A yellow glider plane is captured in flight against a backdrop of a vast, cloudy sky. Below, a rural landscape features a field of tall grass or crops in the foreground, with a cluster of farm buildings, trees, and a windmill visible in the distance.

Chapter 1

Introduction

Potential Applications for Small UAVs

Civil and Commercial:

- 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:

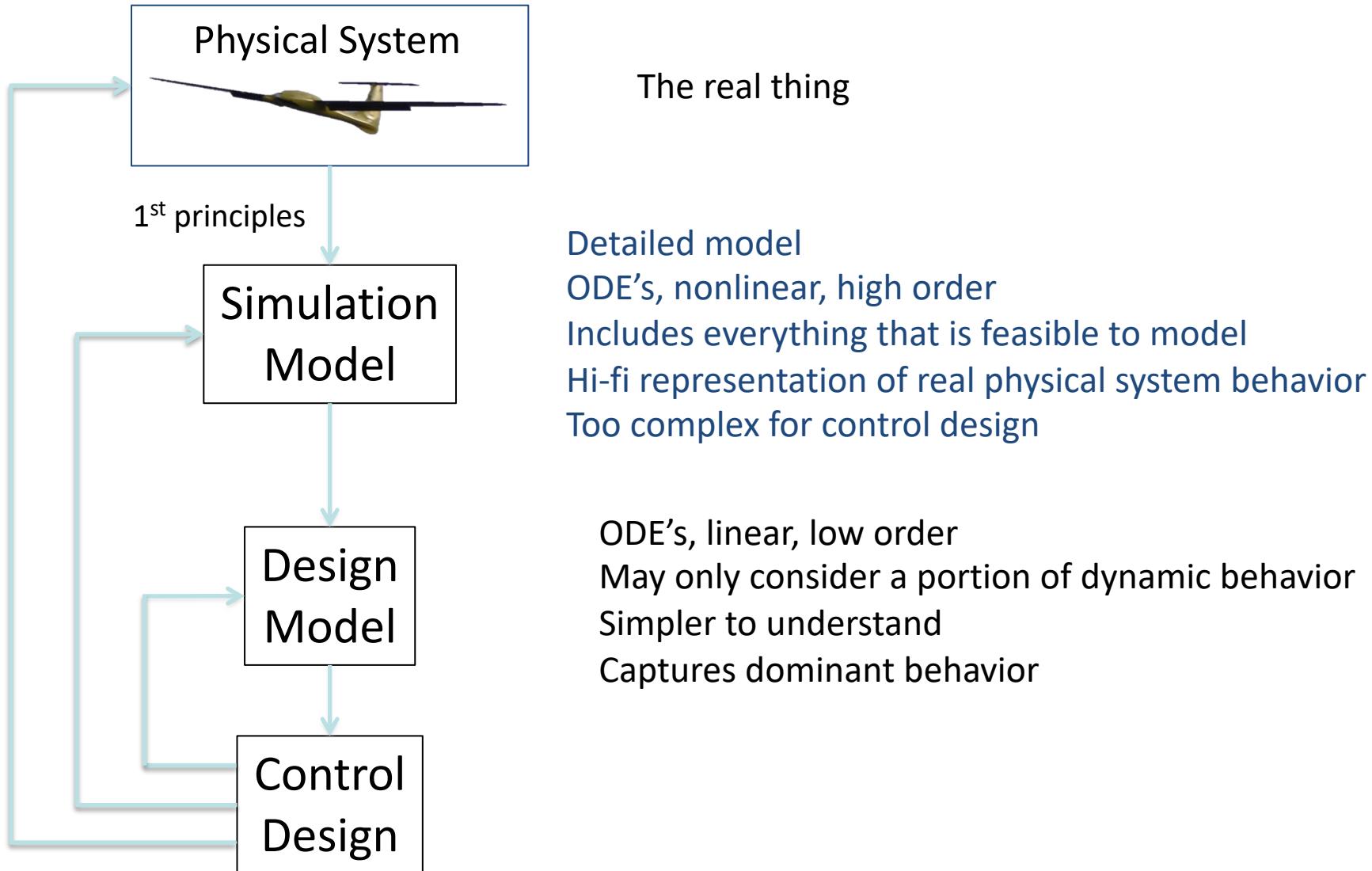
- Border patrol
- Surveillance
- Rural/urban search and rescue

New related area:

- Air mobility of humans, goods, services



Control Design Process



Simulation Model

$$\dot{p}_n = (\cos \theta \cos \psi)u + (\sin \phi \sin \theta \cos \psi - \cos \phi \sin \psi)v + (\cos \phi \sin \theta \cos \psi + \sin \phi \sin \psi)w$$

$$\dot{p}_e = (\cos \theta \sin \psi)u + (\sin \phi \sin \theta \sin \psi + \cos \phi \cos \psi)v + (\cos \phi \sin \theta \sin \psi - \sin \phi \cos \psi)w$$

$$\dot{h} = u \sin \theta - v \sin \phi \cos \theta - w \cos \phi \cos \theta$$

$$\dot{u} = rv - qw - g \sin \theta + \frac{\rho V_a^2 S}{2m} \left[C_X(\alpha) + C_{X_q}(\alpha) \frac{cq}{2V_a} + C_{X_{\delta_e}}(\alpha) \delta_e \right] + \frac{\rho S_{\text{prop}} C_{\text{prop}}}{2m} \left[(k_{\text{motor}} \delta_t)^2 - V_a^2 \right]$$

$$\dot{v} = pw - ru + g \cos \theta \sin \phi + \frac{\rho V_a^2 S}{2m} \left[C_{Y_0} + C_{Y_\beta} \beta + C_{Y_p} \frac{bp}{2V_a} + C_{Y_r} \frac{br}{2V_a} + C_{Y_{\delta_a}} \delta_a + C_{Y_{\delta_r}} \delta_r \right]$$

$$\dot{w} = qu - pv + g \cos \theta \cos \phi + \frac{\rho V_a^2 S}{2m} \left[C_Z(\alpha) + C_{Z_q}(\alpha) \frac{cq}{2V_a} + C_{Z_{\delta_e}}(\alpha) \delta_e \right]$$

$$\dot{\phi} = p + q \sin \phi \tan \theta + r \cos \phi \tan \theta$$

$$\dot{\theta} = q \cos \phi - r \sin \phi$$

$$\dot{\psi} = q \sin \phi \sec \theta + r \cos \phi \sec \theta$$

$$\dot{p} = \Gamma_1 pq - \Gamma_2 qr + \frac{1}{2} \rho V_a^2 S b \left[C_{p_0} + C_{p_\beta} \beta + C_{p_p} \frac{bp}{2V_a} + C_{p_r} \frac{br}{2V_a} + C_{p_{\delta_a}} \delta_a + C_{p_{\delta_r}} \delta_r \right]$$

$$\dot{q} = \Gamma_5 pr - \Gamma_6 (p^2 - r^2) + \frac{\rho V_a^2 Sc}{2J_y} \left[C_{m_0} + C_{m_\alpha} \alpha + C_{m_q} \frac{cq}{2V_a} + C_{m_{\delta_e}} \delta_e \right]$$

$$\dot{r} = \Gamma_7 pq - \Gamma_1 qr + \frac{1}{2} \rho V_a^2 S b \left[C_{r_0} + C_{r_\beta} \beta + C_{r_p} \frac{bp}{2V_a} + C_{r_r} \frac{br}{2V_a} + C_{r_{\delta_a}} \delta_a + C_{r_{\delta_r}} \delta_r \right]$$

Simulation Model

$$C_{p_0} = \Gamma_3 C_{l_0} + \Gamma_4 C_{n_0}$$

$$C_{p_\beta} = \Gamma_3 C_{l_\beta} + \Gamma_4 C_{n_\beta}$$

$$C_{p_p} = \Gamma_3 C_{l_p} + \Gamma_4 C_{n_p}$$

$$C_{p_r} = \Gamma_3 C_{l_r} + \Gamma_4 C_{n_r}$$

$$C_{p_{\delta_a}} = \Gamma_3 C_{l_{\delta_a}} + \Gamma_4 C_{n_{\delta_a}}$$

$$C_{p_{\delta_r}} = \Gamma_3 C_{l_{\delta_r}} + \Gamma_4 C_{n_{\delta_r}}$$

$$C_{r_0} = \Gamma_4 C_{l_0} + \Gamma_8 C_{n_0}$$

$$C_{r_\beta} = \Gamma_4 C_{l_\beta} + \Gamma_8 C_{n_\beta}$$

$$C_{r_p} = \Gamma_4 C_{l_p} + \Gamma_8 C_{n_p}$$

$$C_{r_r} = \Gamma_4 C_{l_r} + \Gamma_8 C_{n_r}$$

$$C_{r_{\delta_a}} = \Gamma_4 C_{l_{\delta_a}} + \Gamma_8 C_{n_{\delta_a}}$$

$$C_{r_{\delta_r}} = \Gamma_4 C_{l_{\delta_r}} + \Gamma_8 C_{n_{\delta_r}}$$

$$C_X(\alpha) \triangleq -C_D(\alpha) \cos \alpha + C_L(\alpha) \sin \alpha$$

$$C_{X_q}(\alpha) \triangleq -C_{D_q} \cos \alpha + C_{L_q} \sin \alpha$$

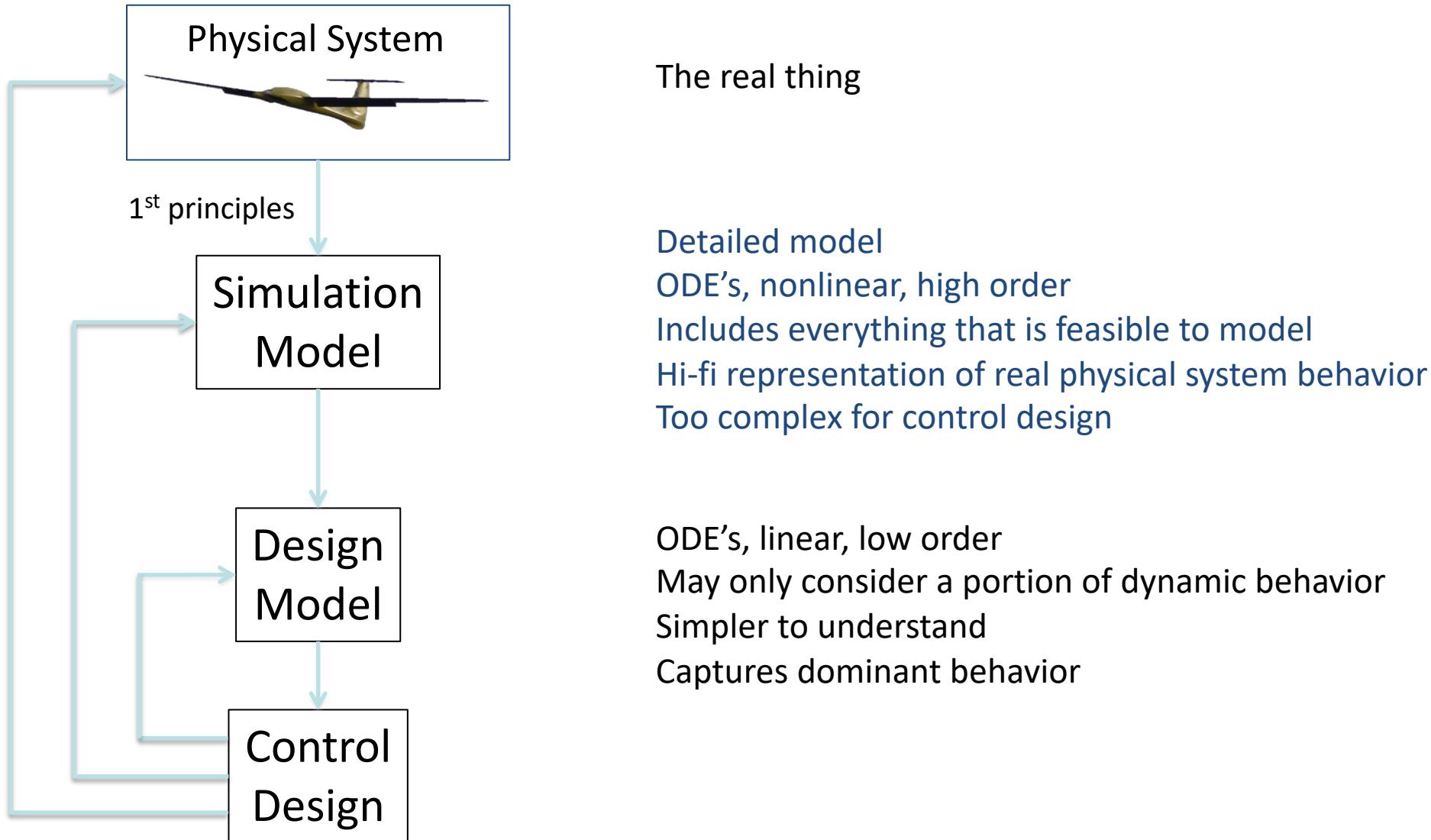
$$C_{X_{\delta_e}}(\alpha) \triangleq -C_{D_{\delta_e}} \cos \alpha + C_{L_{\delta_e}} \sin \alpha$$

$$C_Z(\alpha) \triangleq -C_D(\alpha) \sin \alpha - C_L(\alpha) \cos \alpha$$

$$C_{Z_q}(\alpha) \triangleq -C_{D_q} \sin \alpha - C_{L_q} \cos \alpha$$

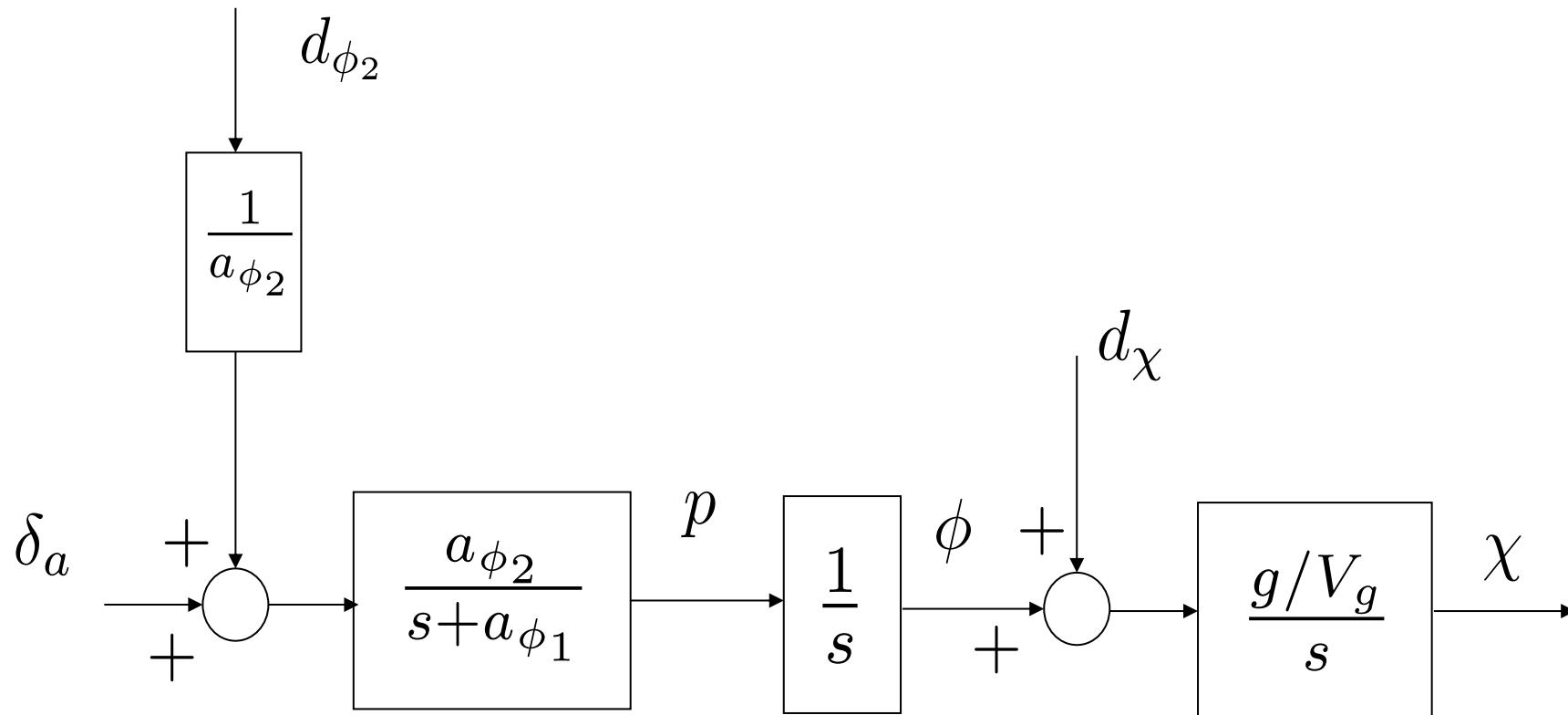
$$C_{Z_{\delta_e}}(\alpha) \triangleq -C_{D_{\delta_e}} \sin \alpha - C_{L_{\delta_e}} \cos \alpha$$

Control Design Process

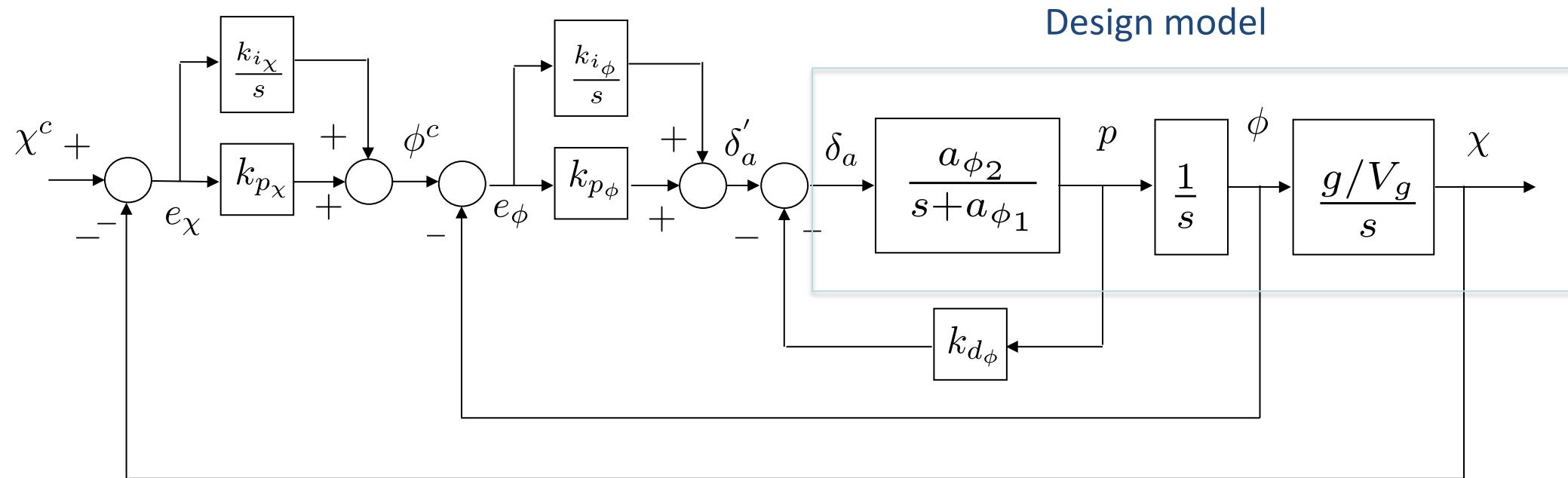


Design Model

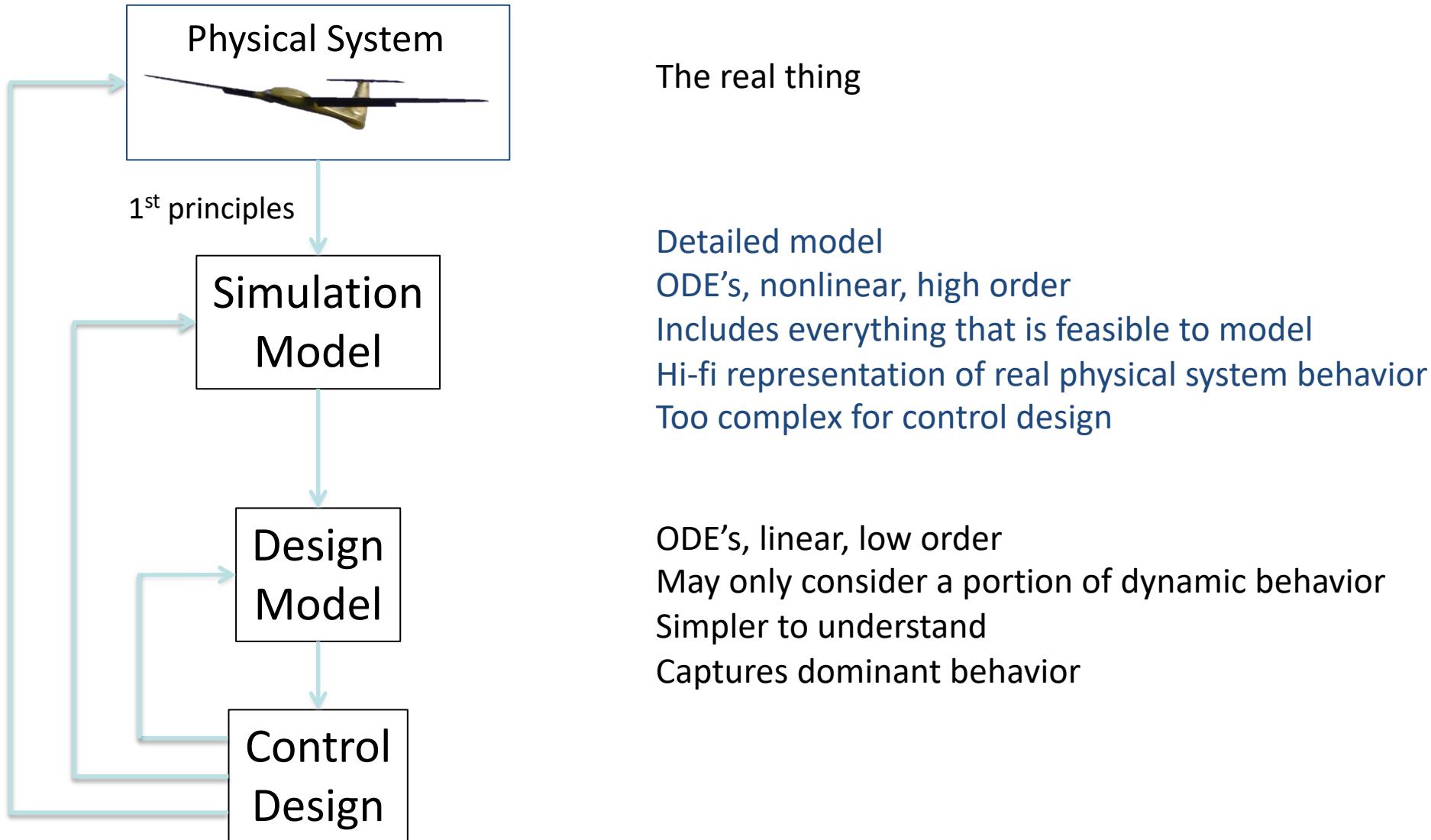
Course from aileron transfer function:



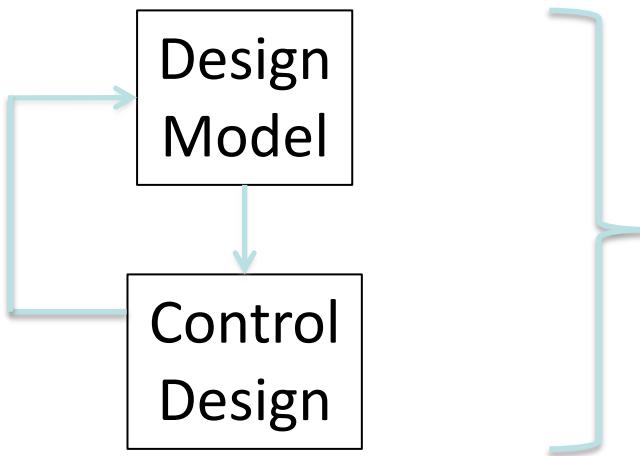
Control Design



Control Design Process

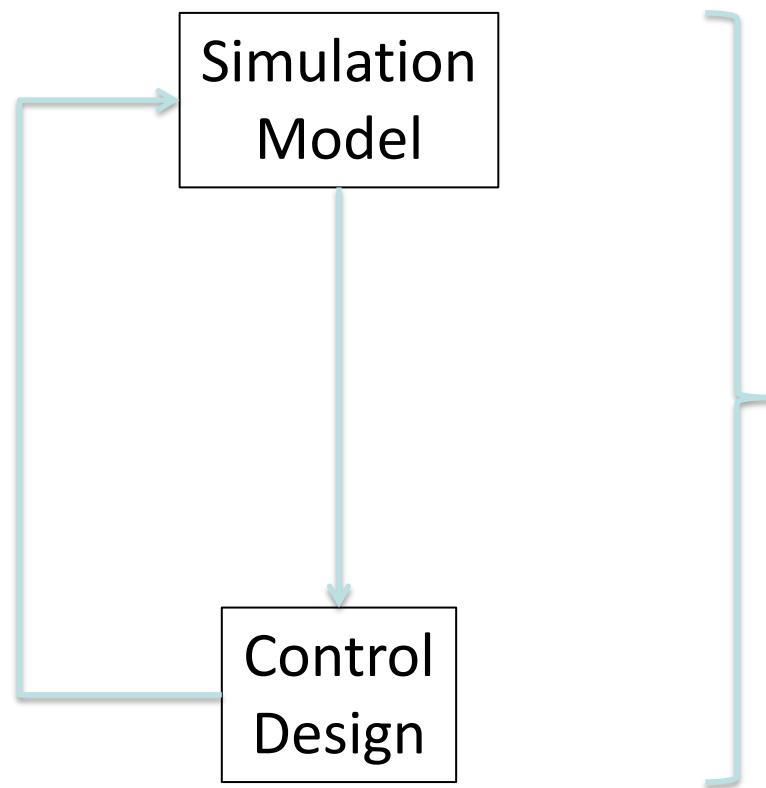


Control Design Process



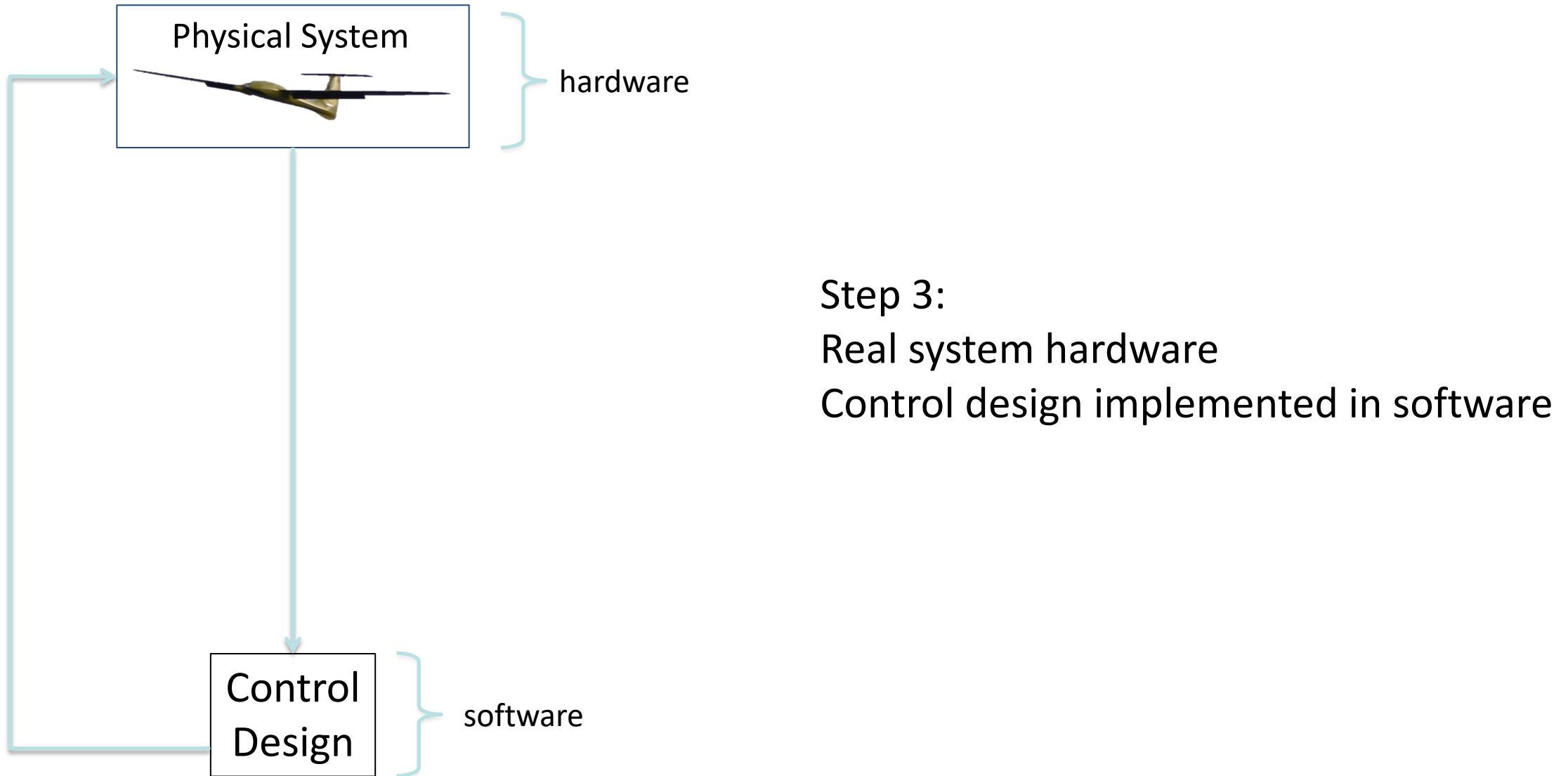
Step 1:
Linear, low-order model
Control design and system model implemented in software

Control Design Process

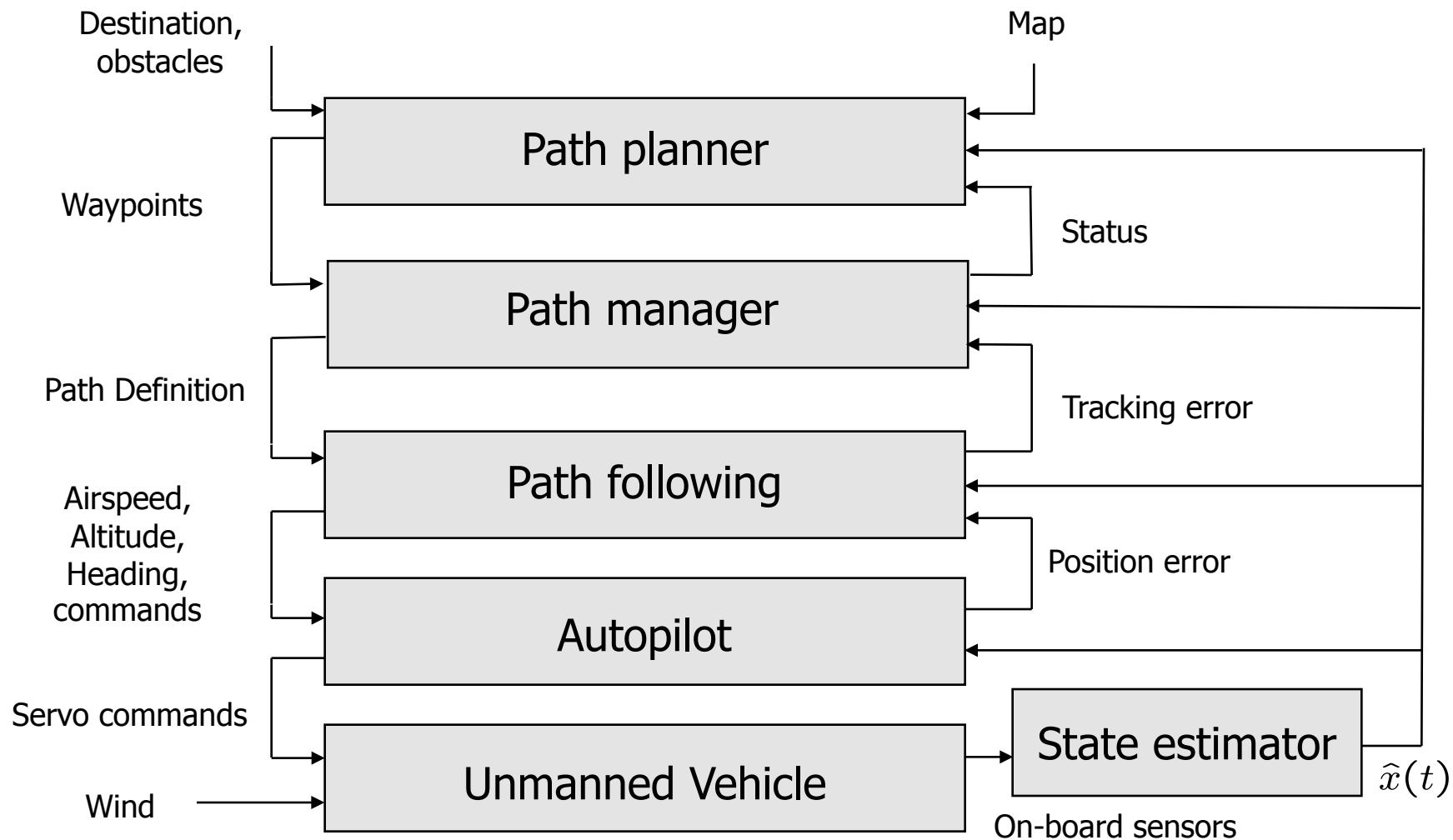


Step 2:
Detailed system model
Control design and system model
implemented in software

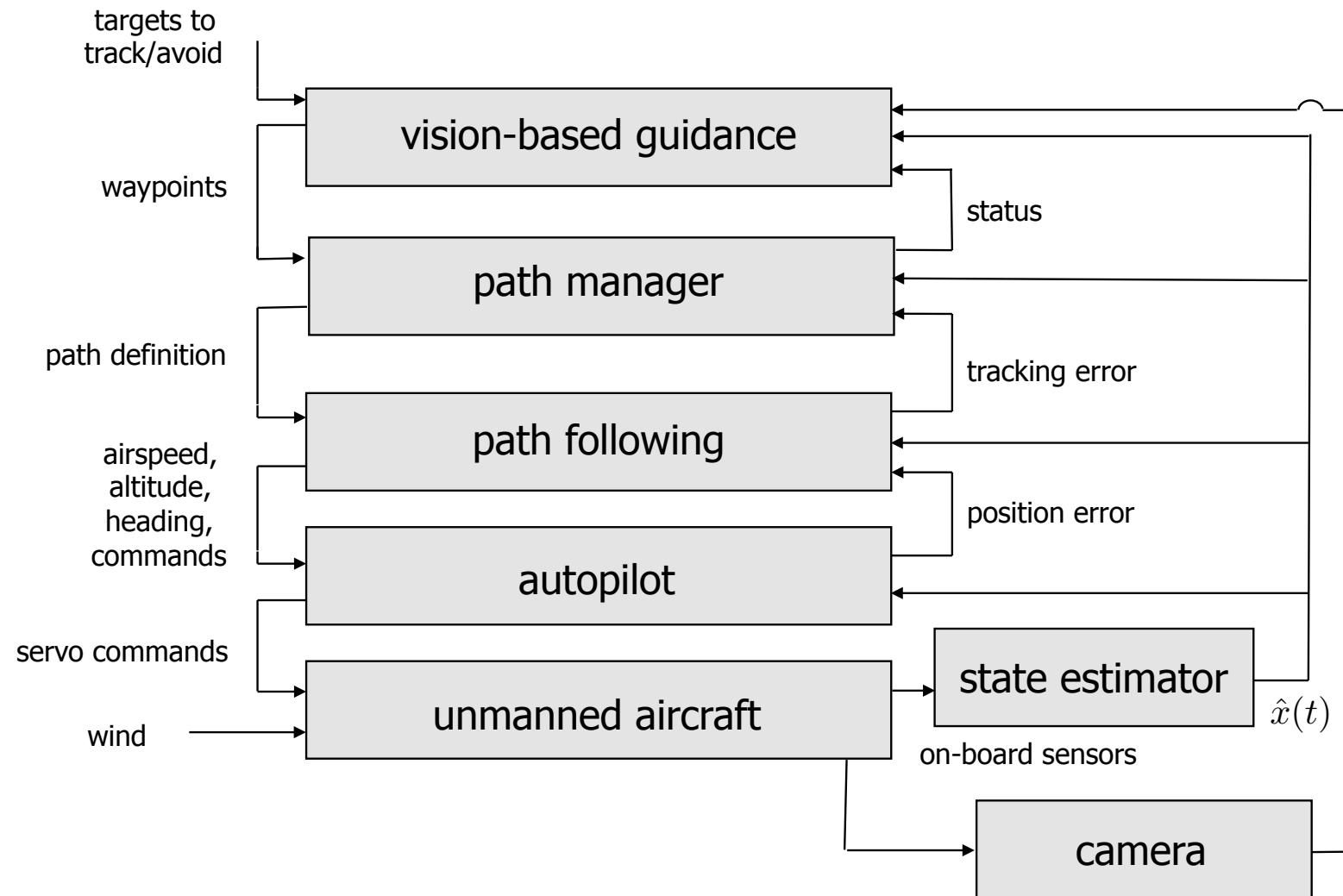
Control Design Process



Architecture



Architecture w/ Camera



Course Project Ideas

- Implement autopilot components on hardware using ROSflight, PixHawk, or another autopilot
- Develop learning modules for the African Drone and Data Academy in Malawi (e.g., Jupyter notebooks)
- Integrate our course simulator into the AirSim simulator
(<https://github.com/microsoft/AirSim>)
- Extend existing autopilot components to do something in a different way (e.g., perform path following using MPC)
- Extend existing autopilot to do something new (e.g., use machine vision to land on a target)
- Something of your own creation that builds on the concepts of this course



Chapter 2

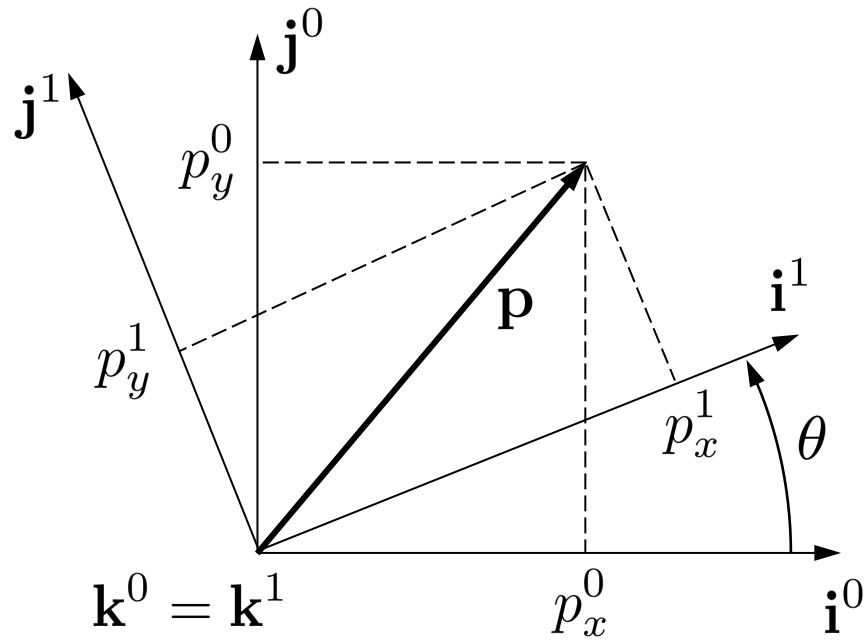
Coordinate Frames

Reference Frames

- In guidance and control of aircraft, reference frames are an essential idea
- Describe relative position and orientation of objects
 - Aircraft relative to direction of wind
 - Camera relative to aircraft
 - Aircraft relative to inertial frame
- Quantities most easily calculated or described in a specific reference frames:
 - Newton's law
 - Aircraft attitude
 - Aerodynamic forces/torques
 - Accelerometers, rate gyros
 - GPS
 - Mission requirements

Must know how to transform
between different reference
frames

Rotation of Reference Frame



Take dot product of both sides –
first with \mathbf{i}^1 , then \mathbf{j}^1 , then \mathbf{k}^1

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

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

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

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

$$\Rightarrow \mathbf{p}^1 = R_0^1 \mathbf{p}^0 \text{ where } R_0^1 \triangleq \begin{pmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

R_0^1 is a rotation about the \mathbf{k} axis.

Rotation of Reference Frame

Right-handed rotation about \mathbf{j} axis:

$$R_0^1 \triangleq \begin{pmatrix} \cos \theta & 0 & -\sin \theta \\ 0 & 1 & 0 \\ \sin \theta & 0 & \cos \theta \end{pmatrix}$$

Right-handed rotation about \mathbf{i} axis:

$$R_0^1 \triangleq \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & \sin \theta \\ 0 & -\sin \theta & \cos \theta \end{pmatrix}$$

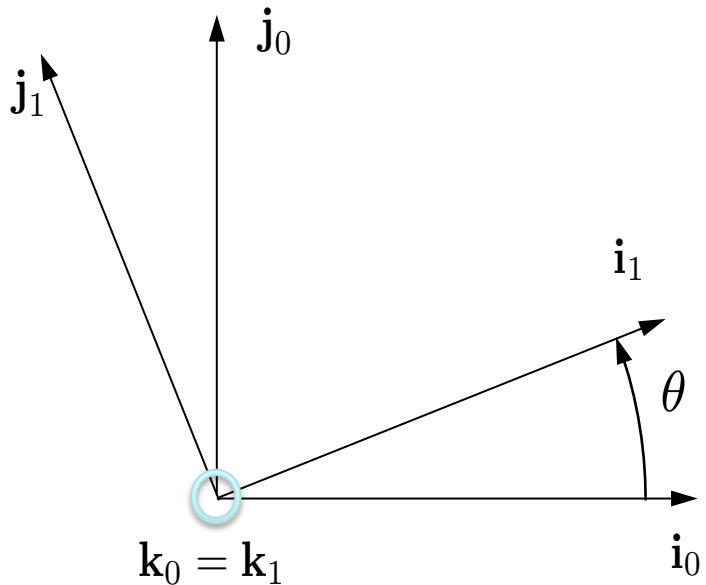
Orthonormal matrix properties:

P.1. $(R_a^b)^{-1} = (R_a^b)^\top = R_b^a$

P.2. $R_b^c R_a^b = R_a^c$

P.3. $\det(R_a^b) = 1$

Rotation of Reference Frame: Alternate View



Note that

$$\mathbf{i}_0^0 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad \mathbf{j}_0^0 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \quad \mathbf{k}_0^0 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

and that

$$\mathbf{i}_1^1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad \mathbf{j}_1^1 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \quad \mathbf{k}_1^1 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

Now express $\mathbf{i}_0, \mathbf{j}_0, \mathbf{k}_0$ in the 1-frame:

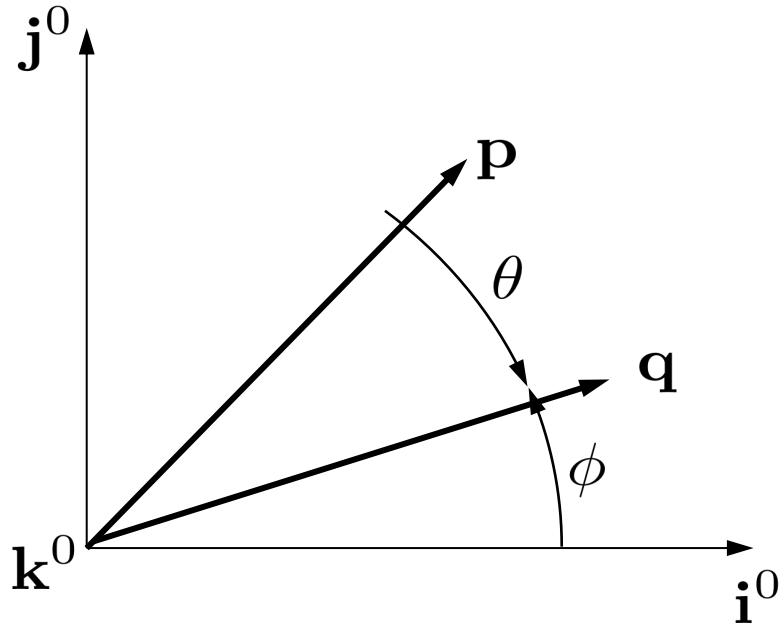
$$\mathbf{i}_0^1 = \begin{pmatrix} \cos \theta \\ -\sin \theta \\ 0 \end{pmatrix}, \quad \mathbf{j}_0^1 = \begin{pmatrix} \sin \theta \\ \cos \theta \\ 0 \end{pmatrix}, \quad \mathbf{k}_0^1 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

Therefore

$$R_0^1 = (\mathbf{i}_0^1 \quad \mathbf{j}_0^1 \quad \mathbf{k}_0^1) = \begin{pmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Rotation of a Vector

Let $p \triangleq |\mathbf{p}| = q \triangleq |\mathbf{q}|$, then



R_0^1 can be interpreted as a left-handed rotation of a vector by an angle θ

$$\begin{aligned}\mathbf{p} &= \begin{pmatrix} p \cos(\theta + \phi) \\ p \sin(\theta + \phi) \\ 0 \end{pmatrix} \\ &= \begin{pmatrix} p \cos \theta \cos \phi - p \sin \theta \sin \phi \\ p \sin \theta \cos \phi + p \cos \theta \sin \phi \\ 0 \end{pmatrix} \\ &= \begin{pmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p \cos \phi \\ p \sin \phi \\ 0 \end{pmatrix}\end{aligned}$$

Define

$$\mathbf{q} = \begin{pmatrix} p \cos \phi \\ p \sin \phi \\ 0 \end{pmatrix}$$

then

$$\mathbf{p} = (R_0^1)^\top \mathbf{q} \quad \Rightarrow \quad \mathbf{q} = R_0^1 \mathbf{p}$$

Passive vs. Active Rotation

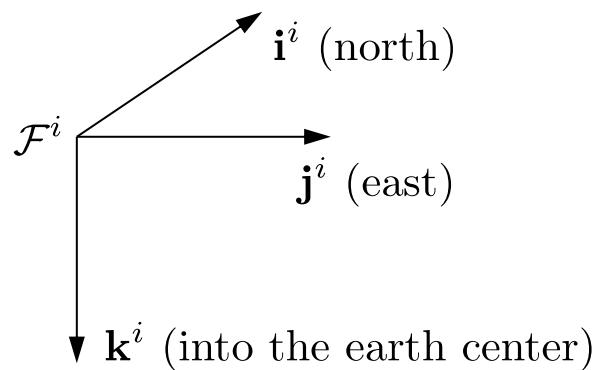
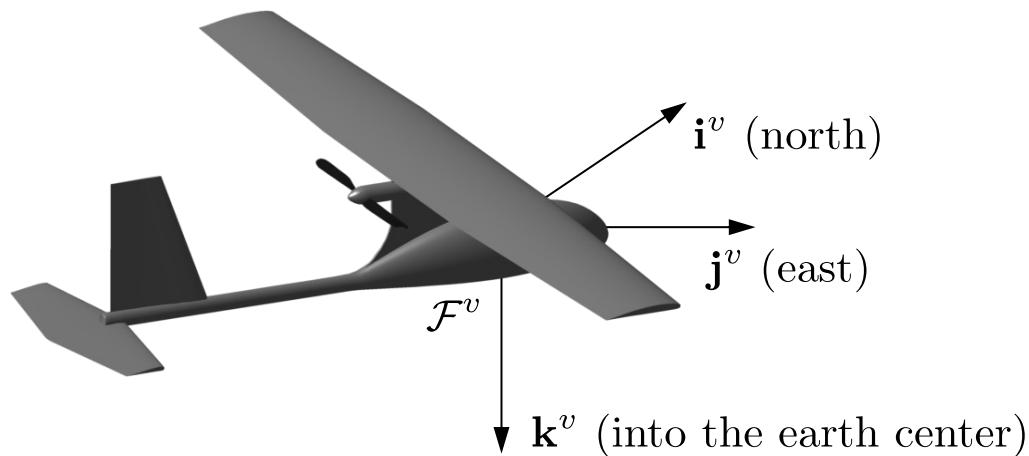
Passive Rotation/Transformation: The vector \mathbf{p} is stationary and the coordinate frame changes

Active Rotation/Transformation: The coordinate frame remains fixed, but the vector is rotated

R_0^1 as defined earlier, represents a right-handed passive transformation, or a left-handed active transformation

If we stick with right-handed rotations, then R_0^1 represents a right-handed passive rotation, and $R_0^{1\top}$ represents a right-handed active rotation

Inertial Frame and Vehicle Frame



- Vehicle frame has same orientation as inertial frame
- Vehicle frame is fixed at cm of aircraft
- Inertial and vehicle frames are referred to as NED frames
- $N \rightarrow x, E \rightarrow y, D \rightarrow z$

Euler Angles

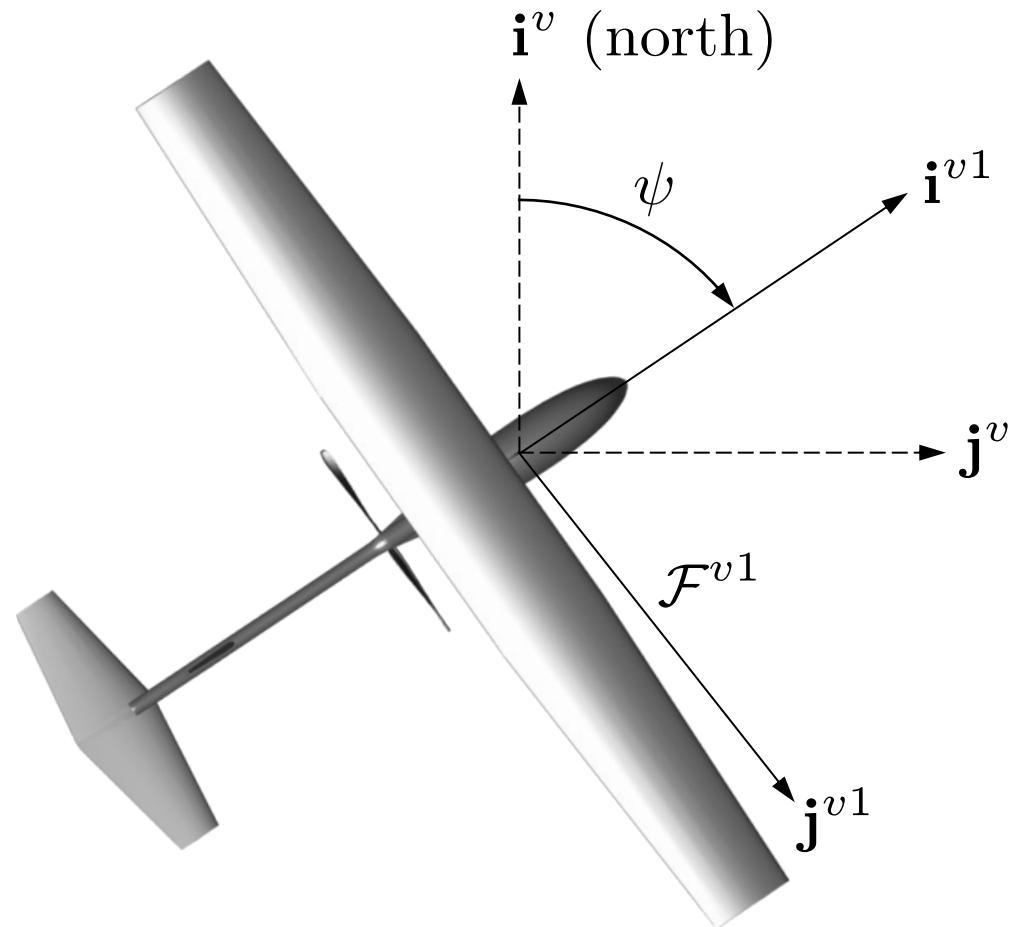
- Need way to describe attitude of aircraft
- Common approach: Euler angles

ψ : heading (yaw)

θ : elevation (pitch)

- Pro: Intuitive ϕ : bank (roll)
- Con: Mathematical singularity
 - Quaternions are alternative for overcoming singularity

