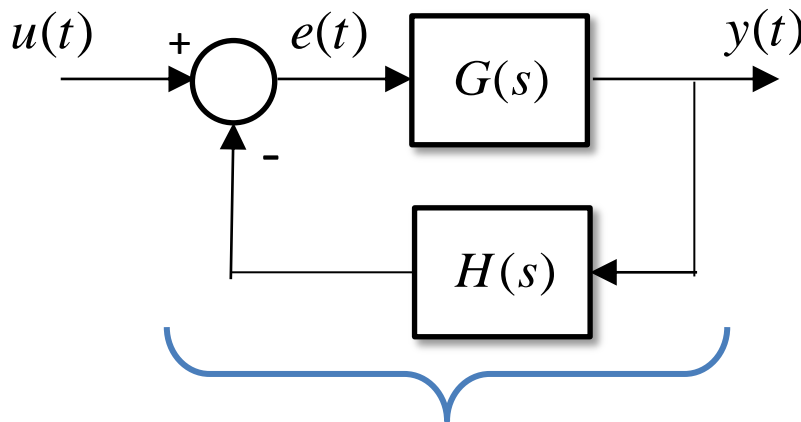


Block Diagrams

- Laplace Transfer Functions make block diagrams useful



- Negative Feedback



Proof:

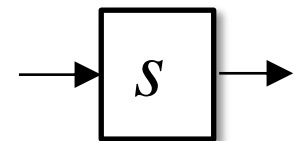
$$E(s) = U(s) - G(s)H(s)E(s)$$

$$(1 + G(s)H(s))E(s) = U(s)$$

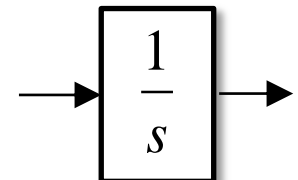
$$\frac{E(s)}{U(s)} = \frac{1}{1 + G(s)H(s)}$$

$$\frac{Y(s)}{U(s)} = \frac{G(s)}{1 + G(s)H(s)}$$

Derivative:



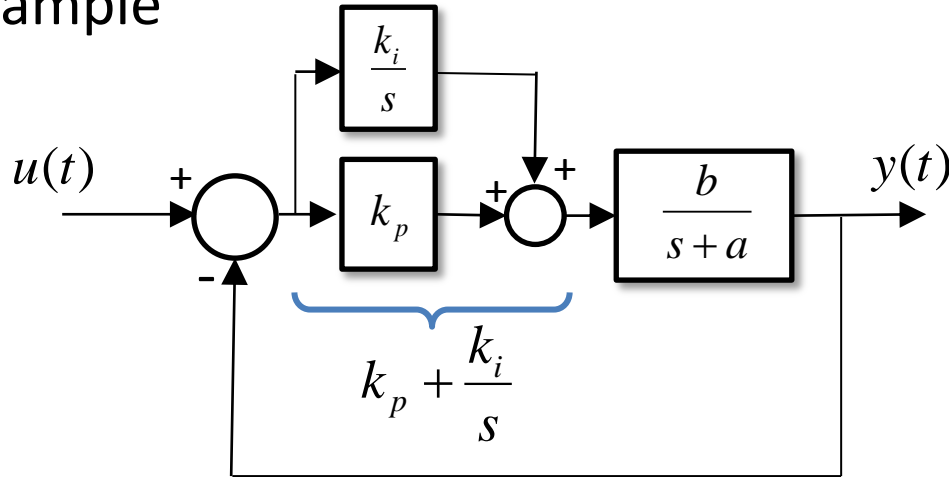
Integral:



$$\frac{G(s)}{1 + G(s)H(s)} = \frac{\{FwdPath\}}{1 + \{FwdPath\}\{FeedbackPath\}}$$

Block Diagrams

- Example



$$\frac{Y(s)}{U(s)} = \frac{\{FwdPath\}}{1 + \{FwdPath\}\{FeedbackPath\}} = \frac{\left(k_p + \frac{k_i}{s}\right)\left(\frac{b}{s+a}\right)}{1 + \left(k_p + \frac{k_i}{s}\right)\left(\frac{b}{s+a}\right)} \quad \leftarrow \text{Multiply by } 1: \frac{s(s+a)}{s(s+a)}$$

$$\frac{Y(s)}{U(s)} = \frac{s(s+a)\left(k_p + \frac{k_i}{s}\right)\left(\frac{b}{s+a}\right)}{s(s+a) + s(s+a)\left(k_p + \frac{k_i}{s}\right)\left(\frac{b}{s+a}\right)} = \frac{(k_p s + k_i)(b)}{s(s+a) + (k_p s + k_i)(b)} = \frac{bk_p s + bk_i}{s^2 + (a + bk_p)s + bk_i}$$

