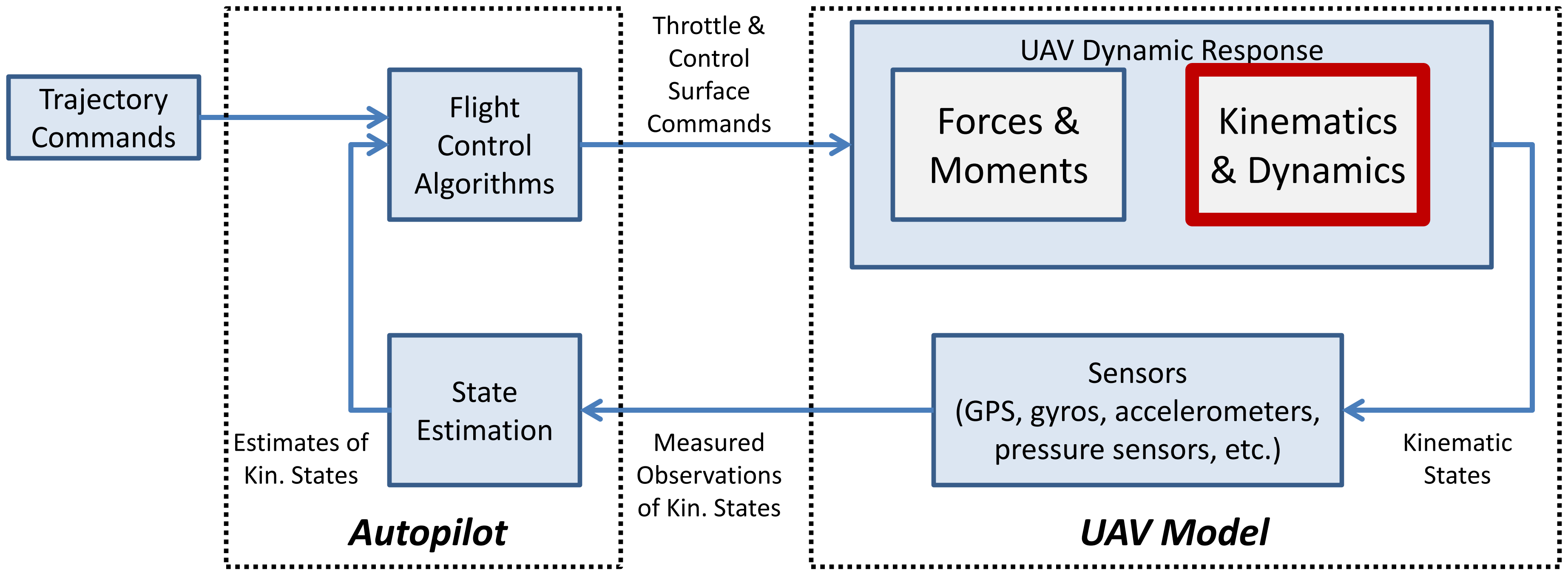
UAV Systems & Control
Lecture 3

Kinematics (Translational & Rotational)

Dynamics (Translational & Rotational)

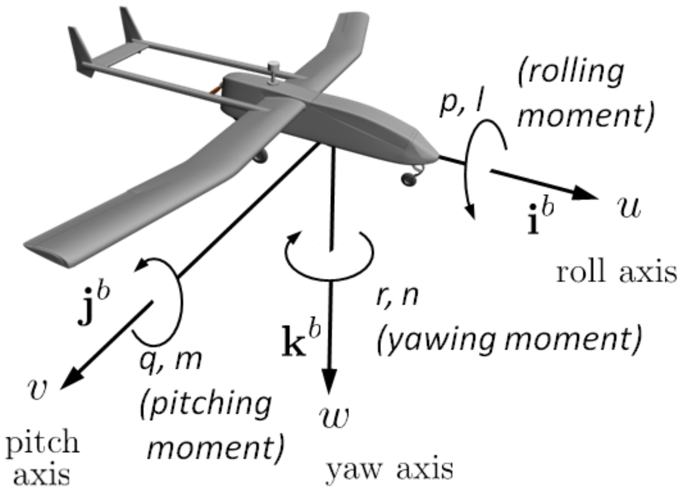
Equations of Motion (Euler & Quaternion)

UAV System


Trajectory Commands
Flight Control Algorithms
Throttle & Control Surface Commands
UAV Dynamic Response
Forces & Moments
Kinematics & Dynamics
State Estimation
Estimates of Kin. States
Measured Observations of Kin. States
Sensors (GPS, gyros, accelerometers, pressure sensors, etc.)
Kinematic States
Autopilot
UAV Model

- In this section we will develop the *kinematics* and *dynamics* equations that govern the motion of an aerodynamic vehicle
  - **Kinematics**: The relationships between translational and rotational positions, velocities and accelerations
  - **Dynamics**: The effects that external forces and moments have on motion
- Note:
  - The book jumps straight into simplifications specific to nominal fixed-wing vehicles
  - We will keep derivations general for as long as possible

Aircraft Variables



12 State Variables

Name	Description	Metric Units
p_n	Inertial North position of MAV expressed along $\mathbf{i}^i$ in $\mathcal{F}^i$.	m
p_e	Inertial East position of MAV expressed along $\mathbf{j}^i$ in $\mathcal{F}^i$.	m
p_d	Inertial Down position of MAV expressed along $\mathbf{k}^i$ in $\mathcal{F}^i$.	m
u	Ground velocity expressed along $\mathbf{i}^b$ in $\mathcal{F}^b$.	m/s
v	Ground velocity expressed along $\mathbf{j}^b$ in $\mathcal{F}^b$.	m/s
w	Ground velocity expressed along $\mathbf{k}^b$ in $\mathcal{F}^b$.	m/s
ϕ	Roll angle defined with respect to $\mathcal{F}^{v2}$.	rad
θ	Pitch angle defined with respect to $\mathcal{F}^{v1}$.	rad
ψ	Heading (yaw) angle defined with respect to $\mathcal{F}^v$.	rad
p	Body angular (roll) rate expressed along $\mathbf{i}^b$ in $\mathcal{F}^b$.	rad/s
q	Body angular (pitch) rate expressed along $\mathbf{j}^b$ in $\mathcal{F}^b$.	rad/s
r	Body angular (yaw) rate expressed along $\mathbf{k}^b$ in $\mathcal{F}^b$.	rad/s

Other Variables

mass	Vehicle mass, assumed constant	kg
J	3x3 Inertia matrix (Common simplifying assumption: $J_{xy}=J_{yz}=0$)	kg-m^2
f_x	Axial force along x-axis (e.g. majority of thrust and drag components)	N
f_y	Lateral force along y-axis (e.g. sideslip-induced force)	N
f_z	Normal force along z-axis (e.g. majority of lift and gravity compnts.)	N
l	Rolling moment, about x-axis	N-m
m	Pitching moment, about y-axis	N-m
n	Yawing moment, about z-axis	N-m

Vector relationships:

$$\underline{v}_g^b = u\,\hat{i}^b + v\,\hat{j}^b + w\,\hat{k}^b$$

$$\underline{\omega}_{b/i}^b = p\,\hat{i}^b + q\,\hat{j}^b + r\,\hat{k}^b$$

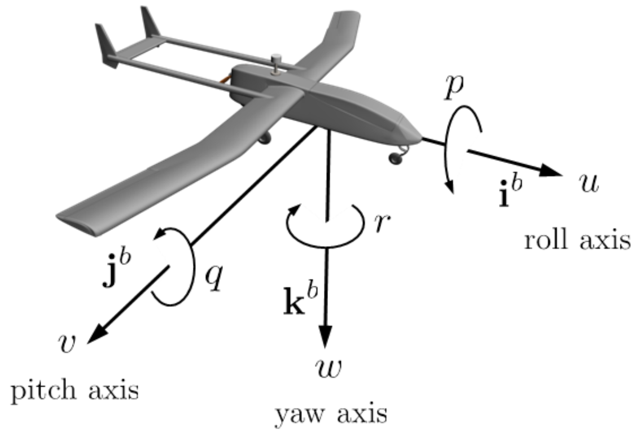
$$\underline{\mathrm{f}}^b = f_x\hat{i}^b + f_y\hat{j}^b + f_z\hat{k}^b$$

$$\underline{\mathrm{m}}^b = l\,\hat{i}^b + m\,\hat{j}^b + n\,\hat{k}^b$$

Portions of slide courtesy of book website: Beard & McLain, “Small Unmanned Aircraft”, *Princeton University Press*, 2012

Translational Kinematics

- Translational kinematics deals with the time-propagation of the inertial position state, $[p_n \; p_e \; p_d]^T$.



- The time-derivative of the inertial position is the inertial velocity:

$$\frac{d}{dt_i}\begin{bmatrix} p_n \\ p_e \\ p_d \end{bmatrix} = \begin{bmatrix} \dot{p}_n \\ \dot{p}_e \\ \dot{p}_d \end{bmatrix} = \begin{bmatrix} v_n \\ v_e \\ v_d \end{bmatrix} = \underline{v}_g^{ned}$$

$\underline{v}_g^{ned}$: Velocity vector wrt ground (i.e. inertial NED) expressed in the NED frame

Note: There is no rotation-rate (Coriolis) component in this differentiation because everything is wrt the same frame

- As we'll see, forces acting on an air vehicle will be easiest to express in body coordinates. So, we want to relate the time-derivative of the inertial positions to the inertial velocity vector *expressed in the BODY frame*:

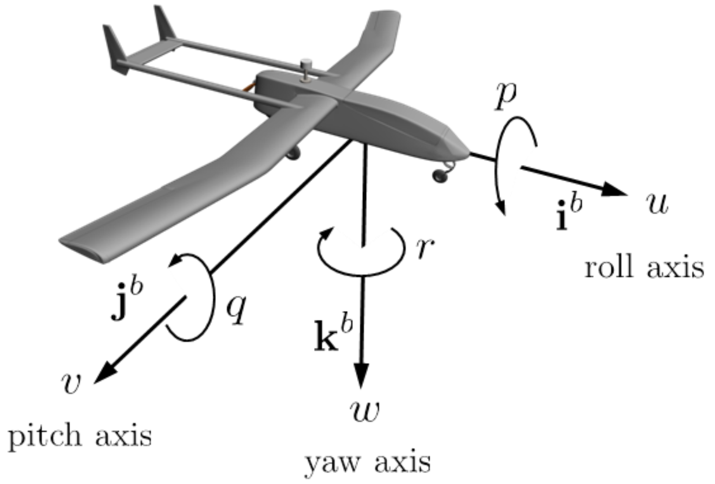
$$\begin{bmatrix} \dot{p}_n \\ \dot{p}_e \\ \dot{p}_d \end{bmatrix} = \underline{v}_g^{ned} = R_b^{ned}\,\underline{v}_g^{b} = R_b^{ned}\begin{bmatrix} u \\ v \\ w \end{bmatrix}$$

This expression tells us how to propagate the vehicle position forward in time based on its velocity

$$\text{Note:}\; R_b^{ned} = \left(R_{ned}^{b}\right)^T$$

Rotational Kinematics

- Rotational kinematics deals with the time-propagation of the body orientation with respect to the inertial (e.g. NED) frame:
  - Euler Angles: roll (ϕ), pitch (θ), yaw (ψ)



- The time-derivative of roll, pitch and yaw are the Euler angle rates:

$$\frac{d}{dt_i}\begin{bmatrix}\phi\\ \theta\\ \psi\end{bmatrix}=\begin{bmatrix}\dot{\phi}\\ \dot{\theta}\\ \dot{\psi}\end{bmatrix}$$

- But, as we'll see, gyros measure the body rates (p, q and r) which are the angular rates about each of the body axes.
  Important: *Euler angle rates are NOT equal to body rates:*

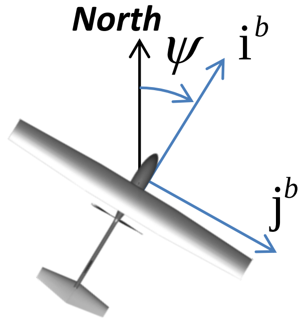
$$\begin{bmatrix}\dot{\phi}\\ \dot{\theta}\\ \dot{\psi}\end{bmatrix}\neq\begin{bmatrix}p\\ q\\ r\end{bmatrix}$$

Thought Experiment:
Consider a vehicle pitched-up at 45 degrees.
Is the yaw rate (rate-of-change of nose axis relative to ground) the same as *r*, the rotation rate about the **k^b** (body z) axis?

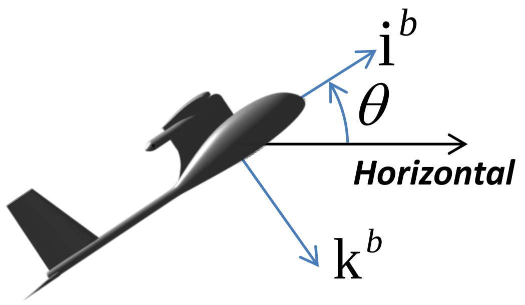
- So, we need to relate the body rates with the Euler angle rates.