Vehicle-1 Frame



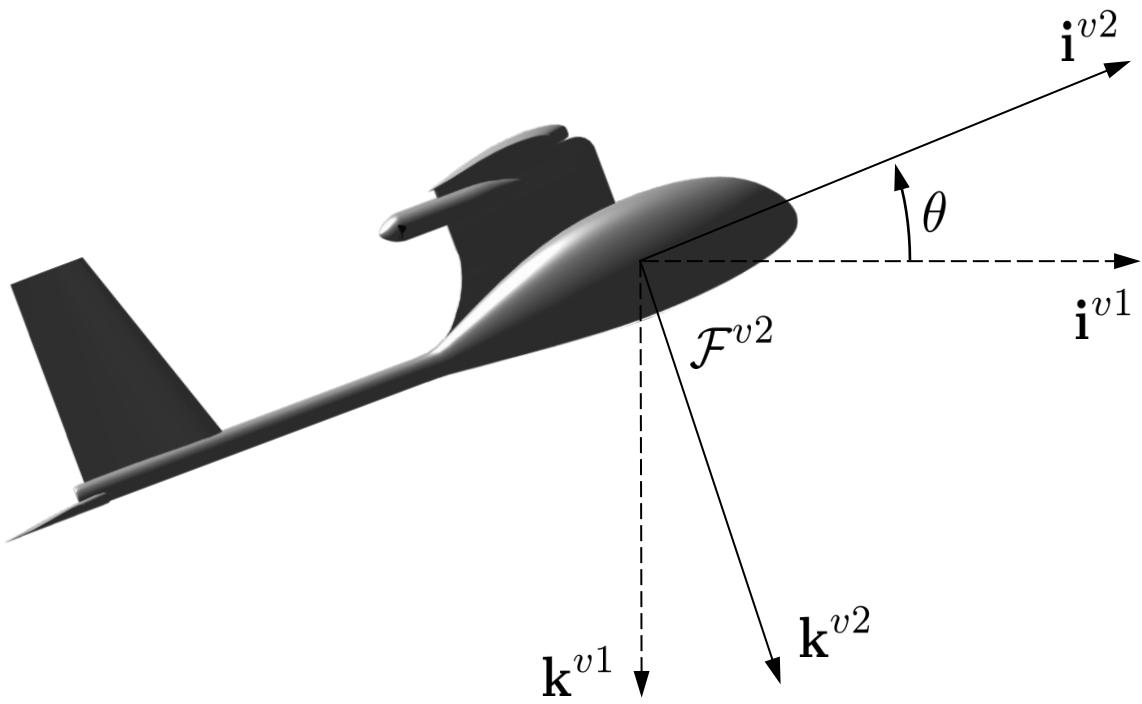
$$R_v^{v1}(\psi) = \begin{pmatrix} \cos \psi & \sin \psi & 0 \\ -\sin \psi & \cos \psi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

gives

$$\mathbf{p}^{v1} = R_v^{v1}(\psi)\mathbf{p}^v$$

where ψ is the heading

Vehicle-2 Frame



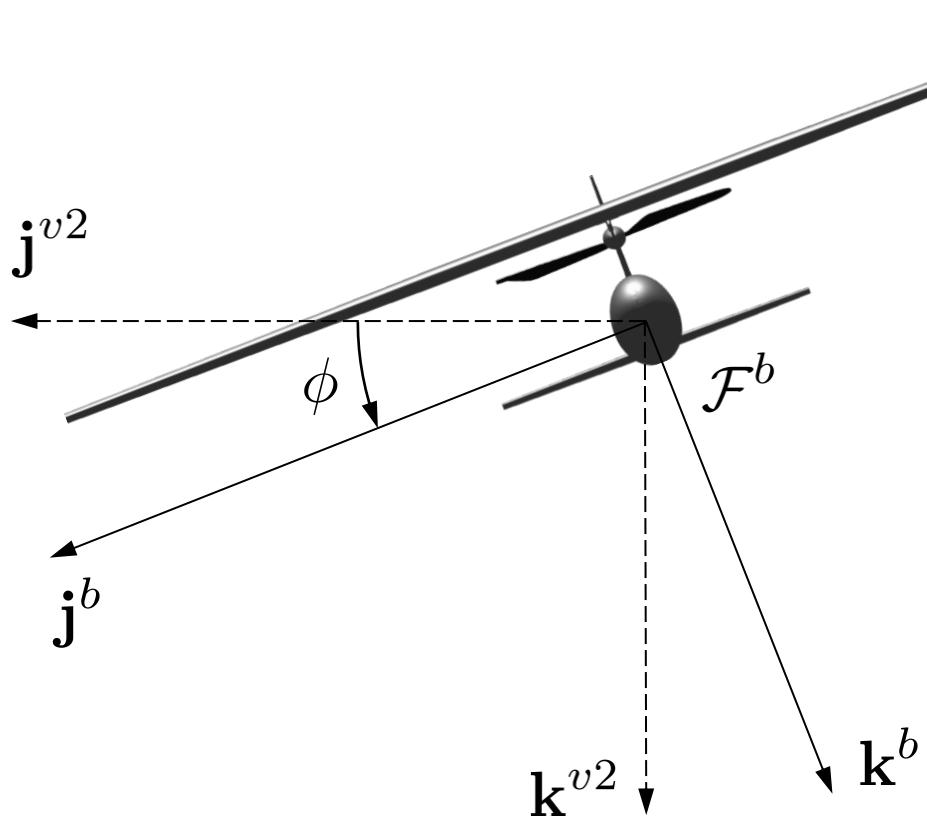
$$R_{v1}^{v2}(\theta) = \begin{pmatrix} \cos \theta & 0 & -\sin \theta \\ 0 & 1 & 0 \\ \sin \theta & 0 & \cos \theta \end{pmatrix}$$

gives

$$\mathbf{p}^{v2} = R_{v1}^{v2}(\theta)\mathbf{p}^{v1}$$

where θ is the pitch angle

Body Frame



$$R_{v2}^b(\phi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \phi & \sin \phi \\ 0 & -\sin \phi & \cos \phi \end{pmatrix}$$

gives

$$\mathbf{p}^b = R_{v2}^b(\phi)\mathbf{p}^{v2}$$

where ϕ is the roll (bank) angle

Inertial Frame to Body Frame Transformation

Define

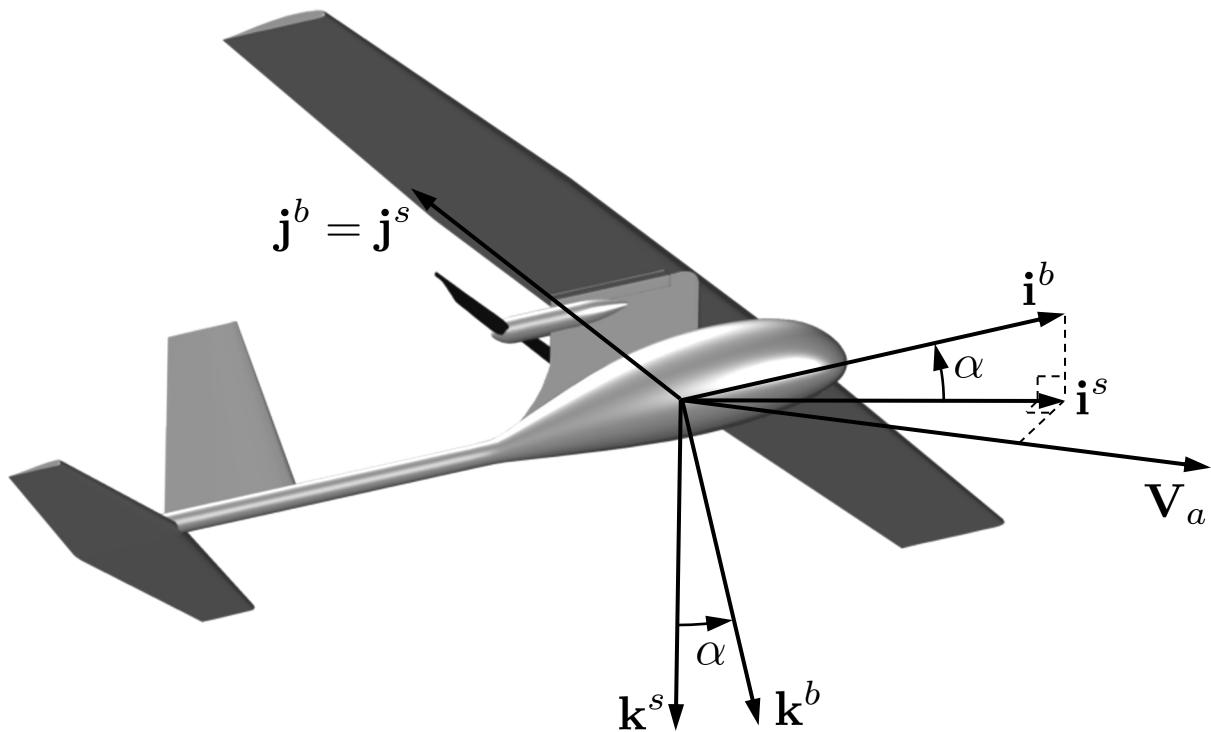
$$\begin{aligned} R_v^b(\phi, \theta, \psi) &= R_{v2}^b(\phi) R_{v1}^{v2}(\theta) R_v^{v1}(\psi) \\ &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \phi & \sin \phi \\ 0 & -\sin \phi & \cos \phi \end{pmatrix} \begin{pmatrix} \cos \theta & 0 & -\sin \theta \\ 0 & 1 & 0 \\ \sin \theta & 0 & \cos \theta \end{pmatrix} \begin{pmatrix} \cos \psi & \sin \psi & 0 \\ -\sin \psi & \cos \psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \\ &= \begin{pmatrix} c_\theta c_\psi & c_\theta s_\psi & -s_\theta \\ s_\phi s_\theta c_\psi - c_\phi s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & s_\phi c_\theta \\ c_\phi s_\theta c_\psi + s_\phi s_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi & c_\phi c_\theta \end{pmatrix} \end{aligned}$$

to give

$$\mathbf{p}^b = R_v^b(\Theta) \mathbf{p}^v$$

where, $\Theta = (\phi, \theta, \psi)^\top$

Stability Frame



Right-handed rotation

$$R_s^b(\alpha) = \begin{pmatrix} \cos \alpha & 0 & -\sin \alpha \\ 0 & 1 & 0 \\ \sin \alpha & 0 & \cos \alpha \end{pmatrix}$$

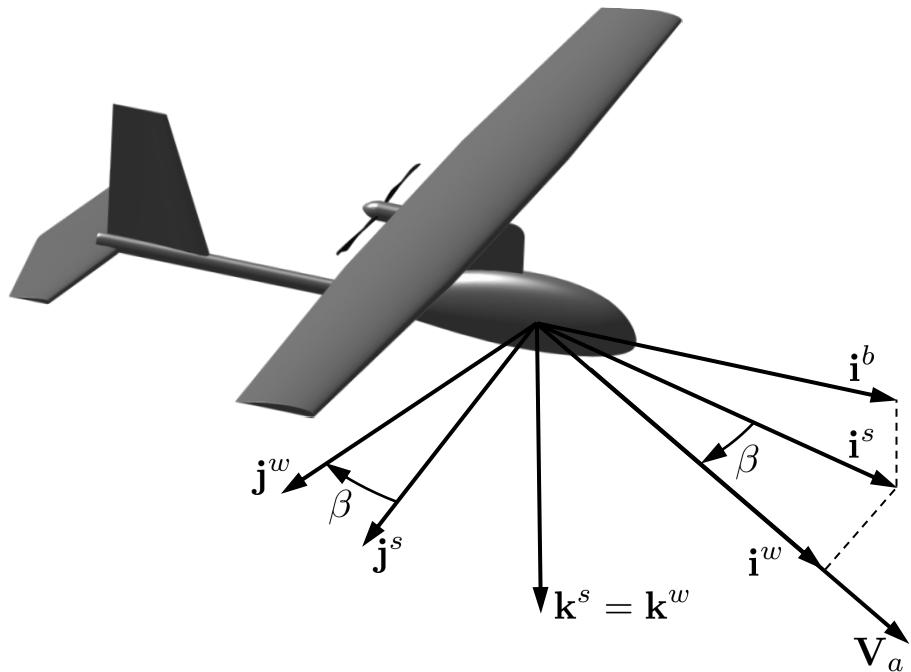
$$\mathbf{p}^b = R_s^b(\alpha)\mathbf{p}^s$$

$$\mathbf{p}^s = R_b^s(\alpha)\mathbf{p}^b = R_s^b(\alpha)^\top \mathbf{p}^b$$

Stability frame helps us rigorously define angle of attack and is useful for analyzing stability of aircraft

Angle of attack defined as a positive RH rotation from stability to body frame

Wind Frame



$$R_s^w(\beta) = \begin{pmatrix} \cos \beta & \sin \beta & 0 \\ -\sin \beta & \cos \beta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

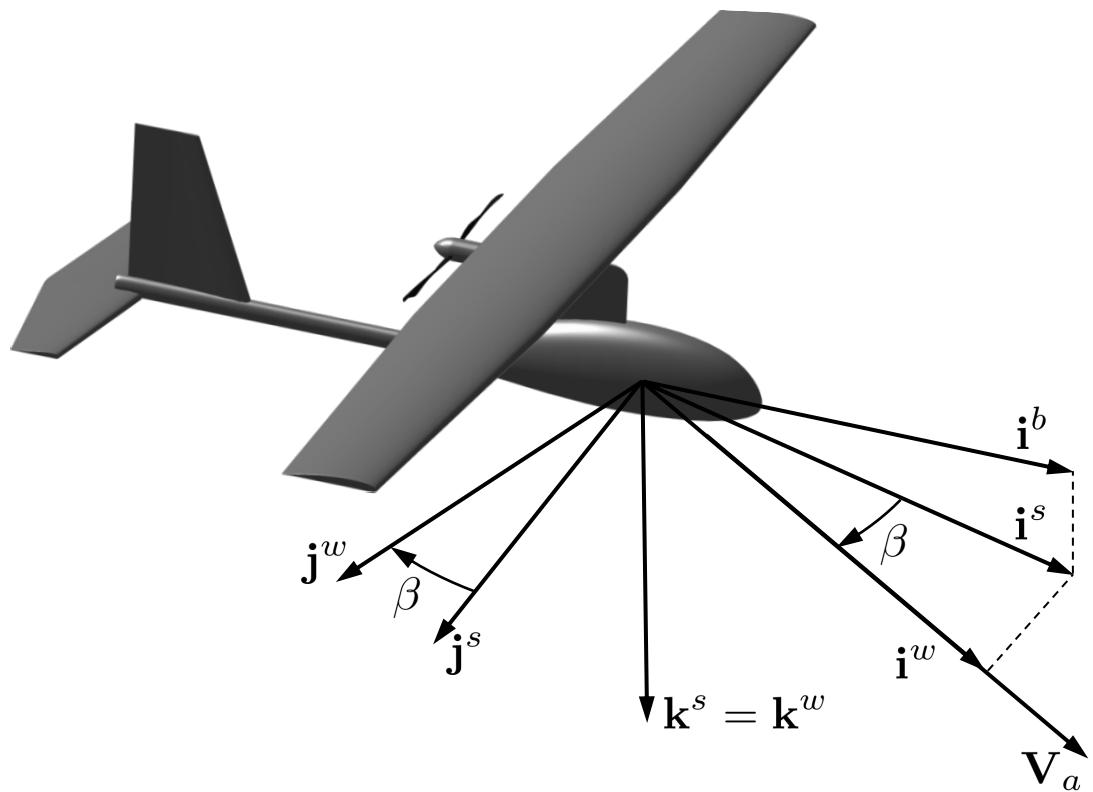
$$\mathbf{p}^w = R_s^w(\beta)\mathbf{p}^s$$

$$R_b^w(\alpha, \beta) = R_s^w(\beta)R_b^s(\alpha)$$

$$= \begin{pmatrix} \cos \beta & \sin \beta & 0 \\ -\sin \beta & \cos \beta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos \alpha & 0 & \sin \alpha \\ 0 & 1 & 0 \\ -\sin \alpha & 0 & \cos \alpha \end{pmatrix}$$

$$= \begin{pmatrix} \cos \beta \cos \alpha & \sin \beta & \cos \beta \sin \alpha \\ -\sin \beta \cos \alpha & \cos \beta & -\sin \beta \sin \alpha \\ -\sin \alpha & 0 & \cos \alpha \end{pmatrix}$$

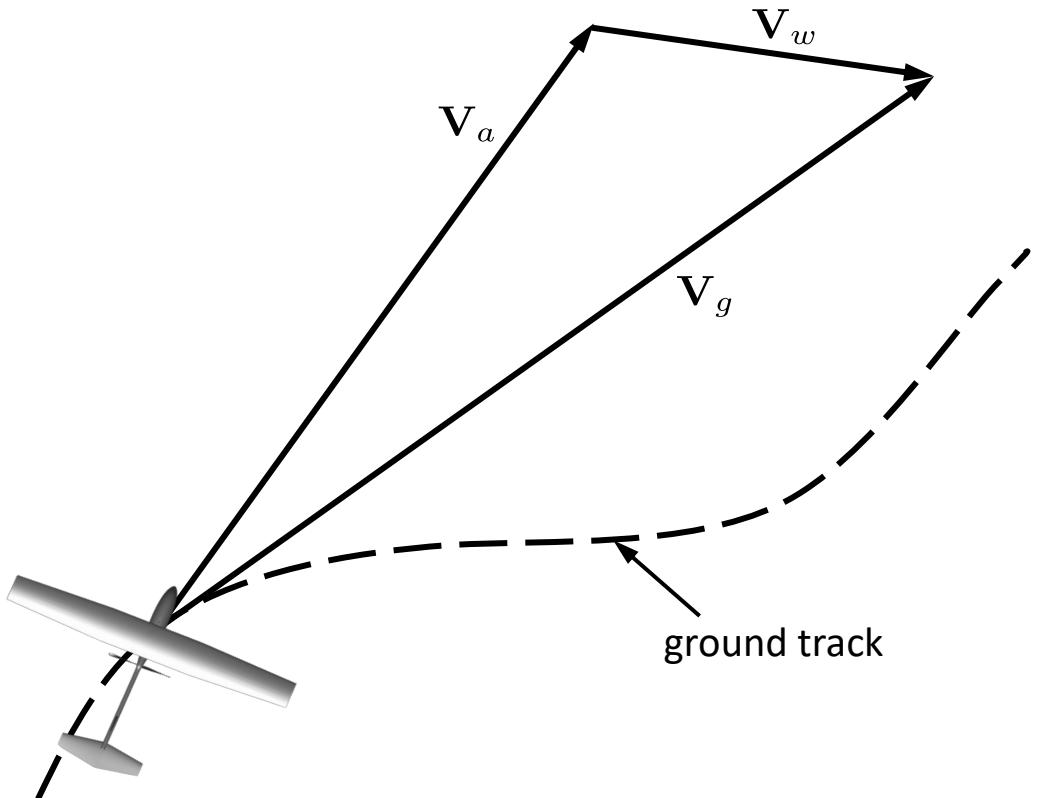
Wind Frame (continued)



- Wind frame helps us rigorously define side-slip angle
- Forces and moments are most naturally defined in wind frame
- Side-slip angle is nominally zero for tailed aircraft

$$R_w^b(\alpha, \beta) = (R_b^w)^\top(\alpha, \beta) = \begin{pmatrix} \cos \beta \cos \alpha & -\sin \beta \cos \alpha & -\sin \alpha \\ \sin \beta & \cos \beta & 0 \\ \cos \beta \sin \alpha & -\sin \beta \sin \alpha & \cos \alpha \end{pmatrix}$$

Airspeed, Wind Speed, Ground Speed



a/c wrt to surrounding air
expressed in body frame

$$\mathbf{V}_a = \mathbf{V}_g - \mathbf{V}_w$$

$$\mathbf{V}_g^b = \begin{pmatrix} u \\ v \\ w \end{pmatrix}$$

a/c wrt to inertial frame
expressed in body frame

$$\mathbf{V}_w^b = \begin{pmatrix} u_w \\ v_w \\ w_w \end{pmatrix} = R_v^b(\phi, \theta, \psi) \begin{pmatrix} w_n \\ w_e \\ w_d \end{pmatrix}$$

$$\mathbf{V}_a^w = \begin{pmatrix} V_a \\ 0 \\ 0 \end{pmatrix}$$

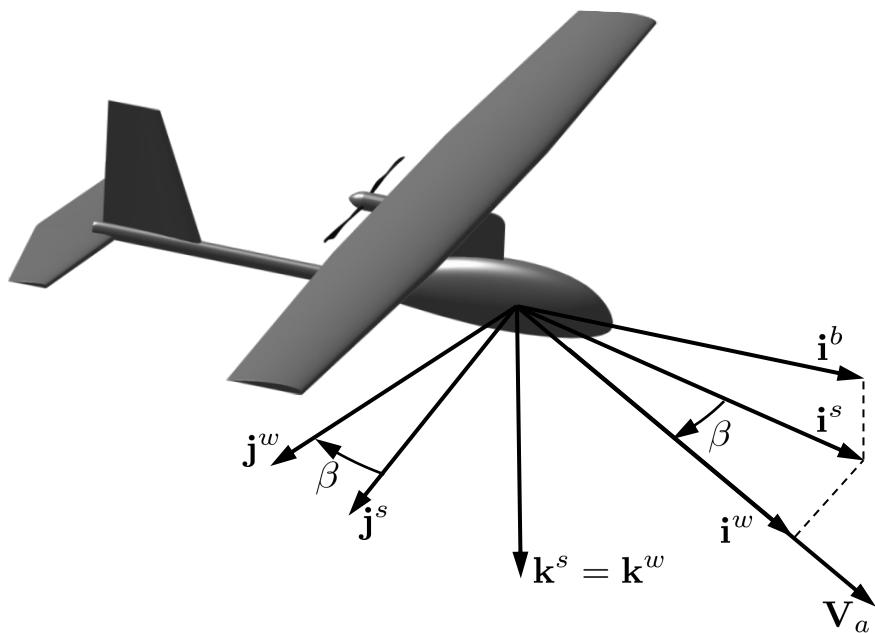
$$\mathbf{V}_a^b = \begin{pmatrix} u_r \\ v_r \\ w_r \end{pmatrix} = \begin{pmatrix} u - u_w \\ v - v_w \\ w - w_w \end{pmatrix}$$

Airspeed, Angle of Attack, Sideslip Angle

$$\mathbf{V}_a^b = \begin{pmatrix} u_r \\ v_r \\ w_r \end{pmatrix} = R_w^b \begin{pmatrix} V_a \\ 0 \\ 0 \end{pmatrix}$$

$$= \begin{pmatrix} \cos \beta \cos \alpha & -\sin \beta \cos \alpha & -\sin \alpha \\ \sin \beta & \cos \beta & -\sin \beta \sin \alpha \\ \cos \beta \sin \alpha & 0 & \cos \alpha \end{pmatrix} \begin{pmatrix} V_a \\ 0 \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} u_r \\ v_r \\ w_r \end{pmatrix} = V_a \begin{pmatrix} \cos \alpha \cos \beta \\ \sin \beta \\ \sin \alpha \cos \beta \end{pmatrix}$$



OR

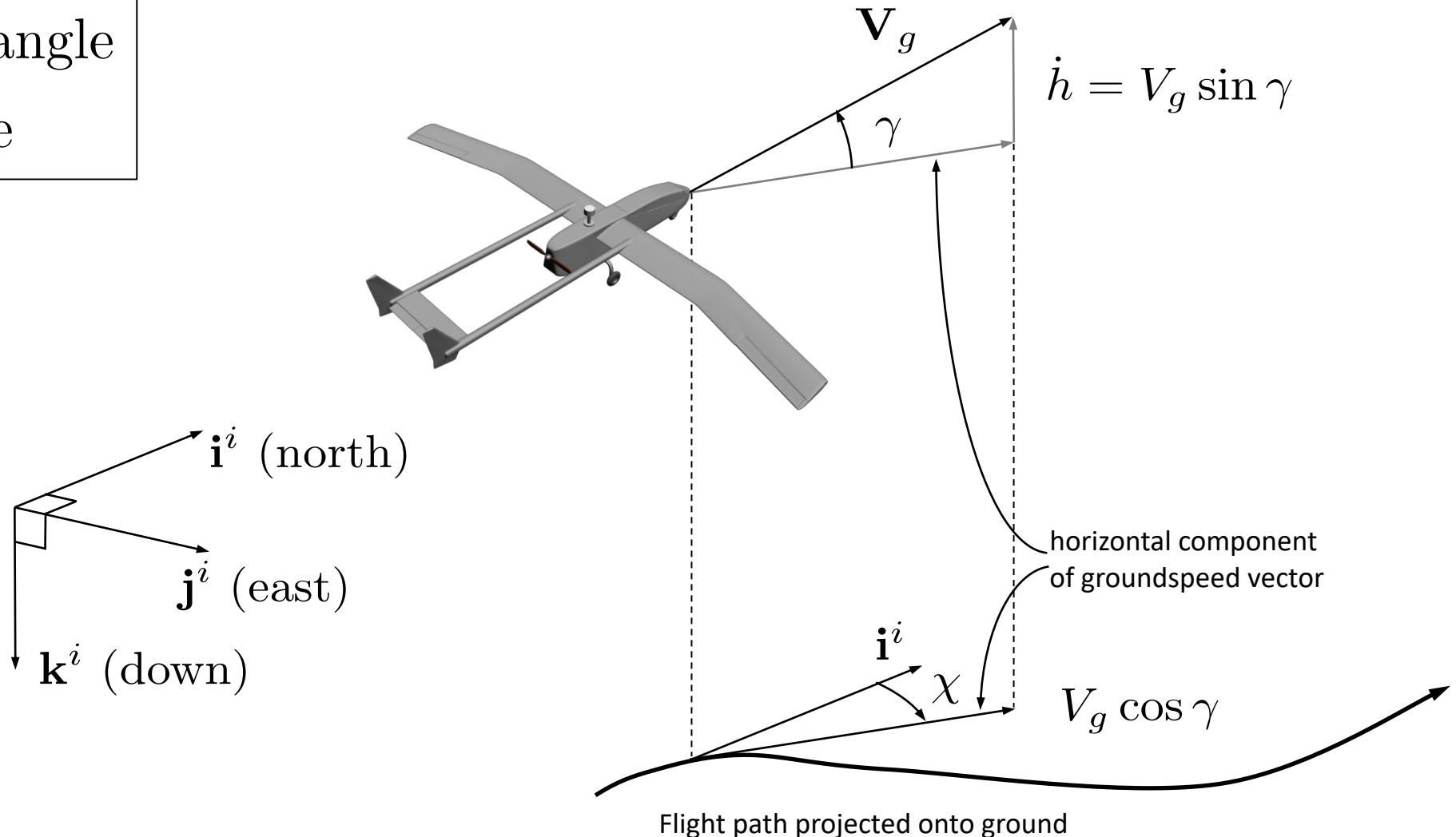
$$V_a = \sqrt{u_r^2 + v_r^2 + w_r^2}$$

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

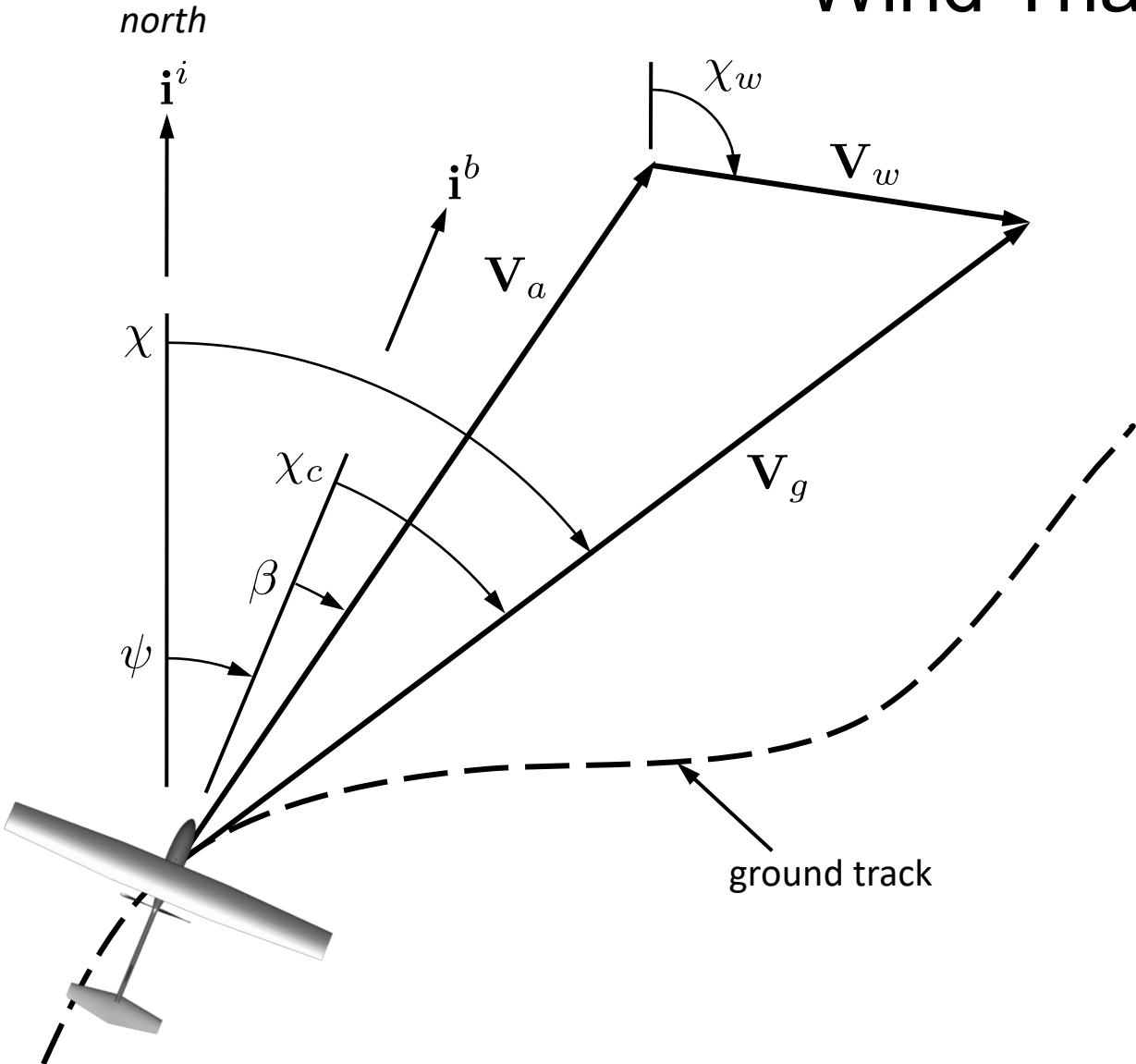
$$\beta = \tan^{-1} \left(\frac{v_r}{\sqrt{u_r^2 + w_r^2}} \right)$$

Course and Flight Path Angles

γ : flight path angle
 χ : course angle



Wind Triangle



χ : course angle

ψ : heading angle

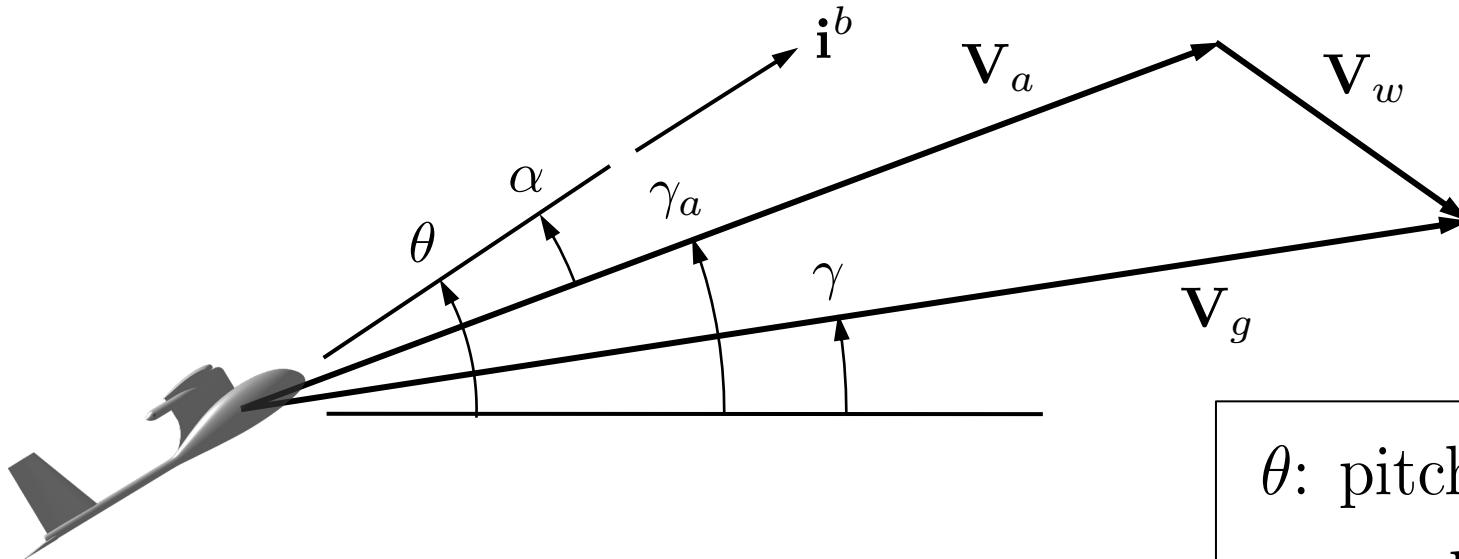
β : side slip angle

χ_c : crab angle

χ_w : wind direction

$$\chi_c \stackrel{\triangle}{=} \chi - \psi$$

Wind Triangle



θ : pitch angle

α : angle of attack

γ : flight path angle

γ_a : air-mass-relative flight path angle

$$\gamma_a = \theta - \alpha$$

When wind speed and sideslip are zero...

If both the windspeed and the sideslip angles are zero, i.e.,

$$V_w = 0$$

$$\beta = 0$$

then we have the following simplifications

$V_a = V_g$ Airspeed equals groundspeed

$u = u_r$ Velocity equals velocity relative to the air mass

$v = v_r$

$w = w_r$

$\psi = \chi$ Heading equals course

$\gamma = \gamma_a$ Flight path angle equals air-mass-referenced flight path angle

Differentiation of a Vector

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

$$\frac{d}{dt_i} \mathbf{p} = \dot{p}_x \mathbf{i}^b + \dot{p}_y \mathbf{j}^b + \dot{p}_z \mathbf{k}^b + p_x \frac{d}{dt_i} \mathbf{i}^b + p_y \frac{d}{dt_i} \mathbf{j}^b + p_z \frac{d}{dt_i} \mathbf{k}^b$$

$$\frac{d}{dt_b} \mathbf{p} = \dot{p}_x \mathbf{i}^b + \dot{p}_y \mathbf{j}^b + \dot{p}_z \mathbf{k}^b$$

$$\dot{\mathbf{i}}^b = \boldsymbol{\omega}_{b/i} \times \mathbf{i}^b$$

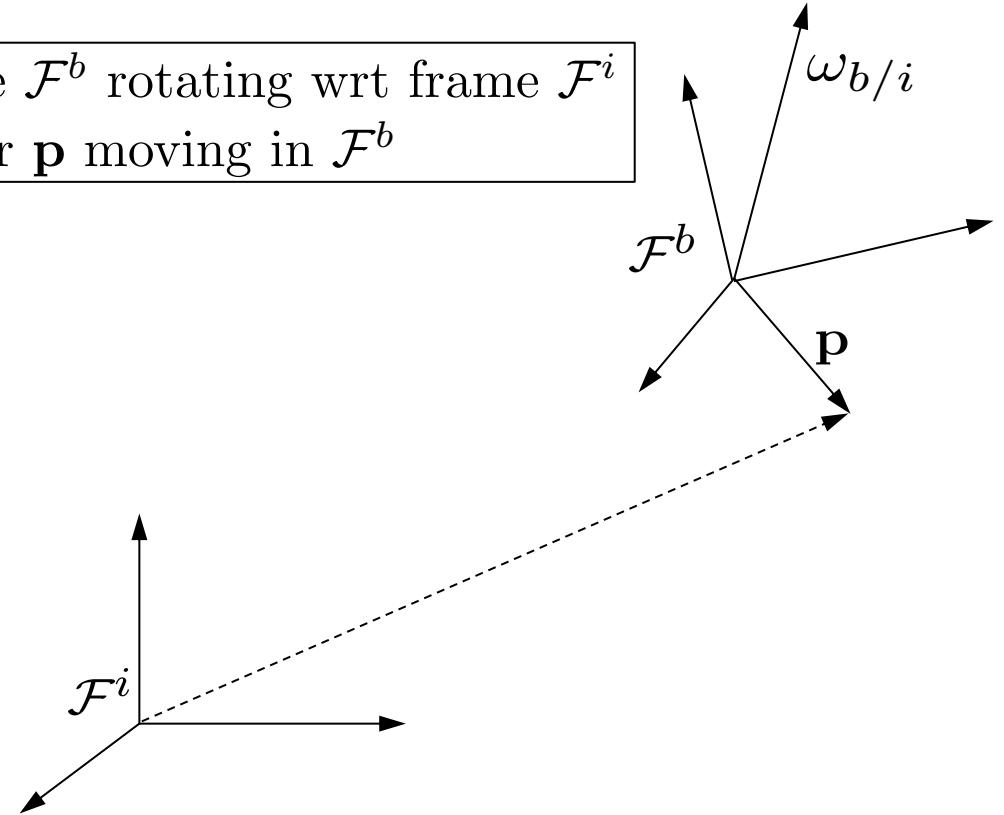
$$\dot{\mathbf{j}}^b = \boldsymbol{\omega}_{b/i} \times \mathbf{j}^b$$

$$\dot{\mathbf{k}}^b = \boldsymbol{\omega}_{b/i} \times \mathbf{k}^b$$

$$\begin{aligned} p_x \dot{\mathbf{i}}^b + p_y \dot{\mathbf{j}}^b + p_z \dot{\mathbf{k}}^b &= p_x (\boldsymbol{\omega}_{b/i} \times \mathbf{i}^b) + p_y (\boldsymbol{\omega}_{b/i} \times \mathbf{j}^b) + p_z (\boldsymbol{\omega}_{b/i} \times \mathbf{k}^b) \\ &= \boldsymbol{\omega}_{b/i} \times \mathbf{p} \end{aligned}$$

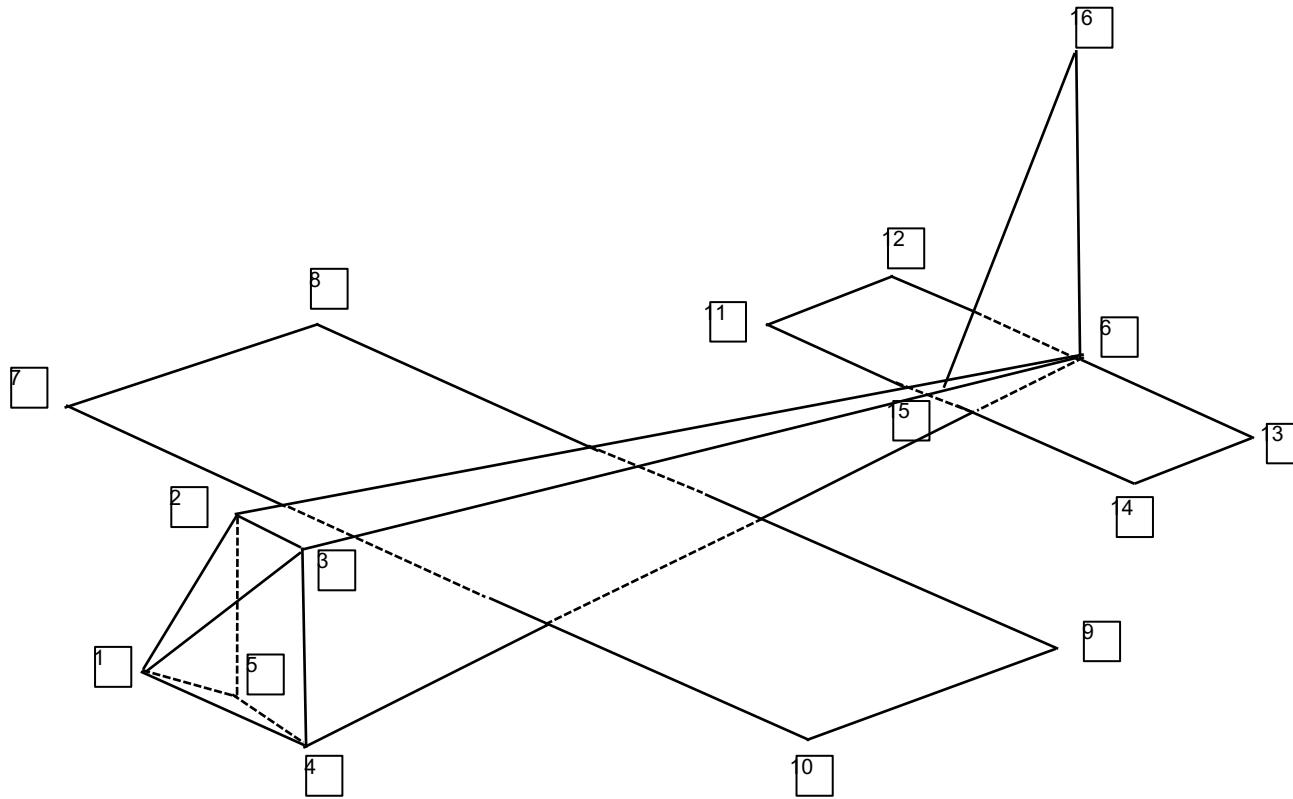
$$\frac{d}{dt_i} \mathbf{p} = \frac{d}{dt_b} \mathbf{p} + \boldsymbol{\omega}_{b/i} \times \mathbf{p}$$

Frame \mathcal{F}^b rotating wrt frame \mathcal{F}^i
Vector \mathbf{p} moving in \mathcal{F}^b

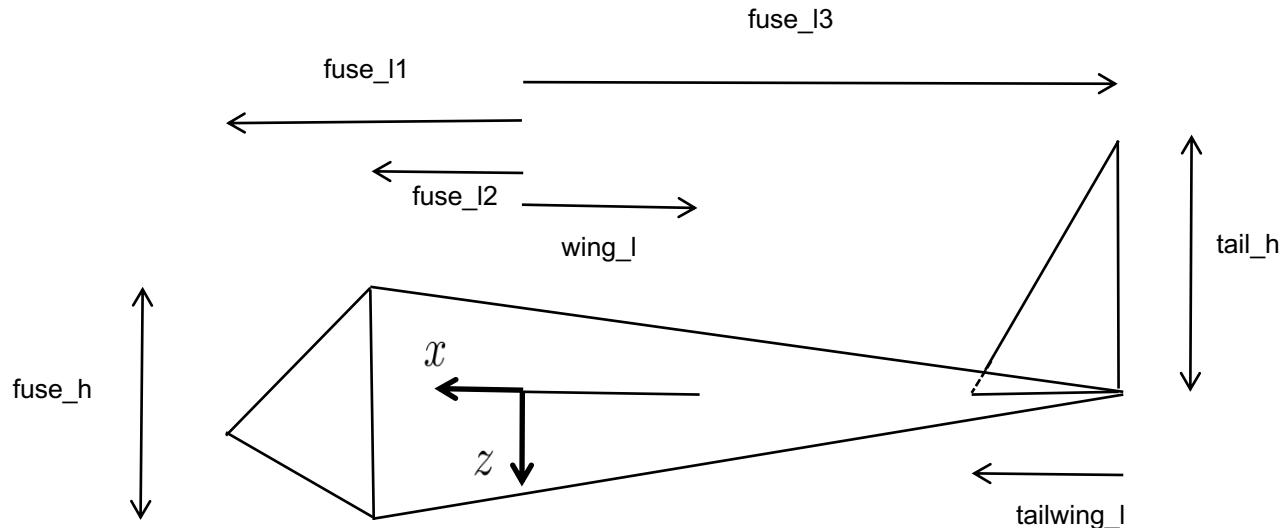


Note: Frame \mathcal{F}^b does not translate
wrt frame \mathcal{F}^i

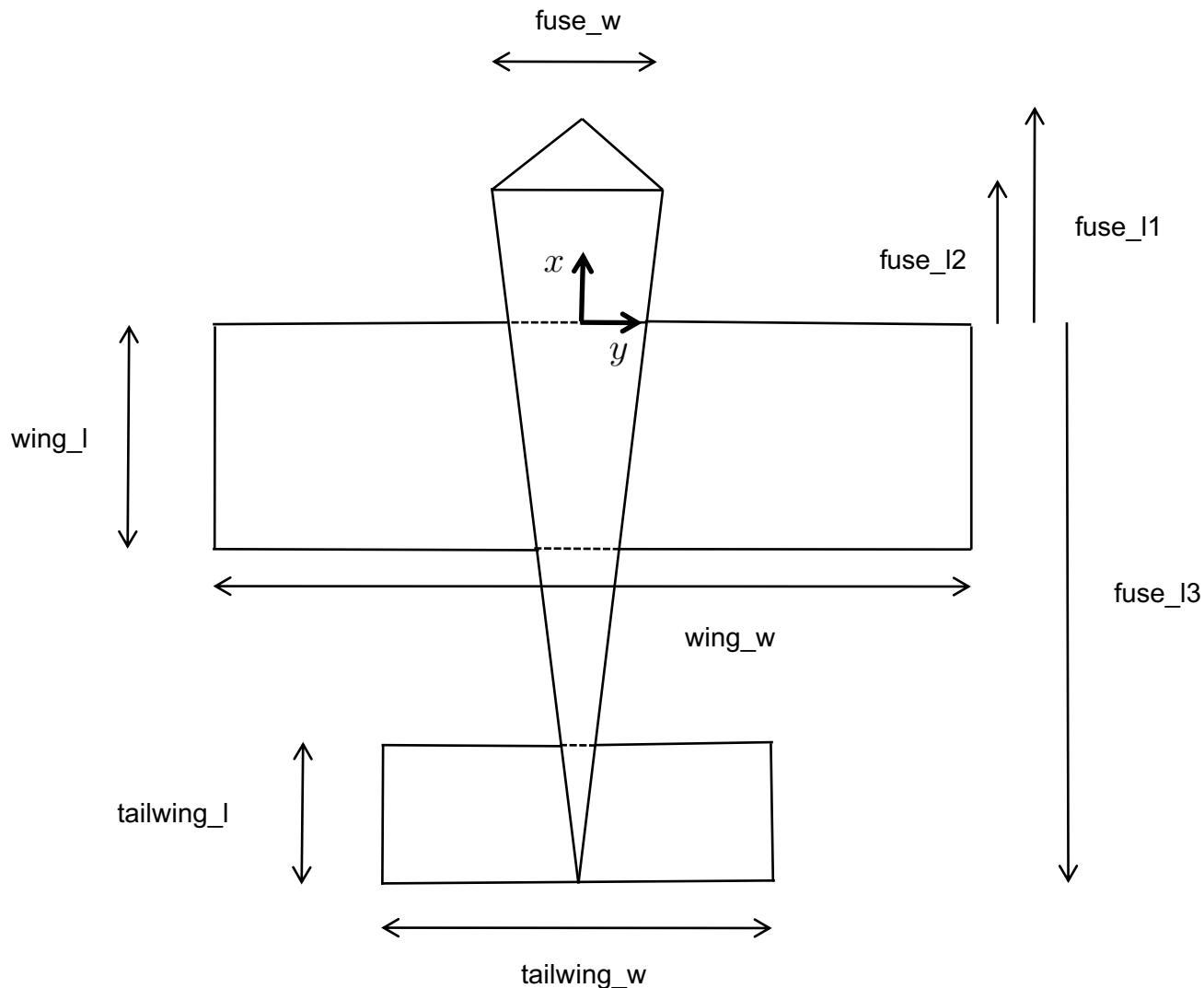
Project Aircraft



Project Aircraft



Project Aircraft

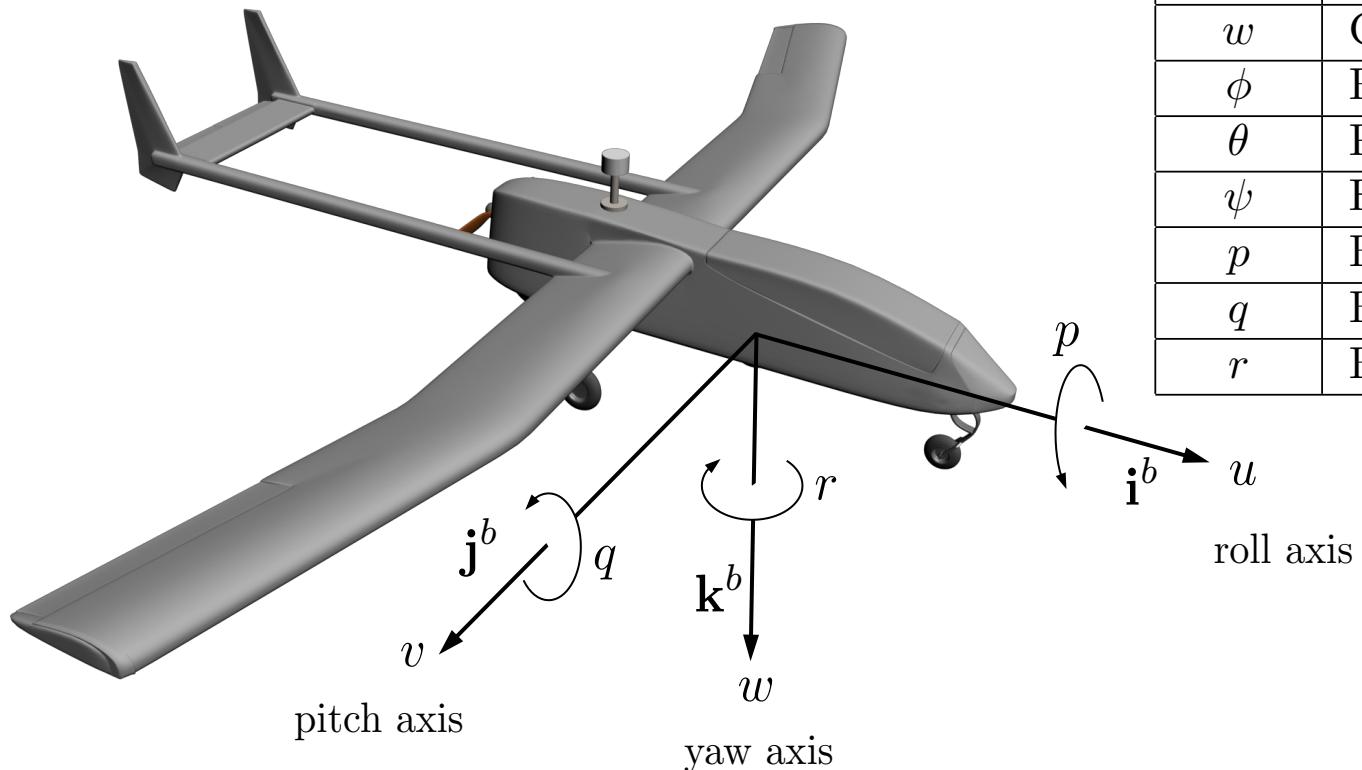




Chapter 3

Kinematics and Dynamics

Aircraft State Variables



Name	Description
p_n	Inertial north position of MAV expressed along \mathbf{i}^i in \mathcal{F}^i
p_e	Inertial east position of MAV expressed along \mathbf{j}^i in \mathcal{F}^i
p_d	Inertial down position of MAV expressed along \mathbf{k}^i in \mathcal{F}^i
u	Ground velocity expressed along \mathbf{i}^b in \mathcal{F}^b
v	Ground velocity expressed along \mathbf{j}^b in \mathcal{F}^b
w	Ground velocity expressed along \mathbf{k}^b in \mathcal{F}^b
ϕ	Roll angle defined with respect to \mathcal{F}^v^2
θ	Pitch angle defined with respect to \mathcal{F}^v^1
ψ	Heading (yaw) angle defined with respect to \mathcal{F}^v
p	Body angular (roll) rate expressed along \mathbf{i}^b in \mathcal{F}^b
q	Body angular (pitch) rate expressed along \mathbf{j}^b in \mathcal{F}^b
r	Body angular (yaw) rate expressed along \mathbf{k}^b in \mathcal{F}^b

Translational Kinematics

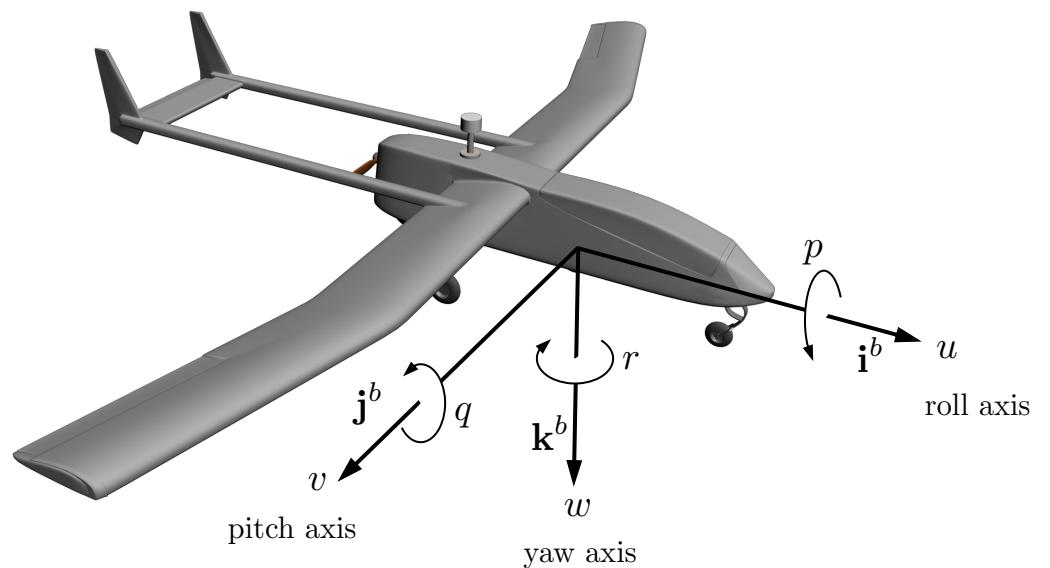
$$\begin{aligned}\begin{pmatrix} \dot{p}_n \\ \dot{p}_e \\ \dot{p}_d \end{pmatrix} &\triangleq \frac{d}{dt} \begin{pmatrix} p_n \\ p_e \\ p_d \end{pmatrix} = \mathbf{v}^v = R_b^v \mathbf{v}^b \\ &= R_b^v \begin{pmatrix} u \\ v \\ w \end{pmatrix} = (R_v^b)^\top \begin{pmatrix} u \\ v \\ w \end{pmatrix} \\ &= \begin{pmatrix} c_\theta c_\psi & s_\phi s_\theta c_\psi - c_\phi s_\psi & c_\phi s_\theta c_\psi + s_\phi s_\psi \\ c_\theta s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi \\ -s_\theta & s_\phi c_\theta & c_\phi c_\theta \end{pmatrix} \begin{pmatrix} u \\ v \\ w \end{pmatrix}\end{aligned}$$

Rotational Kinematics

$$\begin{aligned}
 \begin{pmatrix} p \\ q \\ r \end{pmatrix} &= \underbrace{\begin{pmatrix} \dot{\phi} \\ 0 \\ 0 \end{pmatrix}}_{\dot{\phi} \text{ is defined in } \mathcal{F}^b} + \underbrace{R_{v2}^b(\phi) \begin{pmatrix} 0 \\ \dot{\theta} \\ 0 \end{pmatrix}}_{\dot{\theta} \text{ is defined in } \mathcal{F}^{v^2}} + \underbrace{R_{v2}^b(\phi) R_{v1}^{v^2}(\theta) \begin{pmatrix} 0 \\ 0 \\ \dot{\psi} \end{pmatrix}}_{\dot{\psi} \text{ is defined in } \mathcal{F}^{v^1}} \\
 &= \begin{pmatrix} \dot{\phi} \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \phi & \sin \phi \\ 0 & -\sin \phi & \cos \phi \end{pmatrix} \begin{pmatrix} 0 \\ \dot{\theta} \\ 0 \end{pmatrix} + \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \phi & \sin \phi \\ 0 & -\sin \phi & \cos \phi \end{pmatrix} \begin{pmatrix} \cos \theta & 0 & -\sin \theta \\ 0 & 1 & 0 \\ \sin \theta & 0 & \cos \theta \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ \dot{\psi} \end{pmatrix} \\
 &= \begin{pmatrix} 1 & 0 & -\sin \theta \\ 0 & \cos \phi & \sin \phi \cos \theta \\ 0 & -\sin \phi & \cos \phi \cos \theta \end{pmatrix} \begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix}
 \end{aligned}$$