Useful Matlab Commands for Controls

`% Create Transfer Functions and State Space Models`

`s = tf('s');` `% Make a Laplace s`

`G = (2*s+5)/(4*s^2+7*s+2);` `% Make a Transfer Function`

`num = [2 5];`

`den = [4 7 2];`

`G = tf(num,den);` `% Another way to make a Transfer Function`

`G = ss(A,B,C,D);` `% Create a State Space model from [A B C D]`

`% Convert between Transfer Functions, Zero-Pole-Gain form, and State Space`

`G = tf(G);` `% Convert to a Transfer Function`

`G = ss(G);` `% Convert to a State Space Model`

`G = zpkm(G);` `% Convert to Zero-Pole-Gain form`

`I=eye(size(A));`

`G = C*inv(s*I-A)*B+D;` `% Another conversion to a Transfer Function`

`[z p k] = zpkmdata(G,'v');` `% Extract zeros, poles and gain ('v' to return vectors)`

`[num den] = tfdata(G,'v');` `% Extract numerator and denominator coefficients`

`[A B C D] = ssdata(G);` `% Extract State Space matrices`

`G = minreal(G);` `% Reduce to minimum realization transfer function`

`% (Performs pole/zero cancellations)`

Useful Matlab Commands for Controls

`% Plotting and outputs`

`t=0:.1:10;`

`T=10;`

```
step(G);           % Plot step response
step(G,H);         % Plot multiple step responses
step(G,T);         % Specify duration (first T seconds)
y=step(G,t);       % Output step response at times t
```

`plot(t,step(G,t)); % Another way to plot a step response`

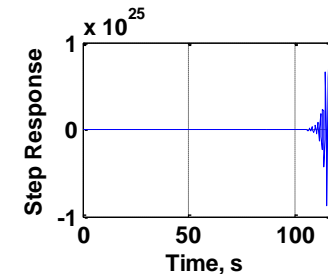
```
impulse(G);        % Plot impulse response
impulse(G,H);      % Plot multiple impulse responses
impulse(G,T);      % Specify duration (first T seconds)
y=impulse(G,t);    % Output impulse response at times t
```

```
u = 4*sin(3*t);
lsim(G,u,t);       % Plot/Output response of G stimulated
y=lsim(G,u,t);     % by input vector u at times t
```

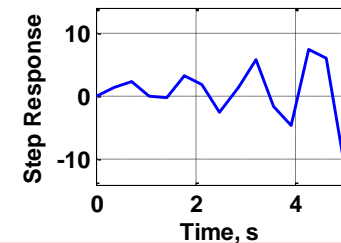
```
bode(G)            % Plot Bode response
bode(G,H)          % Plot multiple Bode responses
```

```
s=tf('s');
wn=5; zeta = -0.1;
G = wn^2/(s^2+2*zeta*wn*s+wn^2);
```

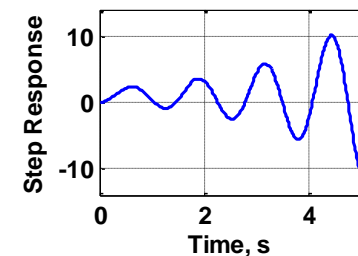
`step(G) % Be Careful Trusting Time!`



`xlim([0 5])`



`step(G,5) % Better to specify time!`



Lecture 1 Homework, Page 1 of 7

- 1) In an airplane, the motion of angle-of-attack (α), pitch rate ($\dot{\theta}$) and pitch (θ) are tightly related and respond very quickly to a change in the “elevator” control surface (δ_e). (In later classes we will derive and utilize these relationships). Consider a dynamic system satisfying:

$$\dot{\alpha} + 3\alpha - 2\dot{\theta} = 0$$

$$\ddot{\theta} + 10\dot{\alpha} + 15\theta = -5\delta_e$$

- a) Re-arrange the above two equations to form three equations with the following form:

$$\dot{\alpha} = ?\alpha + ?\dot{\theta} + ?\theta + ?\delta_e$$

$$\ddot{\theta} = ?\alpha + ?\dot{\theta} + ?\theta + ?\delta_e$$

$$\dot{\theta} = ?\alpha + ?\dot{\theta} + ?\theta + ?\delta_e$$

- b) Let the state vector be: $\underline{x} = [\alpha \quad \dot{\theta} \quad \theta]^T$