Rotational Kinematics

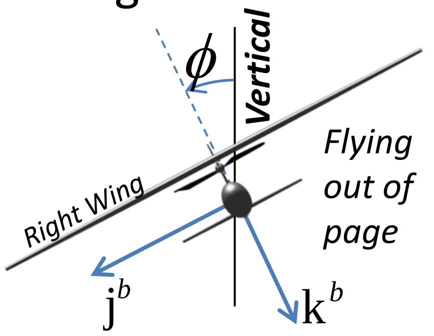
- Recall the yaw (ψ), pitch (θ) and roll (ϕ) <u>ordered</u> Euler angle rotations:



1) Yaw (ψ) about NED z (z is in down direction)



2) Pitch (θ) about resulting y (y is NOT body y or NED y)



3) Roll (ϕ) about resulting x (x is coincident with body-x)

- Because gyros measure rates about each body axis (Body Rates: p, q, r) we need to relate Euler angle rates with body rates

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \underbrace{\begin{bmatrix} \dot\phi \\ 0 \\ 0 \end{bmatrix}}_{\text{Roll-rate expressed in body frame}} + \underbrace{R_x(\phi)\begin{bmatrix} 0 \\ \dot\theta \\ 0 \end{bmatrix}}_{\text{Pitch-rate expressed in body frame}} + \underbrace{R_x(\phi)R_y(\theta)\begin{bmatrix} 0 \\ 0 \\ \dot\psi \end{bmatrix}}_{\text{Yaw-rate expressed in body frame}} = \begin{bmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\phi & \sin\phi\cos\theta \\ 0 & -\sin\phi & \cos\phi\cos\theta \end{bmatrix} \begin{bmatrix} \dot\phi \\ \dot\theta \\ \dot\psi \end{bmatrix}$$

Thus, by inverting $\Rightarrow$

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

This expression tells us how to propagate the vehicle orientation forward in time based on its body rates

State Equations

- 6 of the 12 state equations that govern vehicle motion are provided by the kinematics relating translational and rotational positions and velocities:

6 equations governing translational motion

$$\begin{bmatrix}\dot{p}_n\\ \dot{p}_e\\ \dot{p}_d\end{bmatrix} = R_b^{ned}\underline{v}_g^b = \begin{bmatrix}\cos\theta\cos\psi & \sin\phi\sin\theta\cos\psi-\cos\phi\sin\psi & \cos\phi\sin\theta\cos\psi+\sin\phi\sin\psi\\ \cos\theta\sin\psi & \sin\phi\sin\theta\sin\psi+\cos\phi\cos\psi & \cos\phi\sin\theta\sin\psi-\sin\phi\cos\psi\\ -\sin\theta & \sin\phi\cos\theta & \cos\phi\cos\theta\end{bmatrix}\begin{bmatrix}u\\ v\\ w\end{bmatrix}$$

$$\begin{bmatrix}\dot{u}\\ \dot{v}\\ \dot{w}\end{bmatrix} = \begin{bmatrix}?\\ ?\\ ?\end{bmatrix}$$

Translational Dynamics: How forces affect velocity

6 equations governing rotational motion

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

$$\begin{bmatrix}\dot{p}\\ \dot{q}\\ \dot{r}\end{bmatrix} = \begin{bmatrix}?\\ ?\\ ?\end{bmatrix}$$

Rotational Dynamics: How moments affect angular rates

- The remaining equations will come from an application of Newton's 2nd Law to the translational and rotational motion of the vehicle
  - *i.e. The dynamic effects that external forces and moments have on vehicle motion*

Translational Dynamics

- Assuming constant mass, Newton's 2nd Law states:

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

- $\underline{\mathbf{f}}^{\,i}$: 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

- For our purposes, we assume "ground" to be an inertial reference, so we'll use velocity relative to ground:

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

- $\underline{v}^{\,i}_{\,g}$: Velocity relative to ground, expressed in inertial frame

- For convenience, express those <u>same</u> vectors in body frame:

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

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

i.e. Premultiply both sides of previous expression by: R_i^b (for us, "i"="ned")

- From vector differentiation we can express the <u>*inertial-frame-viewed*</u> rate-of-change of $\underline{v}^b_g$ as a function of the <u>*body-frame-viewed*</u> rate-of-change of $\underline{v}^b_g$:

$$\underline{f}^{\,b} = (mass)\left(\frac{d}{dt_b}\underline{v}_g^{\,b} + \underline{\omega}_{b/i}^{\,b}\times\underline{v}_g^{\,b}\right)$$

- 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)

Translational Dynamics

- From Newton's 2nd Law with constant mass and inertial ground, we derived:

$$\underline{f}^b = (mass)\left(\frac{d}{dt_b}\underline{v}_g^b + \underline{\omega}_{b/i}^b \times \underline{v}_g^b\right)$$

where (with common aero notation):

$$\underline{f}^b = f_x\hat{i}^b + f_y\hat{j}^b + f_z\hat{k}^b \qquad \underline{v}_g^b = u\,\hat{i}^b + v\,\hat{j}^b + w\,\hat{k}^b \qquad \underline{\omega}_{b/i}^b = p\,\hat{i}^b + q\,\hat{j}^b + r\,\hat{k}^b$$

- **$\underline{f}^{\underline{b}}$** : Sum of all external forces, expressed in a body frame
- *mass*: mass of the vehicle (assumed constant)
- $\underline{v}^b_g$: Velocity relative to ground, 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)

- Since $[u, v, w]^T$ is the instantaneous projection of the inertial velocity vector onto the body coordinate frame, then the time-derivative of the inertial velocity as viewed in the body frame is: $\frac{d}{dt_b}\underline{v}_g^b = \dot{u}\,\hat{i}^b + \dot{v}\,\hat{j}^b + \dot{w}\,\hat{k}^b$

- Thus: converting Newton's Law from vectors to scalars:

$$f_x = mass \cdot (\dot{u} + qw - rv)$$
$$f_y = mass \cdot (\dot{v} + ru - pw)$$
$$f_z = mass \cdot (\dot{w} + pv - qu)$$

Note:

$$\underline{\omega}_{b/i}^b \times \underline{v}_g^b = \begin{vmatrix} \hat{i}_b & \hat{j}_b & \hat{k}_b \\ p & q & r \\ u & v & w \end{vmatrix} = \begin{bmatrix} qw - rv \\ ru - pw \\ pv - qu \end{bmatrix}$$

Translational Dynamics

- Newton's 2nd Law for a body coordinate frame (constant mass & inertial ground):

$$\underline{\mathrm{f}}^{b} = (mass)\left(\frac{d}{dt_b}\underline{v}_g^b + \underline{\omega}_{b/i}^b \times \underline{v}_g^b\right) \Longrightarrow \begin{aligned} f_x &= mass\cdot\left(\dot{u} + qw - rv\right) \\ f_y &= mass\cdot\left(\dot{v} + ru - pw\right) \\ f_z &= mass\cdot\left(\dot{w} + pv - qu\right) \end{aligned}$$

- Thus, given the forces acting on the body expressed in body coordinates (f_x, f_y, f_z), the vehicle mass, the inertial rotation rate resolved along body coordinates (*p, q, r*), and the inertial velocity resolved along body coordinates (*u, v, w*), **we can solve for the rate-of-change of the inertial velocity (i.e. acceleration) expressed in the body frame**:

Translational Dynamics in body frame:

$$\frac{d}{dt_b}\underline{v}_g^b = -\underline{\omega}_{b/i}^b \times \underline{v}_g^b + \tfrac{1}{mass}\underline{\mathrm{f}}^{b} \Longrightarrow \begin{aligned} \dot{u} &= rv - qw + \tfrac{1}{mass} f_x \\ \dot{v} &= pw - ru + \tfrac{1}{mass} f_y \\ \dot{w} &= qu - pv + \tfrac{1}{mass} f_z \end{aligned}$$

This expression tells us how to propagate the vehicle velocity vector forward in time as the vehicle is being influenced by external forces (e.g. aero, thrust and gravity)

Rotational Dynamics

- From Newton's 2nd Law (f=mass x acceleration), we can see that mass is an object's resistance to change. The more mass, the less effect a force will have on changing an object's velocity vector, or its translational motion.
- As we'll see, the object's mass as it is distributed throughout the object can also resist angular motion driven by a *torque* or *moment*

- Newton's 2nd Law for rotational motion expressed in body coordinates is:

$$\underline{\mathrm{m}}^b = \frac{d}{dt_i}\underline{\mathrm{h}}_i^b$$

- $\underline{\mathbf{m}}^{b}$: Sum of all external moments, expressed in a body frame
- $\underline{h}^{b}_{i}$: Angular momentum due to rotations relative to inertial, expressed in body frame
- d/dt_i : Time derivative taken wrt an inertial frame

- We'll use Newton's 2nd Law for rotational motion to derive how external moments influence rotational accelerations, or changes in the body rates (*p*,*q*,*r*)

- Method: $\underline{\mathrm{m}}^b = \dfrac{d}{dt_i}\underline{\mathrm{h}}_i^b \Rightarrow \underline{\mathrm{h}}_i^b \equiv \mathrm{J}\ \underline{\omega}_{b/i}^b \Rightarrow \underline{\dot{\omega}}_{b/i}^b = \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \mathrm{J}^{-1}\left[-\underline{\omega}_{b/i}^b \times \left(\mathrm{J}\,\underline{\omega}_{b/i}^b\right) + \underline{\mathrm{m}}^b\right]$

Torque is the deriv. of angular momentum

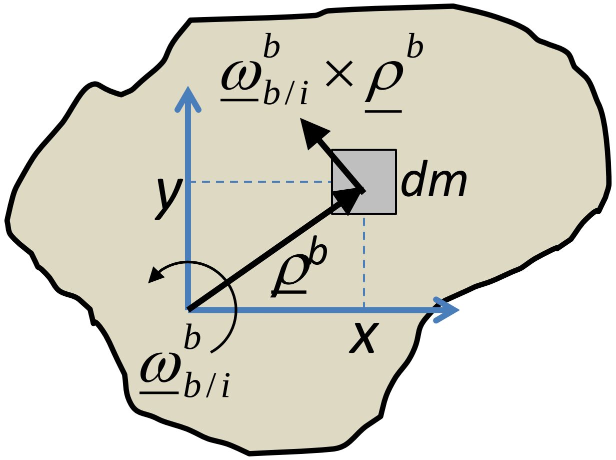
Angular momentum is the <u>inertia matrix</u> times the body rates

Angular accelerations acquired by applying vector differentiation and inverting

Rotational Dynamics

- Linear momentum is (mass)-times-(velocity).
- Angular momentum of a particle with mass *dm* orbiting about an axis of rotation is the cross product of the lever arm from the axis of rotation with the linear momentum:

$$\text{Angular momentum of a particle with mass } dm = \underbrace{\underline{\rho}^b}_{\text{Lever Arm from c.g.}} \times \underbrace{\underbrace{\left(\underline{\omega}^b_{b/i} \times \underline{\rho}^b\right)}_{\text{Tangential velocity}} dm}_{\text{Linear Momentum}}$$



$\underline{\omega}^b_{b/i} \times \underline{\rho}^b$
dm
y
$\underline{\rho}^b$
x
$\underline{\omega}^b_{b/i}$

- Then, the total angular momentum <u>h</u> due to inertial rotations about a body's center-of-gravity is the sum of the contributions of all mass particles in the body:

$$\underline{\mathrm{h}}^b_i = \int \underline{\rho}^b \times \left(\underline{\omega}^b_{b/i} \times \underline{\rho}^b\right) dm$$

Exercise: How does $(\underline{\omega}^b_{b/i} \times \underline{\rho}^b)$ make a linear velocity? What direction is it in?