Inverting gives:

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



Kinematic Equations of Motion

Six of the 12 state equations for the MAV come from the kinematic equations relating positions and velocities:

$$\begin{pmatrix} \dot{p}_n \\ \dot{p}_e \\ \dot{p}_d \end{pmatrix} = \begin{pmatrix} c_\theta c_\psi & s_\phi s_\theta c_\psi - c_\phi s_\psi & c_\phi s_\theta c_\psi + s_\phi s_\psi \\ c_\theta s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi \\ -s_\theta & s_\phi c_\theta & c_\phi c_\theta \end{pmatrix} \begin{pmatrix} u \\ v \\ w \end{pmatrix}$$

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

The remaining six equations will come from applying Newton's 2nd law to the translational and rotational motion of the aircraft

Differentiation of a Vector in Two Reference Frames

Define the vector \mathbf{p} in terms of the axes in the body frame:

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

Frame \mathcal{F}^b rotating wrt frame \mathcal{F}^i
Vector \mathbf{p} moving in \mathcal{F}^b

Differentiation with respect to the inertial frame gives

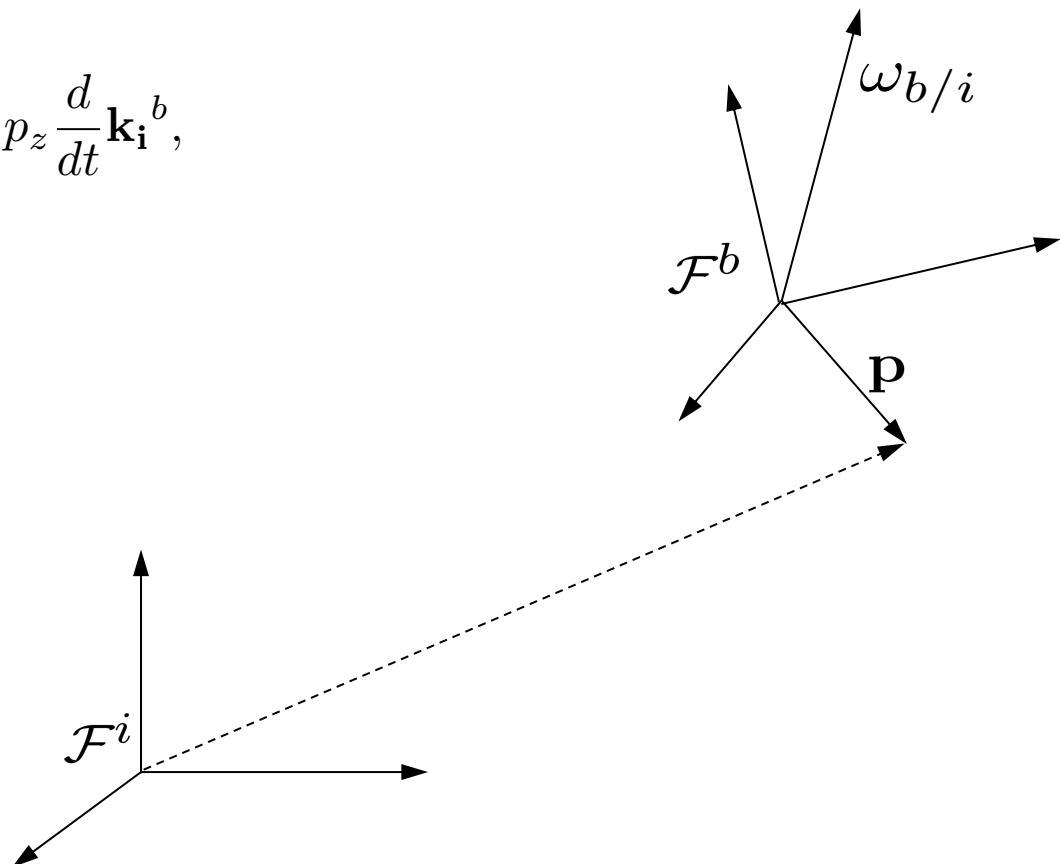
$$\frac{d}{dt_i} \mathbf{p} = \dot{p}_x \mathbf{i}^b + \dot{p}_y \mathbf{j}^b + \dot{p}_z \mathbf{k}^b + p_x \frac{d}{dt_i} \mathbf{i}^b + p_y \frac{d}{dt_i} \mathbf{j}^b + p_z \frac{d}{dt} \mathbf{k}_i^b,$$

where

$$\dot{p}_*^b = \frac{dp_*^b}{dt}.$$

Let

$$\frac{d}{dt_b} \mathbf{p} = \dot{p}_x \mathbf{i}^b + \dot{p}_y \mathbf{j}^b + \dot{p}_z \mathbf{k}^b.$$



Differentiation of a Vector in Two Reference Frames

Recall from physics that for any vector \mathbf{v} fixed in \mathcal{F}^b we have

$$\frac{d}{dt_i} \mathbf{v} = \boldsymbol{\omega}_{b/i} \times \mathbf{v}.$$

Frame \mathcal{F}^b rotating wrt frame \mathcal{F}^i
Vector \mathbf{p} moving in \mathcal{F}^b

Therefore

$$\frac{d}{dt_i} \mathbf{i}^b = \boldsymbol{\omega}_{b/i} \times \mathbf{i}^b$$

$$\frac{d}{dt_i} \mathbf{j}^b = \boldsymbol{\omega}_{b/i} \times \mathbf{j}^b$$

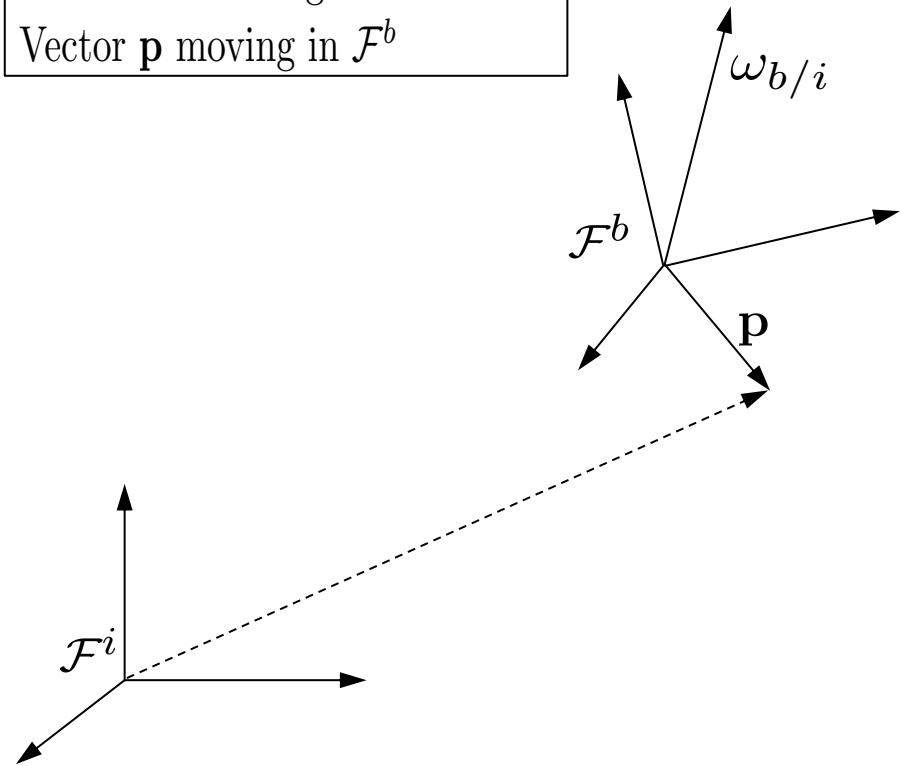
$$\frac{d}{dt_i} \mathbf{k}^b = \boldsymbol{\omega}_{b/i} \times \mathbf{k}^b.$$

Therefore

$$\begin{aligned} p_x \frac{d}{dt_i} \mathbf{i}^b + p_y \frac{d}{dt_i} \mathbf{j}^b + p_z \frac{d}{dt_i} \mathbf{k}^b &= p_x \boldsymbol{\omega}_{b/i} \times \mathbf{i}^b + p_y \boldsymbol{\omega}_{b/i} \times \mathbf{j}^b + p_z \boldsymbol{\omega}_{b/i} \times \mathbf{k}^b \\ &= \boldsymbol{\omega}_{b/i} \times (p_x \mathbf{i}^b + p_y \mathbf{j}^b + p_z \mathbf{k}^b) \\ &= \boldsymbol{\omega}_{b/i} \times \mathbf{p}, \end{aligned}$$

resulting in

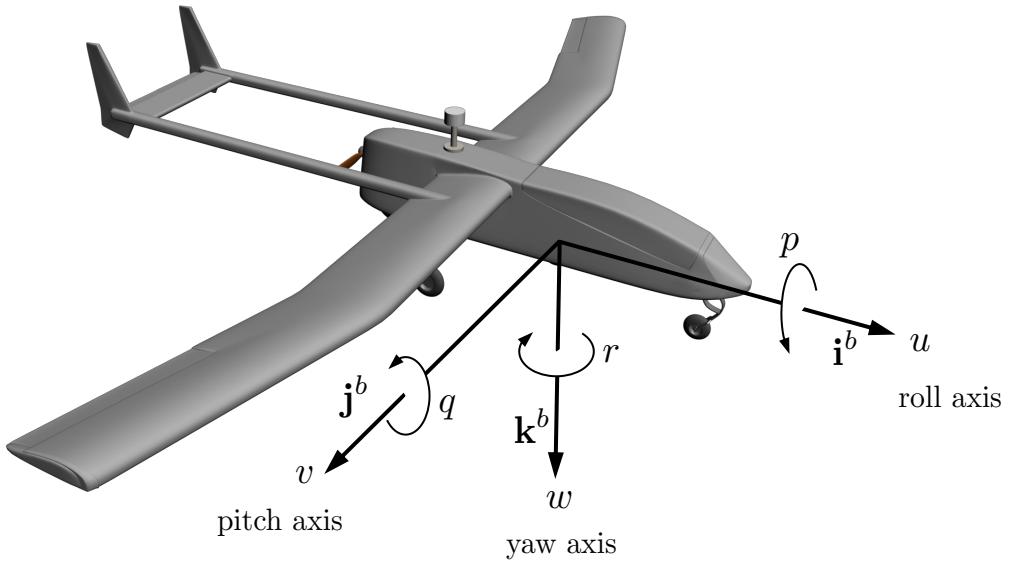
$$\frac{d}{dt_i} \mathbf{p} = \frac{d}{dt_b} \mathbf{p} + \boldsymbol{\omega}_{b/i} \times \mathbf{p}.$$



Translational Dynamics

Newton's 2nd Law:

$$m \frac{d\mathbf{V}_g}{dt_i} = \mathbf{f}$$



What is \mathbf{V}_g ?

- \mathbf{f} is the sum of all external forces
- m is the mass of the aircraft
- Time derivative taken in inertial frame

Using the expression

$$\frac{d\mathbf{V}_g}{dt_i} = \frac{d\mathbf{V}_g}{dt_b} + \boldsymbol{\omega}_{b/i} \times \mathbf{V}_g$$

gives

$$m \left(\frac{d\mathbf{V}_g}{dt_b} + \boldsymbol{\omega}_{b/i} \times \mathbf{V}_g \right) = \mathbf{f}$$

Translational Dynamics

Expressing $m \left(\frac{d\mathbf{V}_g}{dt_b} + \boldsymbol{\omega}_{b/i} \times \mathbf{V}_g \right) = \mathbf{f}$ in the body frame gives

$$m \left(\frac{d\mathbf{V}_g^b}{dt_b} + \boldsymbol{\omega}_{b/i}^b \times \mathbf{V}_g^b \right) = \mathbf{f}^b$$

where

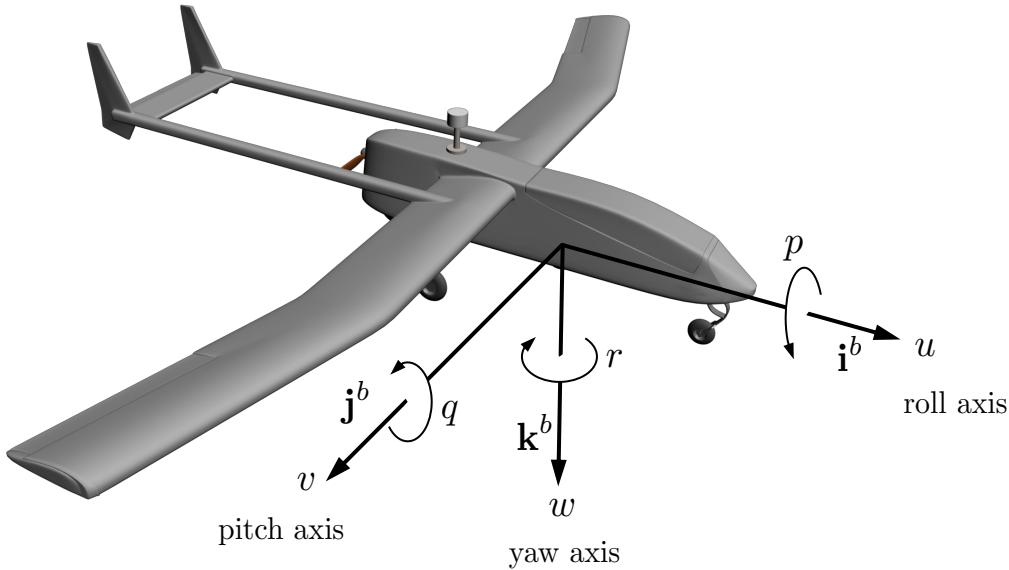
$$\mathbf{V}_g^b = \begin{pmatrix} u \\ v \\ w \end{pmatrix} \quad \boldsymbol{\omega}_{b/i}^b = \begin{pmatrix} p \\ q \\ r \end{pmatrix} \quad \mathbf{f}^b = \begin{pmatrix} f_x \\ f_y \\ f_z \end{pmatrix}$$

Since $\frac{d\mathbf{V}_g^b}{dt_b} = \begin{pmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{pmatrix}$ we have that

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

Rotational Dynamics

Newton's 2nd Law:



- \mathbf{h} is the angular momentum vector
- \mathbf{m} is the sum of all external moments
- Time derivative taken wrt inertial frame

Therefore we have

$$\frac{d\mathbf{h}}{dt_i} = \frac{d\mathbf{h}}{dt_b} + \boldsymbol{\omega}_{b/i} \times \mathbf{h} = \mathbf{m}$$

Expressing in the body frame gives

$$\frac{d\mathbf{h}^b}{dt_b} + \boldsymbol{\omega}_{b/i}^b \times \mathbf{h}^b = \mathbf{m}^b$$

Rotational Dynamics

For a rigid body, angular momentum is defined as the product of the inertia matrix and the angular velocity vector:

$$\mathbf{h}^b \triangleq \mathbf{J}\boldsymbol{\omega}_{b/i}^b$$

where

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

Diagonal elements are called moments of inertia. Off-diagonal elements are called products of inertia

\mathbf{J} determined from mass properties in CAD program or measured experimentally using a bifilar pendulum

Rotational Dynamics

Recalling that

$$\frac{d\mathbf{h}^b}{dt_b} + \boldsymbol{\omega}_{b/i}^b \times \mathbf{h}^b = \mathbf{m}^b$$

Because \mathbf{J} is unchanging in the body frame, $\frac{d\mathbf{J}}{dt_b} = 0$ and

$$\mathbf{J} \frac{d\boldsymbol{\omega}_{b/i}^b}{dt_b} + \boldsymbol{\omega}_{b/i}^b \times (\mathbf{J} \boldsymbol{\omega}_{b/i}^b) = \mathbf{m}^b$$

Rearranging we get

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

where

$$\dot{\boldsymbol{\omega}}_{b/i}^b = \frac{d\boldsymbol{\omega}_{b/i}^b}{dt_b} = \begin{pmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{pmatrix}$$

Rotational Dynamics

If the aircraft is symmetric about the $\mathbf{i}^b\text{-}\mathbf{k}^b$ plane, then $J_{xy} = J_{yz} = 0$ and

$$\mathbf{J} = \begin{pmatrix} J_x & 0 & -J_{xz} \\ 0 & J_y & 0 \\ -J_{xz} & 0 & J_z \end{pmatrix}$$

This symmetry assumption helps to simplify the analysis. The inverse of \mathbf{J} becomes

$$\begin{aligned} \mathbf{J}^{-1} &= \frac{\text{adj}(\mathbf{J})}{\det(\mathbf{J})} = \frac{\begin{pmatrix} J_y J_z & 0 & J_y J_{xz} \\ 0 & J_x J_z - J_{xz}^2 & 0 \\ J_{xz} J_y & 0 & J_x J_y \end{pmatrix}}{J_x J_y J_z - J_{xz}^2 J_y} \\ &= \begin{pmatrix} \frac{J_z}{\Gamma} & 0 & \frac{J_{xz}}{\Gamma} \\ 0 & \frac{1}{J_y} & 0 \\ \frac{J_{xz}}{\Gamma} & 0 & \frac{J_x}{\Gamma} \end{pmatrix} \end{aligned}$$

where

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

Rotational Dynamics

Define $\mathbf{m}^b \triangleq \begin{pmatrix} l \\ m \\ n \end{pmatrix}$ and recall that $\mathbf{a} \times \mathbf{b} = \begin{pmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$

Then

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

can be expressed as

$$\begin{aligned} \begin{pmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{pmatrix} &= \begin{pmatrix} \frac{J_z}{\Gamma} & 0 & \frac{J_{xz}}{\Gamma} \\ 0 & \frac{1}{J_y} & 0 \\ \frac{J_{xz}}{\Gamma} & 0 & \frac{J_x}{\Gamma} \end{pmatrix} \left[\begin{pmatrix} 0 & r & -q \\ -r & 0 & p \\ q & -p & 0 \end{pmatrix} \begin{pmatrix} J_x & 0 & -J_{xz} \\ 0 & J_y & 0 \\ -J_{xz} & 0 & J_z \end{pmatrix} \begin{pmatrix} p \\ q \\ r \end{pmatrix} + \begin{pmatrix} l \\ m \\ n \end{pmatrix} \right] \\ &= \begin{pmatrix} \frac{J_z}{\Gamma} & 0 & \frac{J_{xz}}{\Gamma} \\ 0 & \frac{1}{J_y} & 0 \\ \frac{J_{xz}}{\Gamma} & 0 & \frac{J_x}{\Gamma} \end{pmatrix} \left[\begin{pmatrix} J_{xz}pq + (J_y - J_z)qr \\ J_{xz}(r^2 - p^2) + (J_z - J_x)pr \\ (J_x - J_y)pq - J_{xz}qr \end{pmatrix} + \begin{pmatrix} l \\ m \\ n \end{pmatrix} \right] \\ &= \begin{pmatrix} \Gamma_1 pq - \Gamma_2 qr + \Gamma_3 l + \Gamma_4 n \\ \Gamma_5 pr - \Gamma_6(p^2 - r^2) + \frac{1}{J_y}m \\ \Gamma_7 pq - \Gamma_1 qr + \Gamma_4 l + \Gamma_8 n \end{pmatrix} \end{aligned}$$

where Γ 's are functions of moments and products of inertia

Equation of Motion Summary

The equations of motion are a system of 12 firstorder ODE's:

$$\begin{pmatrix} \dot{p}_n \\ \dot{p}_e \\ \dot{p}_d \end{pmatrix} = \begin{pmatrix} c_\theta c_\psi & s_\phi s_\theta c_\psi - c_\phi s_\psi & c_\phi s_\theta c_\psi + s_\phi s_\psi \\ c_\theta s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi \\ -s_\theta & s_\phi c_\theta & c_\phi c_\theta \end{pmatrix} \begin{pmatrix} u \\ v \\ w \end{pmatrix}$$

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

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

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

where

$$\Gamma_1 = \frac{J_{xz}(J_x - J_y + J_z)}{\Gamma}$$

$$\Gamma_2 = \frac{J_z(J_z - J_y) + J_{xz}^2}{\Gamma}$$

$$\Gamma_3 = \frac{J_z}{\Gamma}$$

$$\Gamma_4 = \frac{J_{xz}}{\Gamma}$$

$$\Gamma_5 = \frac{J_z - J_x}{J_y}$$

$$\Gamma_6 = \frac{J_{xz}}{J_y}$$

$$\Gamma_7 = \frac{(J_x - J_y)J_x + J_{xz}^2}{\Gamma}$$

$$\Gamma_8 = \frac{J_x}{\Gamma}$$

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

Quaternions

The attitude of a rigid body can be represented by a unit quaternion, which is a 4-vector

$$e = \begin{pmatrix} e_0 \\ e_1 \\ e_2 \\ e_3 \end{pmatrix}$$

where e_0 , e_1 , e_2 , and e_3 are scalars, and where $\|e\| = 1$

- e_0 is called the scalar part of the quaternion
- $(e_1, e_2, e_3)^\top$ is called the vector part of the quaternion

Quaternions

$\mathbf{e} = e_0 + ie_x + je_y + ke_z$, where i, j, k satisfy

$$i^2 = j^2 = k^2 = ijk = -1 \quad \Rightarrow \quad ij = k, \quad jk = i, \quad \text{and } ki = j.$$

Therefore

$$\begin{aligned} \mathbf{qe} &= (q_0e_0 - q_xe_x - q_ye_y - q_ze_z) + i(q_xe_0 + q_0e_x + q_ze_y - q_ye_x) \\ &\quad + j(q_ye_0 - q_ze_x + q_0e_y + q_xe_z) + k(q_ze_0 + q_ye_x - q_xe_y + q_0e_z). \end{aligned}$$

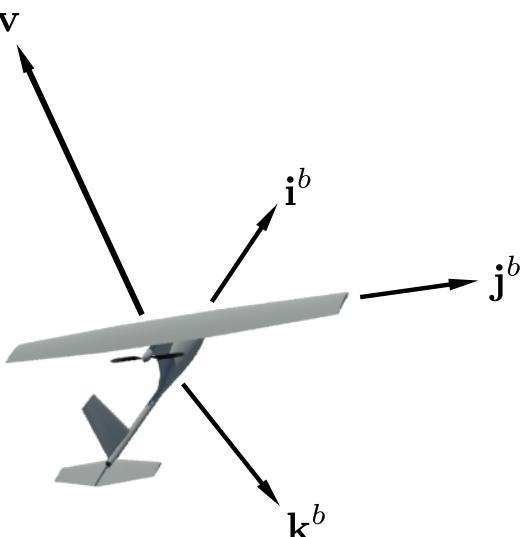
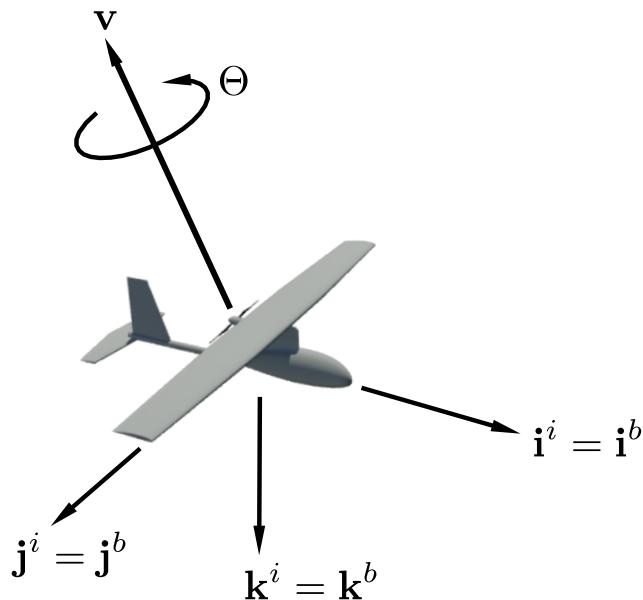
In matrix notation we have

$$\mathbf{qe} = M(\mathbf{q})\mathbf{e},$$

where

$$M(\mathbf{q}) = \begin{pmatrix} q_0 & -q_x & -q_y & -q_z \\ q_x & q_0 & q_z & -q_y \\ q_y & -q_y & q_0 & q_x \\ q_z & q_y & -q_x & q_0 \end{pmatrix}.$$

Quaternions



For a rotation of Θ about unit vector \mathbf{v} the scalar part is defined as

$$e_0 = \cos\left(\frac{\Theta}{2}\right)$$

and the vector part is defined as

$$\mathbf{v} \sin\left(\frac{\Theta}{2}\right) = \begin{pmatrix} e_1 \\ e_2 \\ e_3 \end{pmatrix}$$

Quaternions

Conversion Between Euler Angles and Quaternions

$$\phi = \text{atan2} (2(e_0 e_1 + e_2 e_3), (e_0^2 + e_3^2 - e_1^2 - e_2^2))$$

$$\theta = \text{asin} (2(e_0 e_2 - e_1 e_3))$$

$$\psi = \text{atan2} (2(e_0 e_3 + e_1 e_2), (e_0^2 + e_1^2 - e_2^2 - e_3^2))$$

From the yaw, pitch, and roll Euler angles (ψ, ϕ, θ), the corresponding quaternion elements are

$$e_0 = \cos \frac{\psi}{2} \cos \frac{\theta}{2} \cos \frac{\phi}{2} + \sin \frac{\psi}{2} \sin \frac{\theta}{2} \sin \frac{\phi}{2}$$

$$e_1 = \cos \frac{\psi}{2} \cos \frac{\theta}{2} \sin \frac{\phi}{2} - \sin \frac{\psi}{2} \sin \frac{\theta}{2} \cos \frac{\phi}{2}$$

$$e_2 = \cos \frac{\psi}{2} \sin \frac{\theta}{2} \cos \frac{\phi}{2} + \sin \frac{\psi}{2} \cos \frac{\theta}{2} \sin \frac{\phi}{2}$$

$$e_3 = \sin \frac{\psi}{2} \cos \frac{\theta}{2} \cos \frac{\phi}{2} - \cos \frac{\psi}{2} \sin \frac{\theta}{2} \sin \frac{\phi}{2}$$

Quaternions

Conversion Between Quaternion and Rotation Matrix

If the quaternion $\mathbf{e}_b^i = (e_0, e_1, e_2, e_3)^\top$ represents a rotation from the body to the inertial frame, then the corresponding rotation matrix is

$$R_b^i = \begin{pmatrix} e_1^2 + e_0^2 - e_2^2 - e_3^2 & 2(e_1e_2 - e_3e_0) & 2(e_1e_3 + e_2e_0) \\ 2(e_1e_2 + e_3e_0) & e_2^2 + e_0^2 - e_1^2 - e_3^2 & 2(e_2e_3 - e_1e_0) \\ 2(e_1e_3 - e_2e_0) & 2(e_2e_3 + e_1e_0) & e_3^2 + e_0^2 - e_1^2 - e_2^2 \end{pmatrix}$$

Quaternion Kinematics

be \mathbf{q}_Δ be a small rotation of angle $\Delta\Theta$, i.e.,

$$\mathbf{q}_\Delta = \begin{pmatrix} \cos(\Delta\Theta/2) \\ \sin(\Delta\Theta/2)u_x \\ \sin(\Delta\Theta/2)u_y \\ \sin(\Delta\Theta/2)u_z \end{pmatrix} \approx \begin{pmatrix} 1 \\ \frac{\omega_x\Delta t}{2} \\ \frac{\omega_y\Delta t}{2} \\ \frac{\omega_z\Delta t}{2} \end{pmatrix},$$

where $(\omega_x, \omega_y, \omega_z)^\top$ is the instantaneous angular velocity. Then

$$\mathbf{e}(t + \Delta t) = M(\mathbf{q}_\Delta)\mathbf{e}(t).$$

Therefore

$$\begin{aligned} \dot{\mathbf{e}} &= \lim_{\Delta t \rightarrow 0} \frac{\mathbf{e}(t + \Delta t) - \mathbf{e}(t)}{\Delta t} = \lim_{\Delta t \rightarrow 0} \frac{M(\mathbf{q}_\Delta)\mathbf{e}(t) - \mathbf{e}(t)}{\Delta t} = \lim_{\Delta t \rightarrow 0} \frac{(M(\mathbf{q}_\Delta) - I)\mathbf{e}(t)}{\Delta t} \\ &= \lim_{\Delta t \rightarrow 0} \frac{1}{2\Delta t} \begin{pmatrix} 0 & -\omega_x\Delta t & -\omega_y\Delta t & -\omega_z\Delta t \\ \omega_x\Delta t & 0 & \omega_z\Delta t & -\omega_y\Delta t \\ \omega_y\Delta t & -\omega_z\Delta t & 0 & \omega_x\Delta t \\ \omega_z\Delta t & \omega_y\Delta t & -\omega_x\Delta t & 0 \end{pmatrix} \mathbf{e}(t) = \frac{1}{2}\Omega(\boldsymbol{\omega})\mathbf{e}(t), \end{aligned}$$

where

$$\Omega(\boldsymbol{\omega}) = \begin{pmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & \omega_z & -\omega_y \\ \omega_y & -\omega_z & 0 & \omega_x \\ \omega_z & \omega_y & -\omega_x & 0 \end{pmatrix}$$

Equation of Motion Summary

The equations of motion are a system of 13 first-order ODE's:

$$\begin{pmatrix} \dot{p}_n \\ \dot{p}_e \\ \dot{p}_d \end{pmatrix} = \begin{pmatrix} e_1^2 + e_0^2 - e_2^2 - e_3^2 & 2(e_1e_2 - e_3e_0) & 2(e_1e_3 + e_2e_0) \\ 2(e_1e_2 + e_3e_0) & e_2^2 + e_0^2 - e_1^2 - e_3^2 & 2(e_2e_3 - e_1e_0) \\ 2(e_1e_3 - e_2e_0) & 2(e_2e_3 + e_1e_0) & e_3^2 + e_0^2 - e_1^2 - e_2^2 \end{pmatrix} \begin{pmatrix} u \\ v \\ w \end{pmatrix}$$

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

$$\begin{pmatrix} \dot{e}_0 \\ \dot{e}_1 \\ \dot{e}_2 \\ \dot{e}_3 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 0 & -p & -q & -r \\ p & 0 & r & -q \\ q & -r & 0 & p \\ r & q & -p & 0 \end{pmatrix} \begin{pmatrix} e_0 \\ e_1 \\ e_2 \\ e_3 \end{pmatrix}$$

$$\begin{pmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{pmatrix} = \begin{pmatrix} \Gamma_1pq - \Gamma_2qr \\ \Gamma_5pr - \Gamma_6(p^2 - r^2) \\ \Gamma_7pq - \Gamma_1qr \end{pmatrix} + \begin{pmatrix} \Gamma_3l + \Gamma_4n \\ \frac{1}{J_y}m \\ \Gamma_4l + \Gamma_8n \end{pmatrix}$$

- Quaternion EOM are simpler (linear)
- Require conversion to Euler angles
- Must be normalized after each integration step

Quaternions

In Python, e needs to be normalized after applying the RK4 update step to ensure that $\|e\| = 1$

In Simulink, we can ensure that $\|e\| \approx 1$ by appending the quaternion kinematics as

$$\begin{aligned}\begin{pmatrix} \dot{e}_0 \\ \dot{e}_1 \\ \dot{e}_2 \\ \dot{e}_3 \end{pmatrix} &= \frac{1}{2} \begin{pmatrix} 0 & -p & -q & -r \\ p & 0 & r & -q \\ q & -r & 0 & p \\ r & q & -p & 0 \end{pmatrix} \begin{pmatrix} e_0 \\ e_1 \\ e_2 \\ e_3 \end{pmatrix} - \lambda \frac{\partial J}{\partial e} \\ &= \frac{1}{2} \begin{pmatrix} \lambda(1 - \|e\|^2) & -p & -q & -r \\ p & \lambda(1 - \|e\|^2) & r & -q \\ q & -r & \lambda(1 - \|e\|^2) & p \\ r & q & -p & \lambda(1 - \|e\|^2) \end{pmatrix} \begin{pmatrix} e_0 \\ e_1 \\ e_2 \\ e_3 \end{pmatrix}\end{aligned}$$

where $J = \frac{1}{8}(1 - \|e\|^2)^2$ and where $\lambda > 0$. The second term forces $\|e\| \rightarrow 1$. In our experience, a value of $\lambda = 1000$ seems to work well, but a stiff solver like ODE15s must be used

Runge-Kutta Integration

The objective is to numerically solve the differential equation

$$\frac{dx}{dt}(t) = f(x(t), u(t)), \quad x(t_0) = x_0$$

Integrating both sides gives

$$x(t) = x(t_0) + \int_{t_0}^t f(x(\tau), u(\tau)) d\tau$$

Note: $x(t)$ appears on both sides of the equation! Therefore, approximation is required.

Re-write solution integral as

$$\begin{aligned} x(t) &= x(t_0) + \int_{t_0}^{t-T_s} f(x(\tau), u(\tau)) d\tau + \int_{t-T_s}^t f(x(\tau), u(\tau)) d\tau \\ &= x(t - T_s) + \int_{t-T_s}^t f(x(\tau), x(\tau)) d\tau \end{aligned}$$

RK1 Algorithm

The integral can most easily be approximated using Euler's method

$$\int_a^b f(\xi) d\xi \approx (b - a)f(a)$$

Accordingly,

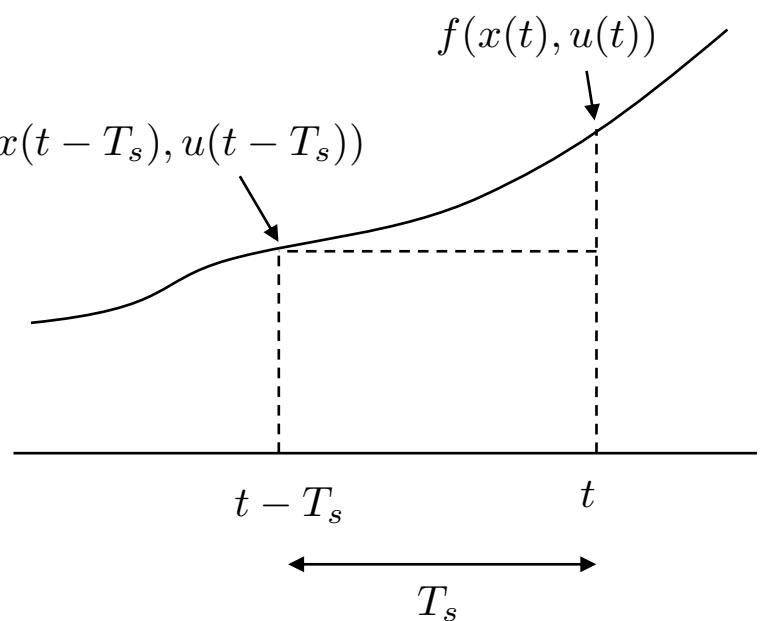
$$\int_{t-T_s}^t f(x(\tau), u(\tau)) d\tau \approx T_s f(x(t - T_s), u(t - T_s))$$

The numerical integration routine can be written as

$$x_0 = x(t_0)$$

$$X_1 = f(x_{k-1}, u_{k-1})$$

$$x_k = x_{k-1} + T_s X_1$$



This is the Runge-Kutta first-order method or RK1

RK2 Algorithm

To improve accuracy, the integral can be approximated using the trapezoidal rule

$$\int_a^b f(\xi) d\xi \approx (b - a) \left(\frac{f(a) + f(b)}{2} \right)$$

Accordingly,

$$\int_{t-T_s}^t f(x(\tau), u(\tau)) d\tau \approx \frac{T_s}{2} [f(x(t - T_s), u(t - T_s)) + f(x(t), u(t))]$$

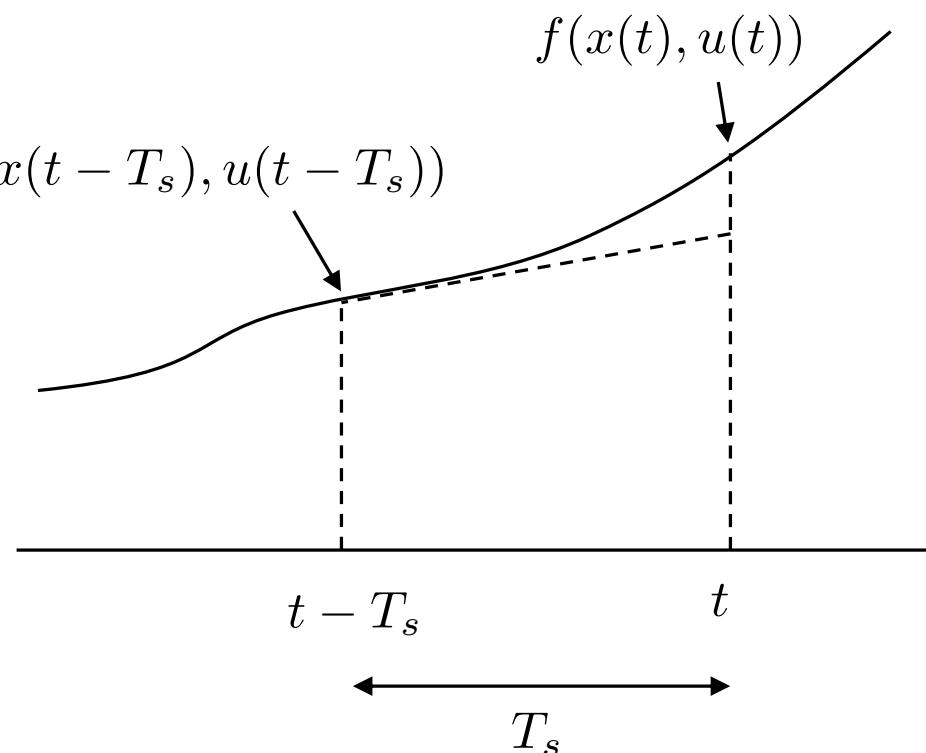
$x(t)$ is unknown so use RK1 to approximate it as

$$\begin{aligned} \int_{t-T_s}^t f(x(\tau), u(\tau)) d\tau &\approx \frac{T_s}{2} [f(x(t - T_s), u(t - T_s)) \\ &+ f(x(t - T_s) + T_s f(x(t - T_s), u(t - T_s)), u(t - T_s))] \end{aligned}$$

The numerical integration routine can be written as

$$\begin{aligned} x_0 &= x(t_0) \\ X_1 &= f(x_{k-1}, u_{k-1}) \\ X_2 &= f(x_{k-1} + T_s X_1, u_{k-1}) \\ x_k &= x_{k-1} + \frac{T_s}{2} (X_1 + X_2) \end{aligned}$$

This is the Runge-Kutta second-order method or RK2



RK4 Algorithm

An even better approximation is obtained using Simpson's Rule (area under a parabola):

$$\int_a^b f(\xi) d\xi \approx \left(\frac{b-a}{6} \right) \left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b) \right)$$

The standard strategy is to write the approximation as

$$\int_a^b f(\xi) \xi d\xi \approx \left(\frac{b-a}{6} \right) \left(f(a) + 2f\left(\frac{a+b}{2}\right) + 2f\left(\frac{a+b}{2}\right) + f(b) \right)$$

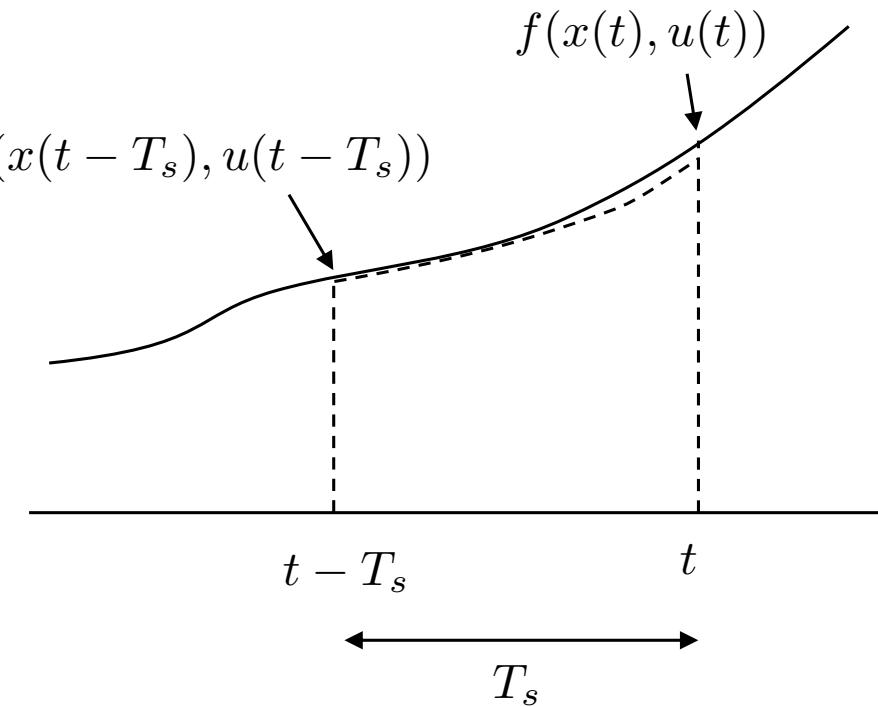
Define

$$X_1 = f(a)$$

$$X_2 = f\left(a + \frac{T_s}{2} X_1\right) \approx f\left(\frac{a+b}{2}\right)$$

$$X_3 = f\left(a + \frac{T_s}{2} X_2\right) \approx f\left(\frac{a+b}{2}\right)$$

$$X_4 = f(a + T_s X_3) \approx f(b)$$



RK4 Algorithm

The numerical integration routine can be written as

$$x_0 = x(t_0)$$

$$X_1 = f(x_{k-1}, u_{k-1})$$

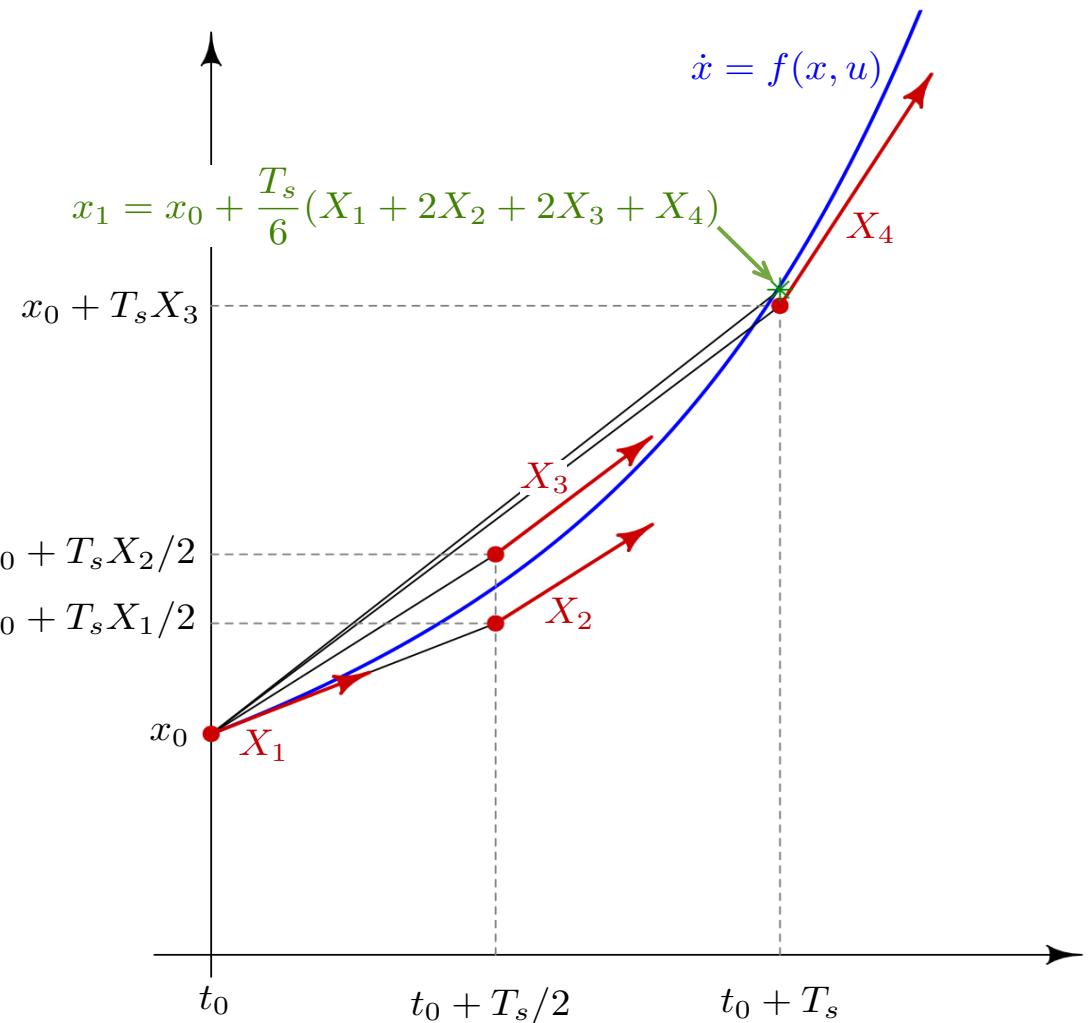
$$X_2 = f(x_{k-1} + \frac{T_s}{2} X_1, u_{k-1})$$

$$X_3 = f(x_{k-1} + \frac{T_s}{2} X_2, u_{k-1})$$

$$X_4 = f(x_{k-1} + T_s X_3, u_{k-1})$$

$$x_k = x_{k-1} + \frac{T_s}{6}(X_1 + 2X_2 + 2X_3 + X_4)$$

This is the Runge-Kutta fourth-order method or RK4



Adapted from HilberTraum - Own work, CC BY-SA 4.0,
<https://commons.wikimedia.org/w/index.php?curid=64366870>

DYNAMICS AND RK4 IMPLEMENTATION

See example code

Project

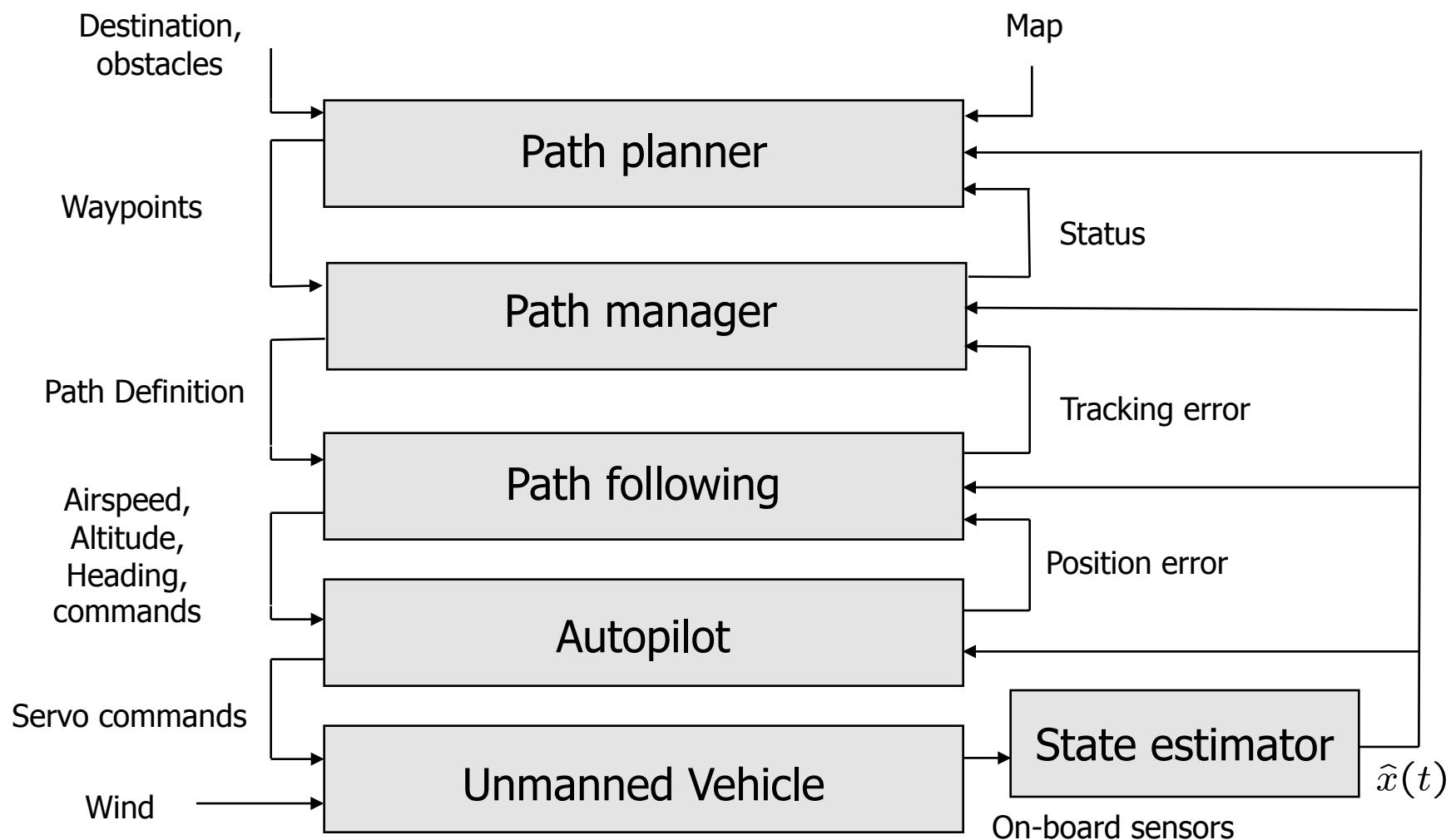
1. Implement the MAV equations of motion given in Equations (3.14) through (3.17). Assume that the inputs to the function are the forces and moments applied to the MAV in the body frame. Changeable parameters should include the mass, the moments and products of inertia, and the initial conditions for each state. Use the parameters given in Appendix E.
2. Verify that the equations of motion are correct by individually setting the forces and moments along each axis to a nonzero value and convincing yourself that the motion is appropriate.
3. Since J_{xz} is non-zero, there is gyroscopic coupling between roll and yaw. To test your simulation, set J_{xz} to zero and place nonzero moments on l and n and verify that there is no coupling between the roll and yaw axes. Verify that when J_{xz} is not zero, there is coupling between the roll and yaw axes.