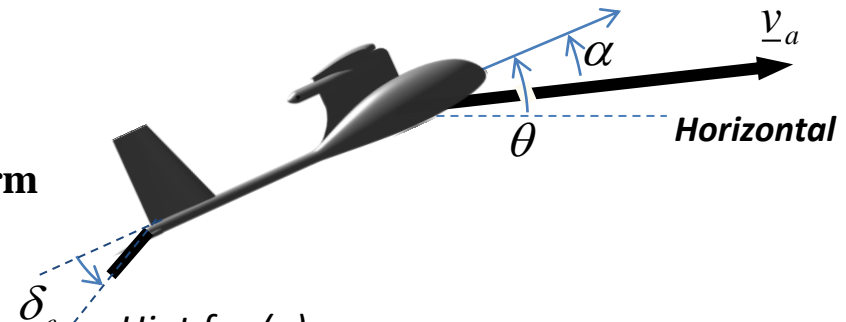
Convert the above dynamic model into a SISO state space model going from an input of δ_e to an output of θ . (i.e. determine the A, B, C, and D matrices)

- c) Using Matlab, convert the resulting state space model into a transfer function in Zero-Pole-Gain form.

For checking:

$$-5 (s+3)$$

$$(s+1.373) (s^2 + \text{XXXX}s + 32.77)$$



Hint for (a):

The 3rd equation may be the trickiest, but is also the most trivial.

(Consider, for example, $x=\dot{x}$)

It may seem trivial, but is necessary for (b)

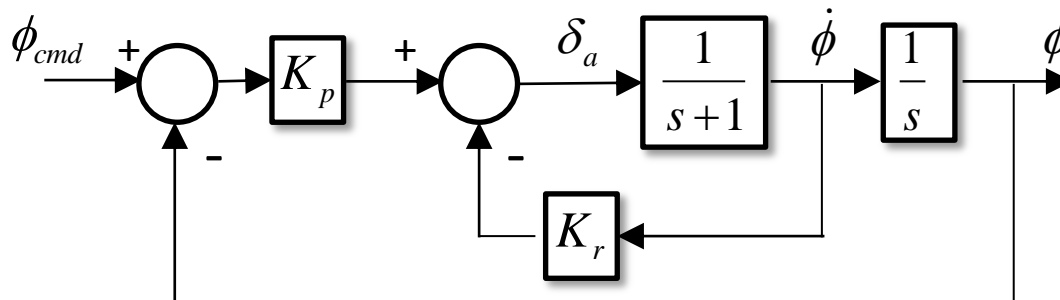
Continued on next slide...

Lecture 1 Homework, Page 2 of 7

- d) **Modify the Matlab script example provided in the lecture slide “State Space response example using Matlab” to generate a time-response of the state-space model from part (b) to a unit-step elevator input (i.e., $u = \delta_e = 1$). Start the script with an initial state vector of $\underline{x} = [0 \ 0 \ 0]^T$, and propagate time from 0 to 5 seconds. Print script and compare the result with a step response of the transfer function you generated in part (c):**

```
plot(tHistory,yHistory, tHistory,step(G,tHistory),'r--');
```

- 2) **Consider the following block diagram that represents a roll control autopilot. In the diagram, the $1/(s+1)$ block represents the “plant” transfer function from the aileron deflection (δ_a) to roll rate ($\dot{\phi}$). As we’ll see, a roll control autopilot might be designed using a roll rate feedback gain (K_r) and a proportional gain on the roll error (K_p).**



For checking:
 $K_r * K_p = 12.5$

- Determine the closed-loop transfer function for the roll control autopilot shown. (Hint: it should look like a 2nd order model)
- For the autopilot to control roll in a stable manner, an autopilot designer must appropriately choose the values K_r and K_p . Determine the gains K_r and K_p to achieve a desired response of: $\omega_n = 2.5$ and $\zeta = 0.6$
- Using “`s=tf('s')`” in Matlab, make the closed loop transfer function using your chosen gains.
- Use the “`step`” function to plot the resulting step response from roll command to achieved roll.