Rotational Dynamics

- Angular momentum <u>h</u> due to inertial rotations about a body's center-of-gravity is defined as:

$$\underline{\mathrm{h}}_i^b = \int \underline{\rho}^b \times \left(\underline{\omega}_{b/i}^b \times \underline{\rho}^b\right) dm$$

Where:

$$\underline{\omega}_{b/i}^b = p\,\hat{i}^b + q\,\hat{j}^b + r\,\hat{k}^b$$

$$\underline{\rho}^b = x\,\hat{i}^b + y\,\hat{j}^b + z\,\hat{k}^b$$

$\underline{\rho}^b$: Lever arm from c.g. to mass element (c.g. is not necessarily the center-of-rotation)

- Expanding:

$$\underline{\omega}_{b/i}^b \times \underline{\rho}^b = \hat{i}^b(qz - ry) + \hat{j}^b(rx - pz) + \hat{k}^b(py - qx)$$

$$\underline{\rho}^b \times \left(\underline{\omega}_{b/i}^b \times \underline{\rho}^b\right) = \begin{aligned}&\hat{i}^b\left[p(y^2+z^2) - xyq - xzr\right]\\ &+\hat{j}^b\left[q(x^2+z^2) - yzr - xyp\right]\\ &+\hat{k}^b\left[r(x^2+y^2) - xzp - yzq\right]\end{aligned}$$

$$\underline{\mathrm{h}}_i^b = \int \underline{\rho}^b \times \left(\underline{\omega}_{b/i}^b \times \underline{\rho}^b\right) dm$$

$$= \begin{aligned}&\hat{i}^b\left[p\int(y^2+z^2)dm - q\int(xy)dm - r\int(xz)dm\right]\\ &+\hat{j}^b\left[q\int(x^2+z^2)dm - r\int(yz)dm - p\int(xy)dm\right]\\ &+\hat{k}^b\left[r\int(x^2+y^2)dm - p\int(xz)dm - q\int(yz)dm\right]\end{aligned} = \begin{aligned}&\hat{i}^b\left[pJ_x - qJ_{xy} - rJ_{xz}\right]\\ &+\hat{j}^b\left[qJ_y - rJ_{yz} - pJ_{xy}\right]\\ &+\hat{k}^b\left[rJ_z - pJ_{xz} - qJ_{yz}\right]\end{aligned} = \begin{bmatrix} J_x & -J_{xy} & -J_{xz} \\ -J_{yx} & J_y & -J_{yz} \\ -J_{xz} & -J_{yz} & J_z \end{bmatrix}\begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

Rotational Dynamics

- Angular momentum $\underline{h}$ due to inertial rotations about a body's center-of-gravity is defined as:

$$\underline{\mathrm{h}}_i^b \equiv \underbrace{\mathrm{J}}_{} \; \underline{\omega}_{b/i}^b = \underbrace{\begin{bmatrix} J_x & -J_{xy} & -J_{xz} \\ -J_{xy} & J_y & -J_{yz} \\ -J_{xz} & -J_{yz} & J_z \end{bmatrix}}_{\text{Inertia Matrix or Inertia Tensor}} \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

- J is the Inertia Matrix
  - J is related to mass and its distribution throughout the object
  - J is symmetric about the diagonal
  - The diagonal matrix elements (J_x, J_y, J_z) are called "*moments of inertia*" and are measures of an object's tendency to oppose angular acceleration about each axis
    - e.g. the larger J_x is, the more an object opposes an angular acceleration about the x axis.
    - J_x, J_y, J_z are always non-negative
  - The off-diagonal elements are the "*products of inertia*"
    - Products of inertia relate how a torque or moment about one axis can *cross-couple* into (or induce) an angular acceleration in another axis
    - Products of inertia can be positive, negative or zero
    - Axes of symmetry and planes of symmetry will cause some off-diagonals to be zero
      - E.g. aircraft are often symmetric about the x-z plane. So, often J_{xy}=J_{yz}=0

For our aircraft:
J_x: Rolling Moment-of-Inertia
J_y: Pitching MOI
J_z: Yawing MOI

Inertia Matrix

- Moments of inertia are measures of the aircraft's tendency to oppose angular acceleration about a specific axis of rotation
- The 3x3 inertia matrix is derived as:

$$J = \begin{bmatrix} \int (y^2+z^2)dm & -\int (xy)dm & -\int (xz)dm \\ -\int (xy)dm & \int (x^2+z^2)dm & -\int (yz)dm \\ -\int (xz)dm & -\int (yz)dm & \int (x^2+y^2)dm \end{bmatrix} = \begin{bmatrix} J_x & -J_{xy} & -J_{xz} \\ -J_{xy} & J_y & -J_{yz} \\ -J_{xz} & -J_{yz} & J_z \end{bmatrix}$$

- In practice, J is either:
  - Measured experimentally (e.g. via oscillations on a bifilar pendulum)
  - Numerically computed via CAD
  - Approximated using discrete mass components

$$J_x = \int (y^2+z^2)dm \quad = \sum_k \left(y_k^2 + z_k^2\right) m_k$$

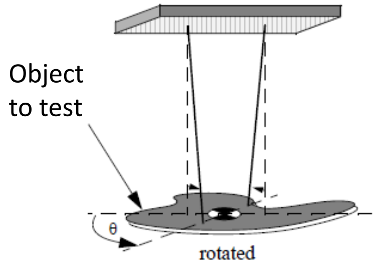
$$J_y = \int (x^2+z^2)dm \quad = \sum_k \left(x_k^2 + z_k^2\right) m_k$$

$$J_z = \int (x^2+y^2)dm \quad = \sum_k \left(x_k^2 + y_k^2\right) m_k$$

$$J_{xy} = \int (xy)dm \quad = \sum_k x_k y_k m_k$$

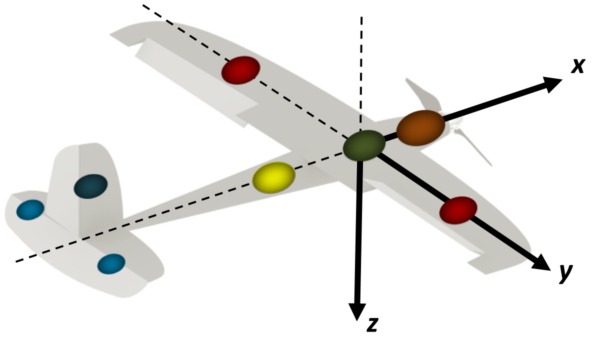
$$J_{xz} = \int (xz)dm \quad = \sum_k x_k z_k m_k$$

$$J_{yz} = \int (yz)dm \quad = \sum_k y_k z_k m_k$$



Bifilar Pendulum

Jardin & Mueller, "Optimized Measurements of a UAV Mass Moment of Inertia with a Bifilar Pendulum"



Inertia Matrix Example

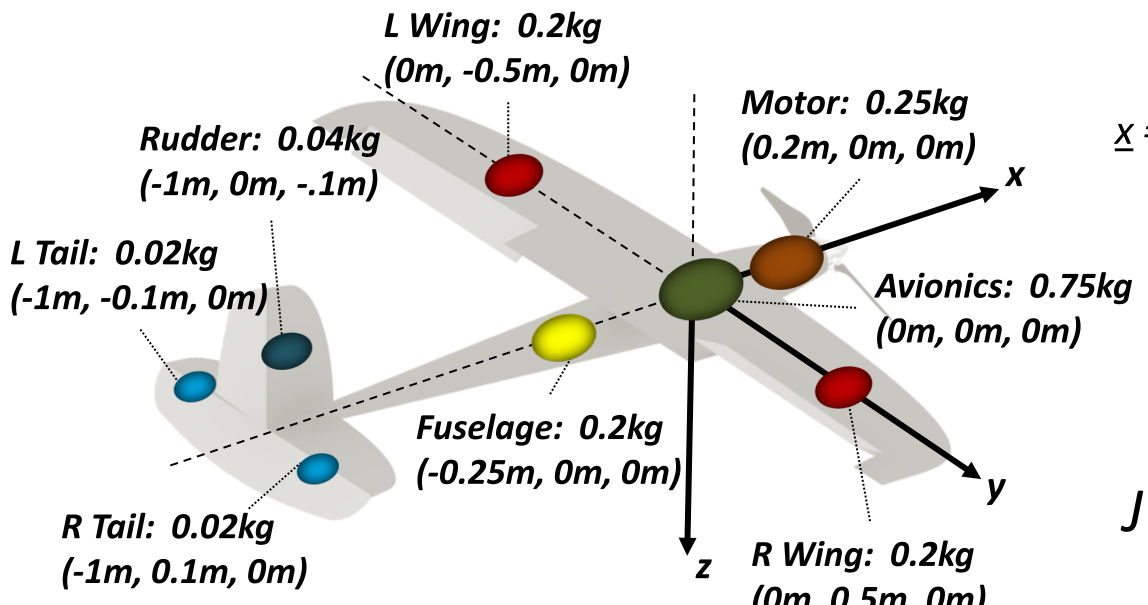
- Inertia Matrix, J, can be approximated using discrete lumped masses:

$$J = \begin{bmatrix} J_x & -J_{xy} & -J_{xz} \\ -J_{xy} & J_y & -J_{yz} \\ -J_{xz} & -J_{yz} & J_z \end{bmatrix}$$

$$J_x = \sum_k \left(y_k^2 + z_k^2\right) m_k \qquad J_{xy} = \sum_k x_k y_k m_k$$

$$J_y = \sum_k \left(x_k^2 + z_k^2\right) m_k \qquad J_{yz} = \sum_k y_k z_k m_k$$

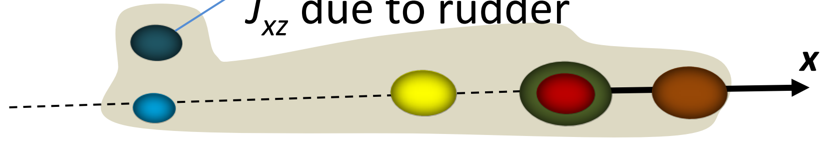
$$J_z = \sum_k \left(x_k^2 + y_k^2\right) m_k \qquad J_{xz} = \sum_k x_k z_k m_k$$



L Wing: 0.2kg (0m, -0.5m, 0m)
Rudder: 0.04kg (-1m, 0m, -.1m)
Motor: 0.25kg (0.2m, 0m, 0m)
x
L Tail: 0.02kg (-1m, -0.1m, 0m)
Avionics: 0.75kg (0m, 0m, 0m)
Fuselage: 0.2kg (-0.25m, 0m, 0m)
y
R Tail: 0.02kg (-1m, 0.1m, 0m)
z
R Wing: 0.2kg (0m, 0.5m, 0m)

$$\underline{x} = \begin{bmatrix} 0.2 \\ 0.0 \\ -0.3 \\ 0.0 \\ 0.0 \\ -1.0 \\ -1.0 \\ -1.0 \end{bmatrix} \quad \underline{y} = \begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \\ 0.5 \\ -0.5 \\ 0.1 \\ -0.1 \\ 0.0 \end{bmatrix} \quad \underline{z} = \begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ -0.1 \end{bmatrix} \quad \underline{m} = \begin{array}{c|l} \begin{bmatrix} 0.25 \\ 0.75 \\ 0.20 \\ 0.20 \\ 0.20 \\ 0.02 \\ 0.02 \\ 0.04 \end{bmatrix} & \begin{array}{l} \textit{Mot} \\ \textit{Avion} \\ \textit{Fus} \\ \textit{RWng} \\ \textit{LWng} \\ \textit{RTail} \\ \textit{LTail} \\ \textit{Rud} \end{array} \end{array}$$

$$J = \begin{bmatrix} 0.1008 & 0 & -0.0040 \\ 0 & 0.1029 & 0 \\ -0.0040 & 0 & 0.2029 \end{bmatrix} \textit{kg-m}^2$$



J_{xz} due to rudder

Matlab: `Jx = sum((y.^2+z.^2).*m);`

Rotational Dynamics

- Newton's 2nd Law for rotational motion expressed in body coordinates is:

$$\underline{\mathrm{m}}^{b} = \frac{d}{dt_i}\underline{\mathrm{h}}_i^{b}$$

$$= \frac{d}{dt_i}\left(\mathrm{J}\,\underline{\omega}_{b/i}^{b}\right)$$

  - **$\underline{\mathbf{m}}^{b}$** : Sum of all external moments, expressed in a body frame
  - $\underline{h}^{b}_{i}$: Angular momentum due to rotations relative to inertial, expressed in body frame. Definition: $\underline{\mathrm{h}}^{b}_{i} = \mathrm{J}\,\underline{\omega}^{b}_{b/i}$
  - d/dt_i : Time derivative taken wrt an inertial frame
  - J : 3x3 Inertia Matrix
  - $\underline{\omega}^{b}_{b/i}$: Rate of change of body frame wrt inertial (*b/i*) expressed in body frame (superscript b)

- From vector differentiation we can express the ***inertial-frame-viewed*** rate-of-change of $\underline{\mathrm{h}}^{b}_{i}$ as a function of the ***body-frame-viewed*** rate-of-change of $\underline{\mathrm{h}}^{b}_{i}$:

$$\underline{\mathrm{m}}^{b} = \frac{d}{dt_i}\underline{\mathrm{h}}^{b} = \frac{d}{dt_b}\underline{\mathrm{h}}^{b} + \underline{\omega}_{b/i}^{b}\times\underline{\mathrm{h}}^{b}$$

$$= \frac{d}{dt_b}\left(\mathrm{J}\,\underline{\omega}_{b/i}^{b}\right) + \underline{\omega}_{b/i}^{b}\times\left(\mathrm{J}\,\underline{\omega}_{b/i}^{b}\right)$$