Chapter 4

Forces and Moments

Architecture



Equations of Motion from Chap 3

The combined equations of motion from Chapter 3 are

$$\begin{pmatrix} \dot{p}_n \\ \dot{p}_e \\ \dot{p}_d \end{pmatrix} = \begin{pmatrix} c_\theta c_\psi & s_\phi s_\theta c_\psi - c_\phi s_\psi & c_\phi s_\theta c_\psi + s_\phi s_\psi \\ c_\theta s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi \\ -s_\theta & s_\phi c_\theta & c_\phi c_\theta \end{pmatrix} \begin{pmatrix} u \\ v \\ w \end{pmatrix}$$

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

System of 12 first-order ODE's

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

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

The objective of this chapter is to show how to compute the force vector

$$\mathbf{f}^b = \begin{pmatrix} f_x \\ f_y \\ f_z \end{pmatrix}$$

and the moment vector

$$\mathbf{m}^b = \begin{pmatrix} \ell \\ m \\ n \end{pmatrix}.$$

External Forces and Moments

The external forces are a combination of gravitational, aerodynamic, and propulsion:

$$\mathbf{f} = \mathbf{f}_g + \mathbf{f}_a + \mathbf{f}_p$$

The external moments are a combination of aerodynamic, and propulsion:

$$\mathbf{m} = \mathbf{m}_a + \mathbf{m}_p$$

Gravity Force

The gravity vector expressed in the vehicle frame is

$$\mathbf{f}_g^v = \begin{pmatrix} 0 \\ 0 \\ mg \end{pmatrix}$$

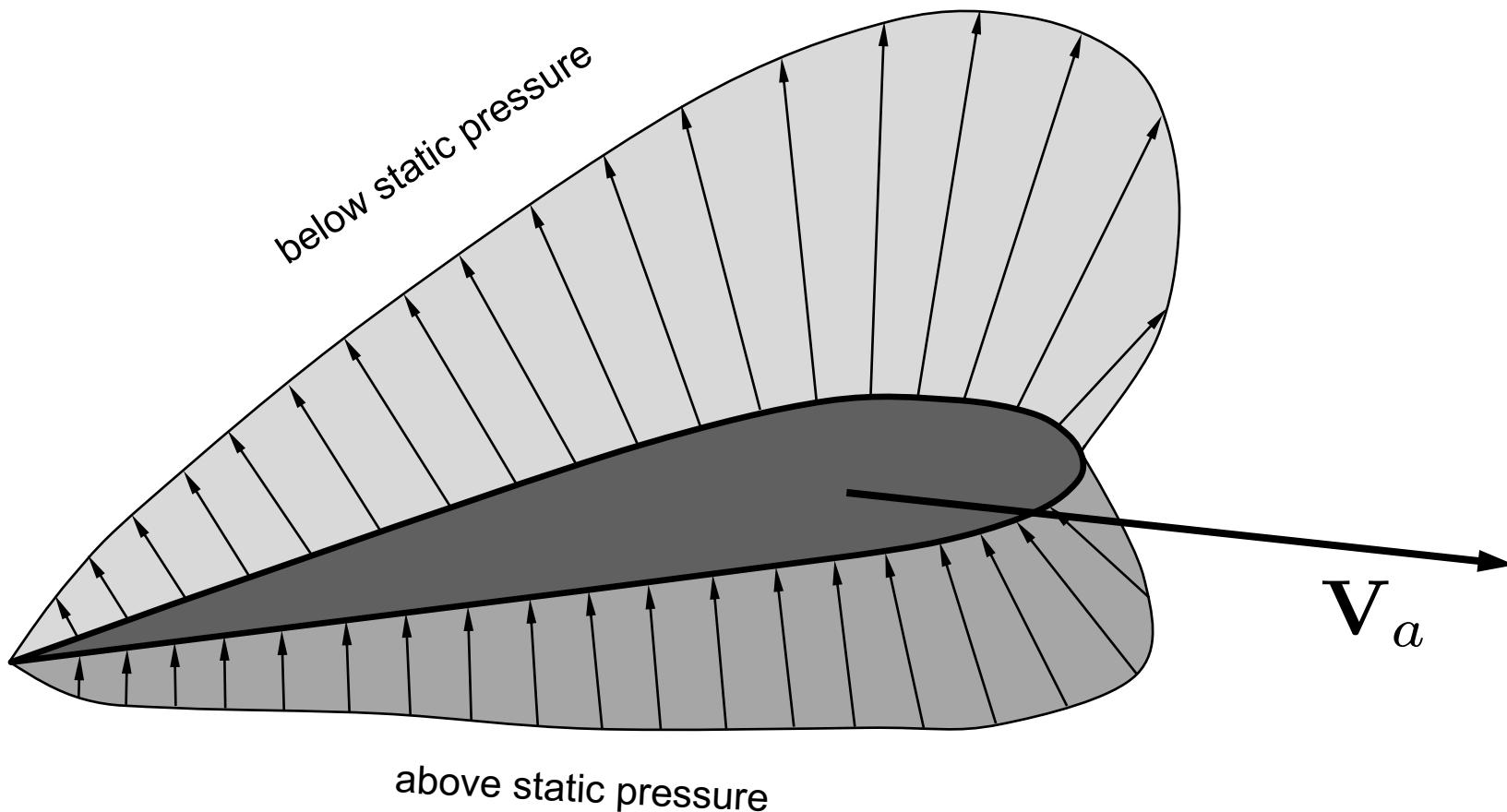
Expressed in the body frame we have

$$\mathbf{f}_g^b = \mathcal{R}_v^b \begin{pmatrix} 0 \\ 0 \\ mg \end{pmatrix} = \begin{pmatrix} c_\theta c_\psi & c_\theta s_\psi & -s_\theta \\ s_\phi s_\theta c_\psi - c_\phi s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & s_\phi c_\theta \\ c_\phi s_\theta c_\psi + s_\phi s_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi & c_\phi c_\theta \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ mg \end{pmatrix} = mg \begin{pmatrix} -\sin \theta \\ \cos \theta \sin \phi \\ \cos \theta \cos \phi \end{pmatrix}$$

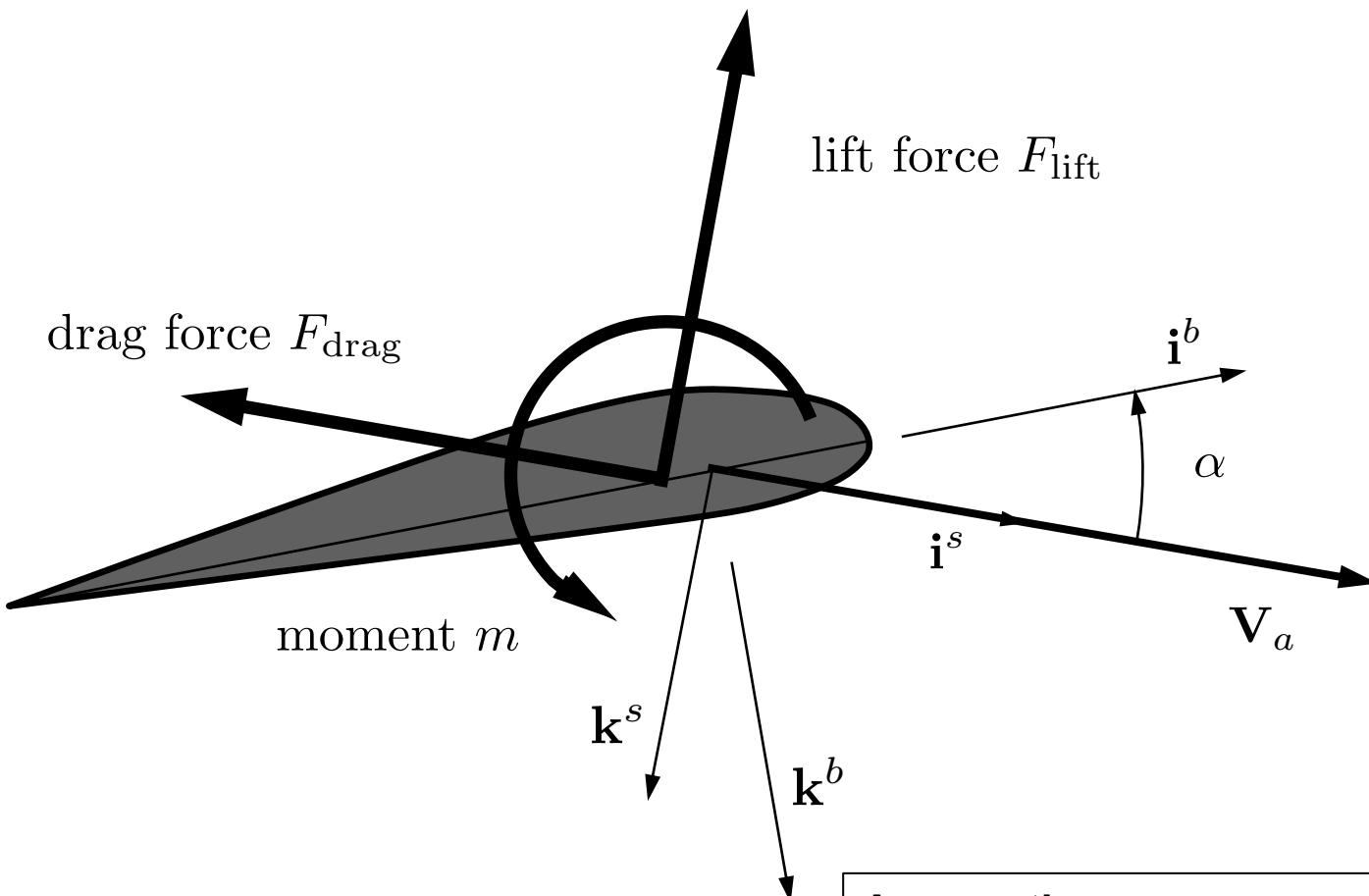
For quaternions, we have

$$\mathbf{f}_g^b = \begin{pmatrix} e_0^2 + e_x^2 - e_y^2 - e_z^2 & 2(e_x e_y - e_0 e_z) & 2(e_x e_z + e_0 e_y) \\ 2(e_x e_y + e_0 e_z) & e_0^2 - e_x^2 + e_y^2 - e_z^2 & 2(e_y e_z - e_0 e_x) \\ 2(e_x e_z - e_0 e_y) & 2(e_y e_z + e_0 e_x) & e_0^2 - e_x^2 - e_y^2 + e_z^2 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ mg \end{pmatrix} = mg \begin{pmatrix} 2(e_x e_z - e_y e_0) \\ 2(e_y e_z + e_x e_0) \\ e_z^2 + e_0^2 - e_x^2 - e_y^2 \end{pmatrix}$$

Airfoil Pressure Distribution



Aerodynamic Approximation



$$F_{\text{lift}} = \frac{1}{2} \rho V_a^2 S C_L$$

$$F_{\text{drag}} = \frac{1}{2} \rho V_a^2 S C_D$$

$$m = \frac{1}{2} \rho V_a^2 S c C_m$$

$$\begin{pmatrix} f_x \\ f_z \end{pmatrix} = \begin{pmatrix} \cos \alpha & -\sin \alpha \\ \sin \alpha & \cos \alpha \end{pmatrix} \begin{pmatrix} -F_{\text{drag}} \\ -F_{\text{lift}} \end{pmatrix}$$

Assumption:

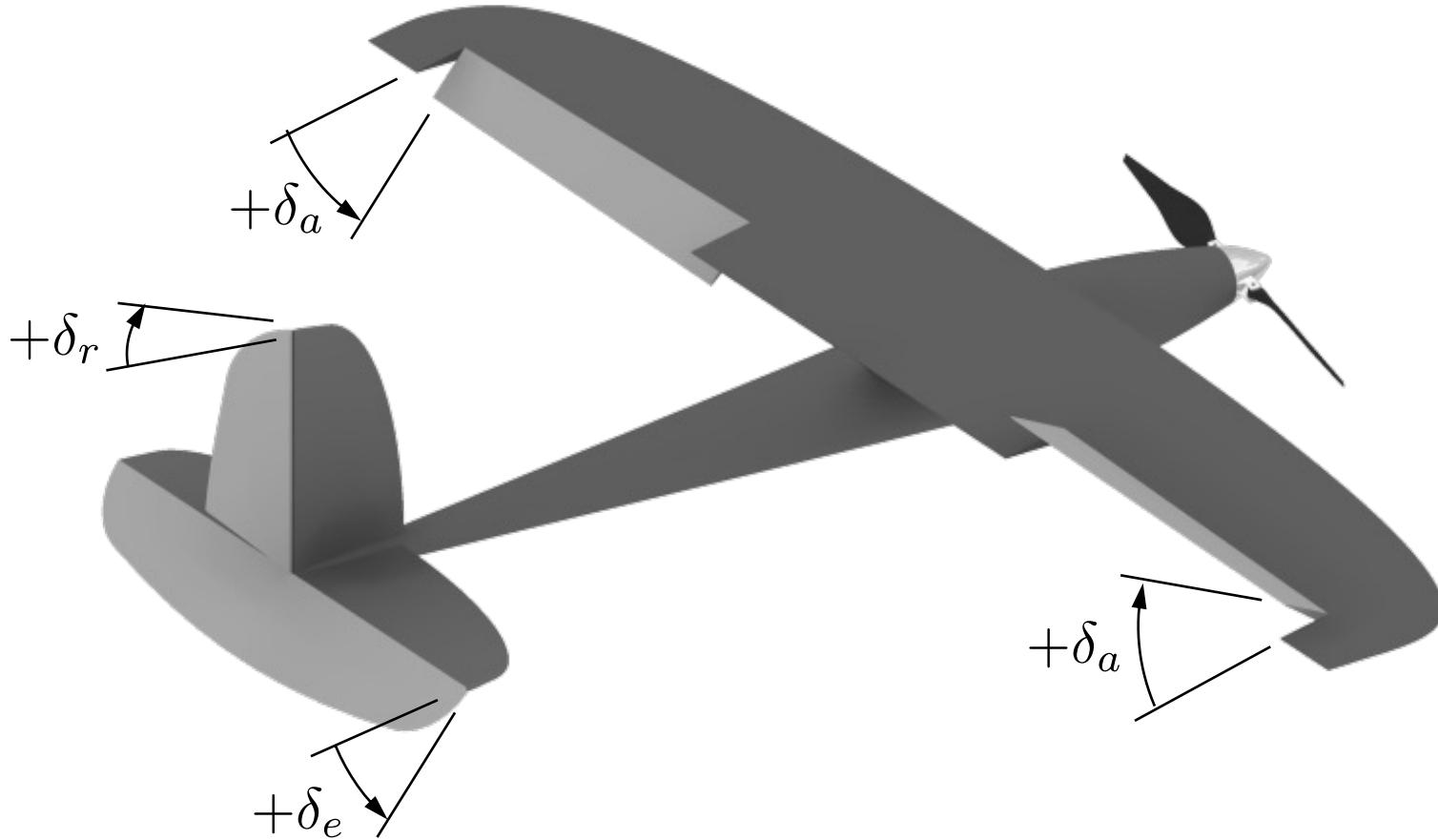
Forces and moment act at *aerodynamic center of wing*

Approximately at quarter chord

Defined as point where moment is constant with angle of attack

Control Surfaces - Conventional

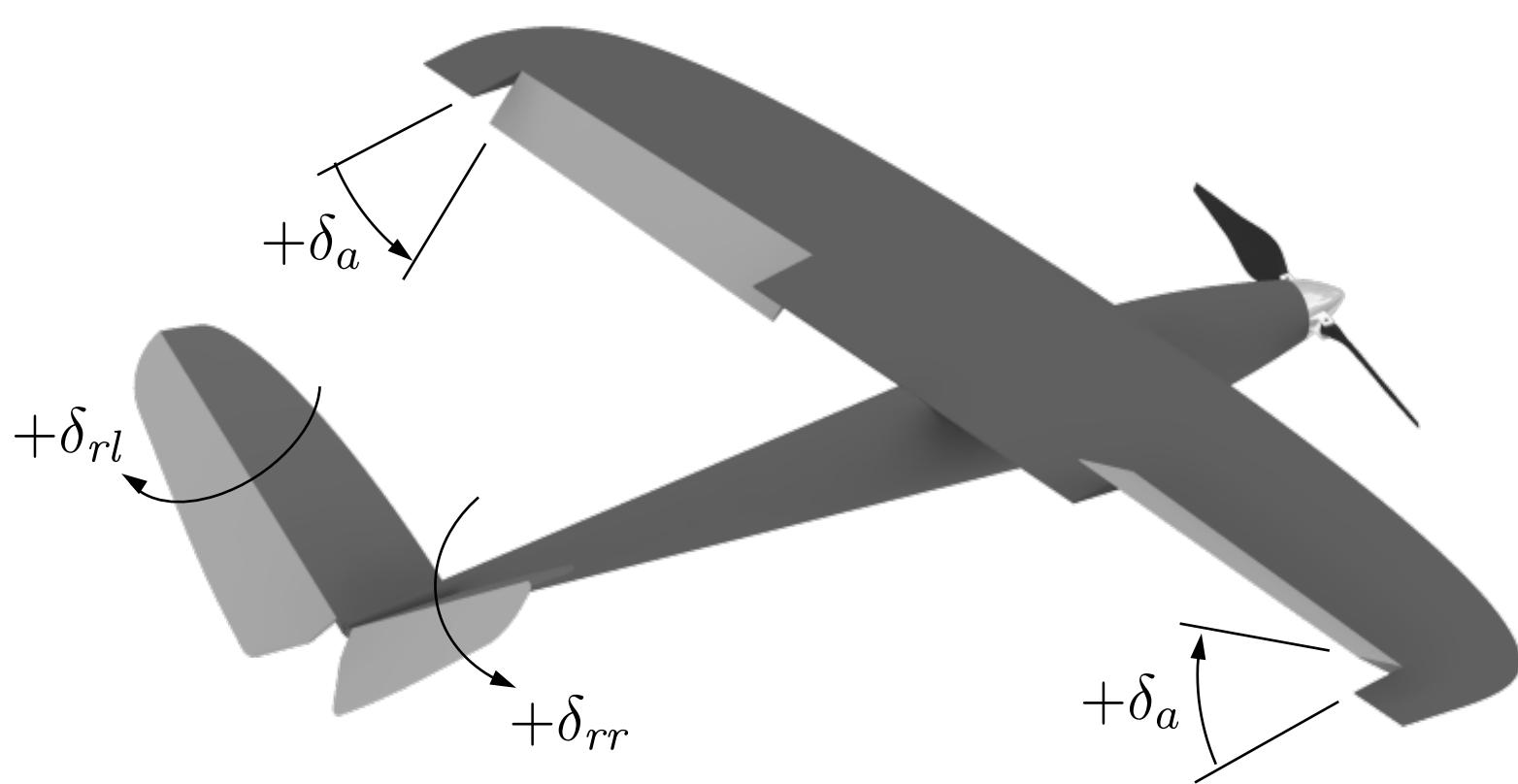
$$\delta_a = \frac{1}{2} (\delta_{a\text{-left}} - \delta_{a\text{-right}})$$



$+δ_a$ results in positive roll rate p
 $+δ_e$ results in negative pitch rate q
 $+δ_r$ results in negative yaw rate r

- Ailerons
- Elevator
- Rudder

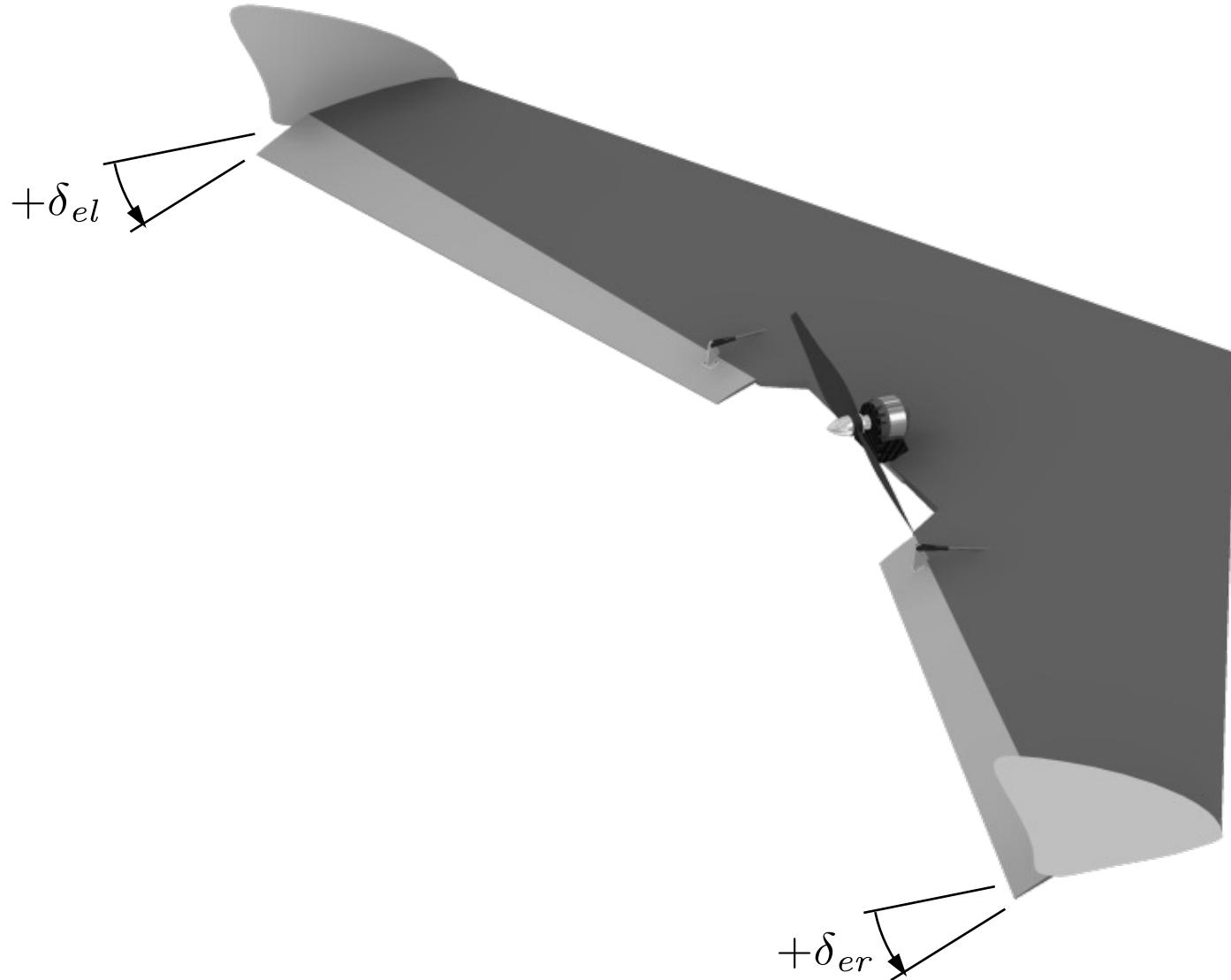
Control Surfaces – V-tail



$$\begin{pmatrix} \delta_e \\ \delta_r \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} \delta_{rr} \\ \delta_{rl} \end{pmatrix}$$

- Ailerons
- Ruddervators

Control Surfaces – Flying Wing



$$\begin{pmatrix} \delta_e \\ \delta_a \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix} \begin{pmatrix} \delta_{er} \\ \delta_{el} \end{pmatrix}$$

- Elevons

Aircraft Dynamics

- Aircraft dynamics and aerodynamics are commonly separated into two groups:
 - Longitudinal
 - Up-down, pitch plane, pitching motions
 - Lateral-directional
 - Side-to-side, turning motions (roll and yaw)

Longitudinal Aerodynamics

- Act in the $\mathbf{i}^b - \mathbf{k}^b$ plane, aka the symmetry/pitch plane
- Heavily influenced by angle of attack
- Also influenced by pitch rate and elevator deflection

$$F_{\text{lift}} \approx \frac{1}{2} \rho V_a^2 S C_L(\alpha, q, \delta_e)$$

$$F_{\text{drag}} \approx \frac{1}{2} \rho V_a^2 S C_D(\alpha, q, \delta_e)$$

$$m \approx \frac{1}{2} \rho V_a^2 S c C_m(\alpha, q, \delta_e)$$

Aerodynamic Approximation

In the general nonlinear case we have

$$F_{\text{lift}} \approx \frac{1}{2} \rho V_a^2 S C_L(\alpha, q, \delta_e)$$

Expanding C_L as a Taylor series and keeping only the first-order (linear) terms gives

$$\begin{aligned} F_{\text{lift}} &= \frac{1}{2} \rho V_a^2 S \left[C_{L_0} + \frac{\partial C_L}{\partial \alpha} \alpha + \frac{\partial C_L}{\partial q} q + \frac{\partial C_L}{\partial \delta_e} \delta_e \right] \\ &= \frac{1}{2} \rho V_a^2 S \left[C_{L_0} + C_{L_\alpha} \alpha + C_{L_q} \frac{c}{2V_a} q + C_{L_{\delta_e}} \delta_e \right] \end{aligned}$$

where the coefficients C_{L_0} , $C_{L_\alpha} \triangleq \frac{\partial C_L}{\partial \alpha}$, $C_{L_q} \triangleq \frac{\partial C_L}{\partial \frac{q c}{2V_a}}$, and $C_{L_{\delta_e}} \triangleq \frac{\partial C_L}{\partial \delta_e}$ are dimensionless quantities



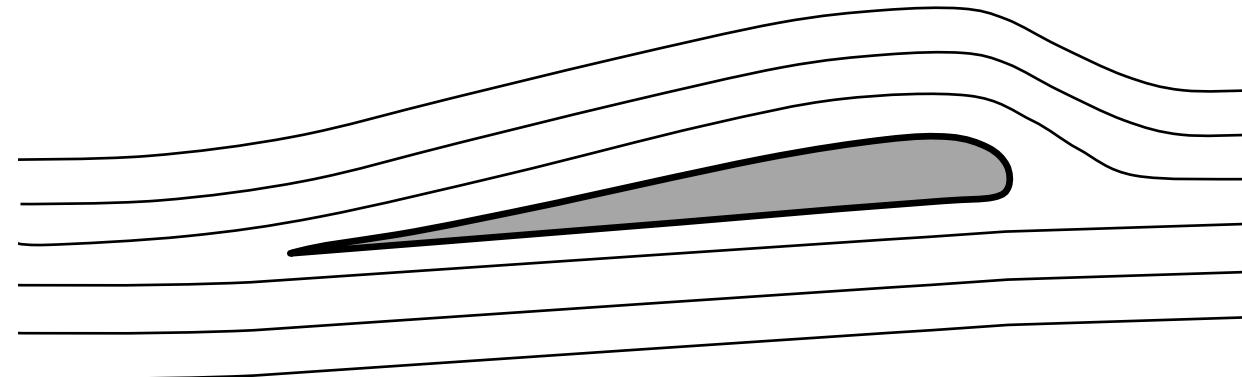
Linear Aerodynamic Model

$$F_{\text{lift}} = \frac{1}{2} \rho V_a^2 S \left[C_{L_0} + C_{L_\alpha} \alpha + C_{L_q} \frac{c}{2V_a} q + C_{L_{\delta_e}} \delta_e \right]$$

$$F_{\text{drag}} = \frac{1}{2} \rho V_a^2 S \left[C_{D_0} + C_{D_\alpha} \alpha + C_{D_q} \frac{c}{2V_a} q + C_{D_{\delta_e}} \delta_e \right]$$

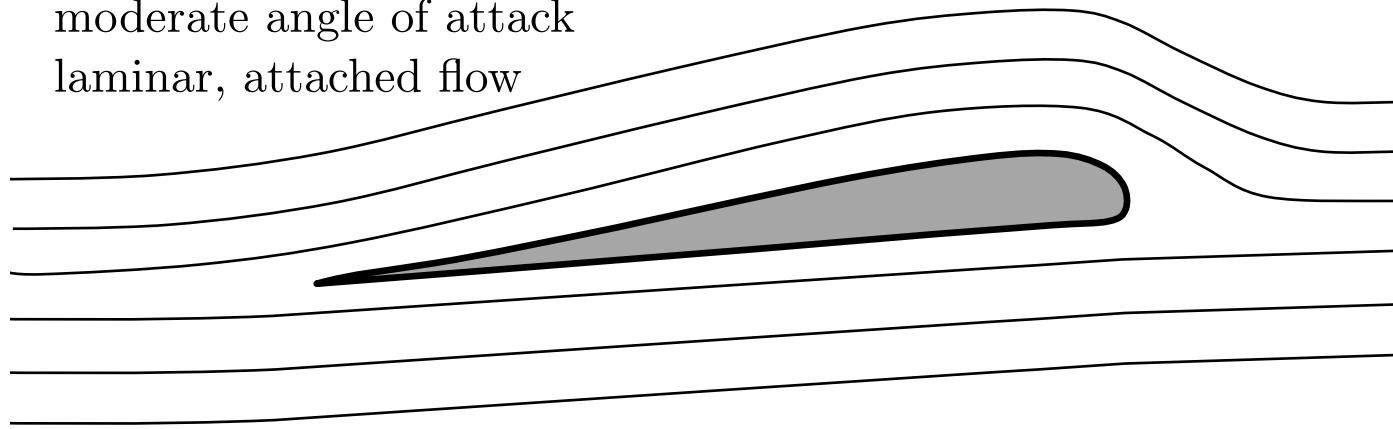
$$m = \frac{1}{2} \rho V_a^2 Sc \left[C_{m_0} + C_{m_\alpha} \alpha + C_{m_q} \frac{c}{2V_a} q + C_{m_{\delta_e}} \delta_e \right]$$

Linear aerodynamic model is valid for small angles of attack – flow remains attached over wing

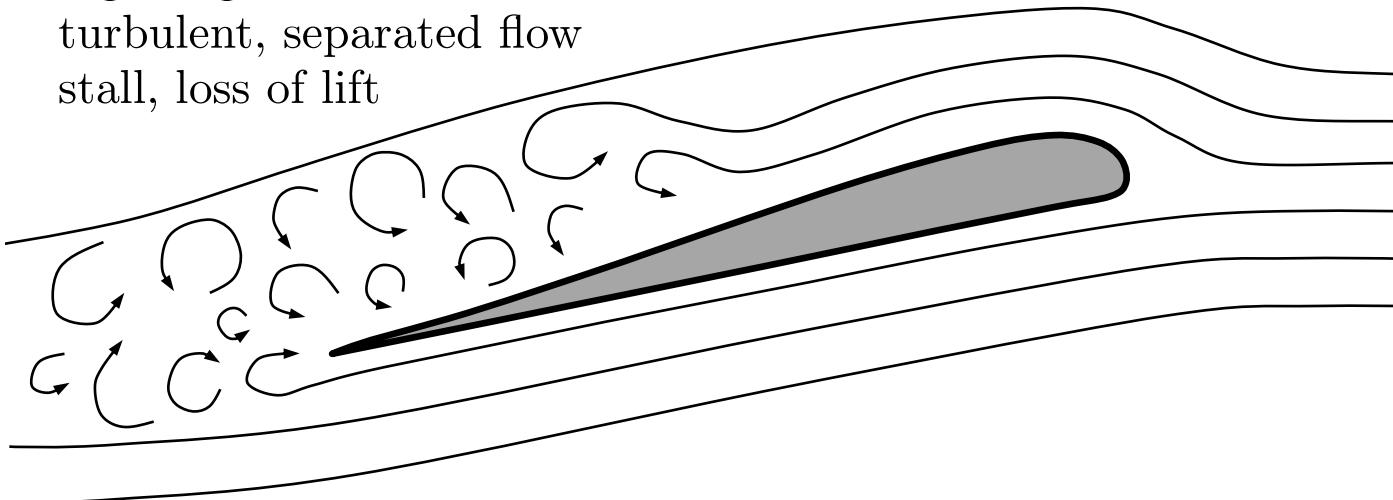


Nonlinear Aerodynamics – Stall

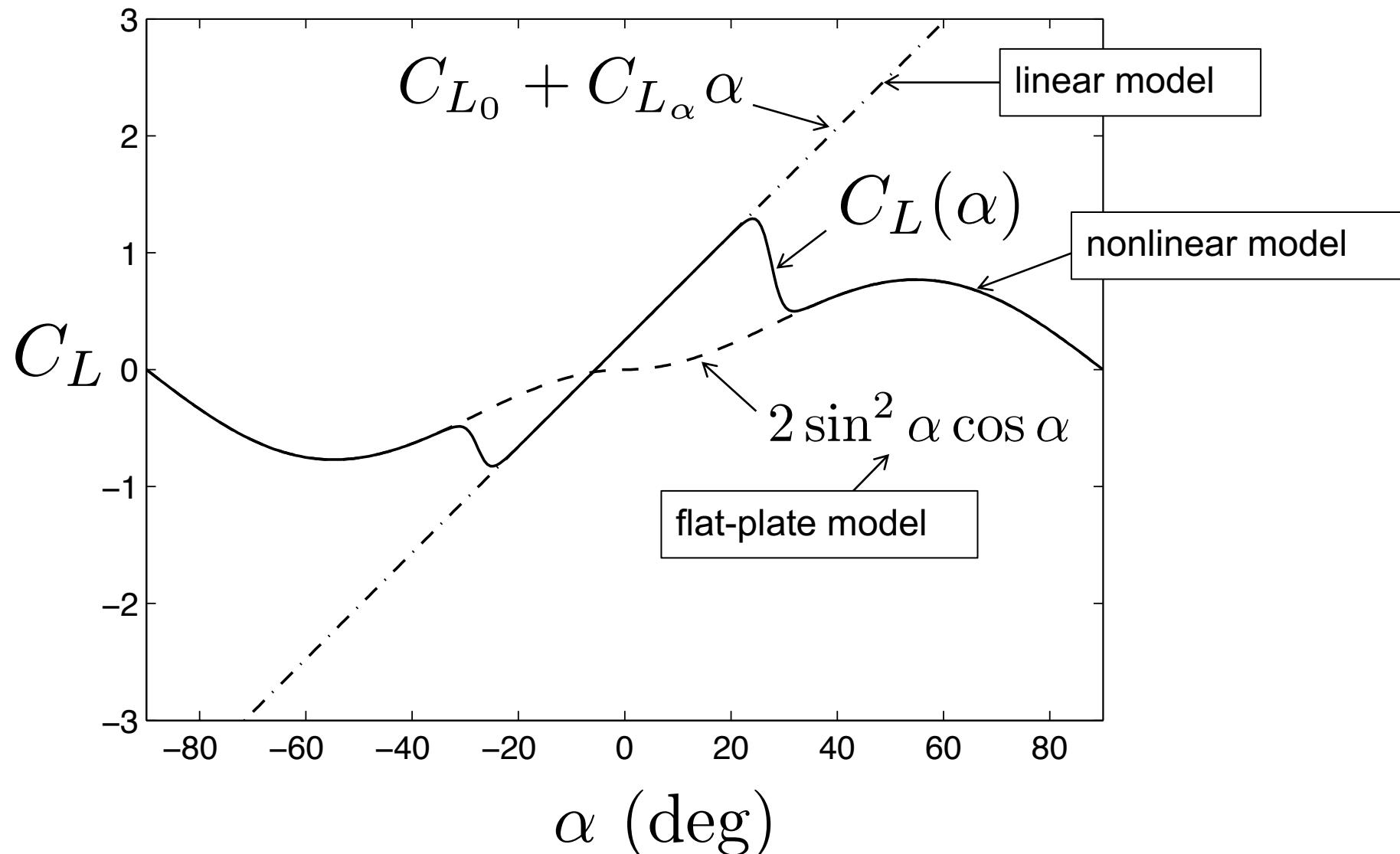
moderate angle of attack
laminar, attached flow



high angle of attack
turbulent, separated flow
stall, loss of lift



Nonlinear Lift Model



Nonlinear Aerodynamic Model

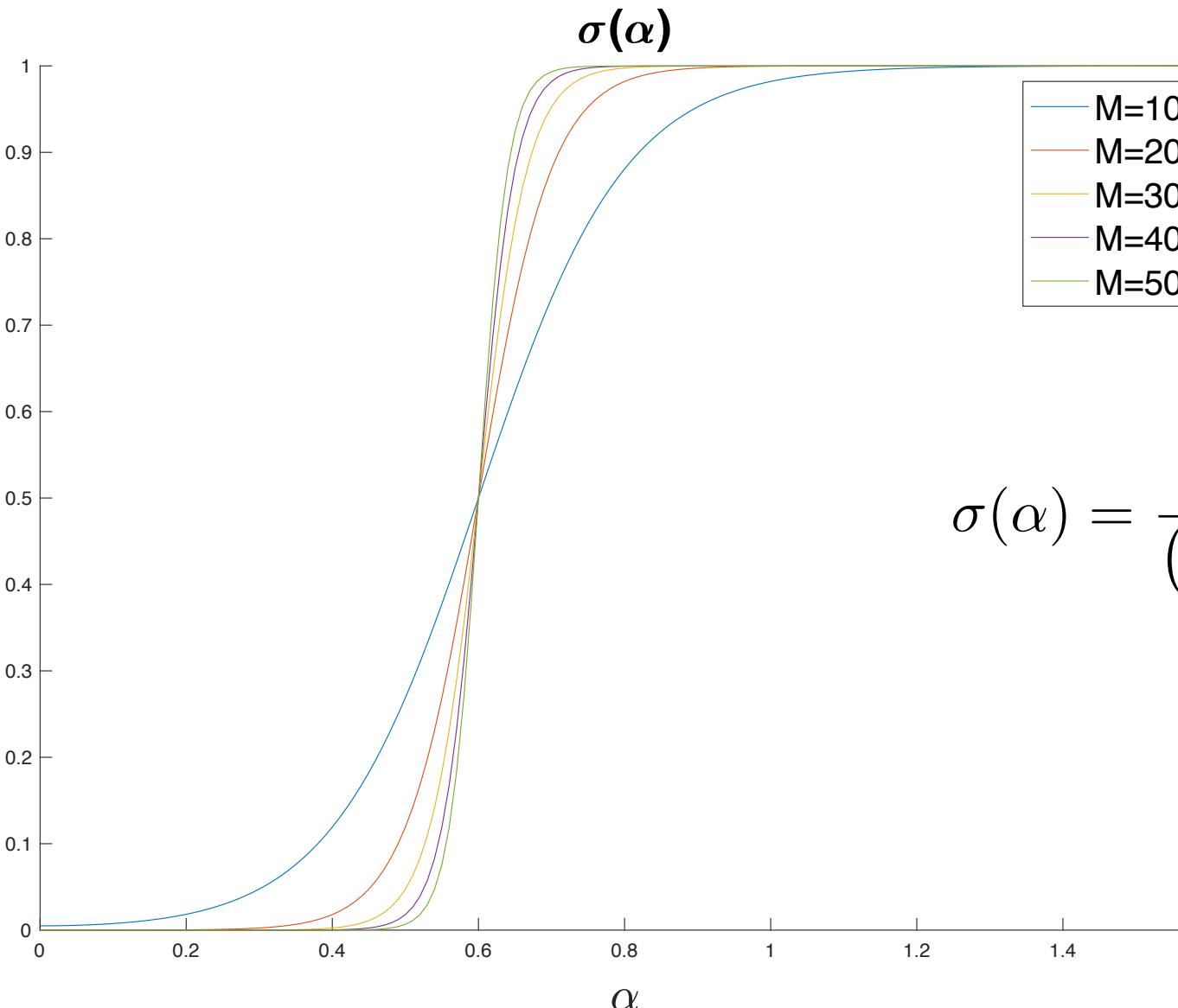
$$F_{\text{lift}} = \frac{1}{2} \rho V_a^2 S \left[C_L(\alpha) + C_{L_q} \frac{c}{2V_a} q + C_{L_{\delta_e}} \delta_e \right]$$

$$F_{\text{drag}} = \frac{1}{2} \rho V_a^2 S \left[C_D(\alpha) + C_{D_q} \frac{c}{2V_a} q + C_{D_{\delta_e}} \delta_e \right]$$

$$C_L(\alpha) = (1 - \sigma(\alpha)) \underbrace{[C_{L_0} + C_{L_\alpha} \alpha]}_{\text{linear model}} + \sigma(\alpha) \underbrace{[2 \operatorname{sign}(\alpha) \sin^2 \alpha \cos \alpha]}_{\text{flat-plate model}}$$

$$\sigma(\alpha) = \frac{1 + e^{-M(\alpha - \alpha_0)} + e^{M(\alpha + \alpha_0)}}{(1 + e^{-M(\alpha - \alpha_0)}) (1 + e^{M(\alpha + \alpha_0)})} \quad \text{blending function}$$

Blending Function



$$\sigma(\alpha) = \frac{1 + e^{-M(\alpha - \alpha_0)} + e^{M(\alpha + \alpha_0)}}{(1 + e^{-M(\alpha - \alpha_0)}) (1 + e^{M(\alpha + \alpha_0)})}$$

Nonlinear Aerodynamic Model

The stability derivative

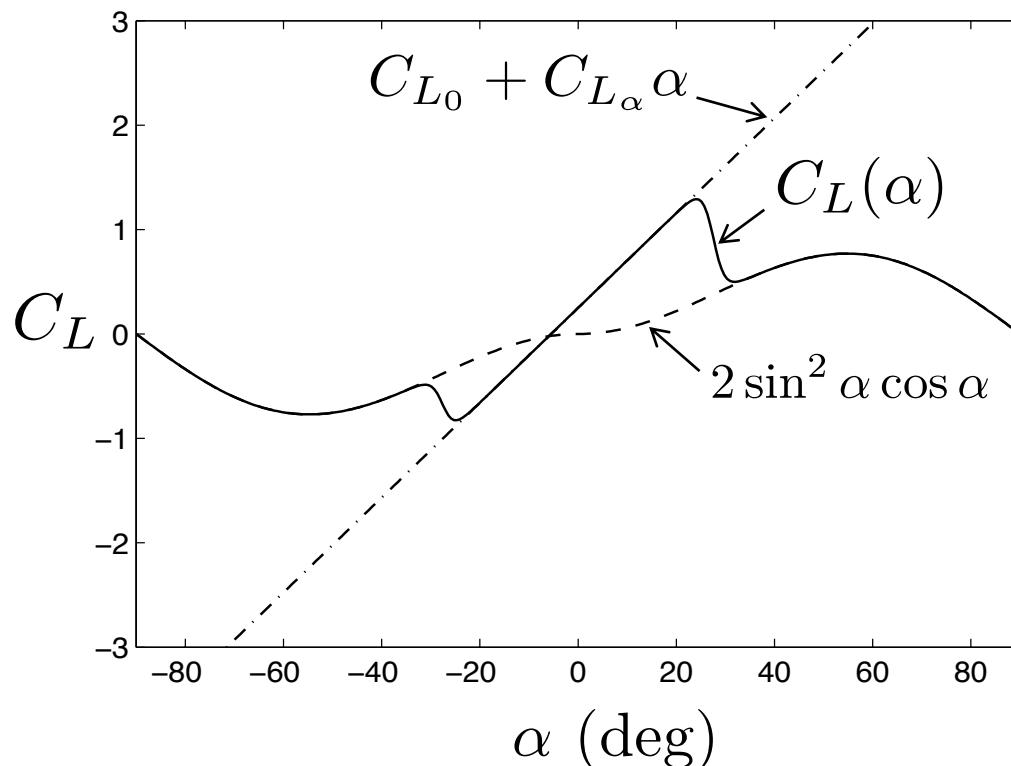
$$C_{L_\alpha} \approx \frac{\pi AR}{1 + \sqrt{1 + (AR/2)^2}}$$

represents the sensitivity of lift to the angle of attack, and can be approximated by physical dimensions of the airfoil, where

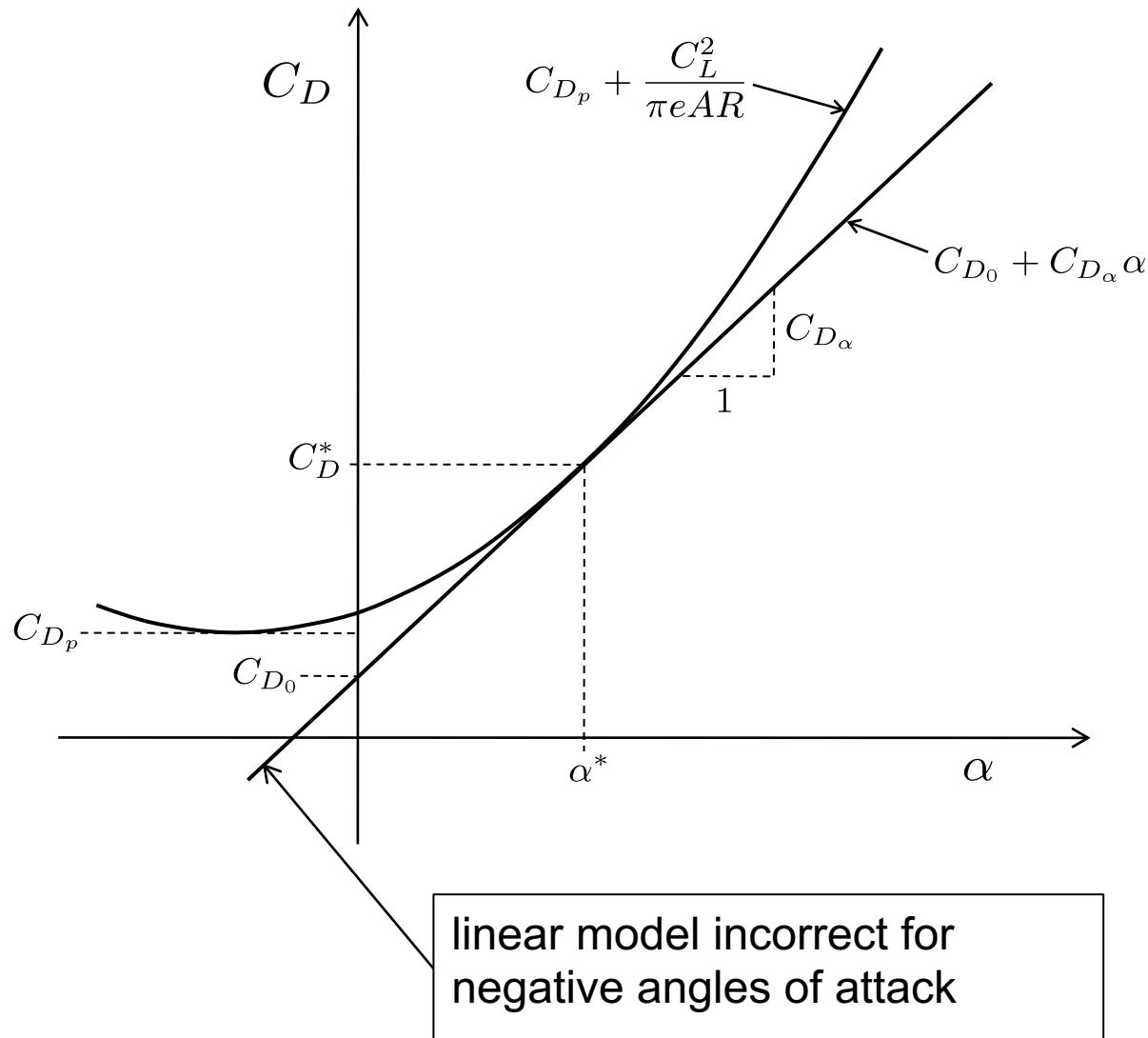
$AR \triangleq b^2/S$ is the wing aspect ratio

b is the wingspan

S is the wing area



Drag vs. Angle of Attack



$$\begin{aligned} C_D(\alpha) &= \underbrace{C_{D_p}}_{\text{parasitic drag}} + \underbrace{\frac{(C_{L_0} + C_{L_\alpha}\alpha)^2}{\pi e AR}}_{\text{induced drag}} \\ &= \left(C_{D_p} + \frac{C_{L_0}^2}{\pi e AR} \right) + \left(\frac{2C_{L_0}C_{L_\alpha}}{\pi e AR} \right)\alpha + \left(\frac{C_{L_\alpha}^2}{\pi e AR} \right)\alpha^2, \end{aligned}$$

where $0.8 \leq e \leq 1.0$ is the Oswald efficiency factor.

Parasitic drag: profile drag of wing, friction and pressure drag of tail surfaces, fuselage, landing gear, etc.

Induced drag: drag due to lift

Linear Lift and Drag Models

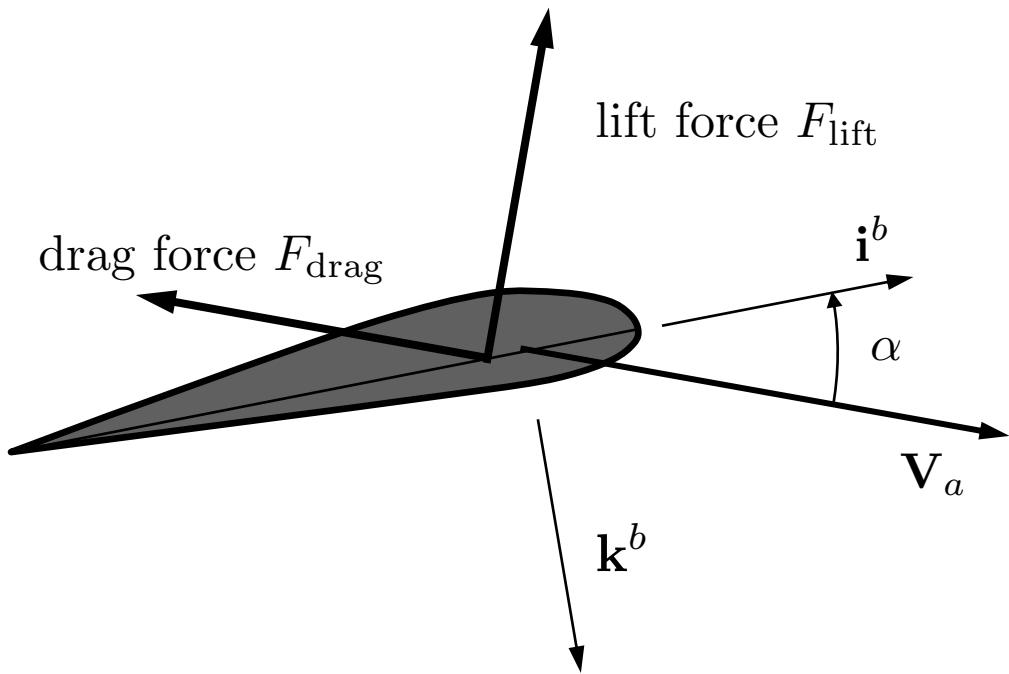
The linear lift and drag models

$$C_L(\alpha) \approx C_{L_0} + C_{L_\alpha} \alpha$$

$$C_D(\alpha) \approx C_{D_0} + C_{D_\alpha} \alpha$$

are valid for small deviations of angle of attack from trim

Longitudinal Forces – Body Frame



$$\begin{pmatrix} f_x \\ f_z \end{pmatrix} = \begin{pmatrix} \cos \alpha & -\sin \alpha \\ \sin \alpha & \cos \alpha \end{pmatrix} \begin{pmatrix} -F_{\text{drag}} \\ -F_{\text{lift}} \end{pmatrix}$$

$$= \frac{1}{2} \rho V_a^2 S \left(\begin{array}{l} [-C_D(\alpha) \cos \alpha + C_L(\alpha) \sin \alpha] \\ + [-C_{D_q} \cos \alpha + C_{L_q} \sin \alpha] \frac{c}{2V_a} q + [-C_{D_{\delta_e}} \cos \alpha + C_{L_{\delta_e}} \sin \alpha] \delta_e \\ \hline \hline \\ [-C_D(\alpha) \sin \alpha - C_L(\alpha) \cos \alpha] \\ + [-C_{D_q} \sin \alpha - C_{L_q} \cos \alpha] \frac{c}{2V_a} q + [-C_{D_{\delta_e}} \sin \alpha - C_{L_{\delta_e}} \cos \alpha] \delta_e \end{array} \right)$$

Pitching Moment

$$m = \frac{1}{2} \rho V_a^2 S c \left[C_{m_0} + C_{m_\alpha} \alpha + C_{m_q} \frac{c}{2V_a} q + C_{m_{\delta_e}} \delta_e \right]$$

- Uses linear model
- No rotation transformation necessary

Lateral Aerodynamics

$$f_y = \frac{1}{2} \rho V_a^2 S C_Y(\beta, p, r, \delta_a, \delta_r)$$

$$l = \frac{1}{2} \rho V_a^2 S b C_l(\beta, p, r, \delta_a, \delta_r)$$

$$n = \frac{1}{2} \rho V_a^2 S b C_n(\beta, p, r, \delta_a, \delta_r)$$

$$f_y \approx \frac{1}{2} \rho V_a^2 S \left[C_{Y_0} + C_{Y_\beta} \beta + C_{Y_p} \frac{b}{2V_a} p + C_{Y_r} \frac{b}{2V_a} r + C_{Y_{\delta_a}} \delta_a + C_{Y_{\delta_r}} \delta_r \right]$$

$$l \approx \frac{1}{2} \rho V_a^2 S b \left[C_{l_0} + C_{l_\beta} \beta + C_{l_p} \frac{b}{2V_a} p + C_{l_r} \frac{b}{2V_a} r + C_{l_{\delta_a}} \delta_a + C_{l_{\delta_r}} \delta_r \right]$$

$$n \approx \frac{1}{2} \rho V_a^2 S b \left[C_{n_0} + C_{n_\beta} \beta + C_{n_p} \frac{b}{2V_a} p + C_{n_r} \frac{b}{2V_a} r + C_{n_{\delta_a}} \delta_a + C_{n_{\delta_r}} \delta_r \right]$$

For symmetric aircraft, $C_{Y_0} = C_{l_0} = C_{n_0} = 0$

Aerodynamic Coefficients

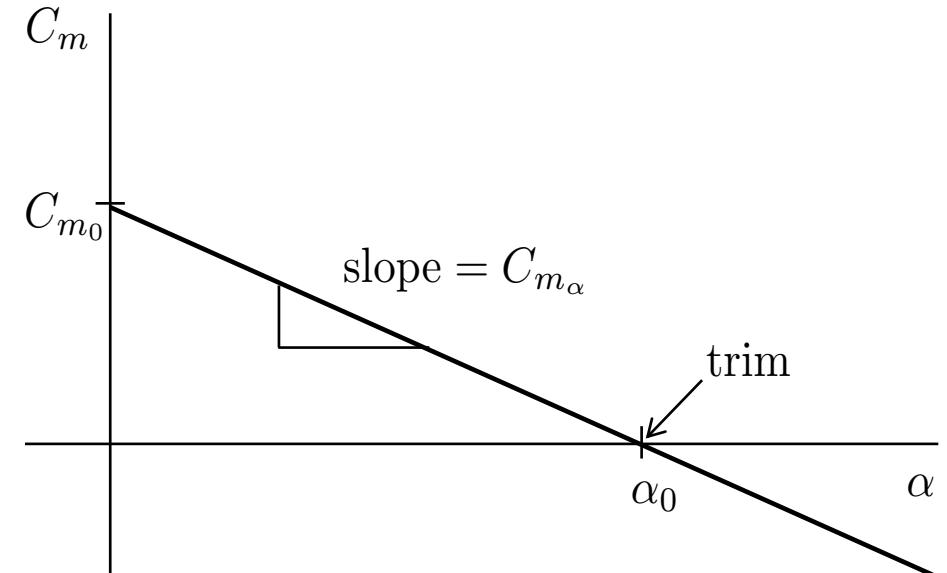
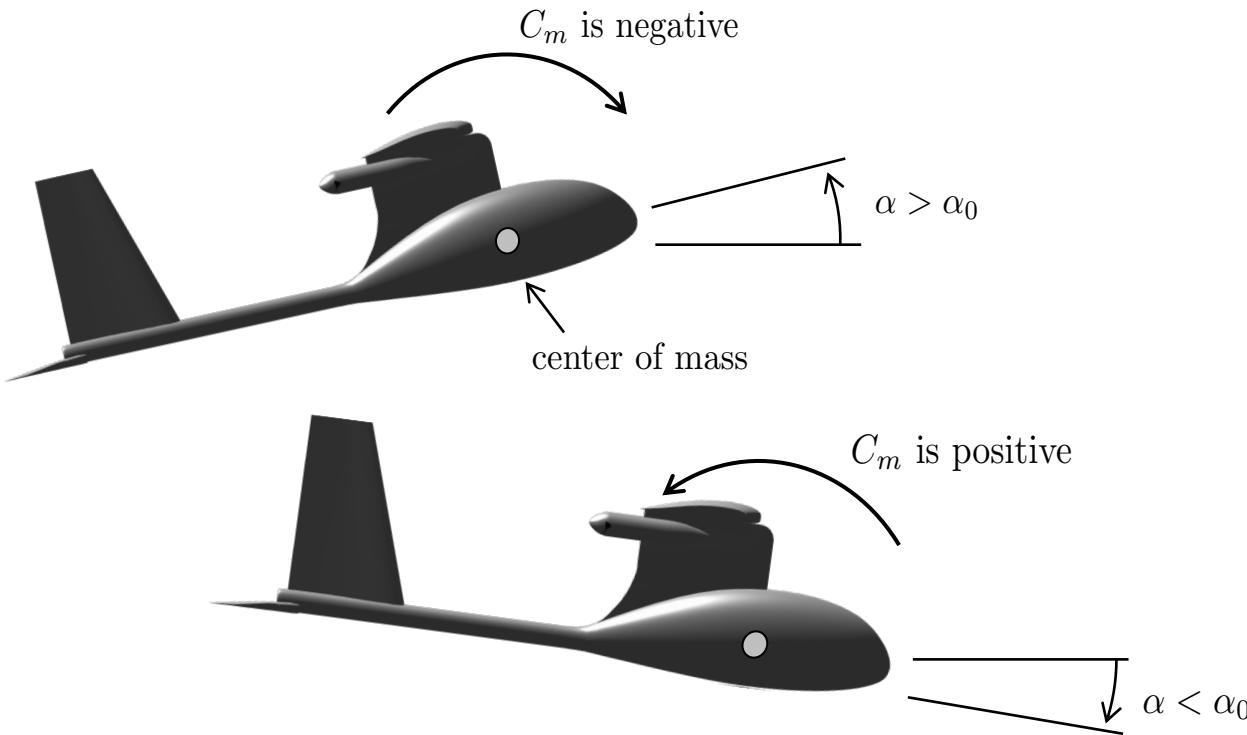
C_{m_α} , C_{ℓ_β} , C_{n_β} , C_{m_q} , C_{ℓ_p} , C_{n_r} are called the *stability derivatives* because their values determine the stability of the aircraft.