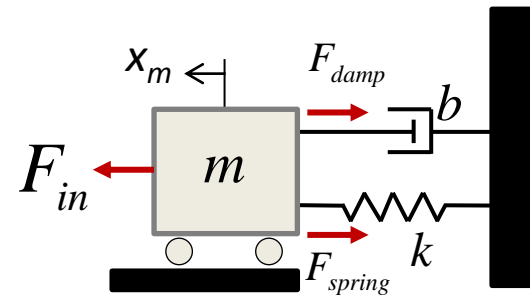
Lecture 1 Homework, Page 3 of 7

3) Consider a spring-damper system with the following values:

$$m = 2 \text{ kg}$$

$$k = 12 \frac{\text{N}}{\text{m}}$$

$$b = 1 \frac{\text{N}}{\text{m/s}}$$



x_m : Mass Pos.

m : mass

$$F_{damp} = b\dot{x}_m$$

$$F_{spring} = kx_m$$

F_{in} = Input Force

- Determine the transfer function (from input force to mass position), and use it to determine the natural frequency and damping. What should be the steady-state position given a unit step force input?
- Using `s=tf('s')` and the `step()` function, plot the input-output step response from input force to mass position. Make sure your step response characteristics match your answers to (a).
- What would be the steady-state mass position, given a constant input force of 10 N (for $t \geq 0$).
- Consider a changeable damper. What value of “b” would result in a damping of 0.7? What value of “b” would result in a critically damped system? Compare both step responses with the step response from the original $b=1 \text{ N/(m/s)}$ system.
- Using the symbols m , b & k , model the 2-state spring damper system as a state-space model with the states of position and velocity (i.e. determine the matrices A , B , C & D as functions of the symbols m , b & k).

$$\begin{bmatrix} \dot{x}_m \\ \ddot{x}_m \end{bmatrix} = \begin{bmatrix} ? & ? \\ ? & ? \end{bmatrix} \begin{bmatrix} x_m \\ \dot{x}_m \end{bmatrix} + \begin{bmatrix} ? \\ ? \end{bmatrix} F_{in}$$

$$\text{mass position} = \begin{bmatrix} ? & ? \end{bmatrix} \begin{bmatrix} x_m \\ \dot{x}_m \end{bmatrix} + \begin{bmatrix} ? \end{bmatrix} F_{in}$$

Lecture 1 Homework, Page 4 of 7

4) Throughout the semester we'll be developing a UAV 6DOF in Simulink/Matlab. The Simulink architecture of the 6DOF will be provided to you, along with “stubbed” Matlab functions. To get started with our simulation, we need some conversion utilities. Using the provided function stubs, create the following matlab functions:

Hint: Use 3 separate matrices:

`R_ned2b = eulerToRotationMatix(phi_rad, theta_rad, psi_rad)`

Make a 3x3 rotation matrix from ned to body coordinates

Assume order: 1) psi_rad about z, 2) theta_rad about y, 3) phi_rad about x

$$R_{NED}^b = R_x(\phi)R_y(\theta)R_z(\psi)$$

`[phi_rad theta_rad psi_rad] = rotationMatrixToEuler(R_ned2b)`

Assume same order as above

Handle theta_rad = $\pm\pi/2$ case ($\pm 90^\circ$)

`[Va alpha_rad beta_rad] = makeVaAlphaBeta(v_rel_b)`

$v_rel_b = \underline{v}_a^b$: wind-relative airspeed vector in body coords

Handle Va=0 case (i.e. set alpha_rad and beta_rad to zero)

**Remember to use
atan2(num,den) !!!**

`[Vg gamma_rad course_rad] = makeVgGammaCourse(v_ned)`

v_ned : groundspeed vector in NED coords

Handle Vg=0 case (i.e. set gamma_rad and course_rad to zero)

(You don't need to turn in the code. See next slide for test cases to submit.)