  - *d/dt_b* : Time derivative taken wrt a body frame

- Using the fact that J is constant in a rigid body with constant mass:

$$\underline{\mathrm{m}}^{b} = \mathrm{J}\,\frac{d}{dt_b}\left(\underline{\omega}_{b/i}^{b}\right) + \underline{\omega}_{b/i}^{b}\times\left(\mathrm{J}\,\underline{\omega}_{b/i}^{b}\right)$$

Rotational Dynamics

- From previous slide, Newton's 2nd Law for rotational motion expressed in body coordinates can be expressed as:

$$\underline{\mathrm{m}}^b = \mathrm{J}\,\frac{d}{dt_b}\left(\underline{\omega}^b_{b/i}\right) + \underline{\omega}^b_{b/i} \times \left(\mathrm{J}\,\underline{\omega}^b_{b/i}\right)$$

  - **$\underline{\mathbf{m}}$** ***^{b}*** : Sum of all external moments, expressed in a body frame
  - J : 3x3 Inertia Matrix
  - $\underline{\omega}^b_{b/i}$: Rate of change of body frame wrt inertial (*b/i*) expressed in body frame (superscript b)

- Since *[p, q, r]*^{T} are the instantaneous projections of $\underline{\omega}^b_{b/i}$ onto the body frame, it follows that the derivative of $\underline{\omega}^b_{b/i}$ as viewed from the body is:

$$\frac{d}{dt_b}\underline{\omega}^b_{b/i} = \underline{\dot{\omega}}^b_{b/i} = \dot{p}\,\hat{i}^b + \dot{q}\,\hat{j}^b + \dot{r}\,\hat{k}^b$$

- Thus, Newton's Law governing rotational motion dynamics can be expressed as:

$$\underline{\mathrm{m}}^b = \mathrm{J}\,\underline{\dot{\omega}}^b_{b/i} + \underline{\omega}^b_{b/i} \times \left(\mathrm{J}\,\underline{\omega}^b_{b/i}\right)$$

- Rearranging, we can determine the effect of external moments on body angular accelerations:

$$\underline{\dot{\omega}}^b_{b/i} = \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \mathrm{J}^{-1}\left[-\underline{\omega}^b_{b/i} \times \left(\mathrm{J}\,\underline{\omega}^b_{b/i}\right) + \underline{\mathrm{m}}^b\right]$$

Where:

$$\underline{\omega}^b_{b/i} = p\,\hat{i}^b + q\,\hat{j}^b + r\,\hat{k}^b$$

Rotational Dynamics

- Using Newton's 2nd Law for rotational motion, we determined the effect of external moments on body angular accelerations:

$$\underline{\dot{\omega}}^b_{b/i} = \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \mathrm{J}^{-1}\left[-\underline{\omega}^b_{b/i} \times \left(\mathrm{J}\,\underline{\omega}^b_{b/i}\right) + \underline{\mathrm{m}}^b\right]$$

- $\underline{\mathbf{m}}^{\boldsymbol{b}}$: Sum of all external moments, expressed in a body frame
- J : 3x3 Inertia Matrix
- $\underline{\omega}^b{}_{b/i}$: Rate of change of body frame wrt inertial expressed in body frame

- Expand using:
  - Skew-symmetric matrix form of the cross product:

$$\left[\underline{\omega}^b_{b/i} \times\right] = \begin{bmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix}$$

  - Common aerodynamic nomenclature for scalar moments (torques) about the body axes, $\underline{\mathbf{m}}^b = [l \;\; m \;\; n]^T$ (The lower case letters L, M and N.)

Rotational Dynamics in body frame:

$$\underline{\dot{\omega}}^b_{b/i} = \mathrm{J}^{-1}\left[-\underline{\omega}^b_{b/i} \times \left(\mathrm{J}\,\underline{\omega}^b_{b/i}\right) + \underline{\mathrm{m}}^b\right]$$

$$\underbrace{\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix}}_{\textbf{Ang. accels.}} = \mathrm{J}^{-1}\left\{ \underbrace{\begin{bmatrix} 0 & r & -q \\ -r & 0 & p \\ q & -p & 0 \end{bmatrix}\begin{bmatrix} J_x & -J_{xy} & -J_{xz} \\ -J_{xy} & J_y & -J_{yz} \\ -J_{xz} & -J_{yz} & J_z \end{bmatrix}\begin{bmatrix} p \\ q \\ r \end{bmatrix}}_{\textbf{“Internal” moments induced by cross-coupling effects}} + \underbrace{\begin{bmatrix} l \\ m \\ n \end{bmatrix}}_{\textbf{External Moments}} \right\}$$

This expression tells us how to propagate the angular rates forward in time as the vehicle is being influenced by external moments (e.g. aero and thrust)

Note: There is a lot of cross-coupling, and this form gives very little insight into relationships.

Equations of Motion

- From Kinematics and Dynamics, the equations of motion are 12 ODEs:

6 equations governing translational motion

$$\begin{bmatrix}\dot p_n\\ \dot p_e\\ \dot p_d\end{bmatrix} = R_b^{ned}\,\underline{v}_g^{\,b}$$

$$\begin{bmatrix}\dot u\\ \dot v\\ \dot w\end{bmatrix} = -\underline{\omega}_{b/i}^{\,b}\times\underline{v}_g^{\,b} + \frac{1}{mass}\underline{\mathrm{f}}^{\,b}$$

6 equations governing rotational motion

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

$$\begin{bmatrix}\dot p\\ \dot q\\ \dot r\end{bmatrix} = \mathrm{J}^{-1}\left[-\underline{\omega}_{b/i}^{\,b}\times\left(\mathrm{J}\,\underline{\omega}_{b/i}^{\,b}\right)+\underline{\mathrm{m}}^{\,b}\right]$$

These are the full equations of motion as we will be implementing them in simulation.

But, for autopilot development, we need to look a little closer at the rotational dynamics equation.

Specifically, we use vehicle symmetry to remove some coupling caused by the J (inertia) matrix.

The 12 equations of motion will be used to propagate vehicle states forward in time, given current states and external forces and moments.

Rotational Dynamics

- We formulated an equation governing rotational dynamics, but it is complicated by cross-coupling effects in the inertia matrix (J) and its inverse (J^{-1}):

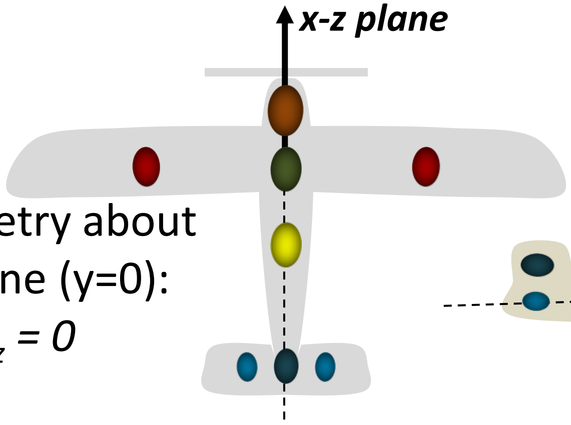
$$\dot{\underline{\omega}}^{b}_{b/i} = \mathrm{J}^{-1}\left[-\underline{\omega}^{b}_{b/i} \times \left(\mathrm{J}\,\underline{\omega}^{b}_{b/i}\right) + \underline{\mathrm{m}}^{b}\right]$$

- This equation can be simplified by taking advantage of symmetries in the airframe
  - Due to symmetry about the x-z plane (in most aircraft), we can assume:

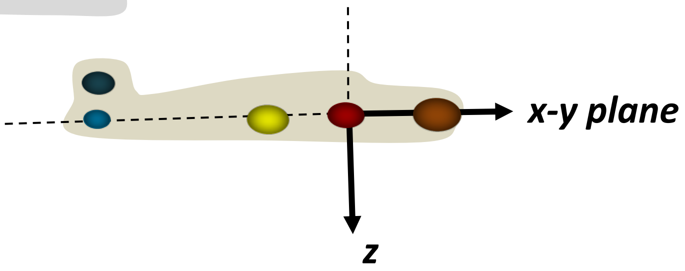
$$J_{xy} = J_{yz} = 0$$

$$\mathrm{J} \cong \begin{bmatrix} J_x & 0 & -J_{xz} \\ 0 & J_y & 0 \\ -J_{xz} & 0 & J_z \end{bmatrix}$$

Symmetry about x-z plane (y=0): $J_{xy} = J_{yz} = 0$



Not symmetric about x-y plane: $J_{xz} \neq 0$



$$\mathrm{J}^{-1} = \frac{\mathrm{adj(J)}}{\det(\mathrm{J})} \cong \frac{\begin{bmatrix} J_y J_z & 0 & J_y J_{xz} \\ 0 & J_x J_z - J_{xz}^2 & 0 \\ J_{xz} J_y & 0 & J_x J_y \end{bmatrix}}{J_x J_y J_z - J_{xz}^2 J_y} = \begin{bmatrix} \frac{J_z}{\Gamma} & 0 & \frac{J_{xz}}{\Gamma} \\ 0 & \frac{1}{J_y} & 0 \\ \frac{J_{xz}}{\Gamma} & 0 & \frac{J_x}{\Gamma} \end{bmatrix}$$

Where :

$$\Gamma = J_x J_z - J_{xz}^2$$

Rotational Dynamics, with Symmetry

- By assuming symmetry about the x-z plane, the inertia matrix loses 4 of 9 terms, and the rotational dynamics equation simplifies, highlighting relationships:

$$\underline{\dot{\omega}}^b_{b/i} = \mathrm{J}^{-1}\left[-\underline{\omega}^b_{b/i} \times \left(\mathrm{J}\,\underline{\omega}^b_{b/i}\right) + \underline{\mathrm{m}}^b\right]$$

$$\begin{bmatrix}\dot{p}\\ \dot{q}\\ \dot{r}\end{bmatrix} = \begin{bmatrix}\frac{J_z}{\Gamma} & 0 & \frac{J_{xz}}{\Gamma}\\ 0 & \frac{1}{J_y} & 0\\ \frac{J_{xz}}{\Gamma} & 0 & \frac{J_x}{\Gamma}\end{bmatrix}\left\{\begin{bmatrix}0 & r & -q\\ -r & 0 & p\\ q & -p & 0\end{bmatrix}\begin{bmatrix}J_x & 0 & -J_{xz}\\ 0 & J_y & 0\\ -J_{xz} & 0 & J_z\end{bmatrix}\begin{bmatrix}p\\ q\\ r\end{bmatrix} + \begin{bmatrix}l\\ m\\ n\end{bmatrix}\right\}$$

$$\begin{bmatrix}\dot{p}\\ \dot{q}\\ \dot{r}\end{bmatrix} = \begin{bmatrix}\frac{J_z}{\Gamma} & 0 & \frac{J_{xz}}{\Gamma}\\ 0 & \frac{1}{J_y} & 0\\ \frac{J_{xz}}{\Gamma} & 0 & \frac{J_x}{\Gamma}\end{bmatrix}\left\{\begin{bmatrix}J_{xz}pq + (J_y - J_z)qr\\ J_{xz}(r^2 - p^2) + (J_z - J_x)pr\\ (J_x - J_y)pq - J_{xz}qr\end{bmatrix} + \begin{bmatrix}l\\ m\\ n\end{bmatrix}\right\}$$

Due to x-z plane symmetry:

$$\mathrm{J} \cong \begin{bmatrix}J_x & 0 & -J_{xz}\\ 0 & J_y & 0\\ -J_{xz} & 0 & J_z\end{bmatrix}$$

$$\mathrm{J}^{-1} = \begin{bmatrix}\frac{J_z}{\Gamma} & 0 & \frac{J_{xz}}{\Gamma}\\ 0 & \frac{1}{J_y} & 0\\ \frac{J_{xz}}{\Gamma} & 0 & \frac{J_x}{\Gamma}\end{bmatrix}$$

$$\text{Where: } \Gamma = J_x J_z - J_{xz}^2$$

$$\underbrace{\begin{bmatrix}\dot{p}\\ \dot{q}\\ \dot{r}\end{bmatrix}}_{\textbf{Ang. accels.}} = \begin{bmatrix}\frac{J_{xz}(J_x - J_y + J_z)}{\Gamma}pq - \frac{J_z(J_z - J_y) + J_{xz}^2}{\Gamma}qr & + & \frac{J_z}{\Gamma}l & + & \frac{J_{xz}}{\Gamma}n\\ \frac{J_z - J_x}{J_y}pr - \frac{J_{xz}}{J_y}(p^2 - r^2) & + & \frac{1}{J_y}m & & \\ \underbrace{\frac{(J_x - J_y)J_x + J_{xz}^2}{\Gamma}pq - \frac{J_{xz}(J_x - J_y + J_z)}{\Gamma}qr}_{\textbf{2nd order cross-coupling}} & + & \underbrace{\frac{J_x}{\Gamma}n}_{\textbf{Direct Moment Effects}} & + & \underbrace{\frac{J_{xz}}{\Gamma}l}_{\textbf{Cross-Coupling}}\end{bmatrix}$$