Static Stability Derivatives

- C_{m_α} - longitudinal static stability derivative. Must be ≤ 0 for stability: increase in α causes a downward pitching moment.
- C_{ℓ_β} - roll static stability derivative. Associated with dihedral in wings. Must be ≤ 0 for stability: positive roll ϕ causes a restoring moment.
- C_{n_β} - yaw static stability derivative. Weathercock stability derivative. Influenced by design of tail. Causes airframe to align with the wind vector. Must be ≥ 0 for stability: cocks airframe into wind driving β to zero.

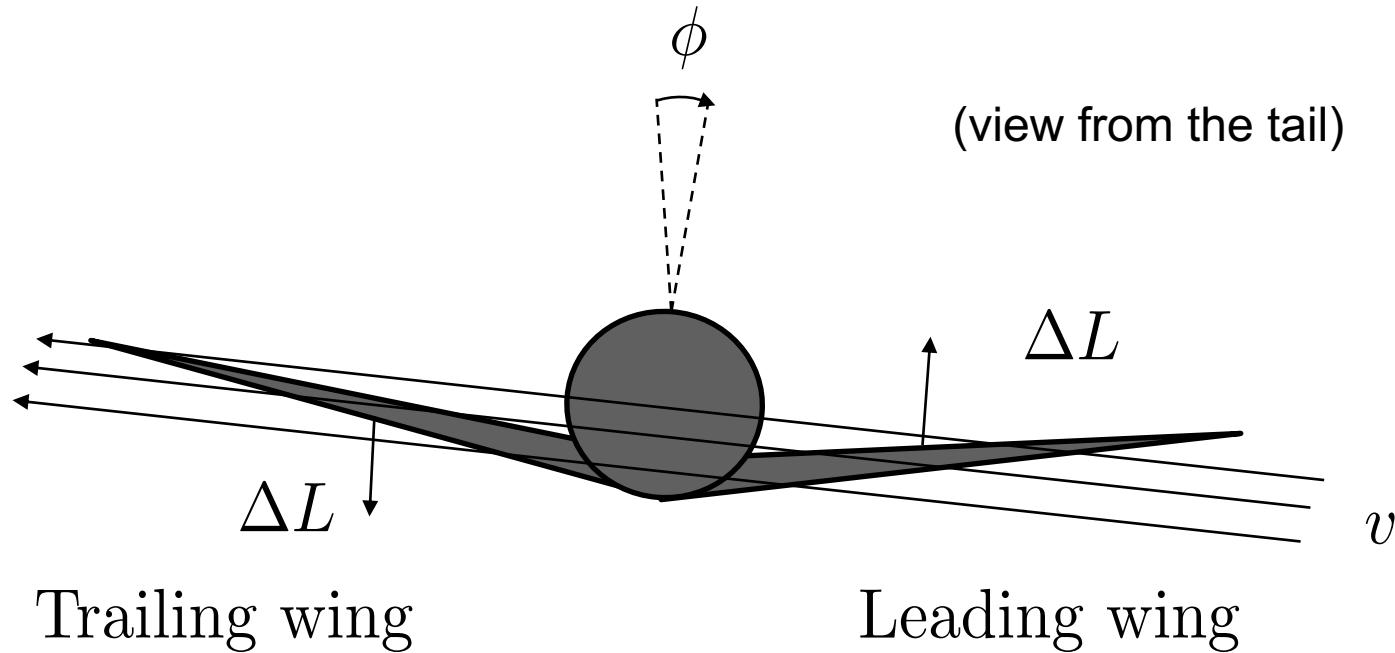
Static stability derivatives describe spring behavior of aerodynamics

Longitudinal Static Stability Derivative



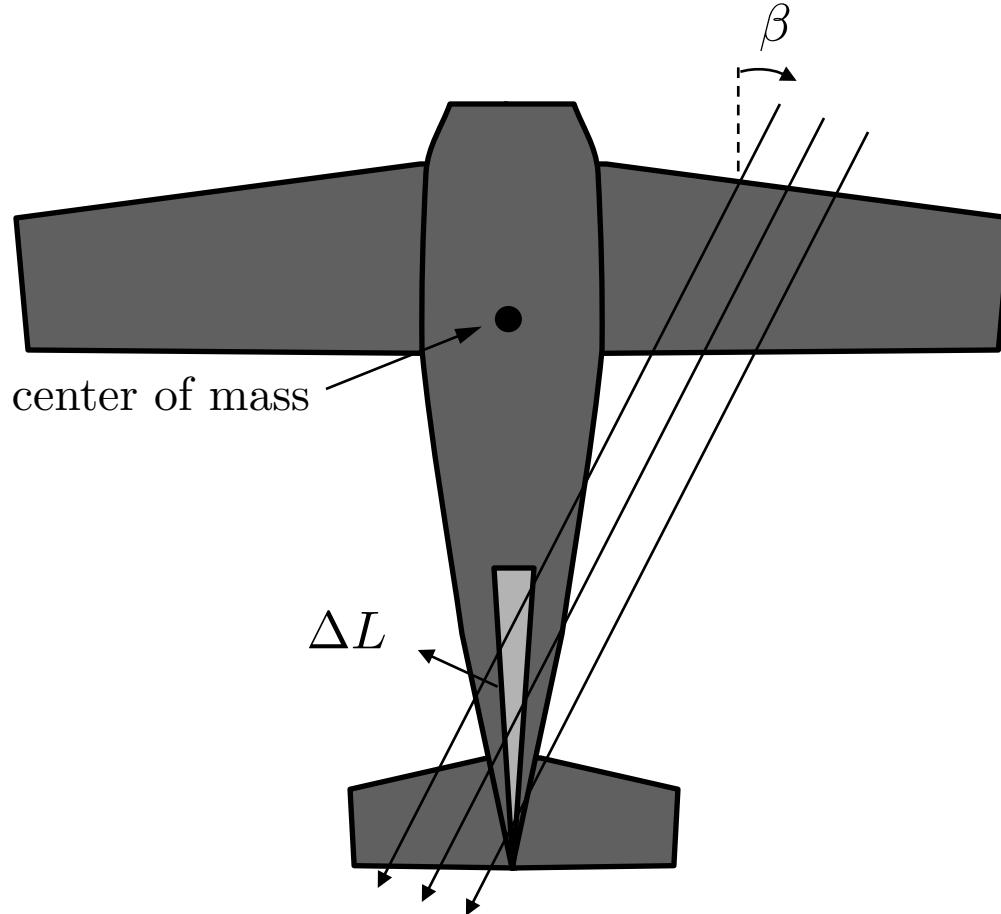
An aircraft is longitudinally statically stable if the pitching moment about the center of mass restores the aircraft to its trim angle of attack whenever it is perturbed away from trim. For this to be true, the pitching moment is positive when the angle of attack is zero ($C_{m_0} > 0$) and the pitching moment decreases as the angle of attack increases ($C_{m_\alpha} < 0$). Note that the pitching moment is zero at the trim condition ($\alpha = \alpha_0$).

Roll Static Stability Derivative



Given the wing dihedral, a roll angle of ϕ will cause a side velocity v , which induces a side slip angle β , which increases the lift on the leading wing, and decreases the lift on the trailing wing, causing a negative rolling moment. Hence the dihedral angle causes $C_{\ell_\beta} < 0$.

Yaw Static Stability Derivative



For a positive side slip angle β , the change in lift on the tail creates a moment about the center of mass, that pushes the nose toward the direction of the wind, or in other words, creates a positive yawing moment n . Hence $C_{n\beta} > 0$.

Aerodynamic Coefficients

Dynamic Stability Derivatives

- C_{m_q} , C_{ℓ_p} , C_{n_r} are known as the pitch damping derivative, roll damping derivative, and yaw damping derivative, respectively. They quantify the level of damping associated with angular motion of the airframe.

Dynamic stability derivatives describe damping behavior of aerodynamics

Control Derivatives

- $C_{m_{\delta_e}}$, $C_{\ell_{\delta_a}}$, and $C_{n_{\delta_r}}$ are the primary control derivatives and quantify the effect on the control surfaces on their primary intended axes of influence.
- $C_{\ell_{\delta_r}}$ and $C_{n_{\delta_a}}$ are the cross-control derivatives.

Propeller Thrust and Torque

If the propeller shaft is aligned along the body frame \mathbf{i}^b axis, then the resulting force and torque are given by

$$\mathbf{f}^b = \begin{pmatrix} T_p \\ 0 \\ 0 \end{pmatrix}$$

$$\mathbf{m}^b = \begin{pmatrix} Q_p \\ 0 \\ 0 \end{pmatrix}$$

Propeller Thrust and Torque

The thrust T_p and the torque Q_p along the propeller axis can be modeled as

$$T_p = \rho \left(\frac{\Omega}{2\pi} \right)^2 D^4 C_T(J)$$

$$Q_p = \rho \left(\frac{\Omega}{2\pi} \right)^2 D^5 C_Q(J),$$

where

$\rho \triangleq$ density of air

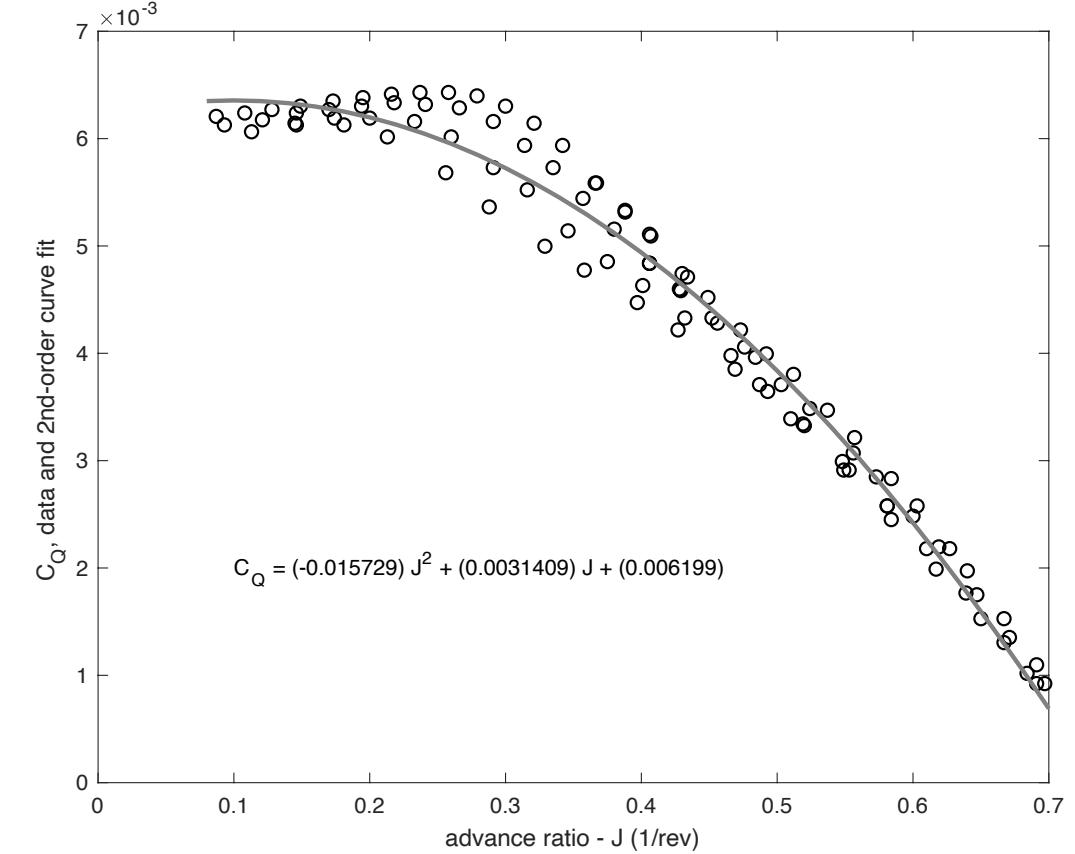
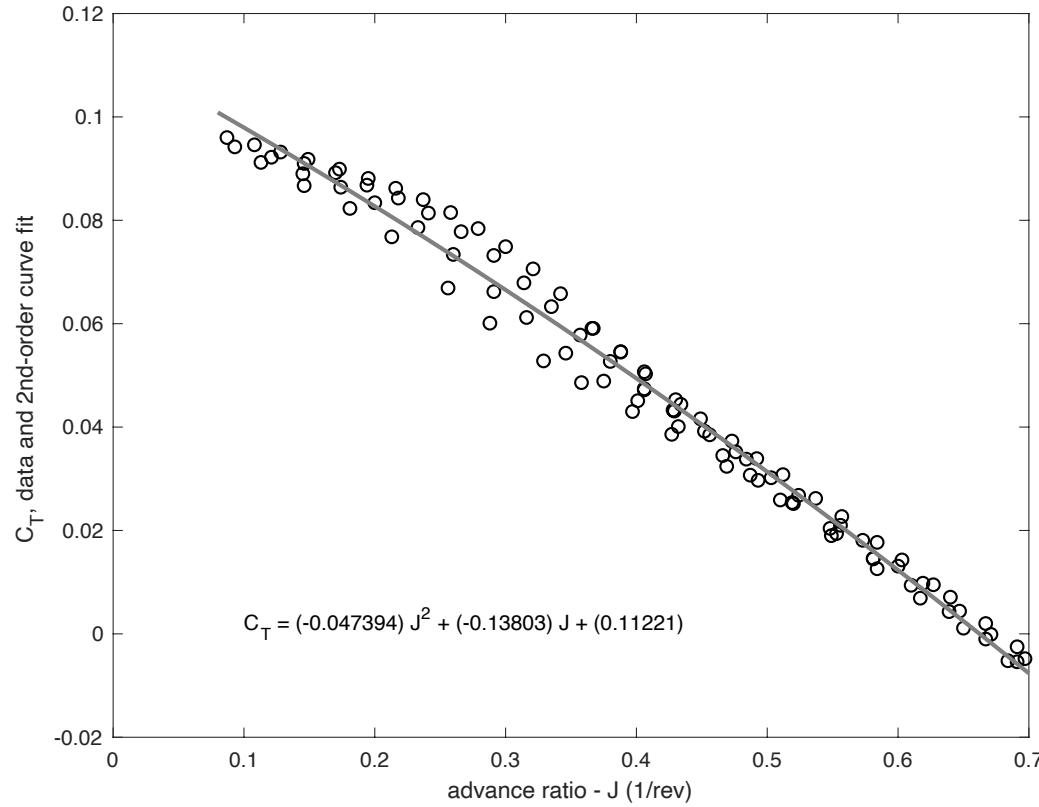
$D \triangleq$ propeller diameter (m)

$C_T, C_Q \triangleq$ aerodynamic coefficients

$J \triangleq \frac{V_a}{nD} = \frac{2\pi V_a}{\Omega D}$, advance ratio (unitless)

$\Omega \triangleq$ propeller speed (rad/sec)

Propeller Thrust and Torque



Experimental data indicates that

$$C_T(J) \approx C_{T2}J^2 + C_{T1}J + C_{T0}$$

$$C_Q(J) \approx C_{Q2}J^2 + C_{Q1}J + C_{Q0}$$

Propeller Thrust and Torque

For a DC motor, the steady-state torque generated for a given input voltage V_{in} is given by

$$Q_m = K_Q \left[\frac{1}{R} (V_{in} - K_V \Omega) - i_0 \right],$$

where

$R \triangleq$ resistance of the motor windings

$K_Q \triangleq$ motor torque constant

$K_V \triangleq$ back-emf voltage constant

$i_0 \triangleq$ zero-torque or no-load current

In our case, $V_{in} = V_{max} \delta_t$ where $\delta_t \in [0, 1]$ and V_{max} is the maximum battery voltage

Propeller Thrust and Torque

Draw the two torque curves – show their intersection as the operating speed.

For a given voltage input to our motor, we want to know the propeller speed and how much thrust the propeller produces. This will depend on the airspeed of the aircraft (and other things).

Propeller torque can be expressed as a function of propeller speed as:

$$Q_p(\Omega_p, C_Q) = \frac{\rho D^5}{4\pi^2} \Omega_p^2 C_Q$$

Motor torque can also be expressed as a function of motor speed:

$$Q_m = K_Q \left[\frac{1}{R} (V_{in} - K_V \Omega) - i_0 \right],$$

We know that the propeller torque and the motor torque are equivalent (they are connected by a shaft that we are assuming is rigid), so we can find the operating speed of the motor/prop combination by equating the torque expressions above and solving for $\Omega = \Omega_p$.

Propeller Thrust and Torque

In equilibrium, the motor torque and the propeller torque are equal. Setting $Q_p = Q_m$ gives

$$\left(\frac{\rho D^5}{(2\pi)^2} C_{Q0} \right) \Omega^2 + \left(\frac{\rho D^4}{2\pi} C_{Q1} V_a + \frac{K_Q K_V}{R} \right) \Omega + \left(\rho D^3 C_{Q2} V_a^2 - \frac{K_Q}{R} V_{in} + K_Q i_0 \right) = 0$$

Solving for the positive root gives the operating propeller speed

$$\Omega_{op} = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

where

$$\begin{aligned} a &= \frac{\rho D^5}{(2\pi)^2} C_{Q0} \\ b &= \frac{\rho D^4}{2\pi} C_{Q1} V_a + \frac{K_Q K_V}{R} \\ c &= \rho D^3 C_{Q2} V_a^2 - \frac{K_Q}{R} V_{in} + K_Q i_0 \end{aligned}$$

The associated advance ratio is: $J_{op} = \frac{2\pi V_a}{\Omega_{op} D}$

Propeller Thrust and Torque: Summary

Given $\delta_t \in [0, 1]$, solve for motor input voltage: $V_{in} = V_{max}\delta_t$

Solve for rotor angular speed as the positive root of:

$$\left(\frac{\rho D^5}{(2\pi)^2} C_{Q0} \right) \Omega^2 + \left(\frac{\rho D^4}{2\pi} C_{Q1} V_a + \frac{K_Q K_V}{R} \right) \Omega + \left(\rho D^3 C_{Q2} V_a^2 - \frac{K_Q}{R} V_{in} + K_Q i_0 \right) = 0$$

Find the advanced ratio $J = \frac{2\pi V_a}{\Omega_{op} D}$, and solve for the aerodynamic coefficients:

$$C_T(J) \approx C_{T2} J^2 + C_{T1} J + C_{T0}$$

$$C_Q(J) \approx C_{Q2} J^2 + C_{Q1} J + C_{Q0}$$

Solve for thrust and torque:

$$T_p = \rho \left(\frac{\Omega}{2\pi} \right)^2 D^4 C_T(J)$$

$$Q_p = \rho \left(\frac{\Omega}{2\pi} \right)^2 D^5 C_Q(J),$$

Propeller Thrust and Torque

```
# compute thrust and torque due to propeller (See addendum by McLain)
# map delta_t throttle command(0 to 1) into motor input voltage
V_in = MAV.V_max * delta_t
# Quadratic formula to solve for motor speed
a = MAV.C_Q0 * MAV.rho * np.power(MAV.D_prop, 5) \
    / ((2.*np.pi)**2)
b = (MAV.C_Q1 * MAV.rho * np.power(MAV.D_prop, 4) \
    / (2.*np.pi)) * self._Va + KQ**2/MAV.R_motor
c = MAV.C_Q2 * MAV.rho * np.power(MAV.D_prop, 3) \
    * self._Va**2 - (KQ / MAV.R_motor) * Volts + KQ * MAV.i0
# Consider only positive root
Omega_op = (-b + np.sqrt(b**2 - 4*a*c)) / (2.*a)
# compute advance ratio
J_op = 2 * np.pi * self._Va / (Omega_op * MAV.D_prop)
# compute non-dimensionalized coefficients of thrust and torque
C_T = MAV.C_T2 * J_op**2 + MAV.C_T1 * J_op + MAV.C_T0
C_Q = MAV.C_Q2 * J_op**2 + MAV.C_Q1 * J_op + MAV.C_Q0
# add thrust and torque due to propeller
n = Omega_op / (2 * np.pi)
fx += MAV.rho * n**2 * np.power(MAV.D_prop, 4) * C_T
Mx += -MAV.rho * n**2 * np.power(MAV.D_prop, 5) * C_Q
```

Wind Model

From the wind triangle:

$$\mathbf{V}_g = \mathbf{V}_a + \mathbf{V}_w$$

The wind vector can be decomposed into steady-state and gust components:

$$\mathbf{V}_w = \mathbf{V}_{w_s} + \mathbf{V}_{w_g}$$

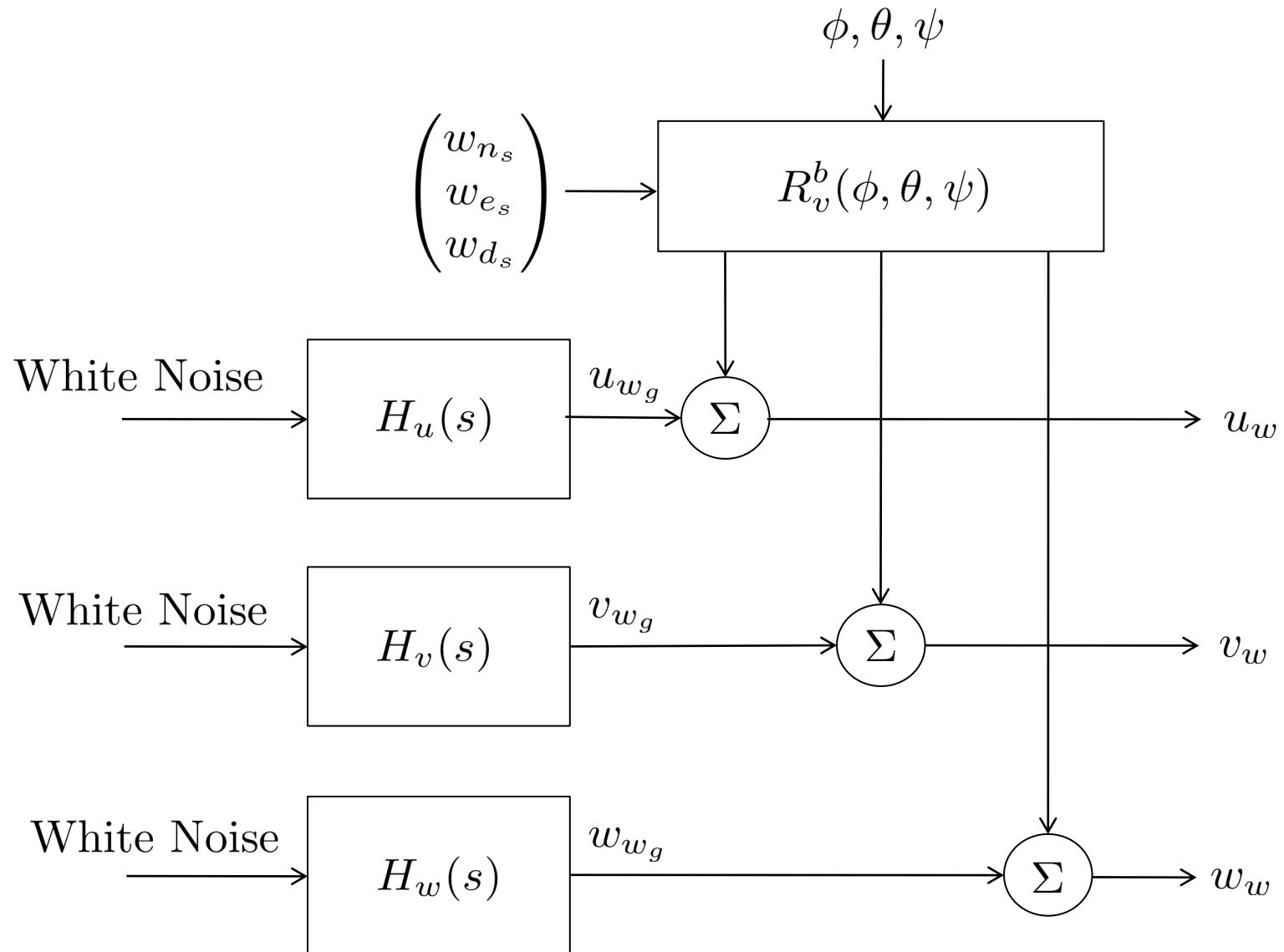
The steady-state component is typically expressed in the NED frame:

$$\mathbf{V}_{w_s}^i = \begin{pmatrix} w_{n_s} \\ w_{e_s} \\ w_{d_s} \end{pmatrix}$$

The gust component is typically expressed in the body frame:

$$\mathbf{V}_{w_g}^b = \begin{pmatrix} u_{w_g} \\ v_{w_g} \\ w_{w_g} \end{pmatrix}$$

Wind Model



Dryden Gust Model

$$H_u(s) = \sigma_u \sqrt{\frac{2V_a}{\pi L_u}} \frac{1}{s + \frac{V_a}{L_u}}$$

$$H_v(s) = \sigma_v \sqrt{\frac{3V_a}{\pi L_v}} \frac{\left(s + \frac{V_a}{\sqrt{3}L_v}\right)}{\left(s + \frac{V_a}{L_v}\right)^2}$$

$$H_w(s) = \sigma_w \sqrt{\frac{3V_a}{\pi L_w}} \frac{\left(s + \frac{V_a}{\sqrt{3}L_w}\right)}{\left(s + \frac{V_a}{L_w}\right)^2}$$

Table 1: Dryden gust model parameters

gust description	altitude (m)	$L_u = L_v$ (m)	L_w (m)	$\sigma_u = \sigma_v$ (m/s)	σ_w (m/s)
low altitude, light turbulence	50	200	50	1.06	0.7
low altitude, moderate turbulence	50	200	50	2.12	1.4
medium altitude, light turbulence	600	533	533	1.5	1.5
medium altitude, moderate turbulence	600	533	533	3.0	3.0

Transfer Function Implementation

To implement the system

$$Y(s) = \frac{as + b}{s^2 + cs + d} U(s),$$

first put the system into control canonical form

$$\begin{aligned}\dot{x} &= \begin{pmatrix} -c & -d \\ 1 & 0 \end{pmatrix} x + \begin{pmatrix} 1 \\ 0 \end{pmatrix} u \\ y &= (a \quad b) x,\end{aligned}$$

and then convert to discrete time using

$$\begin{aligned}x_{k+1} &= x_k + T_s(Ax_k + Bu_k) \\ y_k &= Cx_k\end{aligned}$$

where T_s is the sample rate, to get

$$\begin{aligned}x_{k+1} &= \begin{pmatrix} 1 - T_s c & -T_s d \\ T_s & 1 \end{pmatrix} x_k + \begin{pmatrix} T_s \\ 0 \end{pmatrix} u_k \\ y_k &= (a \quad b) x_k.\end{aligned}$$

For the Dryden model gust models, the input u_k will be zero mean Gaussian noise with unity variance.

Adding in the Effects of Wind

$$\mathbf{V}_w^b = \begin{pmatrix} u_w \\ v_w \\ w_w \end{pmatrix} = \mathcal{R}_v^b(\phi, \theta, \psi) \begin{pmatrix} w_{n_s} \\ w_{e_s} \\ w_{d_s} \end{pmatrix} + \begin{pmatrix} u_{w_g} \\ v_{w_g} \\ w_{w_g} \end{pmatrix}$$

$$\mathbf{V}_a^b = \begin{pmatrix} u_r \\ v_r \\ w_r \end{pmatrix} = \begin{pmatrix} u - u_w \\ v - v_w \\ w - w_w \end{pmatrix}$$

$$V_a = \sqrt{u_r^2 + v_r^2 + w_r^2}$$

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

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

Key concept:

- Wind and turbulence affect airspeed, angle of attack, and sideslip angle
- It is through these parameters that wind and atmospheric effects enter the calculation of aerodynamic forces and moments, and thereby affect motion of aircraft

Summary

$$\begin{pmatrix} f_x \\ f_y \\ f_z \end{pmatrix} = \begin{pmatrix} -mg \sin \theta \\ mg \cos \theta \sin \phi \\ mg \cos \theta \cos \phi \end{pmatrix} + \frac{1}{2} \rho V_a^2 S \begin{pmatrix} C_X(\alpha) + C_{X_q}(\alpha) \frac{c}{2V_a} q \\ C_{Y_0} + C_{Y_\beta} \beta + C_{Y_p} \frac{b}{2V_a} p + C_{Y_r} \frac{b}{2V_a} r \\ C_Z(\alpha) + C_{Z_q}(\alpha) \frac{c}{2V_a} q \end{pmatrix} \\ + \frac{1}{2} \rho V_a^2 S \begin{pmatrix} C_{X_{\delta_e}}(\alpha) \delta_e \\ C_{Y_{\delta_a}} \delta_a + C_{Y_{\delta_r}} \delta_r \\ C_{Z_{\delta_e}}(\alpha) \delta_e \end{pmatrix} + \begin{pmatrix} T_p(\delta_t V_a) \\ 0 \\ 0 \end{pmatrix}$$

$$C_X(\alpha) \triangleq -C_D(\alpha) \cos \alpha + C_L(\alpha) \sin \alpha$$

$$C_{X_q}(\alpha) \triangleq -C_{D_q} \cos \alpha + C_{L_q} \sin \alpha$$

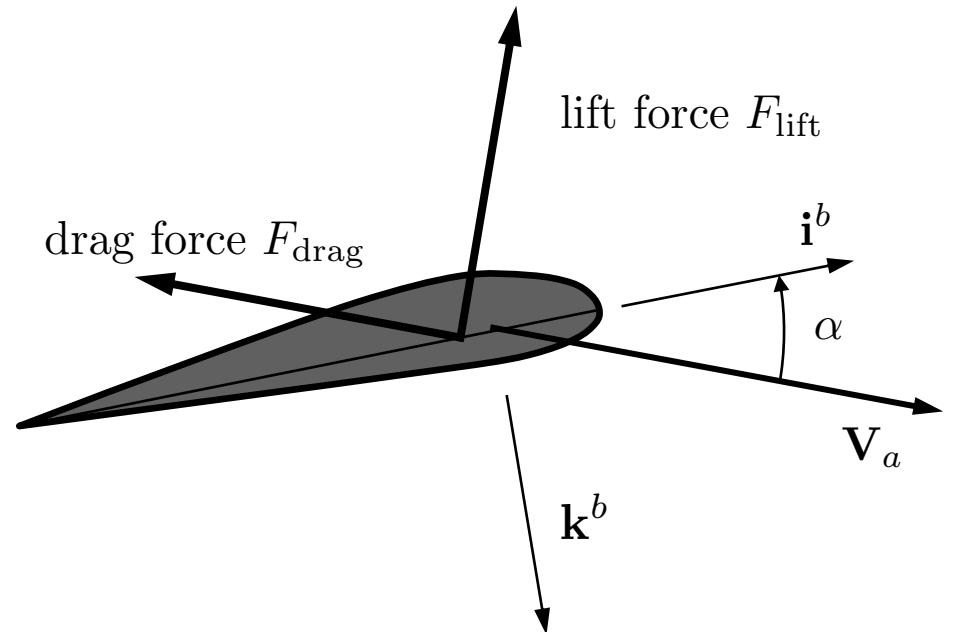
$$C_{X_{\delta_e}}(\alpha) \triangleq -C_{D_{\delta_e}} \cos \alpha + C_{L_{\delta_e}} \sin \alpha$$

$$C_Z(\alpha) \triangleq -C_D(\alpha) \sin \alpha - C_L(\alpha) \cos \alpha$$

$$C_{Z_q}(\alpha) \triangleq -C_{D_q} \sin \alpha - C_{L_q} \cos \alpha$$

$$C_{Z_{\delta_e}}(\alpha) \triangleq -C_{D_{\delta_e}} \sin \alpha - C_{L_{\delta_e}} \cos \alpha$$

$$\begin{pmatrix} l \\ m \\ n \end{pmatrix} = \frac{1}{2} \rho V_a^2 S \begin{pmatrix} b \left[C_{l_0} + C_{l_\beta} \beta + C_{l_p} \frac{b}{2V_a} p + C_{l_r} \frac{b}{2V_a} r \right] \\ c \left[C_{m_0} + C_{m_\alpha} \alpha + C_{m_q} \frac{c}{2V_a} q \right] \\ b \left[C_{n_0} + C_{n_\beta} \beta + C_{n_p} \frac{b}{2V_a} p + C_{n_r} \frac{b}{2V_a} r \right] \end{pmatrix} + \frac{1}{2} \rho V_a^2 S \begin{pmatrix} b \left[C_{l_{\delta_a}} \delta_a + C_{l_{\delta_r}} \delta_r \right] \\ c \left[C_{m_{\delta_e}} \delta_e \right] \\ b \left[C_{n_{\delta_a}} \delta_a + C_{n_{\delta_r}} \delta_r \right] \end{pmatrix} + \begin{pmatrix} Q_p(\delta_t, V_a) \\ 0 \\ 0 \end{pmatrix}$$



Project 4

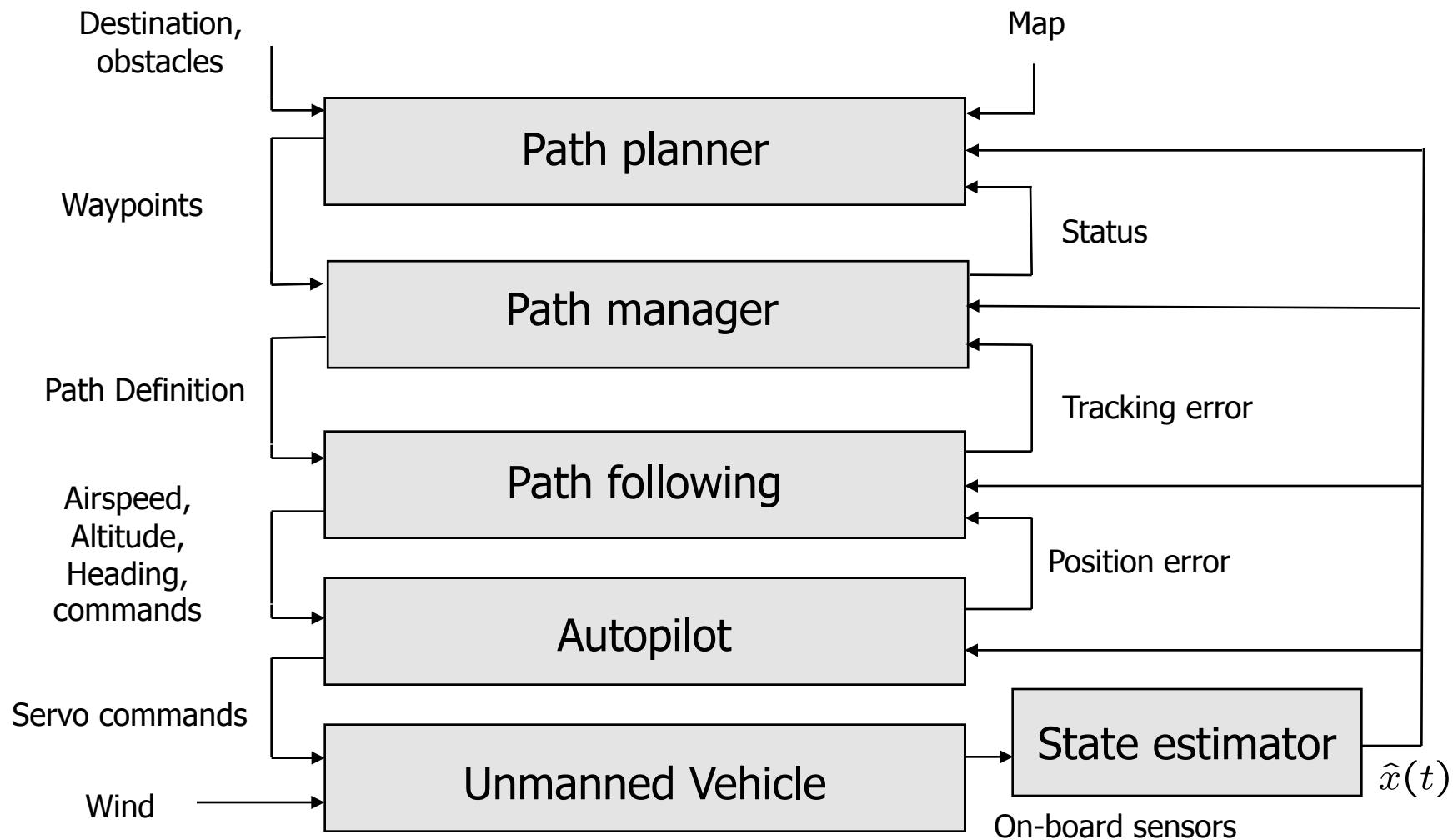
1. Add simulation of the wind to the mavsim simulator. The wind element should produce wind gust along the body axes, and steady state wind along the NED inertial axes.
2. Add forces and moments to the dynamics of the MAV. The inputs to the MAV should now be elevator, throttle, aileron, and rudder. The aerodynamic coefficients are given in Appendix E.
3. Verify your simulation by setting the control surface deflections to different values. Observe the response of the MAV. Does it behave as you think it should?



Chapter 5

Linear Design Models

Architecture



Equations of Motion

$$\dot{p}_n = (\cos \theta \cos \psi)u + (\sin \phi \sin \theta \cos \psi - \cos \phi \sin \psi)v + (\cos \phi \sin \theta \cos \psi + \sin \phi \sin \psi)w$$

$$\dot{p}_e = (\cos \theta \sin \psi)u + (\sin \phi \sin \theta \sin \psi + \cos \phi \cos \psi)v + (\cos \phi \sin \theta \sin \psi - \sin \phi \cos \psi)w$$

$$\dot{h} = u \sin \theta - v \sin \phi \cos \theta - w \cos \phi \cos \theta$$

$$\dot{u} = rv - qw - g \sin \theta + \frac{\rho V_a^2 S}{2m} \left[C_X(\alpha) + C_{X_q}(\alpha) \frac{cq}{2V_a} + C_{X_{\delta_e}}(\alpha) \delta_e \right] + \frac{T_p(\delta_t, V_a)}{m}$$

$$\dot{v} = pw - ru + g \cos \theta \sin \phi + \frac{\rho V_a^2 S}{2m} \left[C_{Y_0} + C_{Y_\beta} \beta + C_{Y_p} \frac{bp}{2V_a} + C_{Y_r} \frac{br}{2V_a} + C_{Y_{\delta_a}} \delta_a + C_{Y_{\delta_r}} \delta_r \right]$$

$$\dot{w} = qu - pv + g \cos \theta \cos \phi + \frac{\rho V_a^2 S}{2m} \left[C_Z(\alpha) + C_{Z_q}(\alpha) \frac{cq}{2V_a} + C_{Z_{\delta_e}}(\alpha) \delta_e \right]$$

$$\dot{\phi} = p + q \sin \phi \tan \theta + r \cos \phi \tan \theta$$

$$\dot{\theta} = q \cos \phi - r \sin \phi$$

$$\dot{\psi} = q \sin \phi \sec \theta + r \cos \phi \sec \theta$$

$$\dot{p} = \Gamma_1 pq - \Gamma_2 qr + \frac{1}{2} \rho V_a^2 S b \left[C_{p_0} + C_{p_\beta} \beta + C_{p_p} \frac{bp}{2V_a} + C_{p_r} \frac{br}{2V_a} + C_{p_{\delta_a}} \delta_a + C_{p_{\delta_r}} \delta_r \right] + \Gamma_3 Q_p(\delta_t, V_a)$$

$$\dot{q} = \Gamma_5 pr - \Gamma_6 (p^2 - r^2) + \frac{\rho V_a^2 Sc}{2J_y} \left[C_{m_0} + C_{m_\alpha} \alpha + C_{m_q} \frac{cq}{2V_a} + C_{m_{\delta_e}} \delta_e \right]$$

$$\dot{r} = \Gamma_7 pq - \Gamma_1 qr + \frac{1}{2} \rho V_a^2 S b \left[C_{r_0} + C_{r_\beta} \beta + C_{r_p} \frac{bp}{2V_a} + C_{r_r} \frac{br}{2V_a} + C_{r_{\delta_a}} \delta_a + C_{r_{\delta_r}} \delta_r \right] + \Gamma_4 Q_p(\delta_t, V_a)$$

Equations of Motion

$$C_{p_0} = \Gamma_3 C_{l_0} + \Gamma_4 C_{n_0}$$

$$C_{p_\beta} = \Gamma_3 C_{l_\beta} + \Gamma_4 C_{n_\beta}$$

$$C_{p_p} = \Gamma_3 C_{l_p} + \Gamma_4 C_{n_p}$$

$$C_{p_r} = \Gamma_3 C_{l_r} + \Gamma_4 C_{n_r}$$

$$C_{p_{\delta_a}} = \Gamma_3 C_{l_{\delta_a}} + \Gamma_4 C_{n_{\delta_a}}$$

$$C_{p_{\delta_r}} = \Gamma_3 C_{l_{\delta_r}} + \Gamma_4 C_{n_{\delta_r}}$$

$$C_{r_0} = \Gamma_4 C_{l_0} + \Gamma_8 C_{n_0}$$

$$C_{r_\beta} = \Gamma_4 C_{l_\beta} + \Gamma_8 C_{n_\beta}$$

$$C_{r_p} = \Gamma_4 C_{l_p} + \Gamma_8 C_{n_p}$$

$$C_{r_r} = \Gamma_4 C_{l_r} + \Gamma_8 C_{n_r}$$

$$C_{r_{\delta_a}} = \Gamma_4 C_{l_{\delta_a}} + \Gamma_8 C_{n_{\delta_a}}$$

$$C_{r_{\delta_r}} = \Gamma_4 C_{l_{\delta_r}} + \Gamma_8 C_{n_{\delta_r}}$$

$$C_X(\alpha) \triangleq -C_D(\alpha) \cos \alpha + C_L(\alpha) \sin \alpha$$

$$C_{X_q}(\alpha) \triangleq -C_{D_q} \cos \alpha + C_{L_q} \sin \alpha$$

$$C_{X_{\delta_e}}(\alpha) \triangleq -C_{D_{\delta_e}} \cos \alpha + C_{L_{\delta_e}} \sin \alpha$$

$$C_Z(\alpha) \triangleq -C_D(\alpha) \sin \alpha - C_L(\alpha) \cos \alpha$$

$$C_{Z_q}(\alpha) \triangleq -C_{D_q} \sin \alpha - C_{L_q} \cos \alpha$$

$$C_{Z_{\delta_e}}(\alpha) \triangleq -C_{D_{\delta_e}} \sin \alpha - C_{L_{\delta_e}} \cos \alpha$$

Lift and Drag Models

Nonlinear model with stall effects

$$C_L(\alpha) = (1 - \sigma(\alpha)) [C_{L_0} + C_{L_\alpha} \alpha] + \sigma(\alpha) [2 \operatorname{sign}(\alpha) \sin^2 \alpha \cos \alpha]$$

$$\sigma(\alpha) = \frac{1 + e^{-M(\alpha - \alpha_0)} + e^{M(\alpha + \alpha_0)}}{(1 + e^{-M(\alpha - \alpha_0)}) (1 + e^{M(\alpha + \alpha_0)})}$$

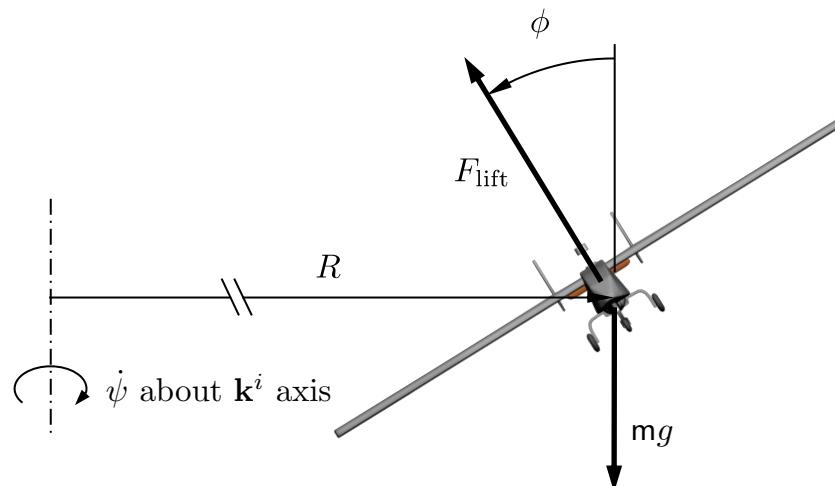
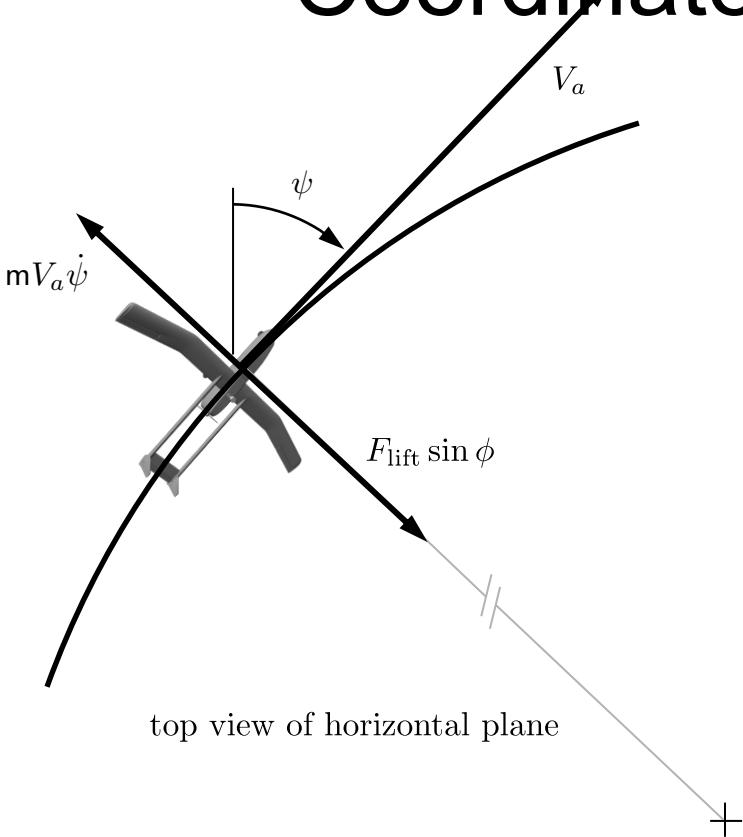
$$C_D(\alpha) = C_{D_p} + \frac{(C_{L_0} + C_{L_\alpha} \alpha)^2}{\pi e A R}$$

Linear model

$$C_L(\alpha) = C_{L_0} + C_{L_\alpha} \alpha$$

$$C_D(\alpha) = C_{D_0} + C_{D_\alpha} \alpha$$

Coordinated Turn – No Wind



$$F_{\text{lift}} \cos \phi = mg$$

$$\begin{aligned} F_{\text{lift}} \sin \phi &= m \frac{v^2}{R} \\ &= mv\omega \\ &= mV_a \dot{\psi} \end{aligned}$$

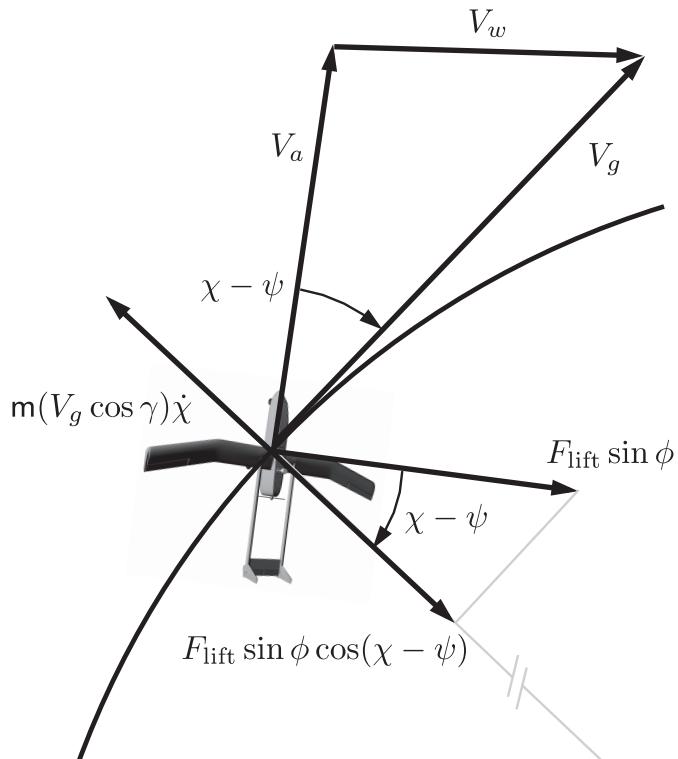
Dividing gives

$$\tan \phi = \frac{V_a \dot{\psi}}{g}$$

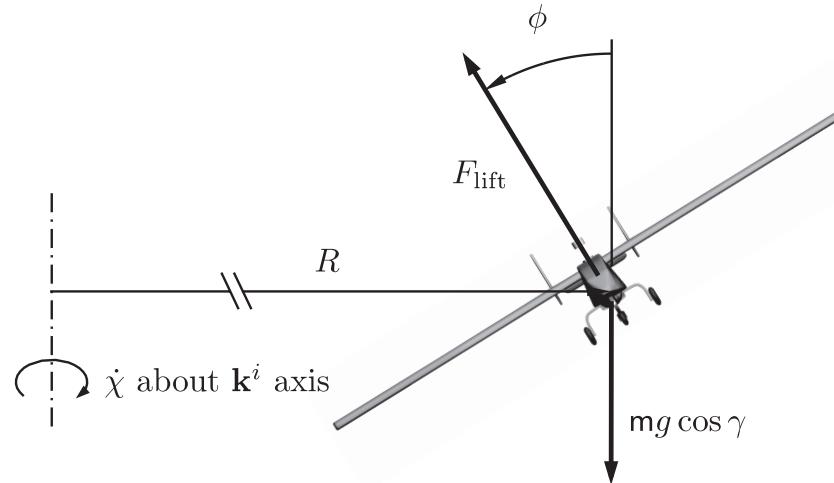
and

$$\dot{\psi} = \frac{g}{V_a} \tan \phi$$

Coordinated Turn



$$\begin{aligned}
 F_{lift} \sin \phi \cos(\chi - \psi) &= m \frac{v^2}{R} \\
 &= m v \omega \\
 &= m (V_g \cos \gamma) \dot{\chi}
 \end{aligned}$$



view in direction of $-\mathbf{i}^b$ axis
forces shown in $\mathbf{j}^b\text{-}\mathbf{k}^b$ plane

$$F_{lift} \cos \phi = mg \cos \gamma$$

Coordinated Turn

Dividing the two expressions gives

$$\dot{\chi} = \frac{g}{V_g} \tan \phi \cos(\chi - \psi)$$

which is the coordinated turn condition in wind

In the absence of wind, we have $V_a = V_g$ and $\psi = \chi$ which gives

$$\dot{\psi} = \frac{g}{V_a} \tan \phi$$

which is the expression commonly seen in the literature

The turning radius is given by

$$R = \frac{V_g \cos \gamma}{\dot{\chi}} = \frac{V_g^2 \cos \gamma}{g \tan \phi \cos(\chi - \psi)}$$

For level flight in the absence of wind we have the standard formula

$$R = \frac{V_a^2}{g \tan \phi}$$

Trim Conditions

Given the nonlinear system

$$\dot{x} = f(x, u)$$

The equilibria (x^*, u^*) are defined by

$$f(x^*, u^*) = 0$$

An aircraft in equilibrium is in a *trim condition*. In general, trim conditions may include states that are not constant. Therefore, trim conditions are given by

$$\dot{x}^* = f(x^*, u^*)$$

Calculating Trim

Objective is to compute trim states and inputs when aircraft simultaneously satisfies three conditions:

- Traveling at constant speed V_a^*
- Climbing at constant flight path angle γ^*
- In constant orbit of radius R^*

V_a^* , γ^* , and R^* , are inputs to the trim calculations

States: $x \triangleq (p_n, p_e, p_d, u, v, w, \phi, \theta, \psi, p, q, r)^\top$

Inputs: $u \triangleq (\delta_e, \delta_a, \delta_r, \delta_t)^\top$

Calculating Trim

For constant-climb orbit:

- speed of aircraft not changing → $\dot{u}^* = \dot{v}^* = \dot{w}^* = 0$
- roll and pitch angles constant → $\dot{\phi}^* = \dot{\theta}^* = \dot{p}^* = \dot{q}^* = 0$

Turn rate constant and given by

$$\dot{\psi}^* = \frac{V_a^*}{R^*} \cos \gamma^* \quad \rightarrow \quad \dot{r}^* = 0$$

Climb rate constant, and given by

$$\dot{h}^* = V_a^* \sin \gamma^*$$

Given parameters V_a^* , γ^* , and R^* , can specify \dot{x}^* as

$$\dot{x}^* = (p_n^* \ p_e^* \ \dot{h}^* \ \dot{u}^* \ \dot{v}^* \ \dot{w}^* \ \dot{\phi}^* \ \dot{\theta}^* \ \dot{\psi}^* \ \dot{p}^* \ \dot{q}^* \ \dot{r}^*)^\top$$

(continued...)