Lecture 1 Homework, Page 5 of 7

4) Continued ... Test cases to submit:

a) Compute R_{ned2b} for:

$$\begin{aligned}\phi_{rad} &= 14 \cdot \pi / 180; & (14 \text{ degrees}) \\ \theta_{rad} &= -30 \cdot \pi / 180; & (-30 \text{ degrees}) \\ \psi_{rad} &= 160 \cdot \pi / 180; & (160 \text{ degrees})\end{aligned}$$

b) Compute Euler angles from the R_{ned2b} computed in (a). (The function computes angles in radians, but provide your answers in degrees. If you didn't get the original values back, you made a mistake.)

c) Extract V_a , α and β from $\underline{v}_a^b = [20 \ -2 \ 3]^T$. Routine generates angles in radians, but provide your answers in degrees. (For checking: $\beta = -5.6478^\circ$)

d) Extract V_g , γ and course from $\underline{v}_g^{ned} = [-15 \ -12 \ 2]^T$. Routine generates angles in radians, but provide your answers in degrees. If this is the velocity vector for a UAV, is it climbing or descending? Approximately what direction is the vehicle heading (N, NE, E, SE, S, SW, W, NW)?

e) A UAV is flying with an attitude of $\phi=12^\circ$, $\theta=2^\circ$ and $\psi=-140^\circ$, and a ground-relative velocity vector of $\underline{v}_g^{ned} = [-15 \ -12 \ 2]^T$ m/s. The local wind vector is $\underline{v}_w^{ned} = [3 \ 4 \ 0]^T$ m/s. Generate \underline{v}_a^b . Then, generate V_a , α and β . (Express angles in degrees). (For checking: $\beta = 2.9929^\circ$)

Lecture 1 Homework, Page 6 of 7

5) Now that you have some necessary working routines, you should be able to run the Simulink simulation, `uavsim.mdl`. To verify, do the following:

- a) The simulation is loaded via the routine “`load_uavsim`”. Run `load_uavsim` and verify that the sim opened and was prepared without error. (The Simulink model should open.) (Yes/No)**
- b) The workspace should contain a structure “P” which contains all the parameters necessary to run the simulation. In addition to opening the Simulink model, “`load_uavsim`” calls “`init_uavsim_params`”, which creates the “P” structure. Likely, none of these parameters make sense to you yet, but they will! Explore “`init_uavsim_params`” and answer the following questions, including units:**
 - What is the vehicle mass?**
 - What is the maximum allowable roll command? (in degrees)**
 - What is the nominal airspeed?**
- c) Run the Simulink model by pressing the “play” button (▶). The number next to the “play” button shows how long the simulation will run. You should see a UAV displayed, but it won’t be moving and it will have zero roll and zero pitch. But, the displayed time on the figure should be updating. Did it run without errors? What is the vehicle altitude in meters?**
- d) The “P” structure contains parameters and initializations, including the initial attitude angles, `P.phi0`, `P.theta0`, and `P.psi0` (all in radians). If you coded the Euler/RotationMatrix functions correctly, you should be able to change these values and see a difference in the visualization. In the Matlab command console, type “`P.phi0 = 90*pi/180;`”, then re-play the Simulink model. Did the visualization change appropriately?**

Lecture 1 Homework, Page 7 of 7

6) For each of the angles below, state:

Variable used to represent quantity (Greek letter and the English spelling of the Greek letter. E.g. χ and chi, or α and alpha)

Description of positive direction (e.g. +EastOfNorth)

Turn in a table similar to:

<i>Angle</i>	<i>Greek Symbol (e.g. α)</i>	<i>English Spelling (e.g. alpha)</i>	<i>Describe of Positive Direction</i>
Roll			
Pitch			+: Nose Up
Yaw			
Angle-of-Attack			
Sideslip			
Course			
Vert. flight path angle			
Crab angle			

Recommended Reading: Beard & McLain Chapter 2

Book Errata:

- Page 19, just above Eq. 2.7: In the wind frame to body frame rotation matrix, the (3,2) and (2,3) entries in the matrix should be swapped. The (2,3) entry should be 0 and the (3,2) entry should be $-\sin(\beta)\sin(\alpha)$.

- Equation (2.9) assumes that the side slip angle $\beta=0$. Otherwise, replace ψ with $\psi+\beta$.