Rotational Dynamics, with Symmetry

Note: Book uses sub-scripted Γ shorthand. I think this adds confusion, so lectures do not rely on this shorthand.

$$\boxed{\begin{array}{l}\text{Where :}\\ \Gamma = J_x J_z - J_{xz}^2\end{array}}$$

$$\begin{bmatrix}\dot{p}\\ \dot{q}\\ \dot{r}\end{bmatrix} = \begin{bmatrix} \frac{J_{xz}(J_x - J_y + J_z)}{\Gamma} pq - \frac{J_z(J_z - J_y) + J_{xz}^2}{\Gamma} qr & + & \frac{J_z}{\Gamma} l & + & \frac{J_{xz}}{\Gamma} n \\ \frac{J_z - J_x}{J_y} pr - \frac{J_{xz}}{J_y}(p^2 - r^2) & + & \frac{1}{J_y} m & & \\ \frac{(J_x - J_y)J_x + J_{xz}^2}{\Gamma} pq - \frac{J_{xz}(J_x - J_y + J_z)}{\Gamma} qr & + & \frac{J_x}{\Gamma} n & + & \frac{J_{xz}}{\Gamma} l \end{bmatrix}$$

$$= \begin{bmatrix} \Gamma_1 pq - \Gamma_2 qr & + & \Gamma_3 l & + & \Gamma_4 n \\ \Gamma_5 pr - \Gamma_6 (p^2 - r^2) & + & \frac{1}{J_y} m & & \\ \Gamma_7 pq - \Gamma_1 qr & + & \Gamma_8 n & + & \Gamma_4 l \end{bmatrix}$$

Equations of Motion

- From Kinematics and Dynamics, the equations of motion are 12 ODEs:

Compact Form

$$\begin{bmatrix}\dot{p}_n\\ \dot{p}_e\\ \dot{p}_d\end{bmatrix} = R_b^{ned}\,\underline{v}_g^b = \left(R_{ned}^b\right)^T \underline{v}_g^b$$

$$\begin{bmatrix}\dot{u}\\ \dot{v}\\ \dot{w}\end{bmatrix} = -\underline{\omega}_{b/i}^b \times \underline{v}_g^b + \frac{1}{mass}\underline{\mathrm{f}}^b$$

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

$$\begin{bmatrix}\dot{p}\\ \dot{q}\\ \dot{r}\end{bmatrix} = \mathrm{J}^{-1}\left[-\underline{\omega}_{b/i}^b \times \left(\mathrm{J}\,\underline{\omega}_{b/i}^b\right) + \underline{\mathrm{m}}^b\right]$$

Expanded Form

$$\begin{bmatrix}\dot{p}_n\\ \dot{p}_e\\ \dot{p}_d\end{bmatrix} = \begin{bmatrix}\cos\theta\cos\psi & \sin\phi\sin\theta\cos\psi-\cos\phi\sin\psi & \cos\phi\sin\theta\cos\psi+\sin\phi\sin\psi\\ \cos\theta\sin\psi & \sin\phi\sin\theta\sin\psi+\cos\phi\cos\psi & \cos\phi\sin\theta\sin\psi-\sin\phi\cos\psi\\ -\sin\theta & \sin\phi\cos\theta & \cos\phi\cos\theta\end{bmatrix}\begin{bmatrix}u\\ v\\ w\end{bmatrix}$$

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

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

$$\left.\begin{bmatrix}\dot{p}\\ \dot{q}\\ \dot{r}\end{bmatrix} = \begin{bmatrix}\frac{J_{xz}(J_x-J_y+J_z)}{\Gamma}pq-\frac{J_z(J_z-J_y)+J_{xz}^2}{\Gamma}qr\\ \frac{J_z-J_x}{J_y}pr-\frac{J_{xz}}{J_y}(p^2-r^2)\\ \frac{(J_x-J_y)J_x+J_{xz}^2}{\Gamma}pq-\frac{J_{xz}(J_x-J_y+J_z)}{\Gamma}qr\end{bmatrix} + \begin{bmatrix}\frac{J_z}{\Gamma}l+\frac{J_{xz}}{\Gamma}n\\ \frac{1}{J_y}m\\ \frac{J_x}{\Gamma}n+\frac{J_{xz}}{\Gamma}l\end{bmatrix}\right\}$$

$$\text{Where: } \Gamma = J_xJ_z - J_{xz}^2$$

Uses $J_{xy}=J_{yz}=0$ Symmetry Assumption

The 12 equations of motion will be used to propagate vehicle states forward in time, given current states and external forces and moments.

Suggestion: To propagate states, use the compact form. Expanded form will be used for insight in developing autopilot.

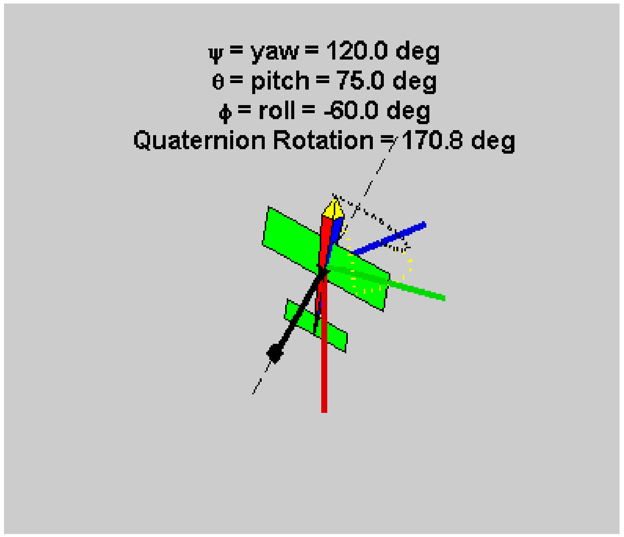
Use of Quaternions, versus Euler Angles

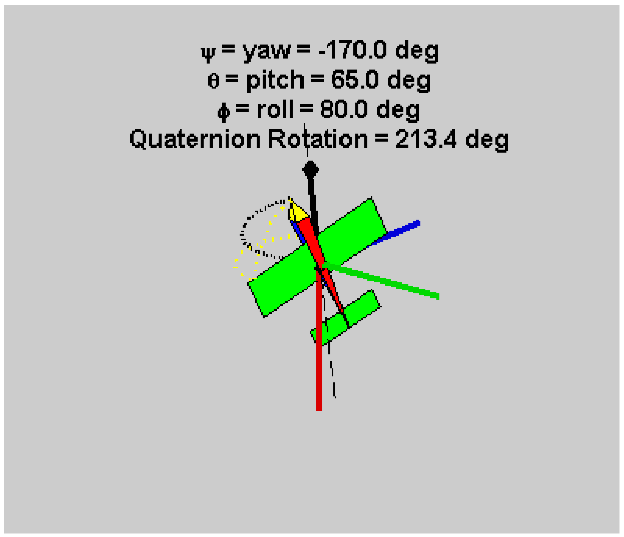
- Notice that the propagation of Euler Angles involves tan(*θ*) and sec(*θ*) =1/cos(*θ*) . These quantities blow up when *θ* (pitch) approaches +/- 90 degrees.

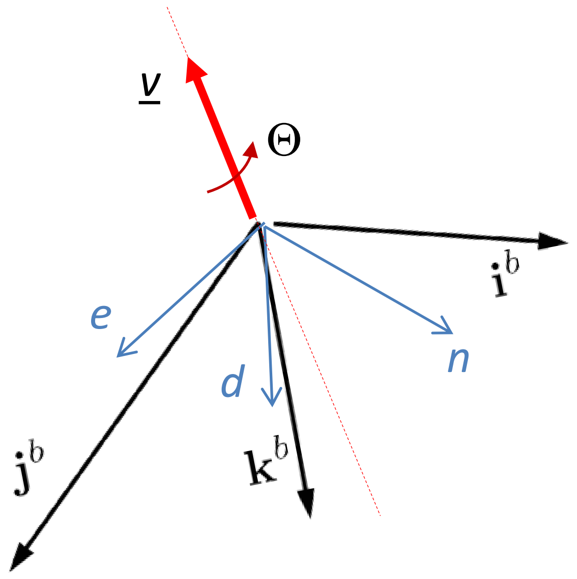
$$\begin{bmatrix}\dot{\phi}\\ \dot{\theta}\\ \dot{\psi}\end{bmatrix}=\begin{bmatrix}1 & \sin\phi\tan\theta & \cos\phi\tan\theta\\ 0 & \cos\phi & -\sin\phi\\ 0 & \sin\phi\sec\theta & \cos\phi\sec\theta\end{bmatrix}\begin{bmatrix}p\\ q\\ r\end{bmatrix}$$

$$\underline{q}=\begin{pmatrix}q_0\\ \begin{bmatrix}q_1\\ q_2\\ q_3\end{bmatrix}\end{pmatrix}=\begin{pmatrix}\cos\left(\frac{\Theta}{2}\right)\\ \hdashline \underline{v}\sin\left(\frac{\Theta}{2}\right)\end{pmatrix}\begin{matrix}\text{Scalar}\\ \\ \text{3x1 Vector}\end{matrix}$$

- To avoid this, attitude is commonly propagated using quaternions
  - A quaternion is a *4x1 unit vector* representing a rotation from one frame to another







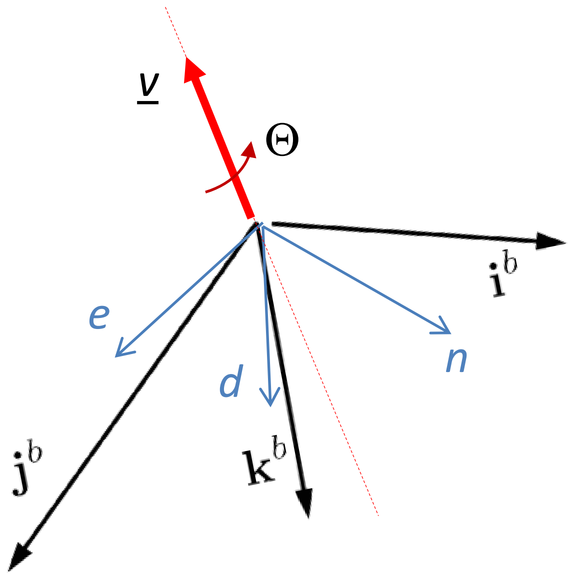
Use of Quaternions, versus Euler Angles

- Quaternions, Euler Angles and DCMs are all ways of expressing a rotation

$$\underbrace{\underline{q}^{b}_{ned}}_{4x1} \text{ is equivalent to } \underbrace{R^{b}_{ned}}_{3x3} \text{ is equivalent to } \underbrace{\begin{bmatrix}\phi\\ \theta\\ \psi\end{bmatrix}^{b}_{ned}}_{3x1}$$

$$\underline{q} = \begin{pmatrix} q_0 \\ \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix} \end{pmatrix} = \begin{pmatrix} \cos\left(\frac{\Theta}{2}\right) \\ \hdashline \underline{v}\sin\left(\frac{\Theta}{2}\right) \end{pmatrix} \begin{array}{l} \text{Scalar} \\ \\ \text{3x1 Vector} \end{array}$$

- Advantages / Disadvantages:
  - Euler Angles: Intuitive, but aren't unique, are order-dependent, and blow up at large θ
  - DCMs: Unique and convenient (matrix math), but have 9 elements to propagate!
  - Quaternions: Unique, but not intuitive.