Calculating Trim

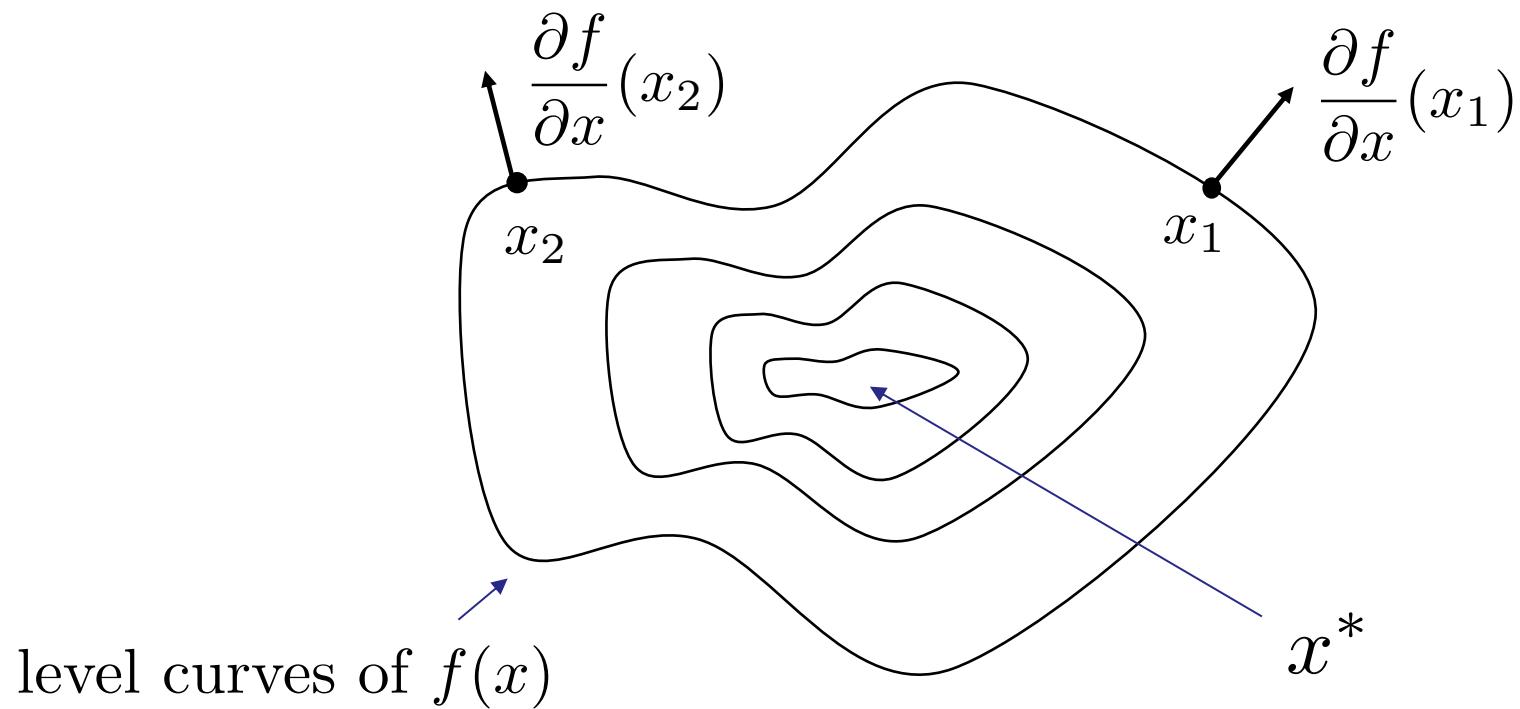
Given parameters V_a^* , γ^* , and R^* , can specify \dot{x}^* as

$$\dot{x}^* = \begin{pmatrix} \dot{p}_n^* \\ \dot{p}_e^* \\ \dot{h}^* \\ \dot{u}^* \\ \dot{v}^* \\ \dot{w}^* \\ \dot{\phi}^* \\ \dot{\theta}^* \\ \dot{\psi}^* \\ \dot{p}^* \\ \dot{q}^* \\ \dot{r}^* \end{pmatrix} = \begin{pmatrix} [\text{don't care}] \\ [\text{don't care}] \\ V_a^* \sin \gamma^* \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \frac{V_a^*}{R^*} \cos \gamma^* \\ 0 \\ 0 \\ 0 \end{pmatrix} = f(x^*, u^*)$$

Problem of finding x^* and u^* such that $\dot{x}^* = f(x^*, u^*)$, reduces to solving nonlinear algebraic systems of equations

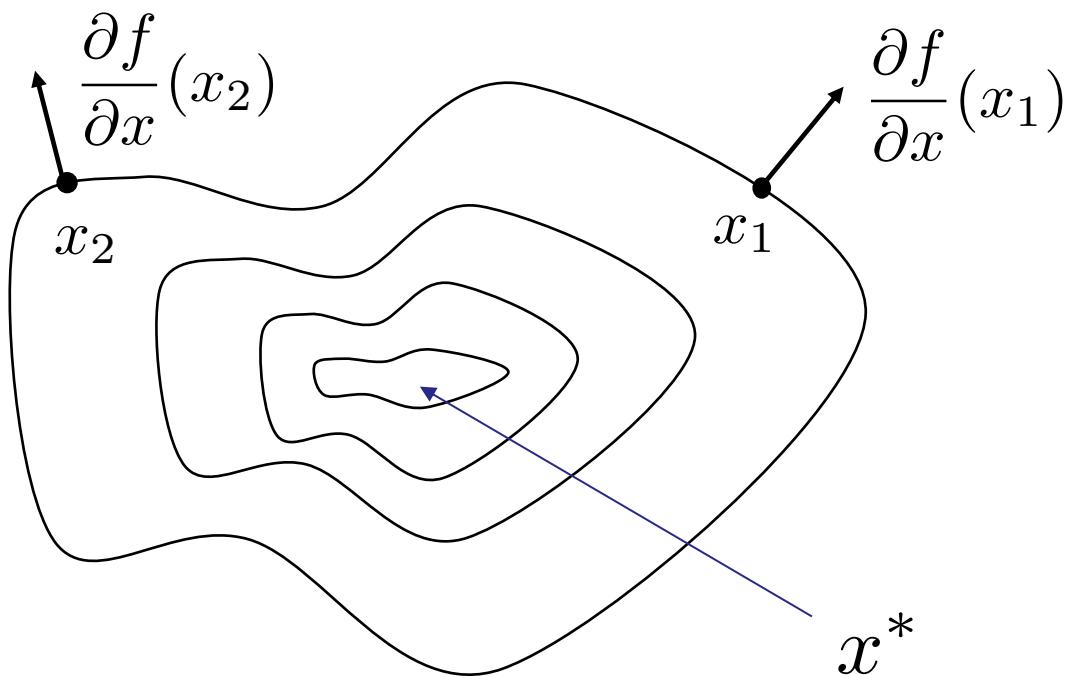
Can use Matlab trim command – see Appendix F

Unconstrained Optimization



$$x^* = \arg \min_{x \in \mathbb{R}^n} f(x)$$

Unconstrained Optimization



Gradient descent algorithm:

$$x^{[k+1]} = x^{[k]} - \alpha_k \frac{\partial f}{\partial x}(x^{[k]})$$

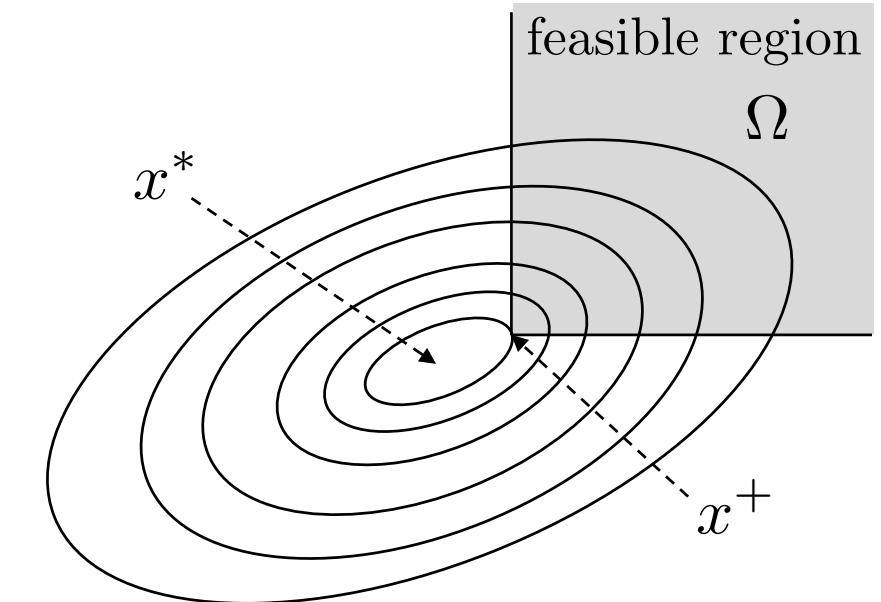
Constrained Optimization

Can generally pose as:

$$\min_{x \in \Omega} f(x)$$

Usually more manageable to pose as:

$$\begin{aligned} & \min_{x \in \mathbb{R}^n} f(x) \\ \text{s.t. } & h_1(x) = 0, \\ & \vdots, \\ & h_m(x) = 0, \\ & g_1(x) \leq 0, \\ & \vdots, \\ & g_p(x) \leq 0 \end{aligned}$$

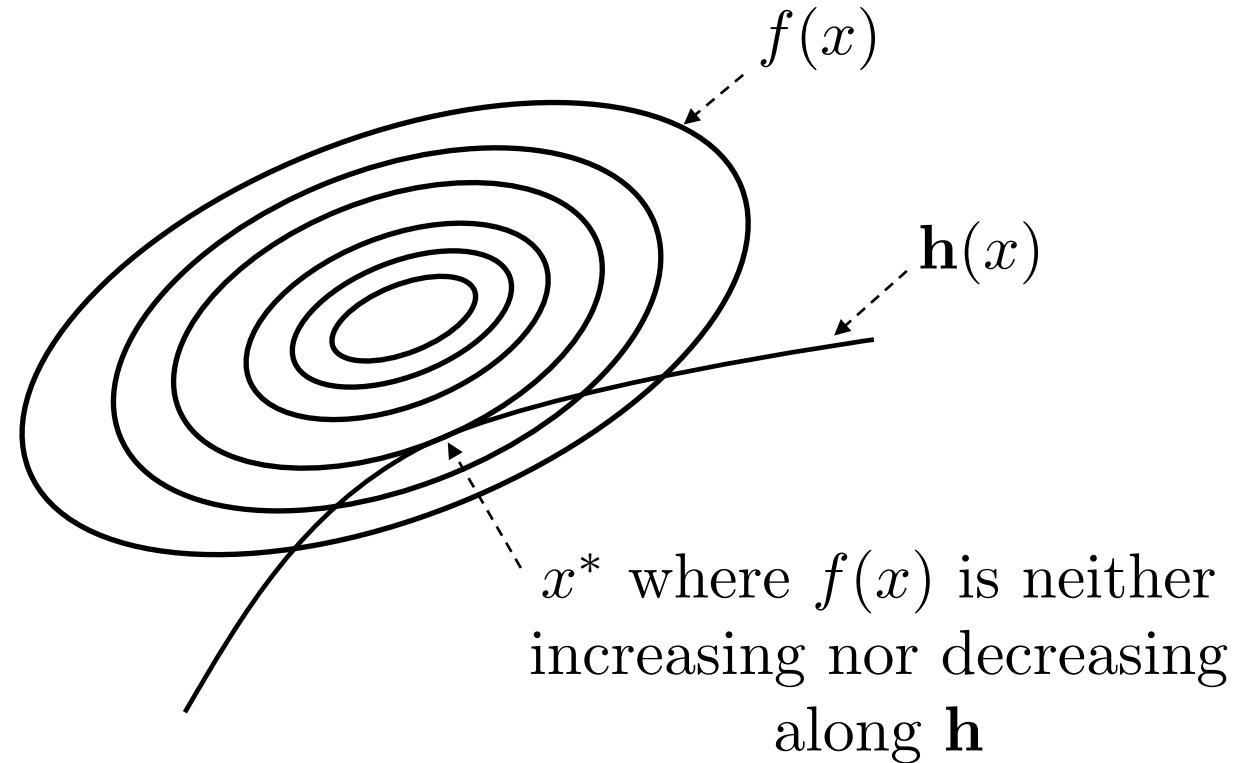


Note that the constrained optimum x^+ does not equal the unconstrained optimum x^* .

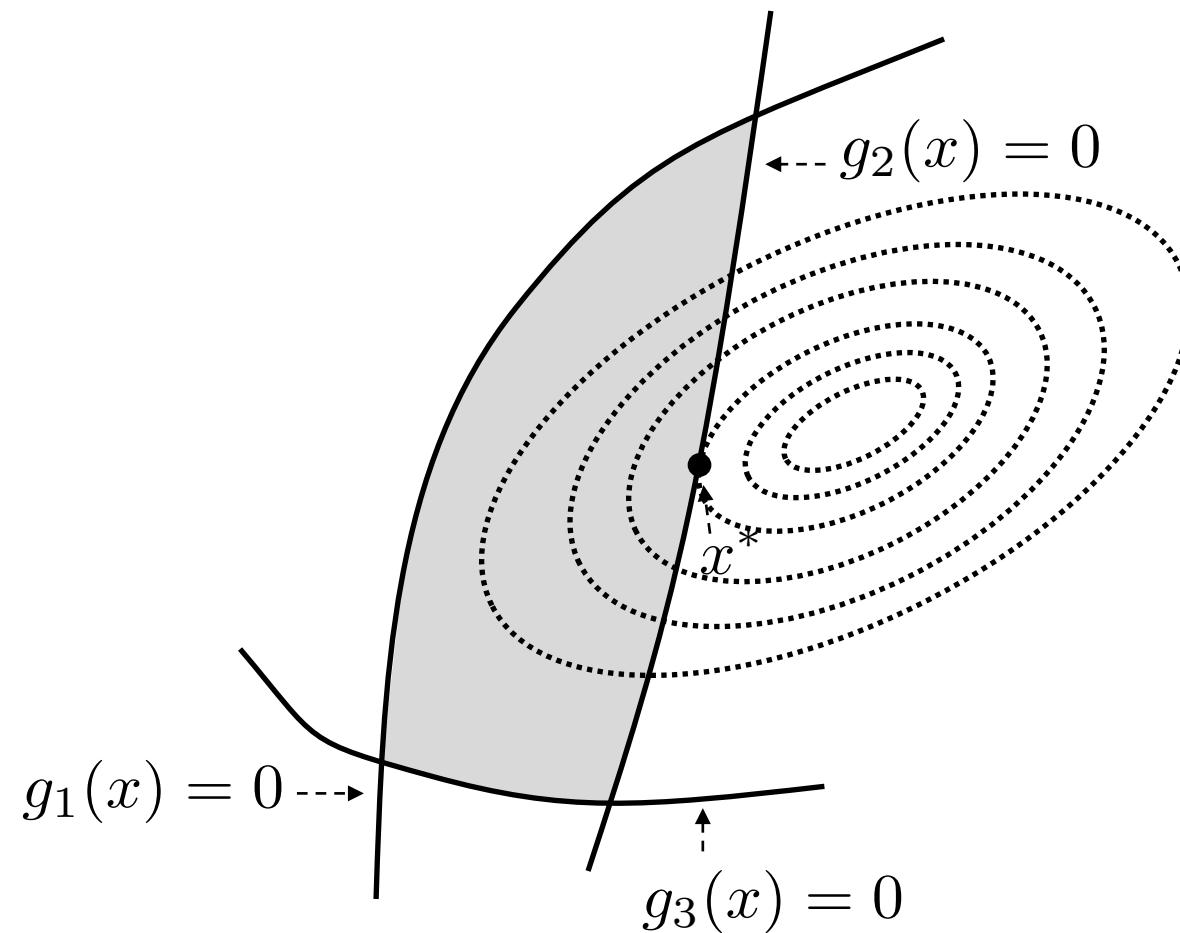
$h_i(x) = 0$ are called “equality constraints”

$g_i(x) \leq 0$ are called “inequality constraints”

Equality Constraints



Inequality Constraints



Optimization in Python

```
from scipy.optimize import minimize

minimize(fun, # function to be minimized
           x0, # initial guess for gradient descent
           args=(), # additional arguments for fun
           method=None, # optimization method
           jac=None, # function that defines Jacobian of f
           hess=None, # function that defines Hessian of f
           hessp=None, # Hessian in a specific direction
           bounds=None, # inequality bounds on x
           constraints=(), # definition of constraints
           tol=None, # tolerance for termination
           callback=None, # function called after each iteration
           options=None) # other algorithm options
```

Example: Optimization in Python

```
from scipy.optimize import minimize
import numpy as np

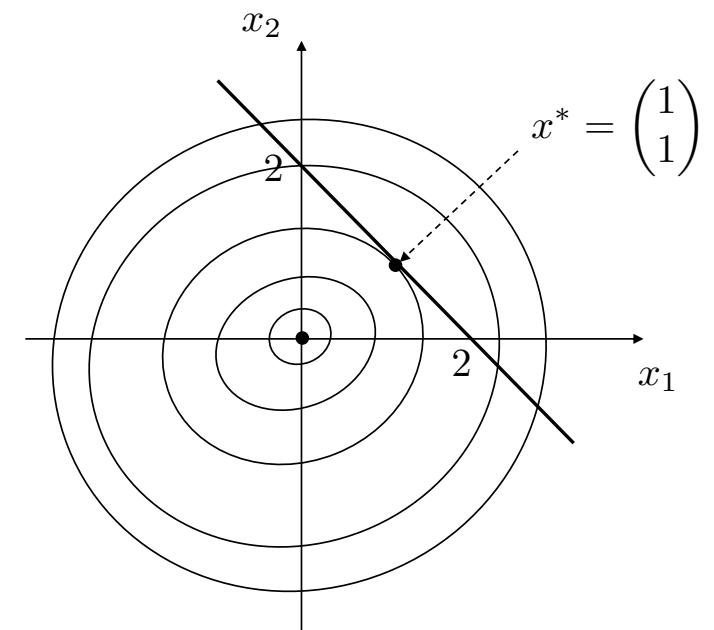
def objective(x):
    return(x[0]**2 + x[1]**2)

x0 = np.array([[5], [2]]) # initial condition
cons = ({'type': 'eq',
          'fun': lambda x: np.array([
              x[0]+x[1]-2, # x1+x2=2
          ]),
      })

res = minimize(objective, x0, method='SLSQP', constraints=cons)
print("xstar =", res.x)
```

$$\min x_1^2 + x_2^2$$

$$\text{s.t. } x_1 + x_2 = 2$$



Objective Function for Trim

```
def trim_objective_fun(x, mav, Va, gamma):
    state = x[0:13]
    delta = MsgDelta(elevator=x.item(13),
                     aileron=x.item(14),
                     rudder=x.item(15),
                     throttle=x.item(16))
    desired_trim_state_dot
        = np.array([[0., 0., -Va*np.sin(gamma), 0.,
                    0., 0., 0., 0., 0., 0., 0., 0.]]) .T
    mav._state = state
    mav._update_velocity_data()
    forces_moments = mav._forces_moments(delta)
    f = mav._derivatives(state, forces_moments)
    tmp = desired_trim_state_dot - f
    J = np.linalg.norm(tmp[2:13])**2.0
    return J
```

Constraints for Trim

```
cons = ({'type': 'eq',
      'fun': lambda x: np.array([
          x[3]**2 + x[4]**2 + x[5]**2 - Va**2, # magnitude of velocity
          # vector is Va
          x[4], # v=0, force side velocity to be zero
          x[6]**2 + x[7]**2 + x[8]**2 + x[9]**2 - 1., # quaternion is unit length
          x[7], # e1=0 - forcing e1=e3=0 ensures zero roll and zero yaw in trim
          x[9], # e3=0
          x[10], # p=0 - angular rates should all be zero
          x[11], # q=0
          x[12], # r=0
      ]),
      'jac': lambda x: np.array([
          [0., 0., 0., 2*x[3], 2*x[4], 2*x[5], 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 2*x[6], 2*x[7], 2*x[8], 2*x[9], 0., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.]
      ])
  })
```

Lateral Transfer Functions - Roll

Objective: Find simple transfer function relationship between ϕ and δ_a

Start with: $\dot{\phi} = p + q \sin \phi \tan \theta + r \cos \phi \tan \theta$

Assuming that θ is small and defining the linearization error (can be considered as a control disturbance) to be $d_{\phi_1} \triangleq q \sin \phi \tan \theta + r \cos \phi \tan \theta$ gives

$$\dot{\phi} = p + d_{\phi_1}$$

Differentiating and substituting for $\dot{\phi}$ gives

$$\begin{aligned}\ddot{\phi} &= \dot{p} + \dot{d}_{\phi_1} \\ &= \Gamma_1 pq - \Gamma_2 qr + \frac{1}{2} \rho V_a^2 Sb \left[C_{p_0} + C_{p_\beta} \beta + C_{p_p} \frac{bp}{2V_a} + C_{p_r} \frac{br}{2V_a} + C_{p_{\delta_a}} \delta_a + C_{p_{\delta_r}} \delta_r \right] + \dot{d}_{\phi_1} \\ &= \Gamma_1 pq - \Gamma_2 qr + \frac{1}{2} \rho V_a^2 Sb \left[C_{p_0} + C_{p_\beta} \beta + C_{p_p} \frac{b}{2V_a} (\dot{\phi} - d_{\phi_1}) + C_{p_r} \frac{br}{2V_a} + C_{p_{\delta_a}} \delta_a + C_{p_{\delta_r}} \delta_r \right] + \dot{d}_{\phi_1} \\ &= \left(\frac{1}{2} \rho V_a^2 Sb C_{p_p} \frac{b}{2V_a} \right) \dot{\phi} + \left(\frac{1}{2} \rho V_a^2 Sb C_{p_{\delta_a}} \right) \delta_a + \left\{ \Gamma_1 pq - \Gamma_2 qr + \frac{1}{2} \rho V_a^2 Sb \left[C_{p_0} + C_{p_\beta} \beta - C_{p_p} \frac{b}{2V_a} (d_{\phi_1}) + C_{p_r} \right. \right. \\ &\quad \left. \left. \left. \frac{br}{2V_a} + C_{p_{\delta_a}} \delta_a + C_{p_{\delta_r}} \delta_r \right] \right\} + \dot{d}_{\phi_1} \\ &= -a_{\phi_1} \dot{\phi} + a_{\phi_2} \delta_a + d_{\phi_2}\end{aligned}$$

Equations of Motion

$$\dot{p}_n = (\cos \theta \cos \psi)u + (\sin \phi \sin \theta \cos \psi - \cos \phi \sin \psi)v + (\cos \phi \sin \theta \cos \psi + \sin \phi \sin \psi)w$$

$$\dot{p}_e = (\cos \theta \sin \psi)u + (\sin \phi \sin \theta \sin \psi + \cos \phi \cos \psi)v + (\cos \phi \sin \theta \sin \psi - \sin \phi \cos \psi)w$$

$$\dot{h} = u \sin \theta - v \sin \phi \cos \theta - w \cos \phi \cos \theta$$

$$\dot{u} = rv - qw - g \sin \theta + \frac{\rho V_a^2 S}{2m} \left[C_X(\alpha) + C_{X_q}(\alpha) \frac{cq}{2V_a} + C_{X_{\delta_e}}(\alpha) \delta_e \right] + \frac{\rho S_{\text{prop}} C_{\text{prop}}}{2m} \left[(k_{\text{motor}} \delta_t)^2 - V_a^2 \right]$$

$$\dot{v} = pw - ru + g \cos \theta \sin \phi + \frac{\rho V_a^2 S}{2m} \left[C_{Y_0} + C_{Y_\beta} \beta + C_{Y_p} \frac{bp}{2V_a} + C_{Y_r} \frac{br}{2V_a} + C_{Y_{\delta_a}} \delta_a + C_{Y_{\delta_r}} \delta_r \right]$$

$$\dot{w} = qu - pv + g \cos \theta \cos \phi + \frac{\rho V_a^2 S}{2m} \left[C_Z(\alpha) + C_{Z_q}(\alpha) \frac{cq}{2V_a} + C_{Z_{\delta_e}}(\alpha) \delta_e \right]$$

$$\dot{\phi} = p + q \sin \phi \tan \theta + r \cos \phi \tan \theta$$

$$\dot{\theta} = q \cos \phi - r \sin \phi$$

$$\dot{\psi} = q \sin \phi \sec \theta + r \cos \phi \sec \theta$$

$$\dot{p} = \Gamma_1 pq - \Gamma_2 qr + \frac{1}{2} \rho V_a^2 S b \left[C_{p_0} + C_{p_\beta} \beta + C_{p_p} \frac{bp}{2V_a} + C_{p_r} \frac{br}{2V_a} + C_{p_{\delta_a}} \delta_a + C_{p_{\delta_r}} \delta_r \right]$$

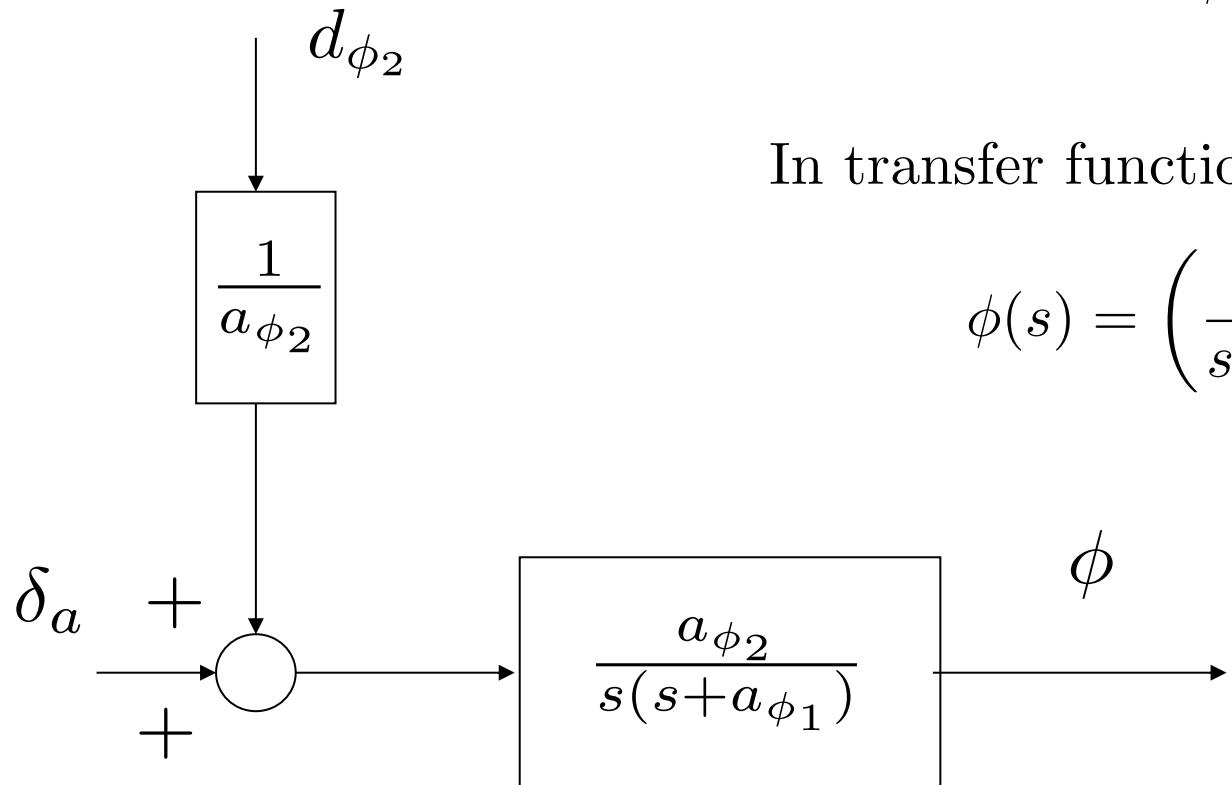
$$\dot{q} = \Gamma_5 pr - \Gamma_6 (p^2 - r^2) + \frac{\rho V_a^2 S c}{2J_y} \left[C_{m_0} + C_{m_\alpha} \alpha + C_{m_q} \frac{cq}{2V_a} + C_{m_{\delta_e}} \delta_e \right]$$

$$\dot{r} = \Gamma_7 pq - \Gamma_1 qr + \frac{1}{2} \rho V_a^2 S b \left[C_{r_0} + C_{r_\beta} \beta + C_{r_p} \frac{bp}{2V_a} + C_{r_r} \frac{br}{2V_a} + C_{r_{\delta_a}} \delta_a + C_{r_{\delta_r}} \delta_r \right]$$

Roll Dynamics

Linearization leads to second-order ordinary differential equation

$$\ddot{\phi} = -a_{\phi_1} \dot{\phi} + a_{\phi_2} \delta_a + d_{\phi_2}$$



In transfer function form we have

$$\phi(s) = \left(\frac{a_{\phi_2}}{s(s+a_{\phi_1})} \right) \left(\delta_a(s) + \frac{1}{a_{\phi_2}} d_{\phi_2}(s) \right)$$

Lateral Transfer Functions - Course

To derive transfer function from roll ϕ to course χ ,
start with no-wind coordinated turn condition

$$\dot{\chi} = \frac{g}{V_g} \tan \phi$$

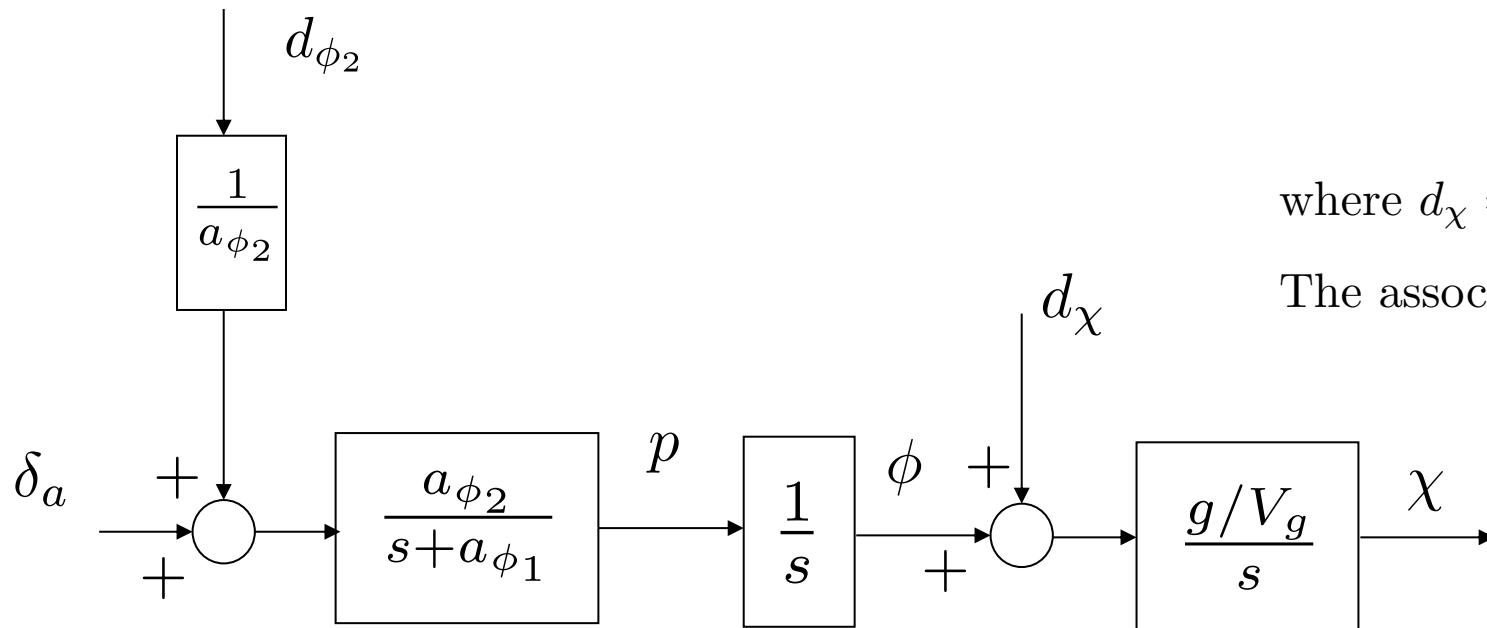
Add and subtract $\frac{g}{V_g}\phi$ to get

$$\begin{aligned}\dot{\chi} &= \frac{g}{V_g}\phi + \frac{g}{V_g}(\tan \phi - \phi) \\ &= \frac{g}{V_g}\phi + \frac{g}{V_g}d_\chi\end{aligned}$$

where $d_\chi = \tan \phi - \phi$ is an input disturbance

The associated transfer function is

$$\chi(s) = \frac{g/V_g}{s} (\phi(s) + d_\chi(s))$$



Lateral Transfer Functions - Sideslip

No-wind conditions: $v = V_a \sin \beta$

Constant airspeed: $\dot{v} = (V_a \cos \beta) \dot{\beta}$

Substituting for \dot{v} from the dynamics gives

$$(V_a \cos \beta) \dot{\beta} = pw - ru + g \cos \theta \sin \phi + \frac{\rho V_a^2 S}{2m} \left[C_{Y_0} + C_{Y_\beta} \beta + C_{Y_p} \frac{bp}{2V_a} + C_{Y_r} \frac{br}{2V_a} + C_{Y_{\delta_a}} \delta_a + C_{Y_{\delta_r}} \delta_r \right]$$

Solving for $\dot{\beta}$ and grouping terms gives

$$\dot{\beta} = -a_{\beta_1} \beta + a_{\beta_2} \delta_r + d_\beta$$

where

$$a_{\beta_1} = -\frac{\rho V_a S}{2m \cos \beta} C_{Y_\beta} \quad a_{\beta_2} = \frac{\rho V_a S}{2m \cos \beta} C_{Y_{\delta_r}}$$

$$d_\beta = \frac{1}{V_a \cos \beta} (pw - ru + g \cos \theta \sin \phi) + \frac{\rho V_a S}{2m \cos \beta} \left[C_{Y_0} + C_{Y_p} \frac{bp}{2V_a} + C_{Y_r} \frac{br}{2V_a} + C_{Y_{\delta_a}} \delta_a \right]$$

Equations of Motion

$$\dot{p}_n = (\cos \theta \cos \psi)u + (\sin \phi \sin \theta \cos \psi - \cos \phi \sin \psi)v + (\cos \phi \sin \theta \cos \psi + \sin \phi \sin \psi)w$$

$$\dot{p}_e = (\cos \theta \sin \psi)u + (\sin \phi \sin \theta \sin \psi + \cos \phi \cos \psi)v + (\cos \phi \sin \theta \sin \psi - \sin \phi \cos \psi)w$$

$$\dot{h} = u \sin \theta - v \sin \phi \cos \theta - w \cos \phi \cos \theta$$

$$\dot{u} = rv - qw - g \sin \theta + \frac{\rho V_a^2 S}{2m} \left[C_X(\alpha) + C_{X_q}(\alpha) \frac{cq}{2V_a} + C_{X_{\delta_e}}(\alpha) \delta_e \right] + \frac{\rho S_{\text{prop}} C_{\text{prop}}}{2m} \left[(k_{\text{motor}} \delta_t)^2 - V_a^2 \right]$$

$$\dot{v} = pw - ru + g \cos \theta \sin \phi + \frac{\rho V_a^2 S}{2m} \left[C_{Y_0} + C_{Y_\beta} \beta + C_{Y_p} \frac{bp}{2V_a} + C_{Y_r} \frac{br}{2V_a} + C_{Y_{\delta_a}} \delta_a + C_{Y_{\delta_r}} \delta_r \right]$$

$$\dot{w} = qu - pv + g \cos \theta \cos \phi + \frac{\rho V_a^2 S}{2m} \left[C_Z(\alpha) + C_{Z_q}(\alpha) \frac{cq}{2V_a} + C_{Z_{\delta_e}}(\alpha) \delta_e \right]$$

$$\dot{\phi} = p + q \sin \phi \tan \theta + r \cos \phi \tan \theta$$

$$\dot{\theta} = q \cos \phi - r \sin \phi$$

$$\dot{\psi} = q \sin \phi \sec \theta + r \cos \phi \sec \theta$$

$$\dot{p} = \Gamma_1 pq - \Gamma_2 qr + \frac{1}{2} \rho V_a^2 S b \left[C_{p_0} + C_{p_\beta} \beta + C_{p_p} \frac{bp}{2V_a} + C_{p_r} \frac{br}{2V_a} + C_{p_{\delta_a}} \delta_a + C_{p_{\delta_r}} \delta_r \right]$$

$$\dot{q} = \Gamma_5 pr - \Gamma_6 (p^2 - r^2) + \frac{\rho V_a^2 S c}{2J_y} \left[C_{m_0} + C_{m_\alpha} \alpha + C_{m_q} \frac{cq}{2V_a} + C_{m_{\delta_e}} \delta_e \right]$$

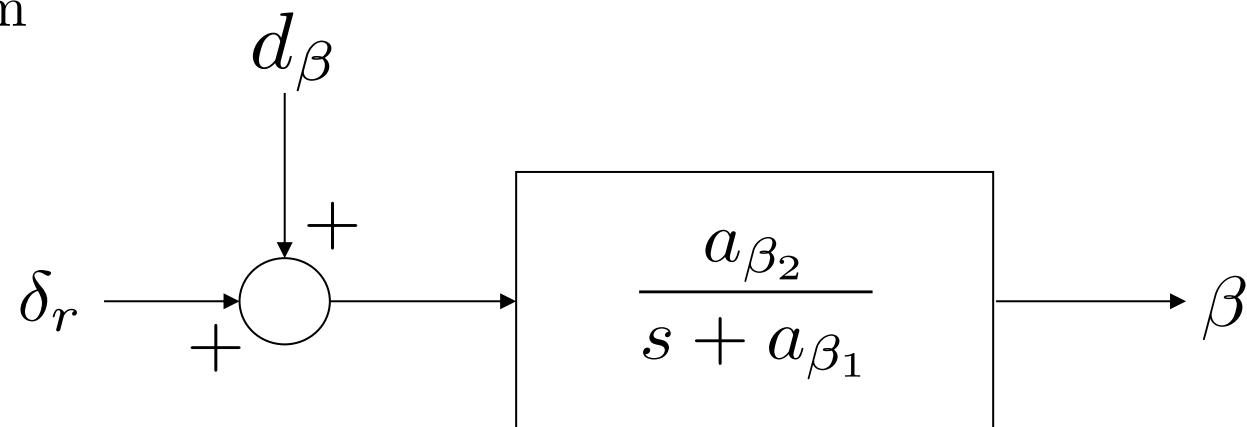
$$\dot{r} = \Gamma_7 pq - \Gamma_1 qr + \frac{1}{2} \rho V_a^2 S b \left[C_{r_0} + C_{r_\beta} \beta + C_{r_p} \frac{bp}{2V_a} + C_{r_r} \frac{br}{2V_a} + C_{r_{\delta_a}} \delta_a + C_{r_{\delta_r}} \delta_r \right]$$

Sideslip Dynamics

The differential equation

$$\dot{\beta} = -a_{\beta_1}\beta + a_{\beta_2}\delta_r + d_\beta$$

leads to the block diagram



$$a_{\beta_1} = -\frac{\rho V_a S}{2m} C_{Y_\beta}$$

$$a_{\beta_2} = \frac{\rho V_a S}{2m} C_{Y_{\delta_r}}$$

$$d_\beta = \frac{1}{V_a} (pw - ru + g \cos \theta \sin \phi) + \frac{\rho V_a S}{2m} \left[C_{Y_0} + C_{Y_p} \frac{bp}{2V_a} + C_{Y_r} \frac{br}{2V_a} + C_{Y_{\delta_a}} \delta_a \right]$$

Longitudinal Transfer Functions - Pitch

To derive the transfer function from elevator to pitch angle

$$\begin{aligned}\dot{\theta} &= q \cos \phi - r \sin \phi \\ &= q + q(\cos \phi - 1) - r \sin \phi \\ &\triangleq q + d_{\theta_1}\end{aligned}$$

Differentiating and substituting \dot{q} from the dynamics gives

$$\begin{aligned}\ddot{\theta} &= \Gamma_6(r^2 - p^2) + \Gamma_5 pr + \frac{\rho V_a^2 c S}{2J_y} \left[C_{m_0} + C_{m_\alpha} \alpha + C_{m_q} \frac{cq}{2V_a} + C_{m_{\delta_e}} \delta_e \right] + \dot{d}_{\theta_1} \\ &= \Gamma_6(r^2 - p^2) + \Gamma_5 pr + \frac{\rho V_a^2 c S}{2J_y} \left[C_{m_0} + C_{m_\alpha} (\theta - \gamma) + C_{m_q} \frac{c}{2V_a} (\dot{\theta} - d_{\theta_1}) + C_{m_{\delta_e}} \delta_e \right] + \dot{d}_{\theta_1} \\ &= \left(\frac{\rho V_a^2 c S}{2J_y} C_{m_q} \frac{c}{2V_a} \right) \dot{\theta} + \left(\frac{\rho V_a^2 c S}{2J_y} C_{m_\alpha} \right) \theta + \left(\frac{\rho V_a^2 c S}{2J_y} C_{m_{\delta_e}} \right) \delta_e + \left\{ \Gamma_6(r^2 - p^2) \right. \\ &\quad \left. + \Gamma_5 pr + \frac{\rho V_a^2 c S}{J_y} \left[C_{m_0} - C_{m_\alpha} \gamma - C_{m_q} \frac{c}{2V_a} d_{\theta_1} \right] + \dot{d}_{\theta_1} \right\} \\ &= -a_{\theta_1} \dot{\theta} - a_{\theta_2} \theta + a_{\theta_3} \delta_e + d_{\theta_2}\end{aligned}$$

Equations of Motion

$$\dot{p}_n = (\cos \theta \cos \psi)u + (\sin \phi \sin \theta \cos \psi - \cos \phi \sin \psi)v + (\cos \phi \sin \theta \cos \psi + \sin \phi \sin \psi)w$$

$$\dot{p}_e = (\cos \theta \sin \psi)u + (\sin \phi \sin \theta \sin \psi + \cos \phi \cos \psi)v + (\cos \phi \sin \theta \sin \psi - \sin \phi \cos \psi)w$$

$$\dot{h} = u \sin \theta - v \sin \phi \cos \theta - w \cos \phi \cos \theta$$

$$\dot{u} = rv - qw - g \sin \theta + \frac{\rho V_a^2 S}{2m} \left[C_X(\alpha) + C_{X_q}(\alpha) \frac{cq}{2V_a} + C_{X_{\delta_e}}(\alpha) \delta_e \right] + \frac{\rho S_{\text{prop}} C_{\text{prop}}}{2m} \left[(k_{\text{motor}} \delta_t)^2 - V_a^2 \right]$$

$$\dot{v} = pw - ru + g \cos \theta \sin \phi + \frac{\rho V_a^2 S}{2m} \left[C_{Y_0} + C_{Y_\beta} \beta + C_{Y_p} \frac{bp}{2V_a} + C_{Y_r} \frac{br}{2V_a} + C_{Y_{\delta_a}} \delta_a + C_{Y_{\delta_r}} \delta_r \right]$$

$$\dot{w} = qu - pv + g \cos \theta \cos \phi + \frac{\rho V_a^2 S}{2m} \left[C_Z(\alpha) + C_{Z_q}(\alpha) \frac{cq}{2V_a} + C_{Z_{\delta_e}}(\alpha) \delta_e \right]$$

$$\dot{\phi} = p + q \sin \phi \tan \theta + r \cos \phi \tan \theta$$

$$\dot{\theta} = q \cos \phi - r \sin \phi$$

$$\dot{\psi} = q \sin \phi \sec \theta + r \cos \phi \sec \theta$$

$$\dot{p} = \Gamma_1 pq - \Gamma_2 qr + \frac{1}{2} \rho V_a^2 S b \left[C_{p_0} + C_{p_\beta} \beta + C_{p_p} \frac{bp}{2V_a} + C_{p_r} \frac{br}{2V_a} + C_{p_{\delta_a}} \delta_a + C_{p_{\delta_r}} \delta_r \right]$$

$$\dot{q} = \Gamma_5 pr - \Gamma_6 (p^2 - r^2) + \frac{\rho V_a^2 S c}{2J_y} \left[C_{m_0} + C_{m_\alpha} \alpha + C_{m_q} \frac{cq}{2V_a} + C_{m_{\delta_e}} \delta_e \right]$$

$$\dot{r} = \Gamma_7 pq - \Gamma_1 qr + \frac{1}{2} \rho V_a^2 S b \left[C_{r_0} + C_{r_\beta} \beta + C_{r_p} \frac{bp}{2V_a} + C_{r_r} \frac{br}{2V_a} + C_{r_{\delta_a}} \delta_a + C_{r_{\delta_r}} \delta_r \right]$$

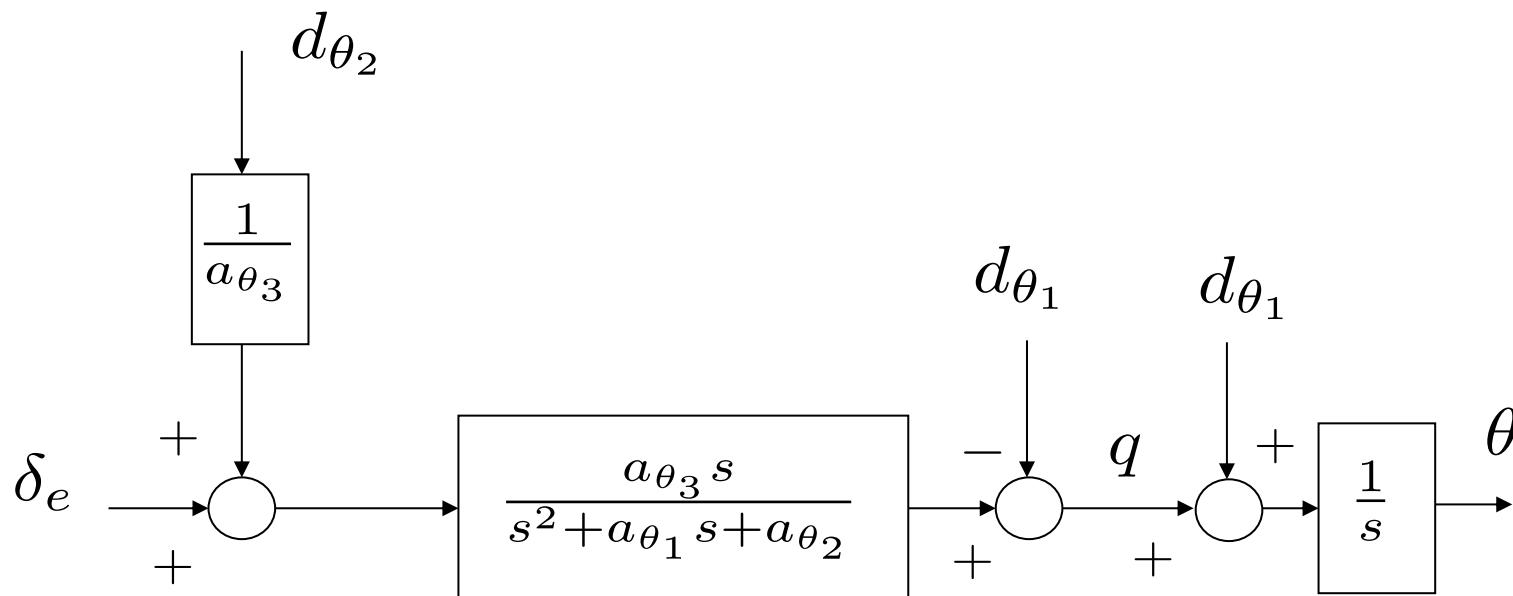
Longitudinal Transfer Functions - Pitch

The differential equation model is

$$\ddot{\theta} = -a_{\theta_1}\dot{\theta} - a_{\theta_2}\theta + a_{\theta_3}\delta_e + d_{\theta_2}$$

with associated transfer function

$$\theta(s) = \left(\frac{a_{\theta_3}}{s^2 + a_{\theta_1}s + a_{\theta_2}} \right) \left(\delta_e(s) + \frac{1}{a_{\theta_3}}d_{\theta_2}(s) \right)$$



Longitudinal TF - Altitude from Pitch

To derive the transfer function from pitch to altitude
(assuming constant airspeed)

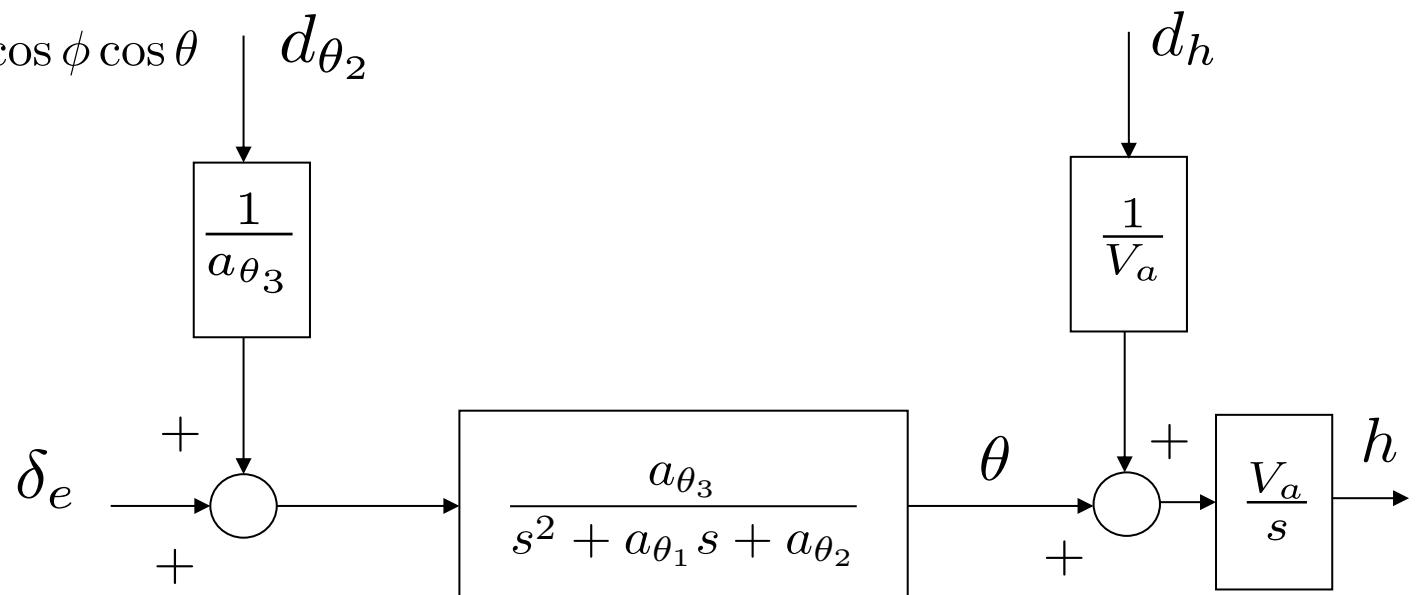
$$\begin{aligned}\dot{h} &= u \sin \theta - v \sin \phi \cos \theta - w \cos \phi \cos \theta \\ &= V_a \theta + (u \sin \theta - V_a \theta) - v \sin \phi \cos \theta - w \cos \phi \cos \theta \\ &= V_a \theta + d_h\end{aligned}$$

where

$$d_h \triangleq (u \sin \theta - V_a \theta) - v \sin \phi \cos \theta - w \cos \phi \cos \theta$$

The associated transfer function is

$$h(s) = \frac{V_a}{s} \left(\theta + \frac{1}{V_a} d_h \right)$$



Equations of Motion

$$\dot{p}_n = (\cos \theta \cos \psi)u + (\sin \phi \sin \theta \cos \psi - \cos \phi \sin \psi)v + (\cos \phi \sin \theta \cos \psi + \sin \phi \sin \psi)w$$

$$\dot{p}_e = (\cos \theta \sin \psi)u + (\sin \phi \sin \theta \sin \psi + \cos \phi \cos \psi)v + (\cos \phi \sin \theta \sin \psi - \sin \phi \cos \psi)w$$

$$\dot{h} = u \sin \theta - v \sin \phi \cos \theta - w \cos \phi \cos \theta$$

$$\dot{u} = rv - qw - g \sin \theta + \frac{\rho V_a^2 S}{2m} \left[C_X(\alpha) + C_{X_q}(\alpha) \frac{cq}{2V_a} + C_{X_{\delta_e}}(\alpha) \delta_e \right] + \frac{\rho S_{\text{prop}} C_{\text{prop}}}{2m} \left[(k_{\text{motor}} \delta_t)^2 - V_a^2 \right]$$

$$\dot{v} = pw - ru + g \cos \theta \sin \phi + \frac{\rho V_a^2 S}{2m} \left[C_{Y_0} + C_{Y_\beta} \beta + C_{Y_p} \frac{bp}{2V_a} + C_{Y_r} \frac{br}{2V_a} + C_{Y_{\delta_a}} \delta_a + C_{Y_{\delta_r}} \delta_r \right]$$

$$\dot{w} = qu - pv + g \cos \theta \cos \phi + \frac{\rho V_a^2 S}{2m} \left[C_Z(\alpha) + C_{Z_q}(\alpha) \frac{cq}{2V_a} + C_{Z_{\delta_e}}(\alpha) \delta_e \right]$$

$$\dot{\phi} = p + q \sin \phi \tan \theta + r \cos \phi \tan \theta$$

$$\dot{\theta} = q \cos \phi - r \sin \phi$$

$$\dot{\psi} = q \sin \phi \sec \theta + r \cos \phi \sec \theta$$

$$\dot{p} = \Gamma_1 pq - \Gamma_2 qr + \frac{1}{2} \rho V_a^2 S b \left[C_{p_0} + C_{p_\beta} \beta + C_{p_p} \frac{bp}{2V_a} + C_{p_r} \frac{br}{2V_a} + C_{p_{\delta_a}} \delta_a + C_{p_{\delta_r}} \delta_r \right]$$

$$\dot{q} = \Gamma_5 pr - \Gamma_6 (p^2 - r^2) + \frac{\rho V_a^2 S c}{2J_y} \left[C_{m_0} + C_{m_\alpha} \alpha + C_{m_q} \frac{cq}{2V_a} + C_{m_{\delta_e}} \delta_e \right]$$

$$\dot{r} = \Gamma_7 pq - \Gamma_1 qr + \frac{1}{2} \rho V_a^2 S b \left[C_{r_0} + C_{r_\beta} \beta + C_{r_p} \frac{bp}{2V_a} + C_{r_r} \frac{br}{2V_a} + C_{r_{\delta_a}} \delta_a + C_{r_{\delta_r}} \delta_r \right]$$

Longitudinal Transfer Functions - Airspeed

Both pitch angle and throttle strongly affect airspeed. We wish to derive the relationship.