Notes:

- Quaternion rotations are unique, but representations aren't. Some people have the scalar as the 4th element.
- Our book uses "*e*" instead of "*q*" for quaternions to avoid confusion with body rate variable

Quaternion Conversions and Propagation

- Conversions (using NED-to-body as an example, and assuming common ZYX Euler Rotations):

$$\underline{q}_{ned}^{b} = \begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{pmatrix} = \begin{pmatrix} \cos(\frac{\psi}{2})\cos(\frac{\theta}{2})\cos(\frac{\phi}{2}) + \sin(\frac{\psi}{2})\sin(\frac{\theta}{2})\sin(\frac{\phi}{2}) \\ \cos(\frac{\psi}{2})\cos(\frac{\theta}{2})\sin(\frac{\phi}{2}) - \sin(\frac{\psi}{2})\sin(\frac{\theta}{2})\cos(\frac{\phi}{2}) \\ \cos(\frac{\psi}{2})\sin(\frac{\theta}{2})\cos(\frac{\phi}{2}) + \sin(\frac{\psi}{2})\cos(\frac{\theta}{2})\sin(\frac{\phi}{2}) \\ \sin(\frac{\psi}{2})\cos(\frac{\theta}{2})\cos(\frac{\phi}{2}) - \cos(\frac{\psi}{2})\sin(\frac{\theta}{2})\sin(\frac{\phi}{2}) \end{pmatrix}$$

$$R_{ned}^{b} = \begin{bmatrix} q_1^2 + q_0^2 - q_2^2 - q_3^2 & 2(q_1 q_2 + q_3 q_0) & 2(q_1 q_3 - q_2 q_0) \\ 2(q_1 q_2 - q_3 q_0) & q_2^2 + q_0^2 - q_1^2 - q_3^2 & 2(q_2 q_3 + q_1 q_0) \\ 2(q_1 q_3 + q_2 q_0) & 2(q_2 q_3 - q_1 q_0) & q_3^2 + q_0^2 - q_1^2 - q_2^2 \end{bmatrix} \Rightarrow \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \left\{ \begin{matrix} \text{Extracted from} \\ R_{ned}^{b} \\ \text{in usual manner} \end{matrix} \right\}$$

- Quaternion attitude propagation in Equations of Motion using body rates *[p, q, r]^T*
  - Replace 3 Euler Angle states with 4 Quaternion states (Resulting in 13 states & ODEs):

$$\cancel{\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi\sec\theta & \cos\phi\sec\theta \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix}} \Rightarrow \begin{pmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 0 & -p & -q & -r \\ p & 0 & r & -q \\ q & -r & 0 & p \\ r & q & -p & 0 \end{pmatrix} \begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} -q_1 & -q_2 & -q_3 \\ q_0 & -q_3 & q_2 \\ q_3 & q_0 & -q_1 \\ -q_2 & q_1 & q_0 \end{pmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

- IMPORTANT: Must re-normalize $\underline{q}=[q_0\ q_1\ q_2\ q_3]^T$ after each propagation! ($\underline{q}$ is a unit vector)
  - e.g. $\underline{q} = \underline{q} / |\underline{q}|$ (Book gives a different method)

Equations of Motion using Quaternions

- Using quaterions, the equations of motion are 13 ODEs:

Compact Form with Quaternion

$$\begin{bmatrix} \dot{p}_n \\ \dot{p}_e \\ \dot{p}_d \end{bmatrix} = R_b^{ned}\, \underline{v}_g^b = \left(R_{ned}^b\right)^T \underline{v}_g^b$$

$$\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = -\underline{\omega}_{b/i}^b \times \underline{v}_g^b + \frac{1}{mass}\underline{\mathrm{f}}^b$$

$$\begin{pmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{pmatrix} = \frac{1}{2}\begin{pmatrix} -q_1 & -q_2 & -q_3 \\ q_0 & -q_3 & q_2 \\ q_3 & q_0 & -q_1 \\ -q_2 & q_1 & q_0 \end{pmatrix}\begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \mathrm{J}^{-1}\left[-\underline{\omega}_{b/i}^b \times \left(\mathrm{J}\,\underline{\omega}_{b/i}^b\right) + \underline{\mathrm{m}}^b\right]$$

At t_0, initialize quaternion with initial Euler Angles:

$$\underline{q}_{ned}^b = \begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{pmatrix} = \begin{pmatrix} \cos(\frac{\psi}{2})\cos(\frac{\theta}{2})\cos(\frac{\phi}{2}) + \sin(\frac{\psi}{2})\sin(\frac{\theta}{2})\sin(\frac{\phi}{2}) \\ \cos(\frac{\psi}{2})\cos(\frac{\theta}{2})\sin(\frac{\phi}{2}) - \sin(\frac{\psi}{2})\sin(\frac{\theta}{2})\cos(\frac{\phi}{2}) \\ \cos(\frac{\psi}{2})\sin(\frac{\theta}{2})\cos(\frac{\phi}{2}) + \sin(\frac{\psi}{2})\cos(\frac{\theta}{2})\sin(\frac{\phi}{2}) \\ \sin(\frac{\psi}{2})\cos(\frac{\theta}{2})\cos(\frac{\phi}{2}) - \cos(\frac{\psi}{2})\sin(\frac{\theta}{2})\sin(\frac{\phi}{2}) \end{pmatrix}$$

After propagating and normalizing quaternion at each time step, construct DCM:

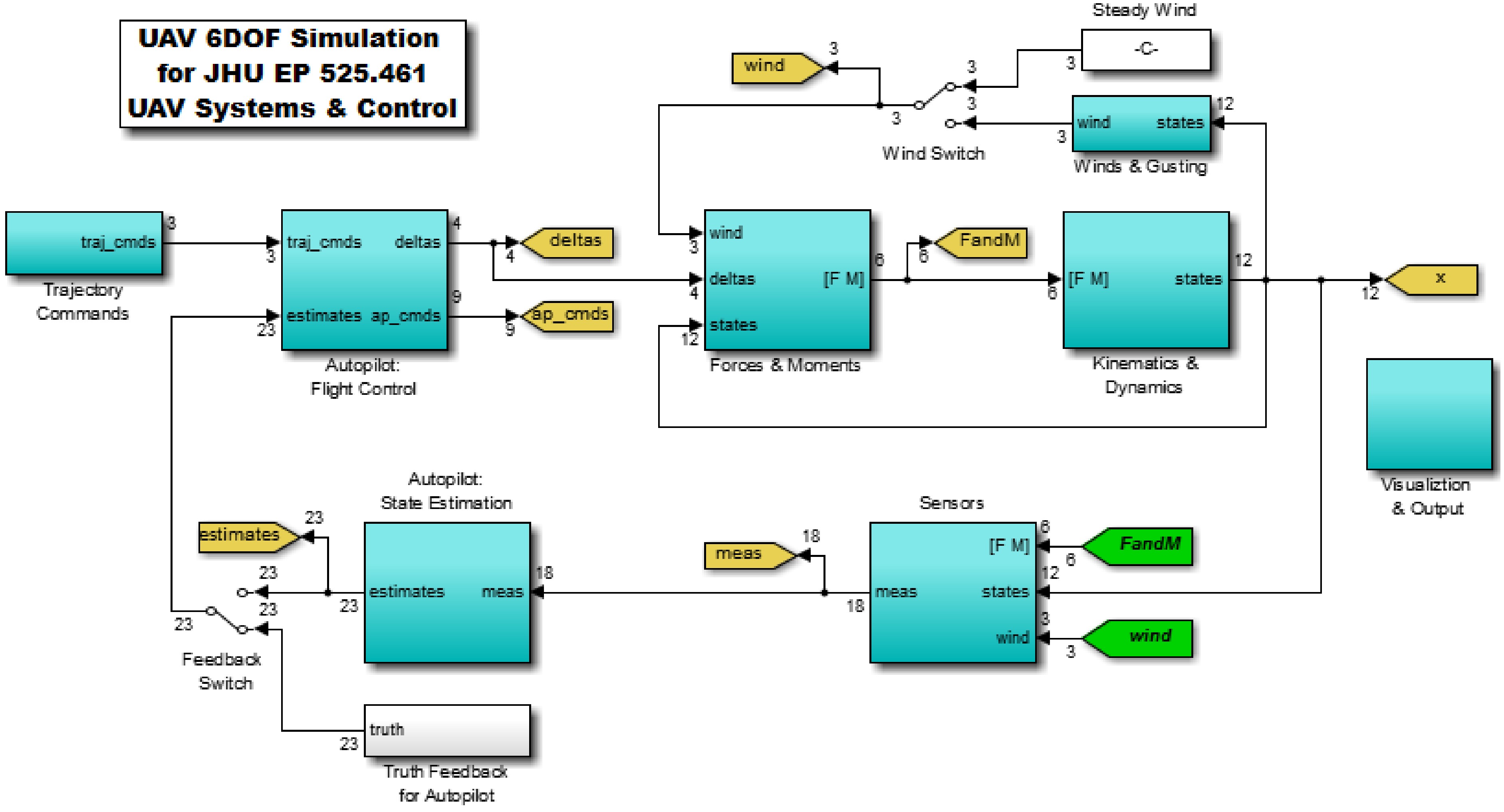
$$R_{ned}^b = \begin{bmatrix} q_1^2 + q_0^2 - q_2^2 - q_3^2 & 2(q_1q_2 + q_3q_0) & 2(q_1q_3 - q_2q_0) \\ 2(q_1q_2 - q_3q_0) & q_2^2 + q_0^2 - q_1^2 - q_3^2 & 2(q_2q_3 + q_1q_0) \\ 2(q_1q_3 + q_2q_0) & 2(q_2q_3 - q_1q_0) & q_3^2 + q_0^2 - q_1^2 - q_2^2 \end{bmatrix}$$

Using the 13-state quaternion-based EoMs is highly preferred if there is any possibility that the vehicle will pitch up or down extremely. Most high fidelity 6DOF simulations, as well as attitude estimation algorithms, will use quaternions.

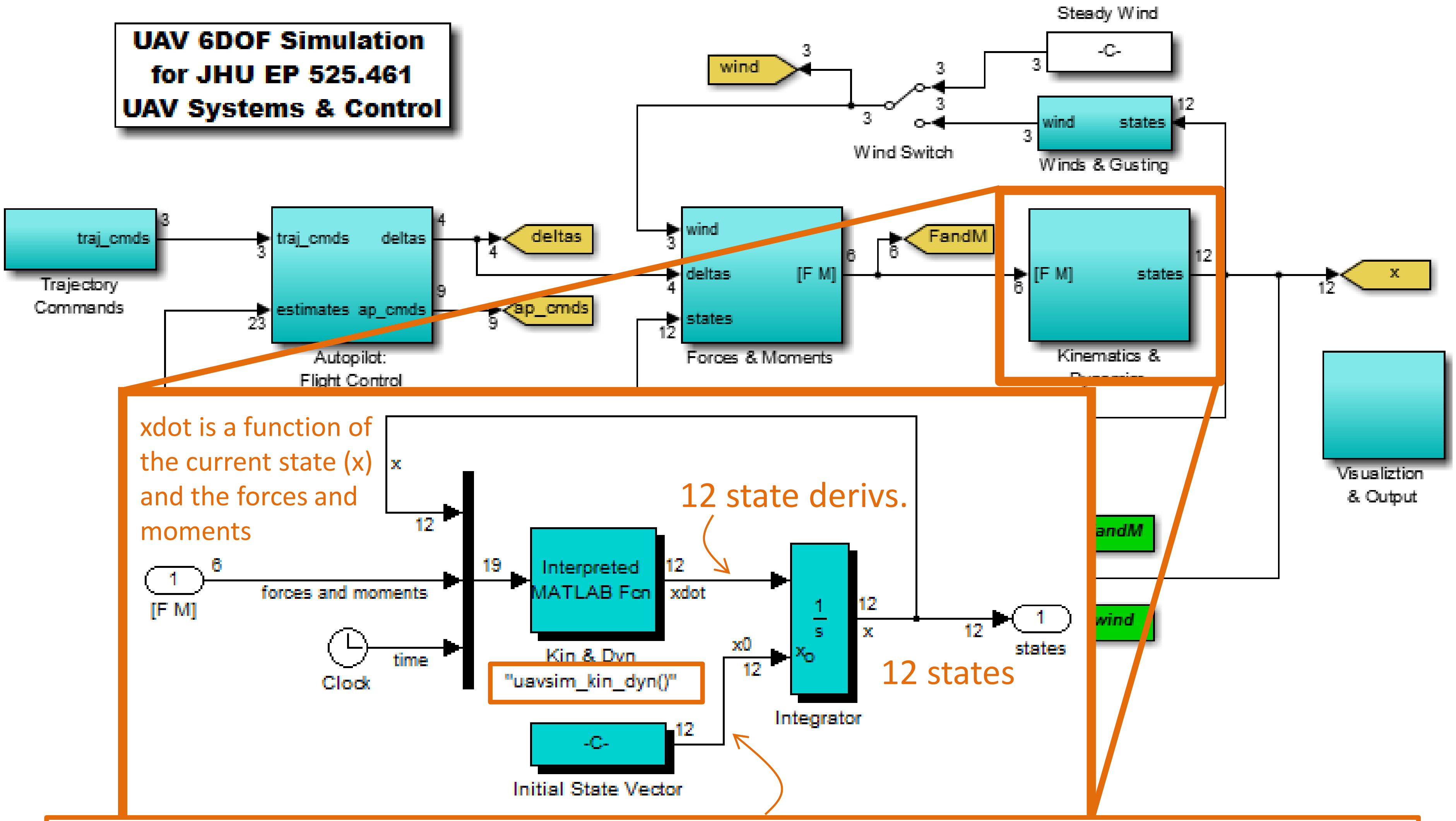
Assumptions/Simplifications we've made:

- NED (attached to ground) is an inertial reference frame
- Presuming a flat earth
- Vehicle has constant mass and inertia
- Vehicle is a rigid body
- Vehicle has symmetry about x-z plane
- Body-frame origin is at center-of-gravity & inertia is defined about c.g.