If there is no wind, then

$$V_a = \sqrt{u^2 + v^2 + w^2}$$

Differentiating gives

$$\dot{V}_a = \dot{u} \cos \alpha \cos \beta + \dot{v} \sin \beta + \dot{w} \sin \alpha \cos \beta = \dot{u} \cos \alpha + \dot{w} \sin \alpha + d_{V_1}$$

where

$$d_{V_1} = -\dot{u}(1 - \cos \beta) \cos \alpha - \dot{w}(1 - \cos \beta) \sin \alpha + \dot{v} \sin \beta$$

Substituting from the dynamics gives

$$\begin{aligned}\dot{V}_a &= rV_a \cos \alpha \sin \beta - pV_a \sin \alpha \sin \beta - g \cos \alpha \sin \theta + g \sin \alpha \cos \theta \cos \phi \\ &\quad + \frac{\rho V_a^2 S}{2m} \left[-C_D(\alpha) - C_{D_\alpha} \alpha - C_{D_q} \frac{cq}{2V_a} - C_{D_{\delta_e}} \delta_e \right] + \frac{1}{m} T_p(\delta_t, V_a) + d_{V_1} \\ &= (rV_a \cos \alpha - pV_a \sin \alpha) \sin \beta - g \sin(\theta - \alpha) - g \sin \alpha \cos \theta (1 - \cos \phi) \\ &\quad + \frac{\rho V_a^2 S}{2m} \left[-C_{D_0} - C_{D_\alpha} \alpha - C_{D_q} \frac{cq}{2V_a} - C_{D_{\delta_e}} \delta_e \right] + \frac{1}{m} T_p(\delta_t, V_a) + d_{V_1} \\ &= -g \sin \gamma + \frac{\rho V_a^2 S}{2m} \left[-C_{D_0} - C_{D_\alpha} \alpha - C_{D_q} \frac{cq}{2V_a} - C_{D_{\delta_e}} \delta_e \right] + \frac{1}{m} T_p(\delta_t, V_a) + d_{V_2}\end{aligned}$$

Longitudinal Transfer Functions - Airspeed

$$\dot{V}_a = -g \sin \gamma + \frac{\rho V_a^2 S}{2m} \left[-C_{D_0} - C_{D_\alpha} \alpha - C_{D_q} \frac{cq}{2V_a} - C_{D_{\delta_e}} \delta_e \right] + \frac{1}{m} T_p(\delta_t, V_a) + d_{V_2}$$

Since this equation is nonlinear in V_a , we will linearize about trim: α^* , V_a^* , θ^* , δ_e^* , δ_t^* , and let

$$\bar{V}_a = V_a - V_a^*, \quad \bar{\theta} = \theta - \theta^*, \quad \bar{\delta}_e = \delta_e - \delta_e^*, \quad \bar{\delta}_t = \delta_e - \delta_t^*$$

to get

$$\begin{aligned} \dot{\bar{V}}_a &= -g \cos(\theta^* - \alpha^*) \bar{\theta} + \left\{ \frac{\rho V_a^* S}{m} \left[-C_{D_0} - C_{D_\alpha} \alpha^* - C_{D_{\delta_e}} \delta_e^* \right] + \frac{1}{m} \frac{\partial T_p}{\partial V_a}(\delta_t^*, V_a^*) \right\} \bar{V}_a + \frac{1}{m} \frac{\partial T_p}{\partial \delta_t}(\delta_t^*, V_a^*) \bar{\delta}_t + d_V \\ &= -a_{V_1} \bar{V}_a + a_{V_2} \bar{\delta}_t - a_{V_3} \bar{\theta} + d_V \end{aligned}$$

where

$$\begin{aligned} a_{V_1} &= \frac{\rho V_a^* S}{m} \left[C_{D_0} + C_{D_\alpha} \alpha^* + C_{D_{\delta_e}} \delta_e^* \right] - \frac{1}{m} \frac{\partial T_p}{\partial V_a}(\delta_t^*, V_a^*) \\ a_{V_2} &= \frac{1}{m} \frac{\partial T_p}{\partial \delta_t}(\delta_t^*, V_a^*) \\ a_{V_3} &= g \cos(\theta^* - \alpha^*) \end{aligned}$$

Longitudinal Transfer Functions - Airspeed

The associated transfer function is

$$\bar{V}_a(s) = \frac{1}{s + a_{V_1}} (a_{V_2} \bar{\delta}_t(s) - a_{V_3} \bar{\theta}(s) + d_V(s))$$

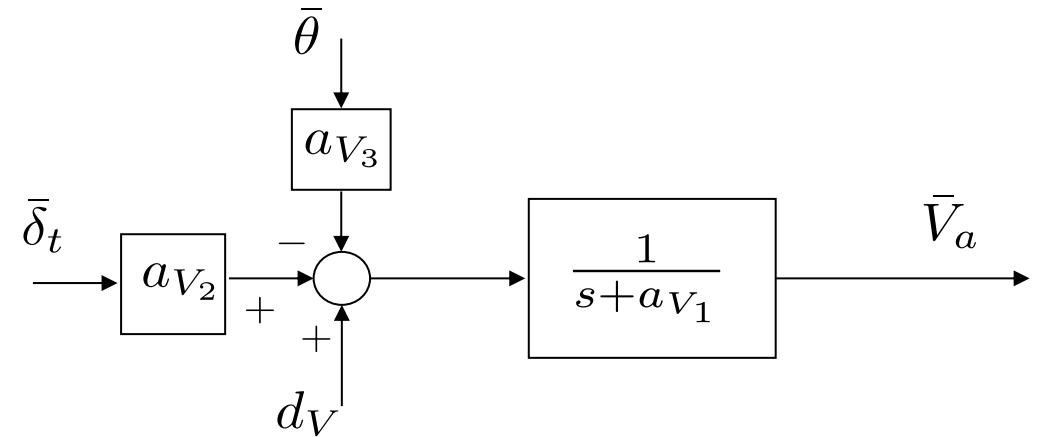
For throttle we have

$$\bar{V}_a(s) = \frac{a_{V_2}}{s + a_{V_1}} (\bar{\delta}_t(s) + d_V(s))$$

Key inputs are pitch angle
and throttle command

and for pitch angle we have

$$\bar{V}_a(s) = \frac{-a_{V_3}}{s + a_{V_1}} (\bar{\theta}(s) + d_V(s))$$



Linear State-space Models

Nonlinear state equations: $\dot{x} = f(x, u)$

Trim condition: $\dot{x}^* = f(x^*, u^*)$

Deviation from trim: $\bar{x} = x - x^*$

Rewriting the state equation in terms of deviation from trim gives

$$\begin{aligned}\dot{\bar{x}} &= \dot{x} - \dot{x}^* \\ &= f(x, u) - f(x^*, u^*) \\ &= f(x + x^* - x^*, u + u^* - u^*) - f(x^*, u^*) \\ &= f(x^* + \bar{x}, u^* + \bar{u}) - f(x^*, u^*)\end{aligned}$$

Using a Taylor series expansion around trim to linearize gives

$$\begin{aligned}\dot{\bar{x}} &= f(x^*, u^*) + \frac{\partial f(x^*, u^*)}{\partial x} \bar{x} + \frac{\partial f(x^*, u^*)}{\partial u} \bar{u} + H.O.T - f(x^*, u^*) \\ &\approx \frac{\partial f(x^*, u^*)}{\partial x} \bar{x} + \frac{\partial f(x^*, u^*)}{\partial u} \bar{u} \\ &\stackrel{\triangle}{=} A\bar{x} + B\bar{u}\end{aligned}$$

Lateral State-space Equations

The lateral states and inputs are defined as

$$\dot{x}_{\text{lat}} \triangleq (v, p, r, \phi, \psi)^\top \quad u_{\text{lat}} \triangleq (\delta_a, \delta_r)^\top$$

The nonlinear equations of motion are

$$\begin{aligned}\dot{v} &= pw - ru + g \cos \theta \sin \phi + \frac{\rho \sqrt{u^2 + v^2 + w^2} S}{2m} \frac{b}{2} [C_{Y_p} p + C_{Y_r} r] \\ &\quad + \frac{\rho(u^2 + v^2 + w^2)S}{2m} [C_{Y_0} + C_{Y_\beta} \tan^{-1} \left(\frac{v}{\sqrt{u^2 + w^2}} \right) + C_{Y_{\delta_a}} \delta_a + C_{Y_{\delta_r}} \delta_r] \\ \dot{p} &= \Gamma_1 pq - \Gamma_2 qr + \frac{\rho \sqrt{u^2 + v^2 + w^2} S}{2} \frac{b^2}{2} [C_{p_p} p + C_{p_r} r] + \Gamma_3 Q_p(\delta_t, V_a) \\ &\quad + \frac{1}{2} \rho(u^2 + v^2 + w^2) S b [C_{p_0} + C_{p_\beta} \tan^{-1} \left(\frac{v}{\sqrt{u^2 + w^2}} \right) + C_{p_{\delta_a}} \delta_a + C_{p_{\delta_r}} \delta_r] \\ \dot{r} &= \Gamma_7 pq - \Gamma_1 qr + \frac{\rho \sqrt{u^2 + v^2 + w^2} S}{2} \frac{b^2}{2} [C_{r_p} p + C_{r_r} r] + \Gamma_4 Q_p(\delta_t, V_a) \\ &\quad + \frac{1}{2} \rho(u^2 + v^2 + w^2) S b [C_{r_0} + C_{r_\beta} \tan^{-1} \left(\frac{v}{\sqrt{u^2 + w^2}} \right) + C_{r_{\delta_a}} \delta_a + C_{r_{\delta_r}} \delta_r] \\ \dot{\phi} &= p + q \sin \phi \tan \theta + r \cos \phi \tan \theta \\ \dot{\psi} &= q \sin \phi \sec \theta + r \cos \phi \sec \theta\end{aligned}$$

where we have used $\beta = \tan^{-1} \left(\frac{v}{\sqrt{u^2 + w^2}} \right)$ and $V_a = \sqrt{u^2 + v^2 + w^2}$

Jacobian Matrices

$$A_{\text{lat}} = \frac{\partial f_{\text{lat}}}{\partial x_{\text{lat}}} = \begin{pmatrix} \frac{\partial \dot{v}}{\partial v} & \frac{\partial \dot{v}}{\partial p} & \frac{\partial \dot{v}}{\partial r} & \frac{\partial \dot{v}}{\partial \phi} & \frac{\partial \dot{v}}{\partial \psi} \\ \frac{\partial \dot{p}}{\partial v} & \frac{\partial \dot{p}}{\partial p} & \frac{\partial \dot{p}}{\partial r} & \frac{\partial \dot{p}}{\partial \phi} & \frac{\partial \dot{p}}{\partial \psi} \\ \frac{\partial \dot{r}}{\partial v} & \frac{\partial \dot{r}}{\partial p} & \frac{\partial \dot{r}}{\partial r} & \frac{\partial \dot{r}}{\partial \phi} & \frac{\partial \dot{r}}{\partial \psi} \\ \frac{\partial \dot{\phi}}{\partial v} & \frac{\partial \dot{\phi}}{\partial p} & \frac{\partial \dot{\phi}}{\partial r} & \frac{\partial \dot{\phi}}{\partial \phi} & \frac{\partial \dot{\phi}}{\partial \psi} \\ \frac{\partial \dot{\psi}}{\partial v} & \frac{\partial \dot{\psi}}{\partial p} & \frac{\partial \dot{\psi}}{\partial r} & \frac{\partial \dot{\psi}}{\partial \phi} & \frac{\partial \dot{\psi}}{\partial \psi} \end{pmatrix}$$

$$B_{\text{lat}} = \frac{\partial f_{\text{lat}}}{\partial u_{\text{lat}}} = \begin{pmatrix} \frac{\partial \dot{v}}{\partial \delta_a} & \frac{\partial \dot{v}}{\partial \delta_r} \\ \frac{\partial \dot{p}}{\partial \delta_a} & \frac{\partial \dot{p}}{\partial \delta_r} \\ \frac{\partial \dot{r}}{\partial \delta_a} & \frac{\partial \dot{r}}{\partial \delta_r} \\ \frac{\partial \dot{\phi}}{\partial \delta_a} & \frac{\partial \dot{\phi}}{\partial \delta_r} \\ \frac{\partial \dot{\psi}}{\partial \delta_a} & \frac{\partial \dot{\psi}}{\partial \delta_r} \end{pmatrix}$$

Take partial derivatives and evaluate them at trim state and trim input

Lateral State-space Equations

$$\begin{pmatrix} \dot{\bar{v}} \\ \dot{\bar{p}} \\ \dot{\bar{r}} \\ \dot{\bar{\phi}} \\ \dot{\bar{\psi}} \end{pmatrix} = \begin{pmatrix} Y_v & Y_p & Y_r & A_{14} & 0 \\ L_v & L_p & L_r & 0 & 0 \\ N_v & N_p & N_r & 0 & 0 \\ 0 & 1 & A_{43} & A_{44} & 0 \\ 0 & 0 & A_{53} & A_{54} & 0 \end{pmatrix} \begin{pmatrix} \bar{v} \\ \bar{p} \\ \bar{r} \\ \bar{\phi} \\ \bar{\psi} \end{pmatrix} + \begin{pmatrix} Y_{\delta_a} & Y_{\delta_r} \\ L_{\delta_a} & L_{\delta_r} \\ N_{\delta_a} & N_{\delta_r} \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \bar{\delta}_a \\ \bar{\delta}_r \end{pmatrix}$$

Lateral Coefficient	Formula
Y_v	$\frac{\rho S b v^*}{4m V_a^*} [C_{Y_p} p^* + C_{Y_r} r^*] + \frac{\rho S v^*}{m} [C_{Y_0} + C_{Y_\beta} \beta^* + C_{Y_{\delta_a}} \delta_a^* + C_{Y_{\delta_r}} \delta_r^*] + \frac{\rho S C_{Y_\beta}}{2m} \sqrt{u^{*2} + w^{*2}}$
Y_p	$\omega^2 \frac{\rho V_a^{*2} S}{2m} C_{Y_{\delta_a}}$
Y_r	$\omega^2 \frac{\rho V_a^{*2} S}{2m} C_{Y_{\delta_r}}$
Y_{δ_a}	
Y_{δ_r}	
L_v	$\frac{\rho S b^2 v^*}{4V_a^*} [C_{p_p} p^* + C_{p_r} r^*] + \rho S b v^* [C_{p_0} + C_{p_\beta} \beta^* + C_{p_{\delta_a}} \delta_a^* + C_{p_{\delta_r}} \delta_r^*] + \frac{\rho S b C_{p_\beta}}{2} \sqrt{u^{*2} + w^{*2}}$
L_p	$\Gamma_1 q^* + \frac{\rho V_a^* S b^2}{4} C_{p_p}$
L_r	$-\Gamma_2 q^* + \frac{\rho V_a^* S b^2}{4} C_{p_r}$
L_{δ_a}	
L_{δ_r}	
N_v	$\frac{\rho S b^2 v^*}{4V_a^*} [C_{r_p} p^* + C_{r_r} r^*] + \rho S b v^* [C_{r_0} + C_{r_\beta} \beta^* + C_{r_{\delta_a}} \delta_a^* + C_{r_{\delta_r}} \delta_r^*] + \frac{\rho S b C_{r_\beta}}{2} \sqrt{u^{*2} + w^{*2}}$
N_p	$\Gamma_7 q^* + \frac{\rho V_a^* S b^2}{4} C_{r_p}$
N_r	$-\Gamma_1 q^* + \frac{\rho V_a^* S b^2}{4} C_{r_r}$
N_{δ_a}	
N_{δ_r}	

dimensional stability derivatives

Lateral State-space Equations

$$\begin{pmatrix} \dot{\bar{v}} \\ \dot{\bar{p}} \\ \dot{\bar{r}} \\ \dot{\bar{\phi}} \\ \dot{\bar{\psi}} \end{pmatrix} = \begin{pmatrix} Y_v & Y_p & Y_r & A_{14} & 0 \\ L_v & L_p & L_r & 0 & 0 \\ N_v & N_p & N_r & 0 & 0 \\ 0 & 1 & A_{43} & A_{44} & 0 \\ 0 & 0 & A_{53} & A_{54} & 0 \end{pmatrix} \begin{pmatrix} \bar{v} \\ \bar{p} \\ \bar{r} \\ \bar{\phi} \\ \bar{\psi} \end{pmatrix} + \begin{pmatrix} Y_{\delta_a} & Y_{\delta_r} \\ L_{\delta_a} & L_{\delta_r} \\ N_{\delta_a} & N_{\delta_r} \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \bar{\delta}_a \\ \bar{\delta}_r \end{pmatrix}$$

Lateral Coefficient	Formula
A_{14}	$g \cos \theta^* \cos \phi^*$
A_{43}	$\cos \phi^* \tan \theta^*$
A_{44}	$q^* \cos \phi^* \tan \theta^* - r^* \sin \phi^* \tan \theta^*$
A_{53}	$\cos \phi^* \sec \theta^*$
A_{54}	$p^* \cos \phi^* \sec \theta^* - r^* \sin \phi^* \sec \theta^*$

Alternative Form - Lateral

The lateral dynamics are often expressed using β instead of v . Recall that

$$v = V_a \sin \beta$$

Therefore

$$\bar{v} = V_a^* \cos \beta^* \bar{\beta}$$

Differentiating gives

$$\dot{\bar{\beta}} = \frac{1}{V_a^* \cos \beta^*} \dot{\bar{v}}$$

Utilizing side-slip angle
instead of lateral velocity

The associated state space equations are

$$\begin{pmatrix} \dot{\bar{\beta}} \\ \dot{\bar{p}} \\ \dot{\bar{r}} \\ \dot{\bar{\phi}} \\ \dot{\bar{\psi}} \end{pmatrix} = \begin{pmatrix} Y_v & \frac{Y_p}{V_a^* \cos \beta^*} & \frac{Y_r}{V_a^* \cos \beta^*} & \frac{g \cos \theta^* \cos \phi^*}{V_a^* \cos \beta^*} & 0 \\ L_v V_a^* \cos \beta^* & L_p & L_r & 0 & 0 \\ N_v V_a^* \cos \beta^* & N_p & N_r & 0 & 0 \\ 0 & 1 & \cos \phi^* \tan \theta^* & q^* \cos \phi^* \tan \theta^* & 0 \\ 0 & 0 & \cos \phi^* \sec \theta^* & p^* \cos \phi^* \sec \theta^* & 0 \\ & & & -r^* \sin \phi^* \sec \theta^* & 0 \end{pmatrix} \begin{pmatrix} \bar{\beta} \\ \bar{p} \\ \bar{r} \\ \bar{\phi} \\ \bar{\psi} \end{pmatrix} + \begin{pmatrix} \frac{Y_{\delta_a}}{V_a^* \cos \beta^*} & \frac{Y_{\delta_r}}{V_a^* \cos \beta^*} \\ L_{\delta_a} & L_{\delta_r} \\ N_{\delta_a} & N_{\delta_r} \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \bar{\delta}_a \\ \bar{\delta}_r \end{pmatrix}$$

Longitudinal State-space Equations

The longitudinal states and inputs are defined as

$$\dot{x}_{\text{lon}} \triangleq (u, w, q, \theta, h)^\top \quad u_{\text{lon}} \triangleq (\delta_e, \delta_t)^\top$$

The nonlinear equations of motion are

$$\begin{aligned} \dot{u} &= -qw - g \sin \theta + \frac{\rho(u^2 + w^2)S}{2m} \left[C_{X_0} + C_{X_\alpha} \tan^{-1} \left(\frac{w}{u} \right) + C_{X_{\delta_e}} \delta_e \right] + \frac{\rho\sqrt{u^2 + w^2}S}{4m} C_{X_q} cq \\ &\quad + \frac{\rho S_{\text{prop}}}{2m} C_{\text{prop}} \left[(k\delta_t)^2 - (u^2 + w^2) \right] \\ \dot{w} &= qu + g \cos \theta + \frac{\rho(u^2 + w^2)S}{2m} \left[C_{Z_0} + C_{Z_\alpha} \tan^{-1} \left(\frac{w}{u} \right) + C_{Z_{\delta_e}} \delta_e \right] + \frac{\rho\sqrt{u^2 + w^2}S}{4m} C_{Z_q} cq \\ \dot{q} &= \frac{1}{2J_y} \rho(u^2 + w^2)cS \left[C_{m_0} + C_{m_\alpha} \tan^{-1} \left(\frac{w}{u} \right) + C_{m_{\delta_e}} \delta_e \right] + \frac{1}{4J_y} \rho\sqrt{u^2 + w^2} S C_{m_q} c^2 q \\ \dot{\theta} &= q \\ \dot{h} &= u \sin \theta - w \cos \theta \end{aligned}$$

where we have used $\alpha = \tan^{-1} \left(\frac{w}{u} \right)$ and $V_a = \sqrt{u^2 + w^2}$

Jacobian Matrices

$$A_{\text{lon}} = \frac{\partial f_{\text{lon}}}{\partial x_{\text{lon}}} = \begin{pmatrix} \frac{\partial \dot{u}}{\partial u} & \frac{\partial \dot{u}}{\partial w} & \frac{\partial \dot{u}}{\partial q} & \frac{\partial \dot{u}}{\partial \theta} & \frac{\partial \dot{u}}{\partial h} \\ \frac{\partial \dot{w}}{\partial u} & \frac{\partial \dot{w}}{\partial w} & \frac{\partial \dot{w}}{\partial q} & \frac{\partial \dot{w}}{\partial \theta} & \frac{\partial \dot{w}}{\partial h} \\ \frac{\partial \dot{q}}{\partial u} & \frac{\partial \dot{q}}{\partial w} & \frac{\partial \dot{q}}{\partial q} & \frac{\partial \dot{q}}{\partial \theta} & \frac{\partial \dot{q}}{\partial h} \\ \frac{\partial \dot{\theta}}{\partial u} & \frac{\partial \dot{\theta}}{\partial w} & \frac{\partial \dot{\theta}}{\partial q} & \frac{\partial \dot{\theta}}{\partial \theta} & \frac{\partial \dot{\theta}}{\partial h} \\ \frac{\partial \dot{h}}{\partial u} & \frac{\partial \dot{h}}{\partial w} & \frac{\partial \dot{h}}{\partial q} & \frac{\partial \dot{h}}{\partial \theta} & \frac{\partial \dot{h}}{\partial h} \end{pmatrix}$$
$$B_{\text{lon}} = \frac{\partial f_{\text{lon}}}{\partial u_{\text{lon}}} = \begin{pmatrix} \frac{\partial \dot{u}}{\partial \delta_e} & \frac{\partial \dot{u}}{\partial \delta_t} \\ \frac{\partial \dot{w}}{\partial \delta_e} & \frac{\partial \dot{w}}{\partial \delta_t} \\ \frac{\partial \dot{q}}{\partial \delta_e} & \frac{\partial \dot{q}}{\partial \delta_t} \\ \frac{\partial \dot{\theta}}{\partial \delta_e} & \frac{\partial \dot{\theta}}{\partial \delta_t} \\ \frac{\partial \dot{h}}{\partial \delta_e} & \frac{\partial \dot{h}}{\partial \delta_t} \end{pmatrix}$$

Take partial derivatives and evaluate them at trim state and trim input

Longitudinal State-space Equations

$$\begin{pmatrix} \dot{\bar{u}} \\ \dot{\bar{w}} \\ \dot{\bar{q}} \\ \dot{\bar{\theta}} \\ \dot{\bar{h}} \end{pmatrix} = \begin{pmatrix} X_u & X_w & X_q & -g \cos \theta^* & 0 \\ Z_u & Z_w & Z_q & -g \sin \theta^* & 0 \\ M_u & M_w & M_q & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ \sin \theta^* & -\cos \theta^* & 0 & u^* \cos \theta^* + w^* \sin \theta^* & 0 \end{pmatrix} \begin{pmatrix} \bar{u} \\ \bar{w} \\ \bar{q} \\ \bar{\theta} \\ \bar{h} \end{pmatrix} + \begin{pmatrix} X_{\delta_e} & X_{\delta_t} \\ Z_{\delta_e} & 0 \\ M_{\delta_e} & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \bar{\delta}_e \\ \bar{\delta}_t \end{pmatrix}$$

Longitudinal	Formula
X_u	$\frac{u^* \rho S}{m} [C_{X_0} + C_{X_\alpha} \alpha^* + C_{X_{\delta_e}} \delta_e^*] - \frac{\rho S w^* C_{X_\alpha}}{2m} + \frac{\rho S c C_{X_q} u^* q^*}{4m V_a^*} - \frac{\rho S_{prop} C_{prop} u^*}{m}$
X_w	$-q^* + \frac{w^* \rho S}{m} [C_{X_0} + C_{X_\alpha} \alpha^* + C_{X_{\delta_e}} \delta_e^*] + \frac{\rho S c C_{X_q} w^* q^*}{4m V_a^*} + \frac{\rho S C_{X_\alpha} u^*}{2m} - \frac{\rho S_{prop} C_{prop} w^*}{m}$
X_q	$-w^* + \frac{\rho V_a^* S C_{X_q} c}{4m}$
X_{δ_e}	$\frac{\rho V_a^{*2} S C_{X_{\delta_e}}}{2m}$
X_{δ_t}	Z_u dimensional stability derivatives
Z_u	$\frac{z_\alpha w^*}{2m} + \frac{u^* \rho S C_{Z_q} c q^*}{4m V_a^*}$
Z_w	$\frac{w^* \rho S}{m} [C_{Z_0} + C_{Z_\alpha} \alpha^* + C_{Z_{\delta_e}} \delta_e^*] + \frac{\rho S C_{Z_\alpha} u^*}{2m} + \frac{\rho w^* S c C_{Z_q} q^*}{4m V_a^*}$
Z_q	$u^* + \frac{\rho V_a^* S C_{Z_q} c}{4m}$
Z_{δ_e}	$\frac{\rho V_a^{*2} S C_{Z_{\delta_e}}}{2m}$
M_u	$\frac{u^* \rho S c}{J_y} [C_{m_0} + C_{m_\alpha} \alpha^* + C_{m_{\delta_e}} \delta_e^*] - \frac{\rho S c C_{m_\alpha} w^*}{2J_y} + \frac{\rho S c^2 C_{m_q} q^* u^*}{4J_y V_a^*}$
M_w	$\frac{w^* \rho S c}{J_y} [C_{m_0} + C_{m_\alpha} \alpha^* + C_{m_{\delta_e}} \delta_e^*] + \frac{\rho S c C_{m_\alpha} u^*}{2J_y} + \frac{\rho S c^2 C_{m_q} q^* w^*}{4J_y V_a^*}$
M_q	$\frac{\rho V_a^* S c^2 C_{m_q}}{4J_y}$
M_{δ_e}	$\frac{\rho V_a^{*2} S c C_{m_{\delta_e}}}{2J_y}$

Alternative Form – Longitudinal

The longitudinal dynamics are often expressed using α instead of w . Recall that

$$w = V_a \sin \alpha \cos \beta.$$

At trim, when $\beta = 0$ we have

$$\bar{w} = V_a^* \cos \alpha^* \bar{\alpha}.$$

Utilizing angle of attack
instead of downward velocity

Differentiating gives

$$\dot{\bar{\alpha}} = \frac{1}{V_a^* \cos \alpha^*} \dot{\bar{w}}.$$

The associated state space equations are

$$\begin{pmatrix} \dot{\bar{u}} \\ \dot{\bar{\alpha}} \\ \dot{\bar{q}} \\ \dot{\bar{\theta}} \\ \dot{\bar{h}} \end{pmatrix} = \begin{pmatrix} X_u & X_w V_a^* \cos \alpha^* & X_q & -g \cos \theta^* & 0 \\ \frac{Z_u}{V_a^* \cos \alpha^*} & Z_w & \frac{Z_q}{V_a^* \cos \alpha^*} & \frac{-g \sin \theta^*}{V_a^* \cos \alpha^*} & 0 \\ M_u & M_w V_a^* \cos \alpha^* & M_q & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ \sin \theta^* & -V_a^* \cos \theta^* \cos \alpha^* & 0 & u^* \cos \theta^* + w^* \sin \theta^* & 0 \end{pmatrix} \begin{pmatrix} \bar{u} \\ \bar{\alpha} \\ \bar{q} \\ \bar{\theta} \\ \bar{h} \end{pmatrix} + \begin{pmatrix} X_{\delta_e} & X_{\delta_t} \\ \frac{Z_{\delta_e}}{V_a^* \cos \alpha^*} & 0 \\ M_{\delta_e} & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \bar{\delta}_e \\ \bar{\delta}_t \end{pmatrix}$$

Numerical Computation of State Space Equations

Let $w = f(x)$ where $f : \mathbb{R}^n \mapsto \mathbb{R}^m$, i.e.,

$$\begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{pmatrix} = \begin{pmatrix} f_1(x_1, x_2, \dots, x_n) \\ f_2(x_1, x_2, \dots, x_n) \\ \vdots \\ f_m(x_1, x_2, \dots, x_n) \end{pmatrix}.$$

The Jacobian of f with respect to x is defined as

$$\frac{\partial f}{\partial x} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & & & \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{pmatrix}.$$

Numerical Computation of State Space Equations

Let

$$\frac{\partial f}{\partial x} = \begin{pmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} & \cdots & \frac{\partial f}{\partial x_n} \end{pmatrix}, \quad \text{where} \quad \frac{\partial f}{\partial x_i} = \begin{pmatrix} \frac{\partial f_1}{\partial x_i} \\ \frac{\partial f_2}{\partial x_i} \\ \vdots \\ \frac{\partial f_n}{\partial x_i} \end{pmatrix}.$$

If ϵ is a small fixed number, then

$$\frac{\partial f}{\partial x_i}(x) \approx \frac{f(x + \epsilon e_i) - f(x)}{\epsilon},$$

which can be computed numerically, with two calls of $f(\cdot)$.

Note that computing $\frac{\partial f}{\partial x}(x)$ will require $n + 1$ functions calls to $f(\cdot)$.

Numerical Computation of State Space Equations

```
import numpy as np
def df_dx(x, z):
    # take partial of f(x, z) with respect to x
    eps = 0.001 # deviation
    A = np.zeros((m, n)) # Jacobian of f wrt x
    f_at_x = f(x, z)
    for i in range(0, n):
        x_eps = np.copy(x)
        x_eps[i][0] += eps # add eps to ith state
        f_at_x_eps = f(x_eps, z)
        df_dx_i = (f_at_x_eps - f_at_x) / eps
        A[:, i] = df_dx_i[:, 0]
    return A
```

Numerical Computation of State Space Equations

Define:

The Euler state: $x_e = (p_n, p_e, p_d, u, v, w, \phi, \theta, \psi, p, q, r)^\top \in \mathbb{R}^{12}$

The Quaternion state: $x_q = (p_n, p_e, p_d, u, v, w, e_0, e_x, e_y, e_z, p, q, r)^\top \in \mathbb{R}^{13}$

Let

$$\mathbf{p} = (p_n, p_e, p_d)^\top$$

$$\mathbf{v} = (u, v, w)^\top$$

$$\boldsymbol{\vartheta} = (\phi, \theta, \psi)^\top$$

$$\mathbf{q} = (e_0, e_1, e_2, e_3)^\top$$

$$\boldsymbol{\omega} = (p, q, r)^\top,$$

Then

$$x_e = (\mathbf{p}^\top, \mathbf{v}^\top, \boldsymbol{\vartheta}^\top, \boldsymbol{\omega}^\top)^\top$$

$$x_q = (\mathbf{p}^\top, \mathbf{v}^\top, \mathbf{q}^\top, \boldsymbol{\omega}^\top)^\top.$$

Numerical Computation of State Space Equations

Let $\vartheta = \Theta(\mathbf{q})$ convert quaternions to Euler angles,
and

$\mathbf{q} = Q(\vartheta)$ converts Euler angles to quaternions.

Define

$$T_e \begin{bmatrix} \mathbf{p} \\ \mathbf{v} \\ \mathbf{q} \\ \boldsymbol{\omega} \end{bmatrix} \triangleq \begin{pmatrix} \mathbf{p} \\ \mathbf{v} \\ \Theta(\mathbf{q}) \\ \boldsymbol{\omega} \end{pmatrix}$$
$$T_q \begin{bmatrix} \mathbf{p} \\ \mathbf{v} \\ \vartheta \\ \boldsymbol{\omega} \end{bmatrix} \triangleq \begin{pmatrix} \mathbf{p} \\ \mathbf{v} \\ Q(\vartheta) \\ \boldsymbol{\omega} \end{pmatrix},$$

where

$$\frac{\partial T_e}{\partial x_q}(x_q) = \begin{pmatrix} I_{3 \times 3} & 0_{3 \times 3} & 0_{3 \times 4} & 0_{3 \times 3} \\ 0_{3 \times 3} & I_{3 \times 3} & 0_{3 \times 4} & 0_{3 \times 3} \\ 0_{3 \times 3} & 0_{3 \times 3} & \frac{\partial \Theta}{\partial \mathbf{q}}(x_q) & 0_{3 \times 3} \\ 0_{3 \times 3} & 0_{3 \times 3} & 0_{3 \times 4} & I_{3 \times 3} \end{pmatrix}.$$

Numerical Computation of State Space Equations

The Python dynamics file returns:

$$\dot{x}_q = f_q(x_q, u).$$

In terms of Euler states we have:

$$\begin{aligned}\dot{x}_e &= \frac{d}{dt}T_e(x_q) = \frac{\partial T_e}{\partial x_q}(x_q)\dot{x}_q = \frac{\partial T_e}{\partial x_q}(x_q)f_q(x_q, u) \\ &= \frac{\partial T_e}{\partial x_q}(T_q(x_e))f_q(T_q(x_e), u).\end{aligned}$$

If $\tilde{x}_e = x_e - x_e^*$, $\tilde{u} = u - u^*$, the state space equations are

$$\dot{\tilde{x}}_e = A\tilde{x}_e + B\tilde{u},$$

where

$$\begin{aligned}A &= \frac{\partial}{\partial x_e} \left(\frac{\partial T_e}{\partial x_q}(T_q(x_e))f_q(T_q(x_e), u) \right) \\ B &= \frac{\partial}{\partial u} \left(\frac{\partial T_e}{\partial x_q}(T_q(x_e))f_q(T_q(x_e), u) \right).\end{aligned}$$

Longitudinal State Space Equations

The longitudinal states and inputs are

$$x_{lon} = \begin{pmatrix} u \\ w \\ q \\ \theta \\ h \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} p_n \\ p_e \\ p_d \\ u \\ v \\ w \\ \phi \\ \theta \\ \psi \\ p \\ q \\ r \end{pmatrix} \triangleq E_1 x_e$$

$$u_{lon} = \begin{pmatrix} \delta_e \\ \delta_t \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \delta_t \\ \delta_e \\ \delta_a \\ \delta_r \end{pmatrix} \triangleq E_2 u.$$

Therefore the longitudinal state space equations are

$$\dot{x}_{lon} = A_{lon} x_{lon} + B_{lon} u_{lon}$$

where

$$A_{lon} = E_1 A E_1^\top$$

$$B_{lon} = E_1 B E_2^\top.$$

Lateral State Space Equations

The lateral states and inputs are

$$x_{lat} = \begin{pmatrix} v \\ p \\ r \\ \phi \\ \psi \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} p_n \\ p_e \\ p_d \\ u \\ v \\ w \\ \phi \\ \theta \\ \psi \\ p \\ q \\ r \end{pmatrix} \triangleq E_3 x_e$$

$$u_{lat} = \begin{pmatrix} \delta_a \\ \delta_r \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \delta_t \\ \delta_e \\ \delta_a \\ \delta_r \end{pmatrix} \triangleq E_4 u,$$

Therefore the lateral state space equations are

$$\dot{x}_{lat} = A_{lat} x_{lat} + B_{lat} u_{lat}$$

where

$$A_{lat} = E_3 A E_3^\top$$

$$B_{lat} = E_3 B E_4^\top.$$

Reduced Order Modes

Longitudinal Modes

- Short-period Mode
 - Short-period mode is the fast mode seen in pitch rate q and pitch angle
- Phugoid Mode
 - Phugoid mode is slow mode seen in pitch angle and u
- Induced by impulse on elevator

Lateral Modes

- Roll Mode
 - First order fast mode between aileron and roll rate
- Spiral-divergence Mode
 - First order (really) slow mode between aileron and yaw/course
- Dutch-roll Mode
 - Second order mode: coupling between roll, yaw, and side slip. Like a duck wagging its tail
- Induced by doublet on aileron or rudder

Simulation Project

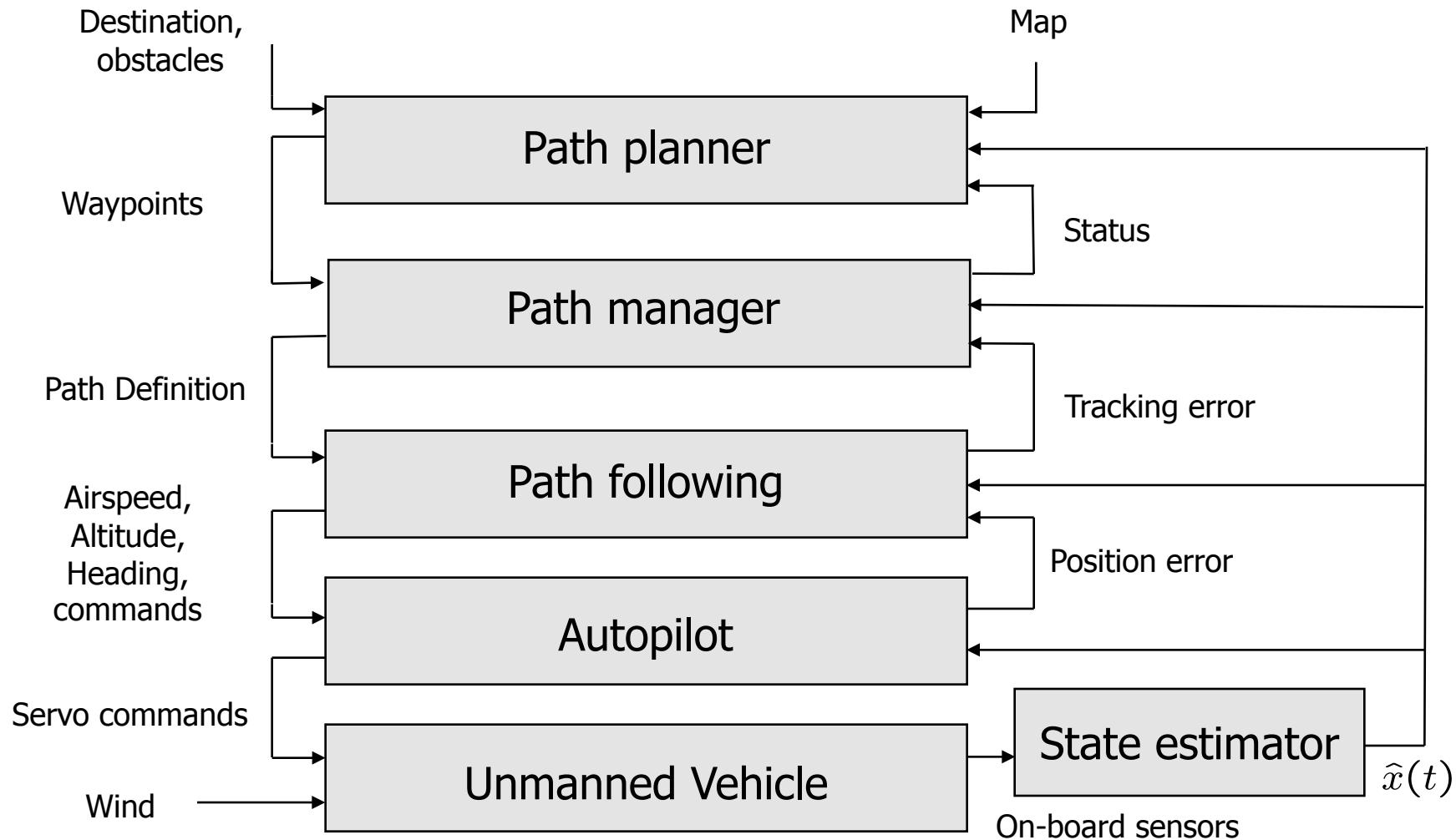
- Find trim states and inputs for straight level flight. For the aerosonde model use $V_a^* = 25$ m/s and $\gamma^* = 0$.
- Find the transfer functions derived in this chapter.
- Find the state space models derived in this chapter.
- Compute the eigenvalues of A_{lon} and A_{lat} and associate them with the phugoid mode, the short period mode, the roll mode, the dutch roll mode, and the spiral mode.



Chapter 6

Autopilot Design

Architecture

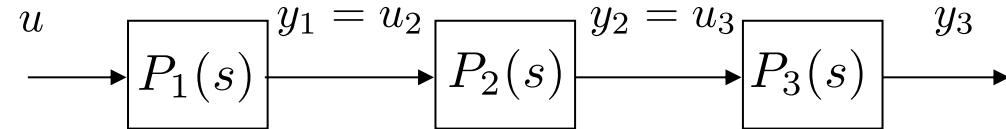


Outline

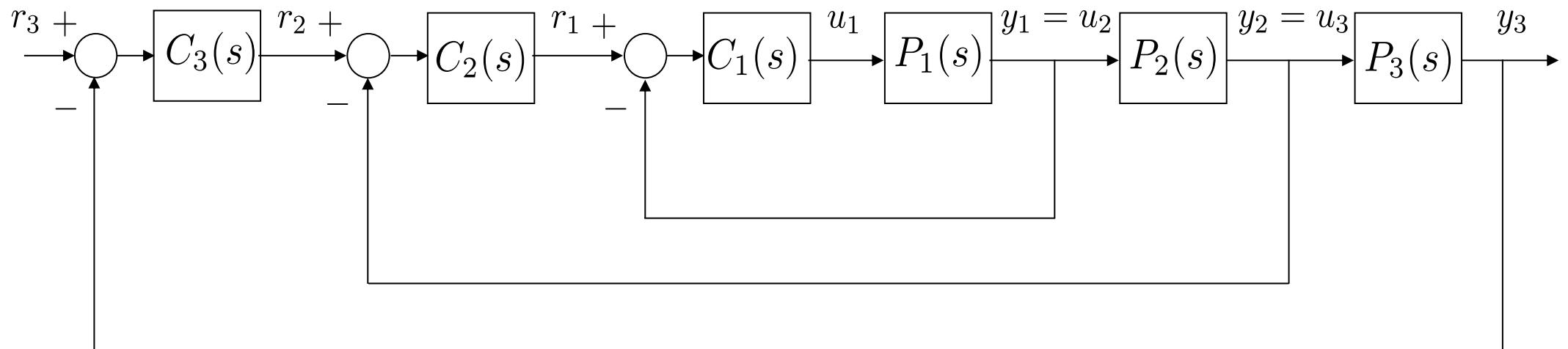
- Different Options for Autopilot Design
 - Successive Loop Closure
 - Total Energy Control
 - LQR Control

Successive Loop Closure

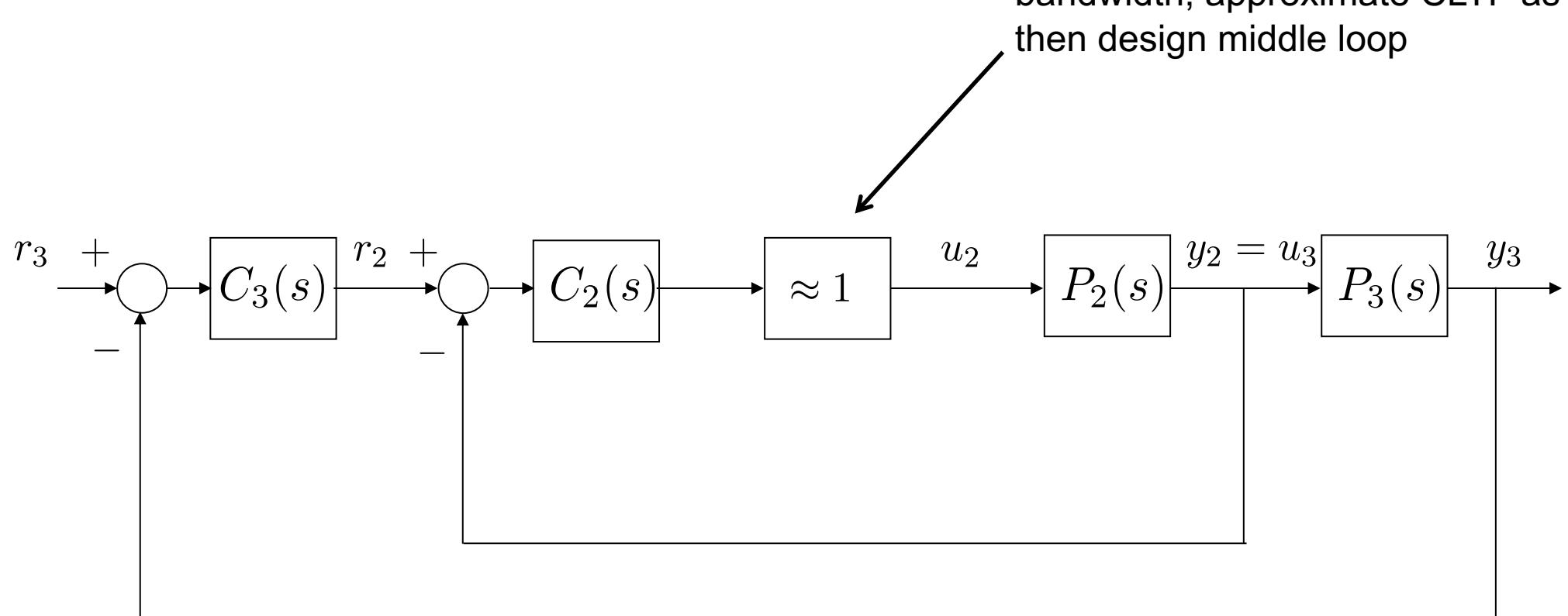
Open-loop system



Closed-loop system

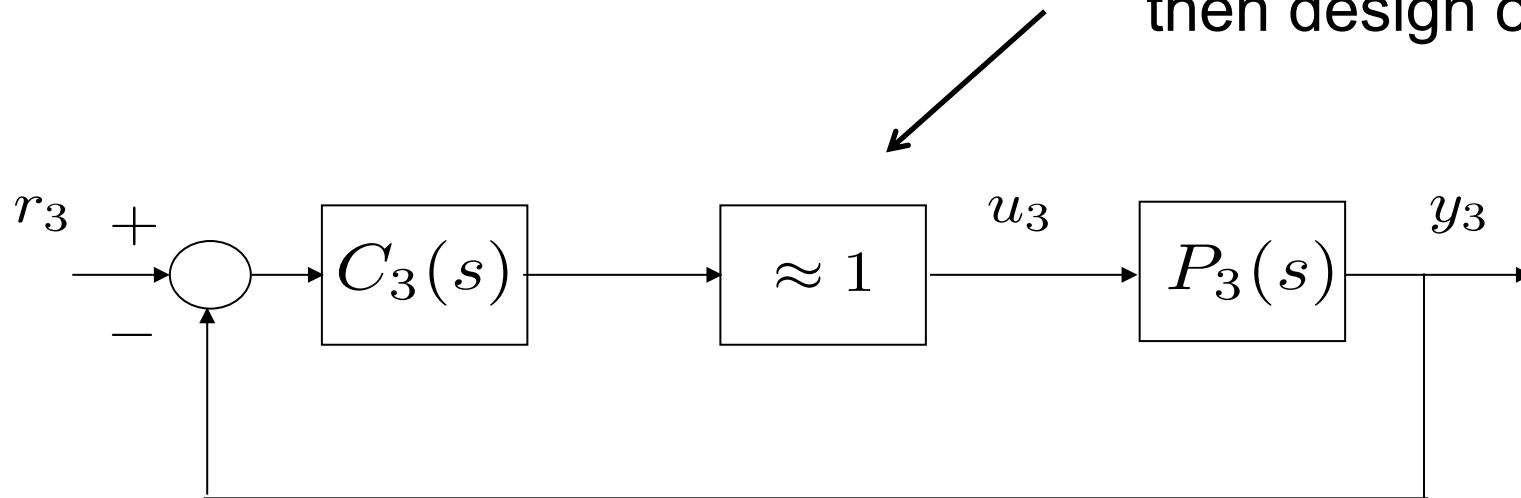


SLC: Inner Loop Closed



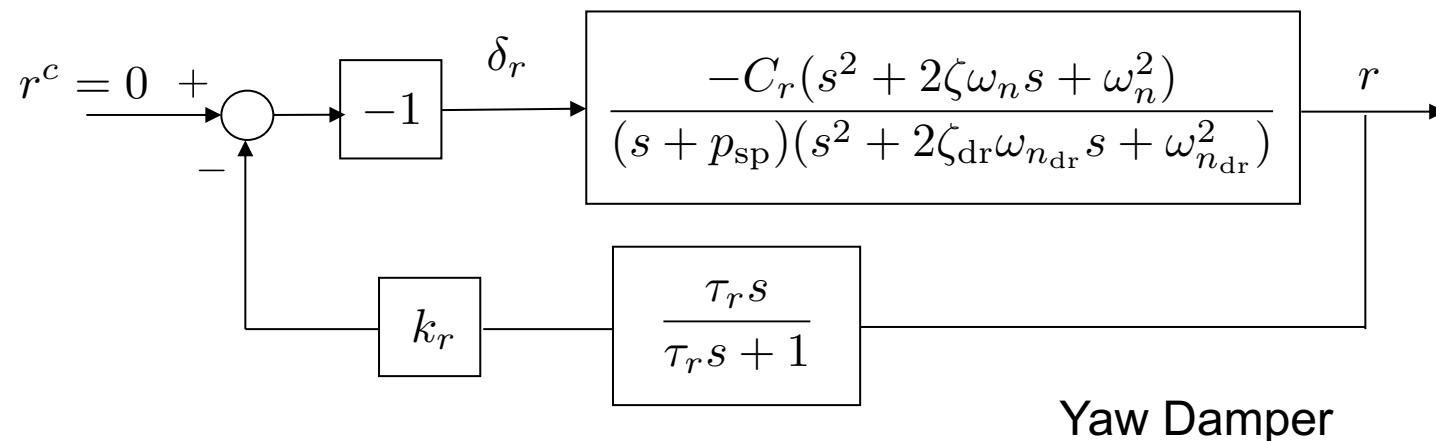
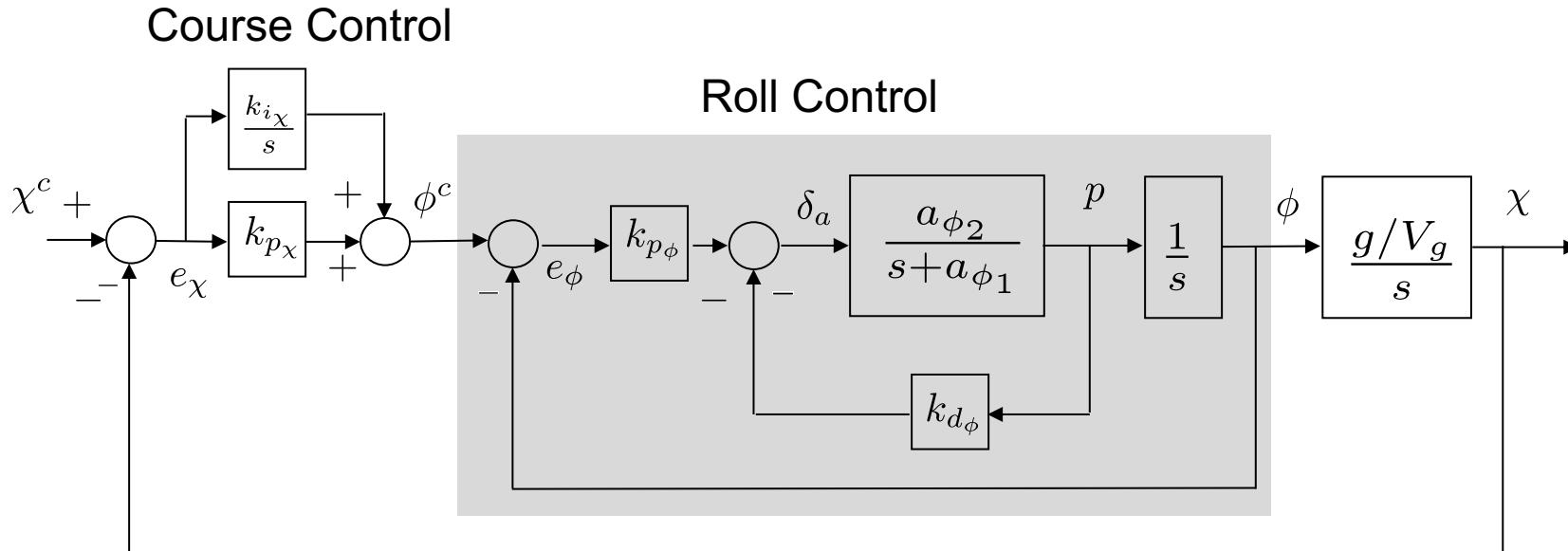
SLC: Two Loops Closed

At frequencies below middle-loop bandwidth, approximate CLTF as 1, then design outer loop



Key idea: Each successive loop must be lower in bandwidth
--- typically by a factor of 10 or more

Lateral-directional Autopilot



Roll Autopilot

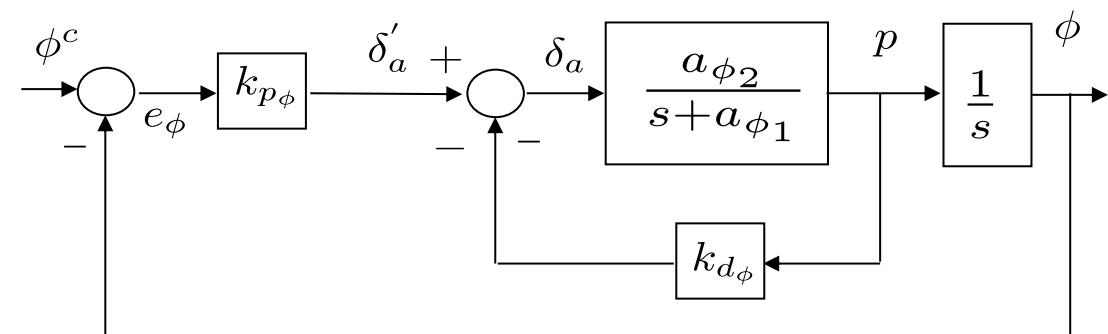
$$H_{\phi/\phi^c}(s) = \underbrace{\frac{k_{p_\phi} a_{\phi_2}}{s^2 + (a_{\phi_1} + a_{\phi_2} k_{d_\phi})s + k_{p_\phi} a_{\phi_2}}}_{\text{Closed Loop TF}} = \underbrace{\frac{\omega_{n_\phi}^2}{s^2 + 2\zeta_\phi \omega_{n_\phi} s + \omega_{n_\phi}^2}}_{\text{Canonical } 2^{nd}\text{-order TF}}$$