Matlab/Simulink UAV 6DOF Tour



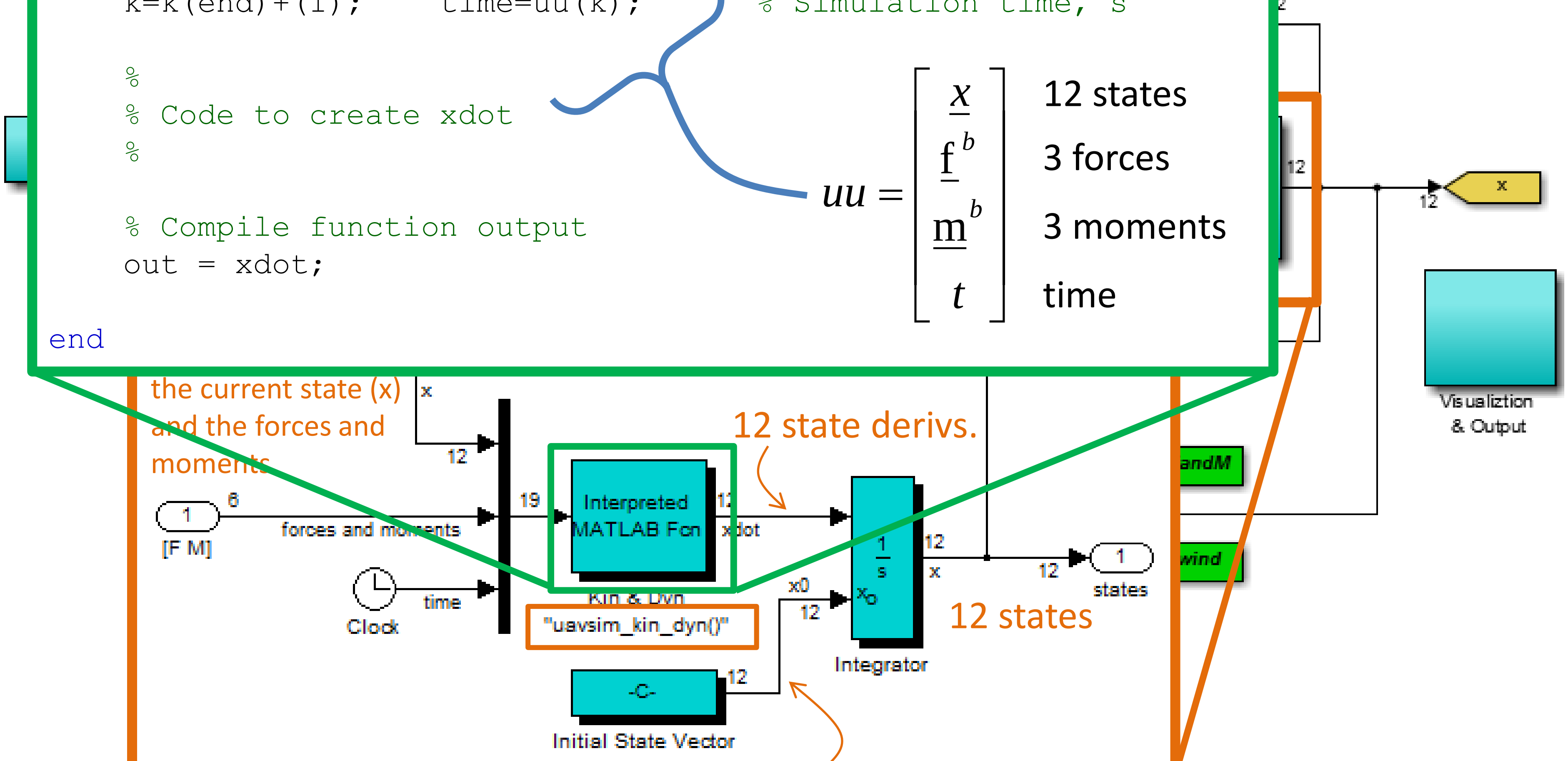
Matlab/Simulink UAV 6DOF Tour



x0 = [P.pn0; P.pe0; P.pd0; P.u0; P.v0; P.w0; P.phi0; P.theta0; P.psi0; P.p0; P.q0; P.r0]'

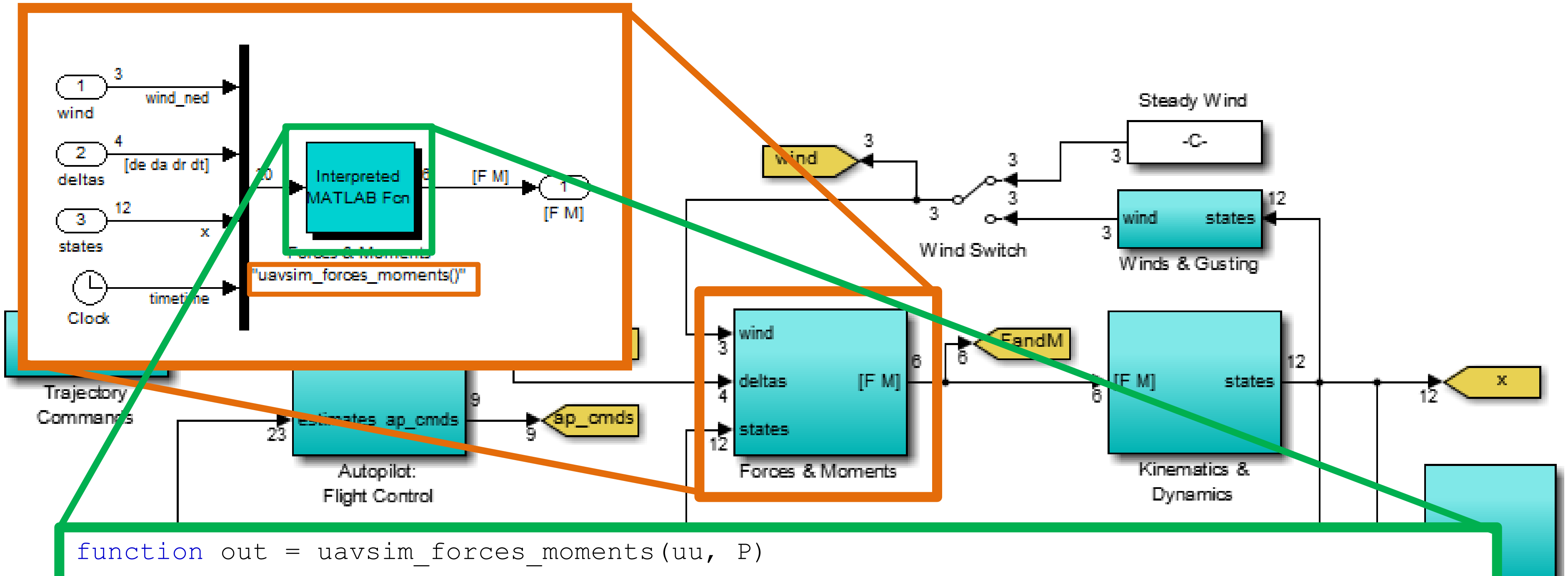
```
function out = uavsim_kin_dyn(uu,P)

    % Extract variables from input vector uu
    %   uu = [x(1:12); f_and_m(1:6); time(1)];
    k=1:12;             x=uu(k);          % States
    k=k(end)+(1:6);     f_and_m=uu(k);    % Forces and Moments, body
    k=k(end)+(1);       time=uu(k);       % Simulation time, s

    %
    % Code to create xdot
    %

    % Compile function output
    out = xdot;

end
```

$$uu = \begin{array}{ll} \begin{bmatrix} \underline{x} \\ \underline{f}^b \\ \underline{m}^b \\ t \end{bmatrix} & \begin{array}{l} \text{12 states} \\ \text{3 forces} \\ \text{3 moments} \\ \text{time} \end{array} \end{array}$$



```
x0 = [P.pn0; P.pe0; P.pd0; P.u0; P.v0; P.w0; P.phi0; P.theta0; P.psi0; P.p0; P.q0; P.r0]'
```

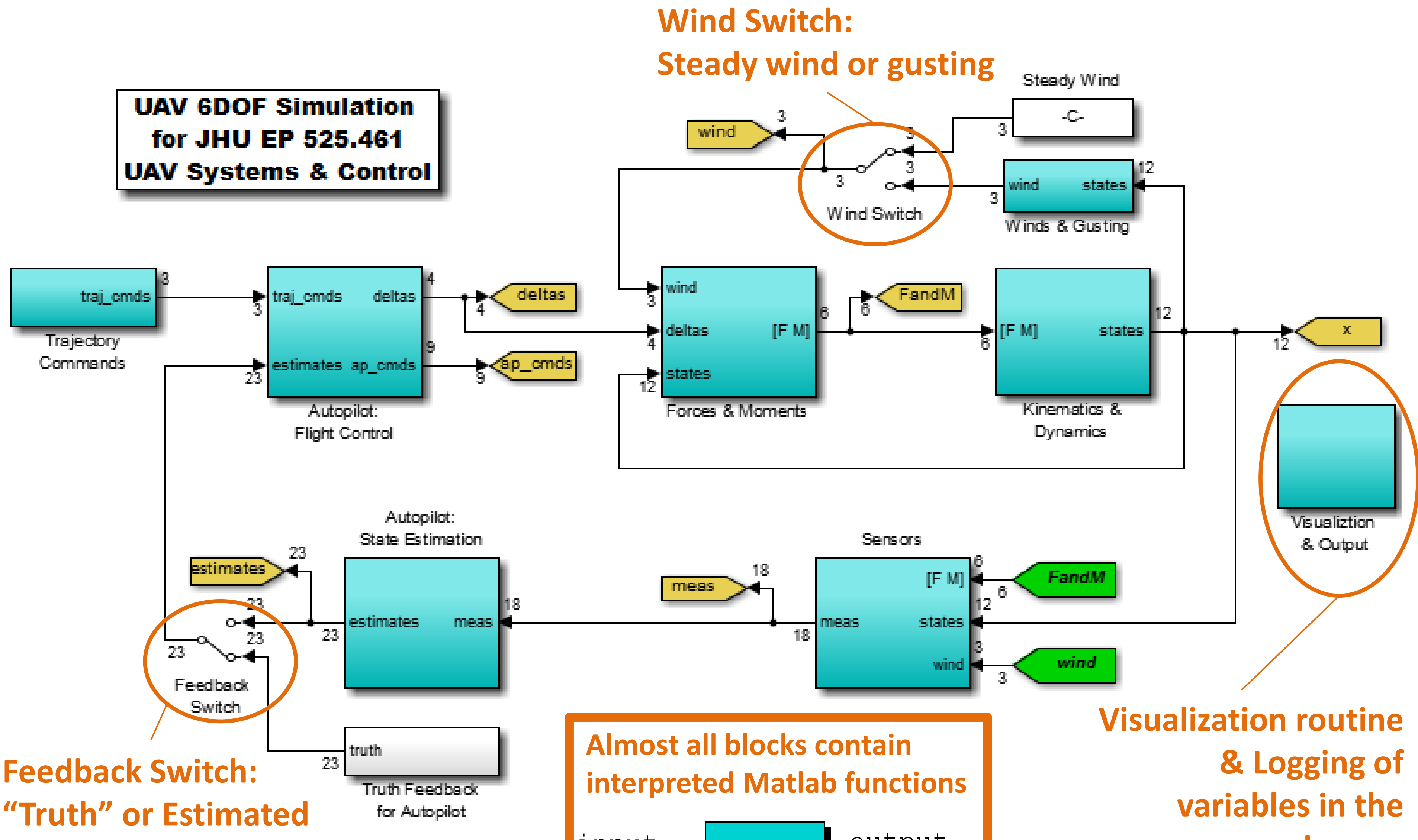
Matlab/Simulink UAV 6DOF Tour



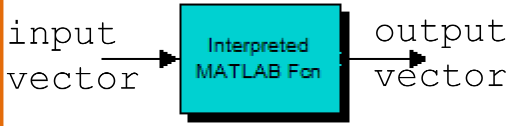
```
function out = uavsim_forces_moments(uu, P)

    % Extract variables from input vector uu
    %   uu = [wind_ned(1:3); deltas(1:4); x(1:12); time(1)];
    k=(1:3);               wind_ned=uu(k);   % Total wind vector, ned, m/s
    k=k(end)+(1:4);        deltas=uu(k);     % 4 Control surface commands
    k=k(end)+(1:12);       x=uu(k);          % states
    k=k(end)+(1);          time=uu(k);       % Simulation time, s

    % Make f_b & m_b

    % Compile function output
    out = [f_b; m_b]; % Length 3+3=6

end
```

$$uu = \begin{bmatrix} \underline{v}_w^{ned} \\ [\delta_e \quad \delta_a \quad \delta_r \quad \delta_t]^T \\ \underline{x} \\ t \end{bmatrix}$$

Matlab/Simulink UAV 6DOF Tour



Almost all blocks contain interpreted Matlab functions



Matlab/Simulink UAV 6DOF Tour

Use load_uavsim.m to prepare simulation and/or re-load the sim parameters

```
% load_uavsim.m

% Bring up simulink model
open('uavsim')

% Load UAV parameters
P = init_uavsim_params;
```

Initialize simulation parameters. Returns `P` *structure.*

```
function P = init_uavsim_params

    P = [];

    % Acquire UAV vehicle model params
    %   e.g. P.mass_kg, P.C_L_alpha, etc.
    uav_props_website;

    P.Ts = 0.01;      % Autopilot sample time step, s

    P.Tlog = 0.01;    % Logging time step, s
    P.Tvis = 0.2;     % Visualization time step, s

    % and more and more and more sim parameters…
```

Initialize vehicle-specific parameters. (Source: book website.)

```
% physical parameters of airframe
P.mass = 1.56;      % kg
P.Jx   = 0.1147;    % kg-m^2
P.Jy   = 0.0576;    % kg-m^2
P.Jz   = 0.1712;    % kg-m^2
P.Jxz  = 0.0015;    % kg-m^2

% aerodynamic coefficients
P.c               = 0.3302;    % Wing chord, m
P.b               = 1.4224;    % Wing span, m
P.S_wing          = 0.2589;    % Wing area, m^2

% and more and more and more vehicle params...
```

Matlab/Simulink UAV 6DOF Tour

uavsim_logging.m generates an "out" structure that is saved to the Matlab workspace:

```
out.time_s            % Time
out.north_m           % Position
out.east_m
out.alt_m
out.groundspeed_mps   % Velocity
out.gamma_deg         %   (Mag & angles)
out.course_deg
out.roll_deg          % Attitude
out.pitch_deg
out.yaw_deg
out.p_dps             % pqr: body rates
out.q_dps
out.r_dps

out.airspeed_mps      % Airspeed
out.alpha_deg         % Angle-of-Attack
out.beta_deg          % Sideslip
```

```
out.wind_north_mps  % Wind
out.wind_east_mps
out.wind_down_mps

out.de_deg        % Elevator
out.da_deg        % Aileron
out.dr_deg        % Rudder
out.throttle      % Throttle

out.fx_N          % Forces
out.fy_N
out.fz_N
out.mx_Nm         % Moments
out.my_Nm
out.mz_Nm
```

Note: Internal computations are done in radians, but outputs are in degrees for convenience

uavsim_logging.m also contains flags to enable additional outputs when necessary

```
% Logging flags
log_commands     = 0;
log_measurements = 0; % Set to 1 to enable
log_estimates    = 0;
```

Lecture 3 Homework, 1/4

Recommended Reading: Beard & McLain Chapter 3 & Appendix B (lightly)

Some matrices in book are difficult to read. (e.g. 3.1., 3.3, 3.14, 3.16) Use notes as reference.

1) **Modify "uavsim_kin_dyn.m" to implement the Euler-version of Equations of Motion in the UAV 6DOF. Specifically, implement the <u>Compact Form</u> of the EoMs. The vehicle parameters you need are available via the "P" structure (e.g. P.mass, P.Jxz, etc.). (Note: The quaternion form is generally a better choice, but we'll use the Euler form in this class for simplicity.)**

   a. **Print out the resulting uavsim_kin_dyn.m.**
   (Just print the portion from "Your code goes below…**" to "**out = xdot;**")**

   b. **After modifying kin&dyn, initialize the sim with "load_uavsim", run the simulation, and describe (briefly!) the vehicle motion. (Note: all forces and moments are still zero.)**

   c. **The "P" structure contains vehicle and simulation parameters, including the 12 initial kinematic states (all units are metric and angles are in radians):**
   ```
   pn0, pe0, pd0, u0, v0, w0, phi0, theta0, psi0, p0, q0, r0
   ```
   Values in "P" can be modified in "init_uavsim_params", <u>or</u> they can be set directly in the Matlab command console. <u>In the command console</u>, temporarily overwrite the following initial kinematic states, then re-run the simulation:
   ```
   P.psi0=30*pi/180; P.theta0=15*pi/180;
   ```
   After running the simulation, the "Visualization & Output" block produces an "out" structure in the Matlab workspace. Using the "out" structure, plot the resulting variables versus time:

   alt_m, east_m, and north_m (e.g. `plot(out.time_s, out.alt_m)` **)**

Lecture 3 Homework, 2/4

2) **The file "uavsim_forces_moments.m" will be modified in the future to generate the forces and moments that feed into the kinematics and dynamics equations. As a test, modify "uavsim_forces_moments" in the following ways. (Note: re-run "load_uavsim" before doing this problem so that theta0 and psi0 are both reverted to zero.)**
   a. **m_b=[0.005; 0; 0]; % N-m (Constant torque about x)**
   b. **m_b=[0; 0.005; 0]; % N-m (Constant torque about y)**
   c. **m_b=[0; 0; 0.005]; % N-m (Constant torque about z)**
   For each: plot roll, pitch & yaw, and describe the resulting motion.
   <u>Should it behave this way?</u>

   Reset m_b to a zero vector before continuing…

3) **Gravity points down. In NED coordinates, the force due to gravity is:**

```
f_grav_ned=P.mass*[0; 0; P.gravity]; % Newtons
```

Use `f_grav_ned` **to set** `f_grav_b` **(i.e. rotate the vector appropriately) in "uavsim_forces_moments.m". You have now made a ballistic trajectory generator! Test your ballistic trajectory generator using the following initial conditions:**

```
P.pd0 = -50; P.u0 = 30; P.psi0 = 0; P.theta0 = 45*pi/180;
```

   a. **Plot altitude versus north distance:** `plot(out.north_m, out.alt_m);`
   b. **What is the vehicle's maximum altitude?**
   c. **What is the north position at ground impact?**
   d. **What are the minimum and maximum groundspeeds during the trajectory?**
   (Note: the attitude won't change because the moments are zero.)

Hint: Impacts ground at about t=6 seconds

Lecture 3 Homework, 3/4

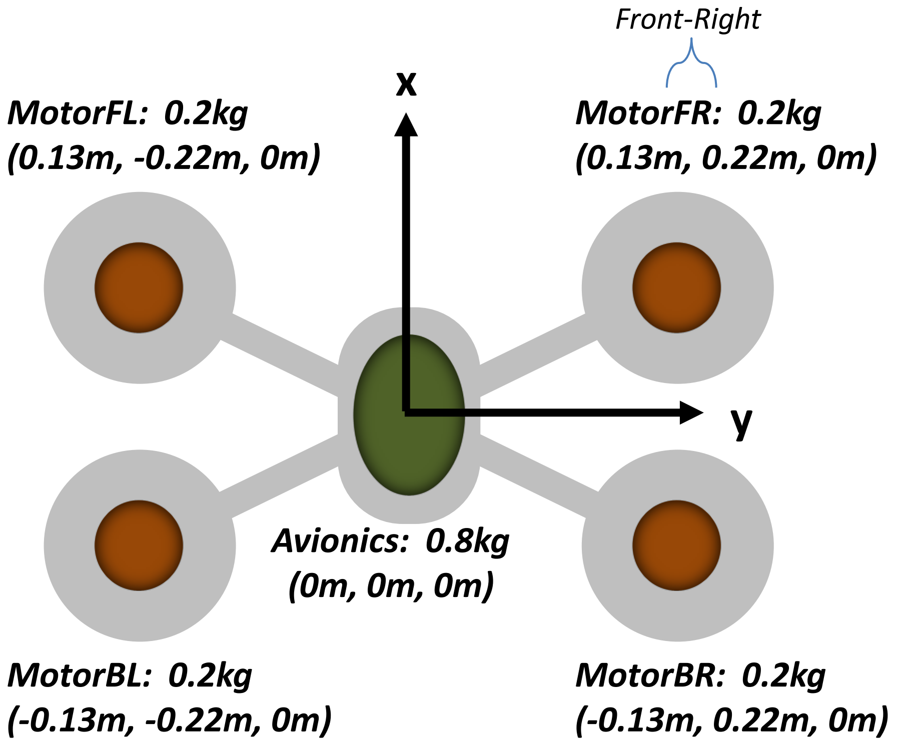
4) Know your aircraft variables…

- **a. Write variable names for the 6 translational kinematic states and concisely describe each.**
- **b. Write variable names for the 6 rotational kinematic states and concisely describe each.**
- **c. Write variable names for the body-frame force components and concisely describe each.**
- **d. Write variable names for the body-frame moment components and concisely describe each.**
- **e. Write the variable names for angle-of-attack and sideslip (specify which is which).**

5) Consider a symmetric quadcopter as shown.

- **a. Derive the 3x3 inertia matrix J for the quadcopter using the lumped mass approximations shown. Note: Provided positions are (x,y,z) coordinates.**

- **b. What would the inertia matrix be if the owner of the quadcopter mounted a 0.3kg camera below the quadcopter at the x=0.05m, y=0m, and z=0.10m position?**



http://xproheli.com/products/3dr-iris-quadcopter-for-gopro

Lecture 3 Homework, 4/4

6) **Use $\underline{\omega}_{b/i}^{b} \times \underline{v}_{g}^{b}$ to show that left-multiplying with the skew-symmetric matrix is equivalent to performing the cross product. In other words, show that the following are equivalent:**

$$\underline{\omega}_{b/i}^{b} \times \underline{v}_{g}^{b} \Leftrightarrow \left[\underline{\omega}_{b/i}^{b} \times\right]\underline{v}_{g}^{b} \qquad \text{where: } \left[\underline{\omega}_{b/i}^{b} \times\right] = \begin{bmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix}$$

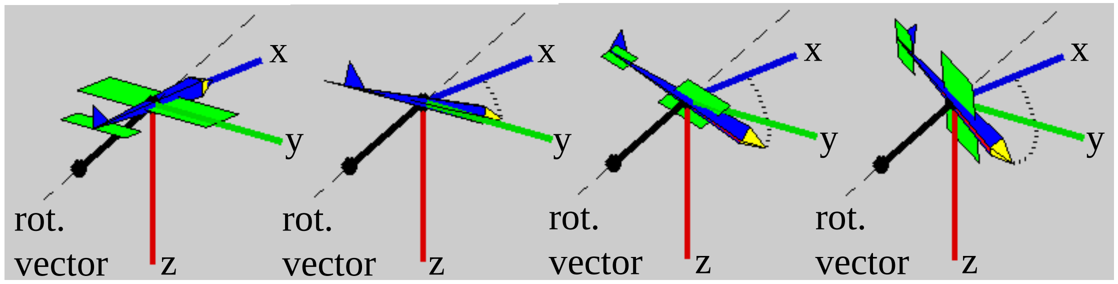
7) **Consider the Euler angle rotation from NED to body:**

$\psi = 95^o$, $\theta = -30^o$, $\phi = -60^o$

Using the Euler-to-quaternion method shown in the lecture, one can generate a 4-element quaternion that is equivalent to this ZYX Euler rotation, as shown:

$$\underline{q}_{ned}^{b} = \begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{pmatrix} = \begin{pmatrix} \cos(\frac{\psi}{2})\cos(\frac{\theta}{2})\cos(\frac{\phi}{2}) + \sin(\frac{\psi}{2})\sin(\frac{\theta}{2})\sin(\frac{\phi}{2}) \\ \cos(\frac{\psi}{2})\cos(\frac{\theta}{2})\sin(\frac{\phi}{2}) - \sin(\frac{\psi}{2})\sin(\frac{\theta}{2})\cos(\frac{\phi}{2}) \\ \cos(\frac{\psi}{2})\sin(\frac{\theta}{2})\cos(\frac{\phi}{2}) + \sin(\frac{\psi}{2})\cos(\frac{\theta}{2})\sin(\frac{\phi}{2}) \\ \sin(\frac{\psi}{2})\cos(\frac{\theta}{2})\cos(\frac{\phi}{2}) - \cos(\frac{\psi}{2})\sin(\frac{\theta}{2})\sin(\frac{\phi}{2}) \end{pmatrix} = \begin{pmatrix} 0.661 \\ -0.161 \\ -0.508 \\ 0.529 \end{pmatrix}$$

The 4-element quaternion consists of a scalar component and a vector component. From the quaternion definition, determine the angular magnitude of this rotation (in degrees), along with the unit vector defining the axis of rotation.


x
y
z
rot.
vector
x
y
z
rot.
vector
x
y
z
rot.
vector
x
y
z
rot.
vector