Design parameters are ω_{n_ϕ} and ζ_ϕ

Gains are given by

$$k_{p_\phi} = \frac{\omega_{n_\phi}^2}{a_{\phi_2}}$$

$$k_{d_\phi} = \frac{2\zeta_\phi \omega_{n_\phi} - a_{\phi_1}}{a_{\phi_2}}$$



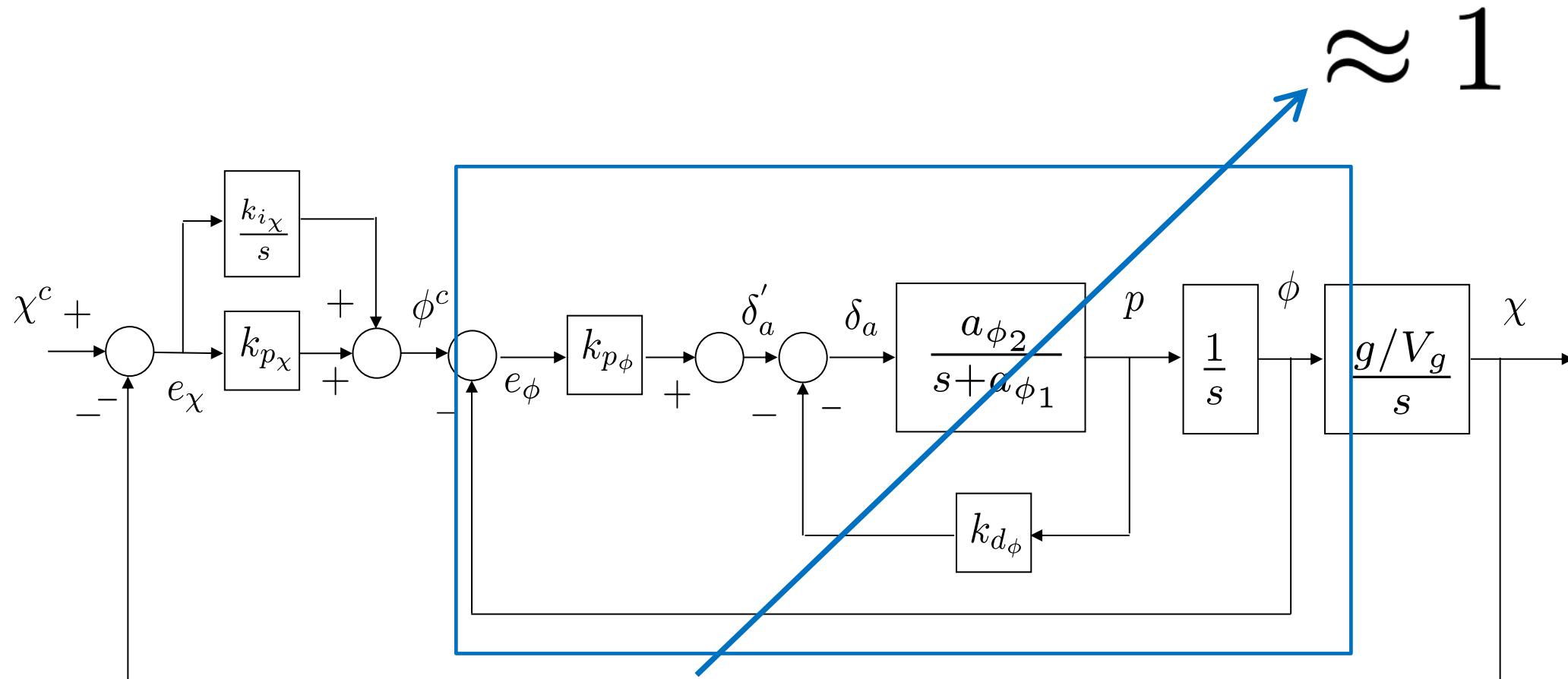
Implementation:

$$\delta_a(t) = k_{p_\phi}(\phi^c(t) - \phi(t)) - k_{d_\phi}p(t).$$

Roll Autopilot

- The book suggests using an integrator on roll in the roll loop to correct for any steady-state error due to disturbances
- Our current suggestion is to not have an integrator on inner loops, including the roll loop
 - Integrators add delay and instability -> not a good idea for inner-most loops
 - An integrator will be used on the course loop to correct for steady-state errors

Lateral-directional Autopilot



Course Hold Loop

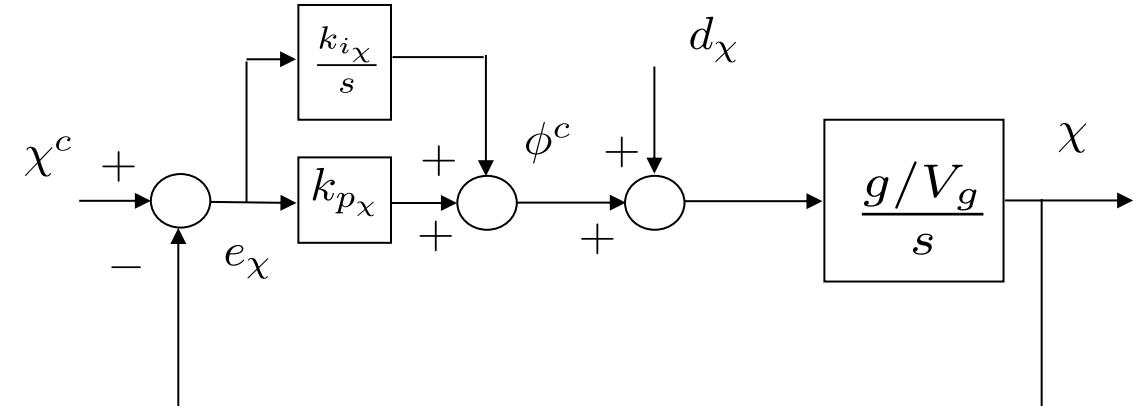
For the course loop, note the presence of the input disturbance

Using a PI controller for course, the response to the course command and disturbance is given by

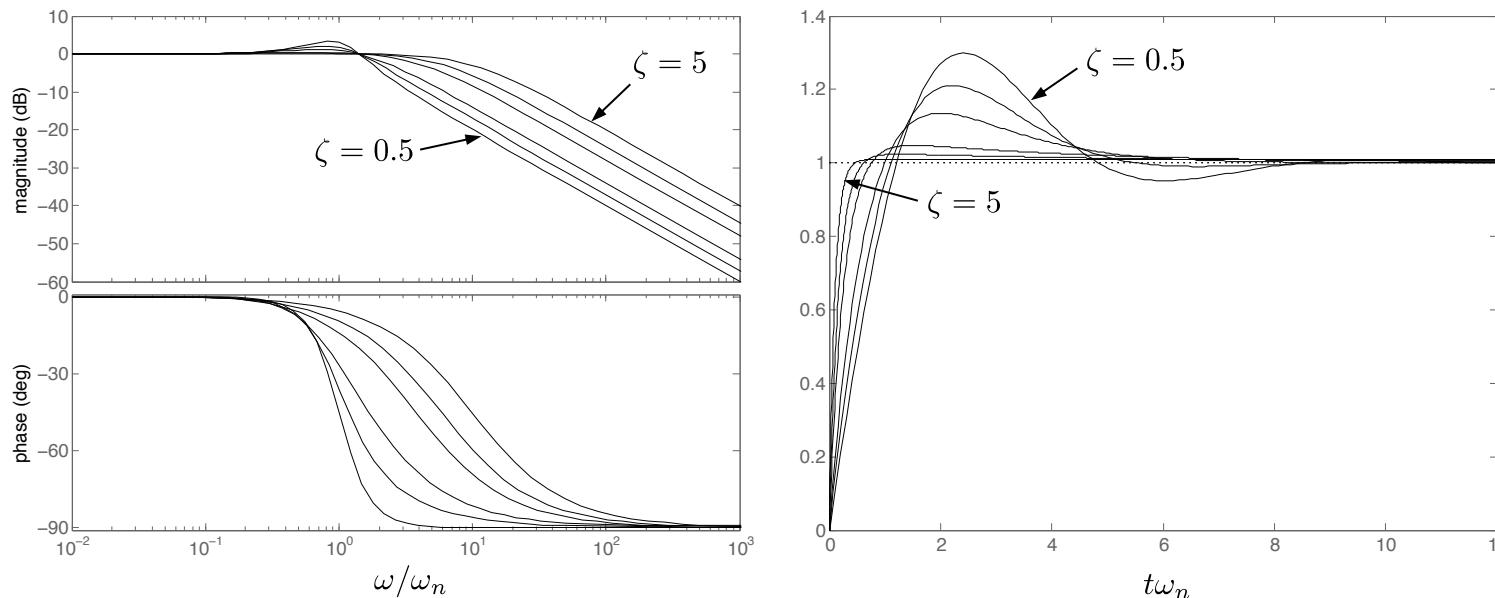
$$\chi = \frac{k_{p\chi}g/V_g s + k_{i\chi}g/V_g}{s^2 + k_{p\chi}g/V_g s + k_{i\chi}g/V_g} \chi^c + \frac{g/V_g s}{s^2 + k_{p\chi}g/V_g s + k_{i\chi}g/V_g} d_\chi$$

Note:

- There is a zero in the response to the course command χ^c
- The presence of the zero at the origin ensures rejection of low-frequency disturbances



TF Zero Affects Response

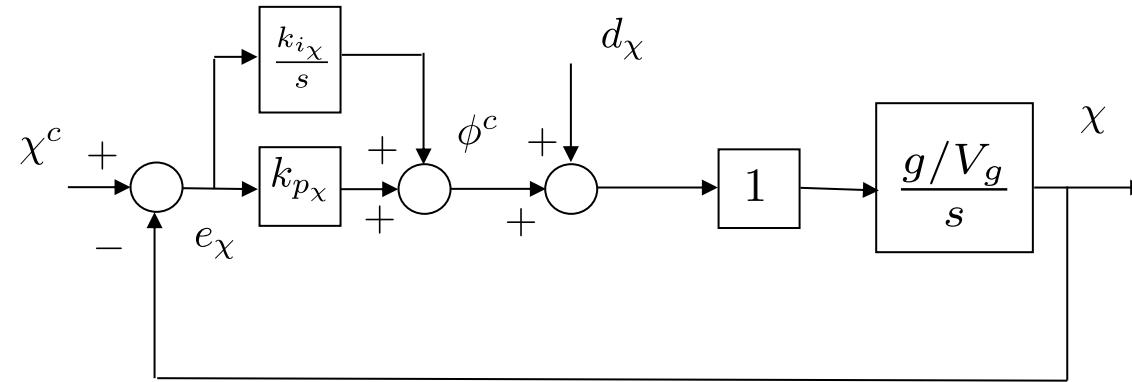


With a zero, the canonical second-order TF is given by

$$H(s) = \frac{2\zeta\omega_n s + \omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} = \left(\frac{2\zeta}{\omega_n}\right) \underbrace{\frac{\omega_n^2 s}{s^2 + 2\zeta\omega_n s + \omega_n^2}}_{d/dt \text{ of 2}^{\text{nd}}\text{-order TF}} + \underbrace{\frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}}_{2^{\text{nd}}\text{-order TF}}$$

Note that ζ has a different effect when the zero is present

Course Hold Loop



$$\chi = \underbrace{\frac{(k_{p_\chi} g/V_g)s + (k_{i_\chi} g/V_g)}{s^2 + (k_{p_\chi} g/V_g)s + (k_{i_\chi} g/V_g)} \chi^c}_{\text{Response to course command}} + \underbrace{\frac{(g/V_g)s}{s^2 + (k_{p_\chi} g/V_g)s + (k_{i_\chi} g/V_g)} d_\chi}_{\text{Response to disturbance}}$$

Equating coefficients to canonical TF gives:

$$\omega_{n_\chi}^2 = k_{i_\chi} g/V_g \quad \text{and} \quad 2\zeta_\chi \omega_{n_\chi} = k_{p_\chi} g/V_g$$

or

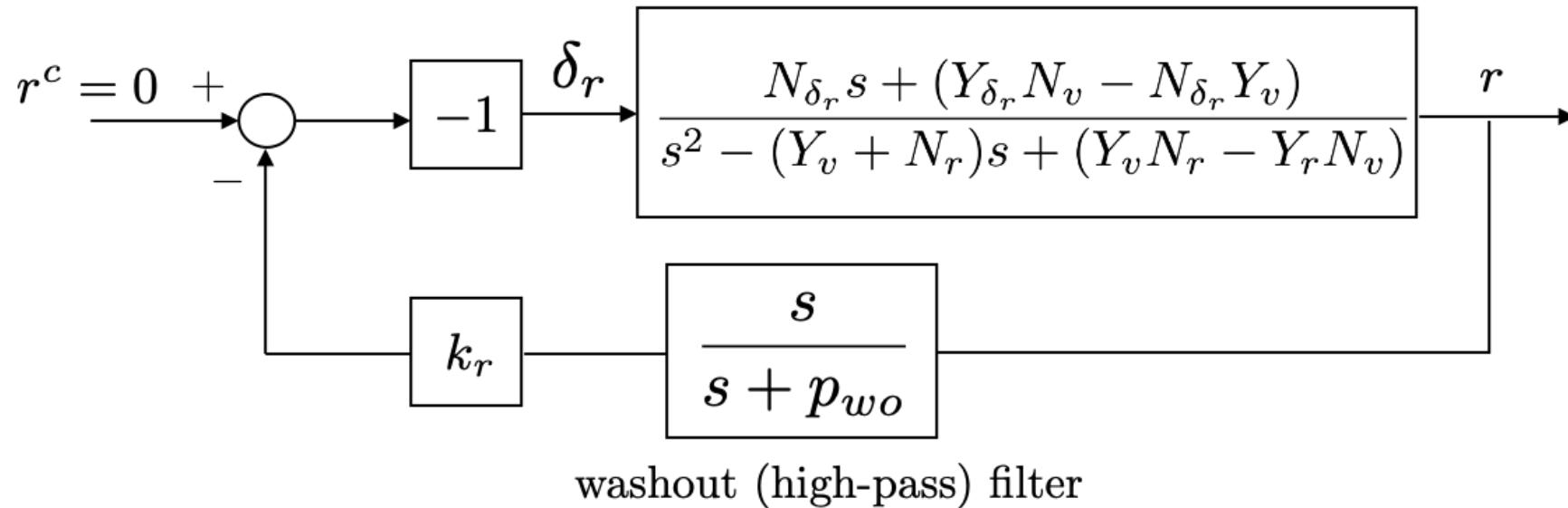
$$\omega_{n_\chi} = \frac{1}{W_\chi} \omega_{n_\phi}$$

$$k_{p_\chi} = 2\zeta_\chi \omega_{n_\chi} V_g/g$$

$$k_{i_\chi} = \omega_{n_\chi}^2 V_g/g$$

Design parameters are bandwidth separation W_χ and damping ratio ζ_χ

Yaw Damper



- The rudder is used to counteract the yaw rate caused by adverse yaw
- The washout filter makes it so that the yaw damper only counteracts high-frequency yaw rate
- The washout filter is a high-pass filter with cut-off frequency p_{wo}

Lateral Autopilot - Summary

If model is known, the the design parameters are

Inner Loop (roll attitude hold)

- ω_{n_ϕ} - Error in roll when aileron just saturates
- ζ_ϕ - Damping ratio for roll attitude loop

Outer Loop (course hold)

- $W_\chi > 1$ - Bandwidth separation between roll and course loops
- ζ_χ - Damping ratio for course hold loop

Yaw damper (if rudder is available)

- τ_r - cut off frequency for wash-out filter
- k_r - gain for yaw damper

Lateral Autopilot – In Flight Tuning

If model is not known, and autopilot must be tuned in flight, then the following gains are tuned one at a time, in this specific order:

Inner Loop (roll attitude hold)

- $k_{d\phi}$ - Increase $k_{d\phi}$ until onset of instability, and then back off by 20%
- $k_{p\phi}$ - Tune $k_{p\phi}$ to get acceptable step response

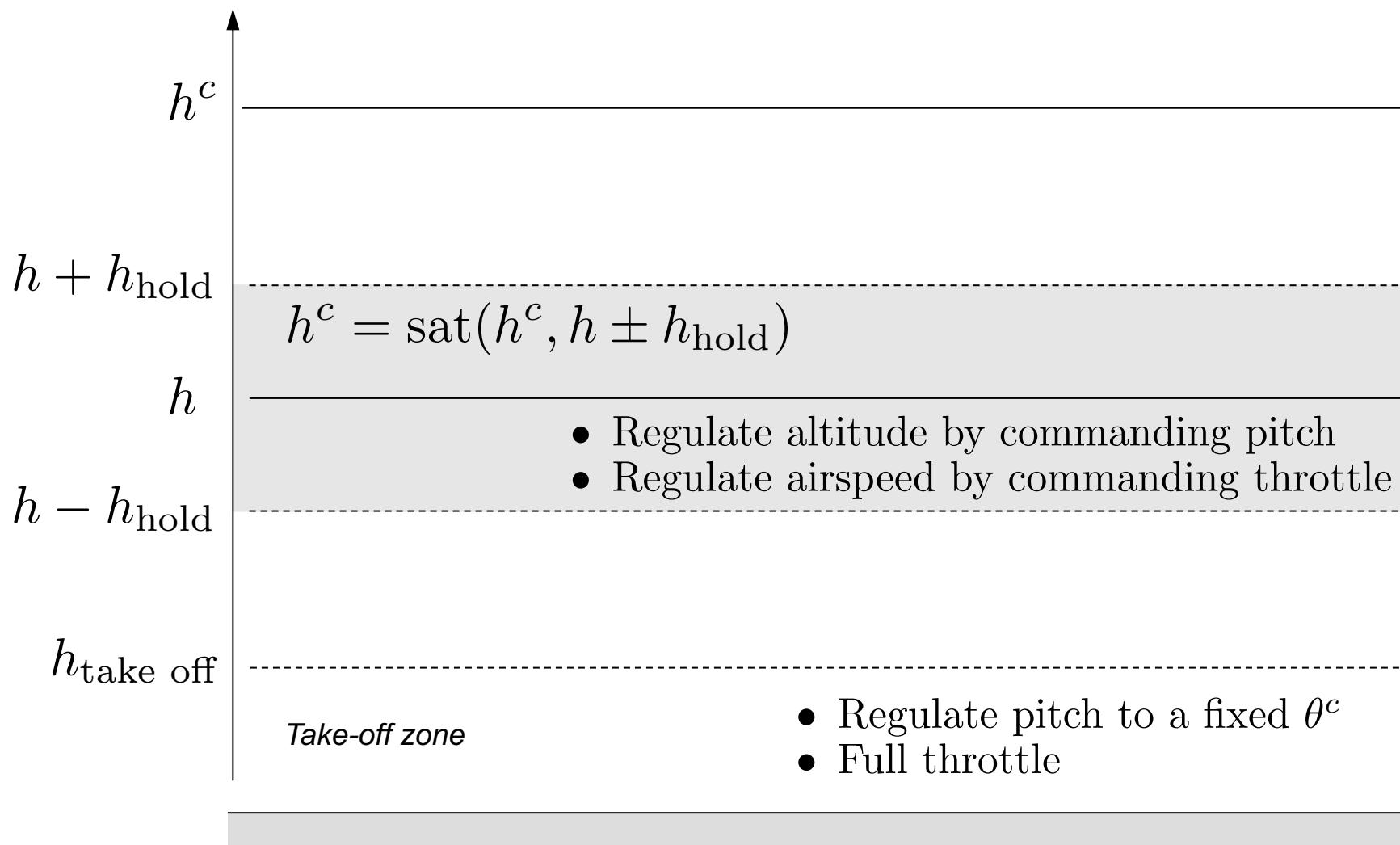
Outer Loop (course hold)

- k_{p_x} - Tune k_{p_x} to get acceptable step response
- k_{i_x} - Tune k_{i_x} to remove steady state error

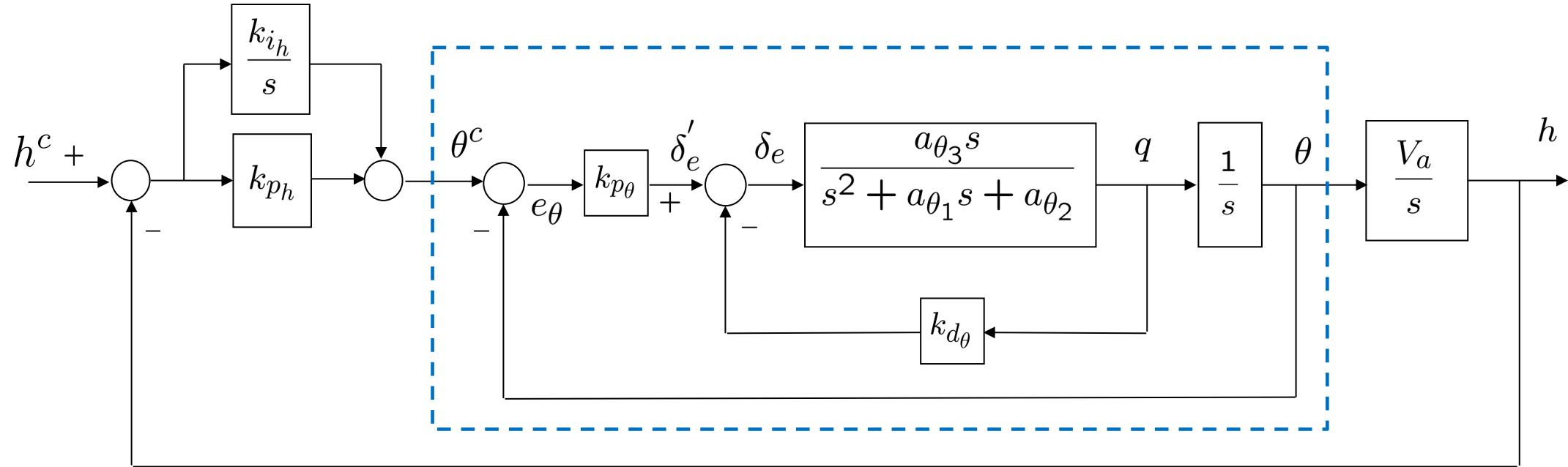
Sideslip hold (if rudder is available)

- τ_r - Tune τ_r to get acceptable step response
- k_r - Tune k_r to remove steady state error

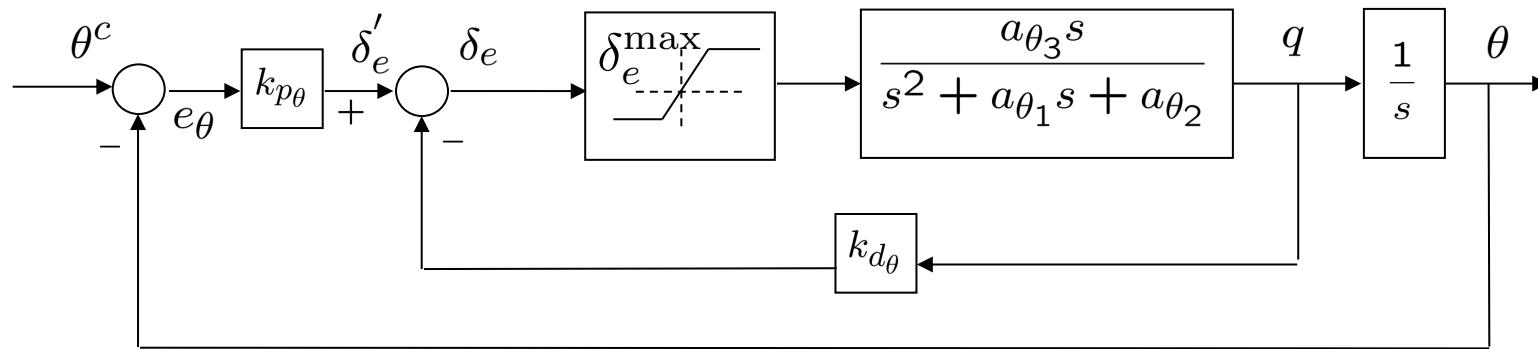
Longitudinal Flight Regimes



Altitude Hold Using Commanded Pitch



Pitch Attitude Hold



$$H_{\theta/\theta^c}(s) = \underbrace{\frac{k_{p\theta} a_{\theta_3}}{s^2 + (a_{\theta_1} + k_{d\theta} a_{\theta_3})s + (a_{\theta_2} + k_{p\theta} a_{\theta_3})}}_{\text{Closed Loop TF}} = \underbrace{\frac{K_{\theta_{DC}} \omega_{n_\theta}^2}{s^2 + 2\zeta_\theta \omega_{n_\theta} s + \omega_{n_\theta}^2}}_{\text{Note: Non-unity DC Gain}}$$

Equating coefficients, the gains are given by

$$k_{p\theta} = \frac{\omega_{n_\theta}^2 - a_{\theta_2}}{a_{\theta_3}}$$

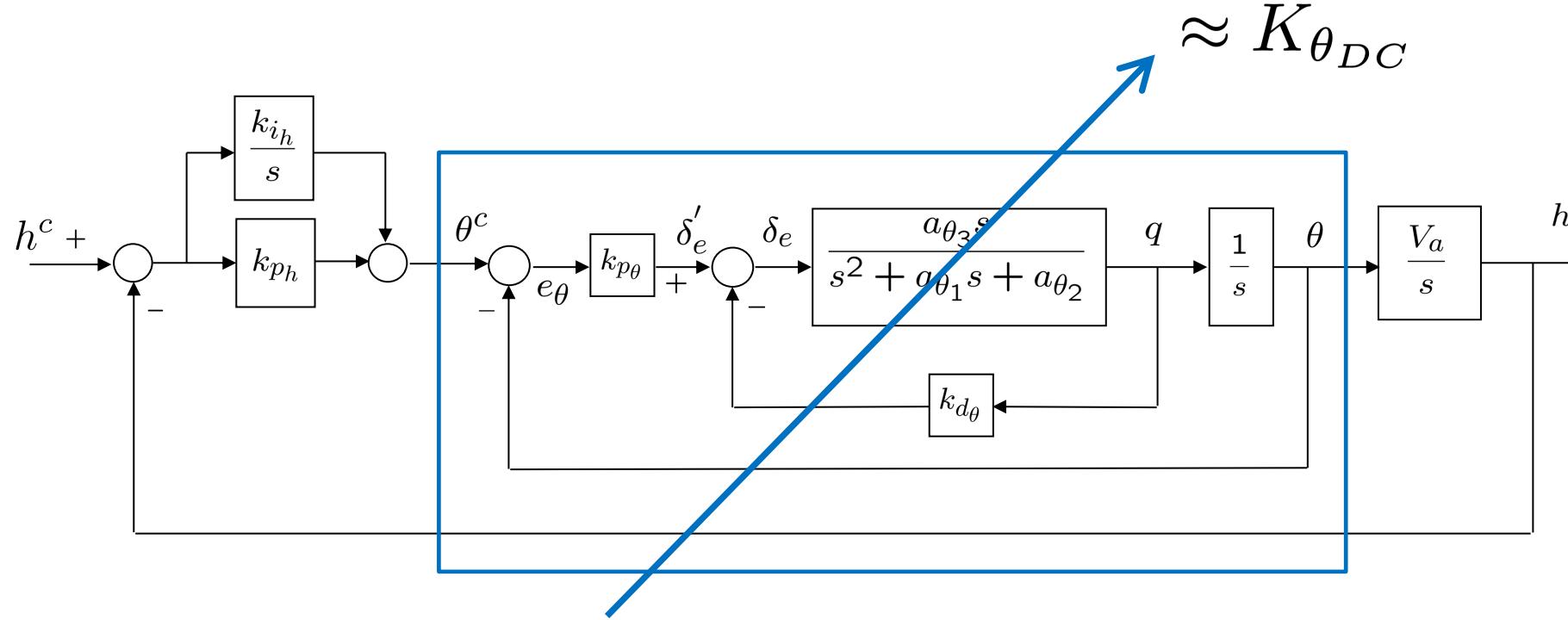
$$k_{d\phi} = \frac{2\zeta_\theta \omega_{n_\theta} - a_{\theta_1}}{a_{\theta_3}}$$

Design parameters are ω_{n_θ} and ζ_θ

The DC gain is

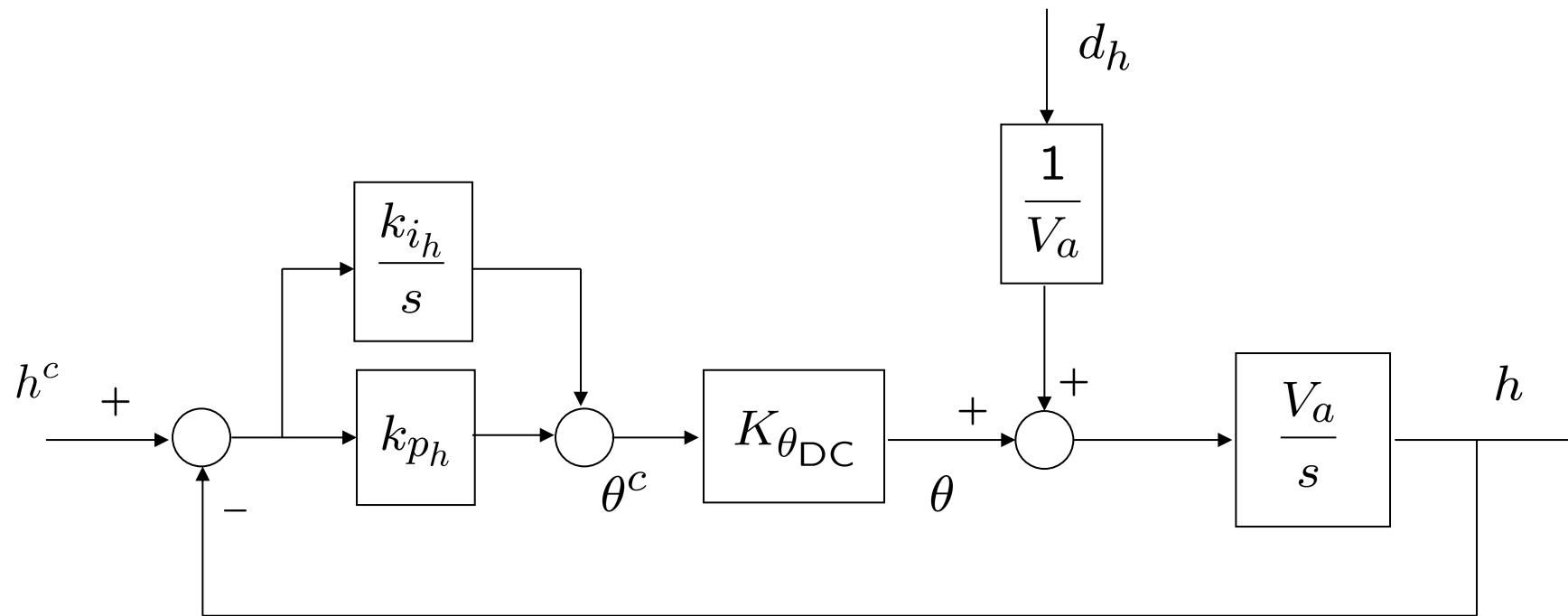
$$K_{\theta_{DC}} = \frac{k_{p\theta} a_{\theta_3}}{a_{\theta_2} + k_{p\theta} a_{\theta_3}}$$

Altitude Hold Using Commanded Pitch



Provided pitch loop functions as intended, we can simplify the inner-loop dynamics to $\theta \approx K_{\theta_{DC}} \theta^c$

Altitude from Pitch – Simplified



$$H(s) = \left(\frac{K_{\theta_{DC}} V_a k_{p_h} s + K_{\theta_{DC}} V_a k_{i_h}}{s^2 + K_{\theta_{DC}} V_a k_{p_h} s + K_{\theta_{DC}} V_a k_{i_h}} \right) h^c(s) + \left(\frac{s}{s^2 + K_{\theta_{DC}} V_a k_{p_h} s + K_{\theta_{DC}} V_a k_{i_h}} \right) d_h(s)$$

A PI control on altitude ensures that h tracks constant h^c with zero steady-state error and rejects constant disturbances

Altitude from Pitch Gain Calculations

Equating the transfer functions

$$H_{h/h^c}(s) = \frac{K_{\theta_{DC}} V_a k_{p_h} s + K_{\theta_{DC}} V_a k_{i_h}}{s^2 + K_{\theta_{DC}} V_a k_{p_h} s + K_{\theta_{DC}} V_a k_{i_h}} \quad \text{and} \quad H_c(s) = \frac{2\zeta\omega_n s + \omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}$$

gives the coefficients

$$k_{i_h} = \frac{\omega_{n_h}^2}{K_{\theta_{DC}} V_a}$$

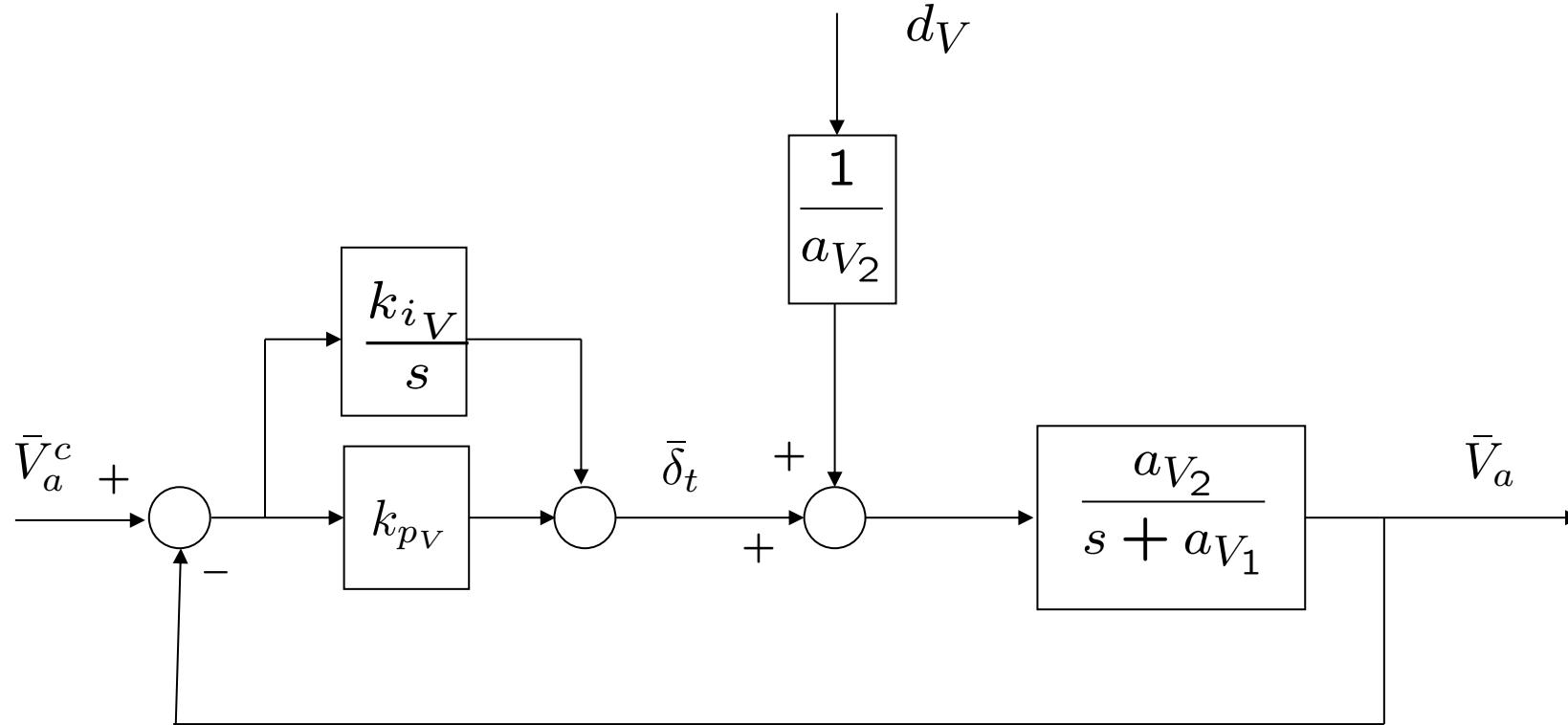
$$k_{p_h} = \frac{2\zeta_h \omega_{n_h}}{K_{\theta_{DC}} V_a}$$

where bandwidth separation is achieved by selecting

$$\omega_{n_h} = \frac{1}{W_h} \omega_{n_\theta}$$

Design parameters are bandwidth separation W_h and damping ratio ζ_θ

Airspeed Hold Using Throttle



$$V_a = \left(\frac{a_{V_2}(k_{pV}s + k_{iV})}{s^2 + (a_{V_1} + a_{V_2}k_{pV})s + a_{V_2}k_{iV}} \right) V_a^c + \left(\frac{s}{s^2 + (a_{V_1} + a_{V_2}k_{pV})s + a_{V_2}k_{iV}} \right) d_V$$

A PI control on the throttle-to-airspeed loop ensures that V_a tracks a constant V_a^c with zero steady-state error and rejects constant disturbances

Airspeed from Throttle Gain Calculations

Equating the transfer functions and their coefficients

$$H_{V_a/V_a^c}(s) = \left(\frac{a_{V_2} k_{p_V} s + a_{V_2} k_{i_V}}{s^2 + (a_{V_1} + a_{V_2} k_{p_V})s + a_{V_2} k_{i_V}} \right) = \frac{\varrho 2\zeta \omega_n s + \omega_n^2}{s^2 + 2\zeta \omega_n s + \omega_n^2}$$

gives the gains

$$k_{i_V} = \frac{\omega_{n_V}^2}{a_{V_2}}$$

$$k_{p_V} = \frac{2\zeta_V \omega_{n_V} - a_{V_1}}{a_{V_2}}$$

Design parameters are natural frequency ω_{n_V} and damping ratio ζ_V

The control signal is

$$\begin{aligned}\delta_t &= \delta_t^* + \bar{\delta}_t \\ &= \delta_t^* + k_{p_V} (V_a^c - V_a) + k_{i_V} \int (V_a^c - V_a) dt\end{aligned}$$

Note that if δ_t^* is imprecisely known, the integrator will compensate.

Longitudinal Autopilot - Summary

If model is known, the the design parameters are

Inner Loop (pitch attitude hold)

- $\omega_{n\theta}$ - natural frequency for pitch.
- ζ_θ - Damping ratio for pitch attitude loop.

Altitude Hold Outer Loop

- $W_h > 1$ - Bandwidth separation between pitch and altitude loops.
- ζ_h - Damping ratio for altitude hold loop.

Airspeed Hold Outer Loop

- $W_{V_2} > 1$ - Bandwidth separation between pitch and airspeed loops.
- ζ_{V_2} - Damping ratio for airspeed hold loop.

Throttle hold (inner loop)

- ω_{n_V} - Natural frequency for throttle loop.
- ζ_V - Damping ratio for throttle loop.

Longitudinal Autopilot – In Flight Tuning

If model is not known, and autopilot must be tuned in flight, then the following gains are tuned one at a time, in this specific order:

Inner Loop (pitch attitude hold)

- $k_{d\theta}$ - Increase $k_{d\theta}$ until onset of instability, and then back off by 20%.
- $k_{p\theta}$ - Tune $k_{p\theta}$ to get acceptable step response.

Altitude Hold Outer Loop

- k_{ph} - Tune k_{ph} to get acceptable step response.
- k_{ih} - Tune k_{ih} to remove steady state error.

Airspeed Hold Outer Loop

- k_{pv_2} - Tune k_{pv_2} to get acceptable step response.
- k_{iv_2} - Tune k_{iv_2} to remove steady state error.

Throttle hold (inner loop)

- k_{pv} - Tune k_{pv} to get acceptable step response.
- k_{iv} - Tune k_{iv} to remove steady state error.

Python Implementation

```
class autopilot:
    def __init__(self, ts_control):
        self.roll_from_aileron = pdControlWithRate(kp, kd, limit)
        self.course_from_roll = piControl(kp, ki, Ts, limit)
        self.yaw_damper = transferFunction(num, den, Ts)
        self.pitch_from_elevator = pdControlWithRate(kp, kd, limit)
        self.altitude_from_pitch = piControl(kp, ki, Ts, limit)
        self.airspeed_from_throttle = piControl(kp, ki, Ts, limit)

    def update(self, cmd, state):
        # lateral autopilot
        chi_c = wrap(cmd.course_command, state.chi)
        phi_c = self.saturate(
            cmd.phi_feedforward + self.course_from_roll.update(chi_c, state.chi)
            -np.radians(30), np.radians(30))
        delta_a = self.roll_from_aileron.update(phi_c, state.phi, state.p)
        delta_r = self.yaw_damper.update(state.r)

        # longitudinal autopilot
        # saturate the altitude command
        h_c = self.saturate(cmd.altitude_command,
                            state.h - AP.altitude_zone, state.h + AP.altitude_zone)
        theta_c = self.altitude_from_pitch.update(h_c, state.h)
        delta_e = self.pitch_from_elevator.update(theta_c, state.theta, state.q)
        delta_t = self.airspeed_from_throttle.update(cmd.airspeed_command,
                                                    state.Va)
        delta_t = self.saturate(delta_t, 0.0, 1.0)

    return delta, self.commanded_state
```

Wrap Function

```
import numpy as np

def wrap(chi_1, chi_2):
    while chi_1 - chi_2 > np.pi:
        chi_1 = chi_1 - 2.0 * np.pi
    while chi_1 - chi_2 < -np.pi:
        chi_1 = chi_1 + 2.0 * np.pi
    return chi_1
```

PID Loop Implementation

$$u(t) = k_p e(t) + k_i \int_{-\infty}^t e(\tau) d\tau + k_d \frac{de}{dt}(t) \quad \text{PID continuous time}$$

$$e(t) = y^c(t) - y(t)$$

Taking Laplace transform...

$$U(s) = k_p E(s) + k_i \frac{E(s)}{s} + k_d s E(s)$$

Use bandwidth-limited differentiator to reduce noise

$$U(s) = k_p E(s) + k_i \frac{E(s)}{s} + k_d \frac{s}{\tau s + 1} E(s)$$

PID Loop Implementation

Tustin's rule or trapezoidal rule: $s \mapsto \frac{2}{T_s} \left(\frac{1 - z^{-1}}{1 + z^{-1}} \right)$

Integrator: $I(z) = \frac{T_s}{2} \left(\frac{1 + z^{-1}}{1 - z^{-1}} \right) E(z)$ 

$$I[n] = I[n - 1] + \frac{T_s}{2} (E[n] + E[n - 1])$$

Differentiator: $D(z) = \frac{\frac{2}{T_s} \left(\frac{1-z^{-1}}{1+z^{-1}} \right)}{\frac{2\tau}{T_s} \left(\frac{1-z^{-1}}{1+z^{-1}} \right) + 1} E(z) = \frac{\left(\frac{2}{2\tau+T_s} \right) (1 - z^{-1})}{1 - \left(\frac{2\tau-T_s}{2\tau+T_s} \right) z^{-1}} E(z)$



$$D[n] = \left(\frac{2\tau - T_s}{2\tau + T_s} \right) D[n - 1] + \left(\frac{2}{2\tau + T_s} \right) (E[n] - E[n - 1])$$

Integrator Anti-wind-up

$$u_{\text{unsat}}^- = k_p e + k_d D + k_i I^- \quad \text{control before anti-wind-up update}$$

$$u_{\text{unsat}}^+ = k_p e + k_d D + k_i I^+ \quad \text{control after anti-wind-up update}$$

$$I^+ = I^- + \Delta I \quad \Delta I \text{ is anti-wind-up update}$$

$$u_{\text{unsat}}^+ = u_{\text{unsat}}^- + k_i \Delta I \quad \text{subtracting top two equations}$$

Let $u_{\text{unsat}}^+ = u$ value of control after saturation is applied

$$\Delta I = \frac{1}{k_i} (u - u_{\text{unsat}}^-) \quad \text{solving...}$$

$$I^+ = I^- + \frac{1}{k_i} (u - u_{\text{unsat}}^-)$$

PID Implementation (Python)

```
class PIDControl:  
    def __init__(self, kp=0.0, ki=0.0, kd=0.0, Ts=0.01, sigma=0.05, limit=1.0):  
        self.kp = kp  
        self.ki = ki  
        self.kd = kd  
        self.Ts = Ts  
        self.limit = limit  
        self.integrator = 0.0  
        self.error_delay_1 = 0.0  
        self.error_dot_delay_1 = 0.0  
        self.y_dot = 0.0  
        self.y_delay_1 = 0.0  
        self.y_dot_delay_1 = 0.0  
        # gains for differentiator  
        self.a1 = (2.0 * sigma - Ts) / (2.0 * sigma + Ts)  
        self.a2 = 2.0 / (2.0 * sigma + Ts)  
  
    def update(self, y_ref, y, reset_flag=False):  
        if reset_flag is True:  
            self.integrator = 0.0  
            self.error_delay_1 = 0.0  
            self.y_dot = 0.0  
            self.y_delay_1 = 0.0  
            self.y_dot_delay_1 = 0.0  
        # compute the error  
        error = y_ref - y  
        # update the integrator using trapazoidal rule  
        self.integrator = self.integrator \  
                         + (self.Ts/2) * (error + self.error_delay_1)  
        # update the differentiator  
        error_dot = self.a1 * self.error_dot_delay_1 \  
                   + self.a2 * (error - self.error_delay_1)  
        # PID control  
        u = self.kp * error \  
           + self.ki * self.integrator \  
           + self.kd * error_dot  
        # saturate PID control at limit  
        u_sat = self._saturate(u)  
        # integral anti-windup  
        # adjust integrator to keep u out of saturation  
        if np.abs(self.ki) > 0.0001:  
            self.integrator = self.integrator \  
                           + (self.Ts / self.ki) * (u_sat - u)  
        # update the delayed variables  
        self.error_delay_1 = error  
        self.error_dot_delay_1 = error_dot
```

Integrator:

$$I[n] = I[n-1] + \frac{T_s}{2} (E[n] + E[n-1])$$

Differentiator:

$$D[n] = \left(\frac{2\tau - T_s}{2\tau + T_s} \right) D[n-1] + \left(\frac{2}{2\tau + T_s} \right) (E[n] - E[n-1])$$

Integrator Anti-windup:

$$I^+ = I^- + \frac{1}{k_i} (u - u_{\text{unsat}}^-)$$

PID Implementation (Matlab)

```
1 function u = pidloop(y_c, y, flag, kp, ki, kd, limit, Ts, tau)
2     persistent integrator;
3     persistent differentiator;
4     persistent error_d1;
5     if flag==1, % reset (initialize) persistent variables
6         % when flag==1
7         integrator = 0;
8         differentiator = 0;
9         error_d1 = 0; % _d1 means delayed by one time step
10    end
11    error = y_c - y; % compute the current error
12    integrator = integrator + (Ts/2)*(error + error_d1);
13    % update integrator
14    differentiator = (2*tau-Ts)/(2*tau+Ts)*differentiator...
15        + 2/(2*tau+Ts)*(error - error_d1);
16    % update differentiator
17    error_d1 = error; % update the error for next time through
18        % the loop
19    u = sat(... % implement PID control
20        kp * error +... % proportional term
21        ki * integrator +... % integral term
22        kd * differentiator,... % derivative term
23        limit... % ensure abs(u)<=limit
24    );
25    % implement integrator anti-windup
26    if ki~=0
27        u_unsat = kp*error + ki*integrator + kd*differentiator;
28        integrator = integrator + Ts/ki * (u - u_unsat);
29    end
30
31 function out = sat(in, limit)
32     if in > limit, out = limit;
33     elseif in < -limit; out = -limit;
34     else out = in;
35     end
```

Integrator:

$$I[n] = I[n-1] + \frac{T_s}{2} (E[n] + E[n-1])$$

Differentiator:

$$D[n] = \left(\frac{2\tau - T_s}{2\tau + T_s} \right) D[n-1] + \left(\frac{2}{2\tau + T_s} \right) (E[n] - E[n-1])$$

Integrator Anti-windup:

$$I^+ = I^- + \frac{1}{k_i} (u - u_{\text{unsat}}^-)$$

Yaw Damper Implementation

For a general, first-order continuous-time transfer function of the form

$$H(s) = \frac{U(s)}{Y(s)} = k \frac{n_1 s + n_0}{d_1 s + d_0}$$

we want to be able to calculate the equivalent discrete-time transfer function

$$H(z) = \frac{U(z)}{Y(z)} = \frac{b_0 + b_1 z^{-1}}{1 + a_1 z^{-1}}$$

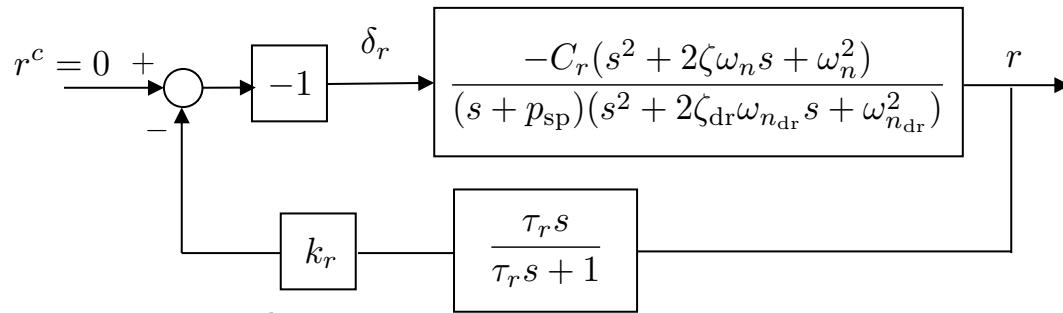
for a specific sample rate T_s . Using Tustin's method (trapezoidal rule) by substituting $s \rightarrow \frac{2}{T_s} \left(\frac{1-z^{-1}}{1+z^{-1}} \right)$, we can calculate the coefficients of $H(z)$ as

$$\begin{aligned} b_0 &= \frac{k(2n_1 + T_s n_0)}{2d_1 + T_s d_0} & b_1 &= -\frac{k(2n_1 - T_s n_0)}{2d_1 + T_s d_0} \\ a_1 &= -\frac{k(2d_1 - T_s d_0)}{2d_1 + T_s d_0} \end{aligned}$$

By taking the inverse z transform, we can convert our discrete transfer function to an discrete-time difference equation:

$$u_k = -a_1 u_{k-1} + b_0 y_k + b_1 y_{k-1}$$

Yaw Damper Implementation



For our yaw damper,

$$H(s) = \frac{\delta_r(s)}{r(s)} = k_r \left(\frac{\tau_r s}{\tau_r s + 1} \right) = k_r \left(\frac{s}{s + p_{wo}} \right),$$

where $k_r = 0.2$, $p_{wo} = 0.45$ rad/s, and $T_s = 0.01$ s

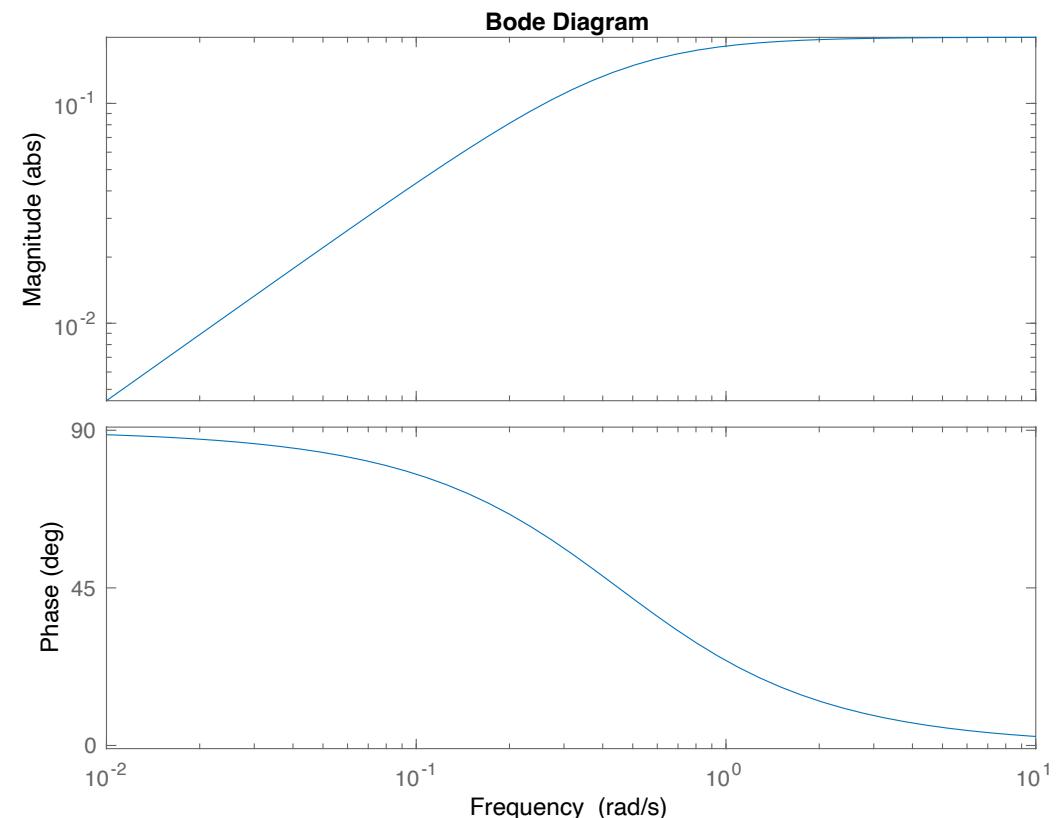
This gives a discrete-time transfer function of

$$H(z) = \frac{\delta_r(z)}{r(z)} = \frac{0.1996 - 0.1996 z^{-1}}{1 - 0.9955 z^{-1}}$$

with the corresponding difference equation

$$\delta_{r,k} = 0.9955 \delta_{r,k-1} + 0.1996 r_k - 0.1996 r_{k-1}$$

The coefficients of $H(z)$ can be calculated manually as we have done here or they can be computed in MATLAB using the `c2d` command with the '`tustin`' option or in Python using the `control.matlab.c2d` function. This is especially convenient for higher-order transfer functions.

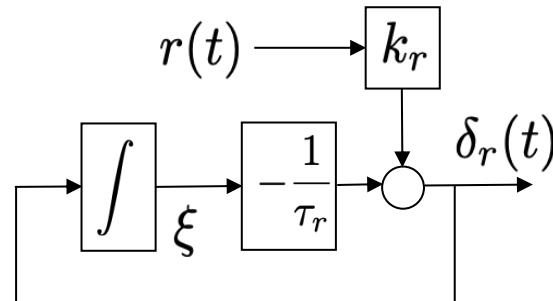


Yaw Damper Implementation

Yaw damper: $\delta_r(s) = k_r \left(\frac{s}{s + \frac{1}{\tau_r}} \right) r(s)$

In time domain: $\dot{\delta}_r = -\frac{1}{\tau_r} \delta_r + k_r \dot{r}$

Integrating: $\delta_r(t) = -\frac{1}{\tau_r} \int_{-\infty}^t \delta_r(\sigma) d\sigma + k_r r(t)$



State-space form:

$$\dot{\xi} = -\frac{1}{\tau_r} \xi + k_r r$$

$$\delta_r = -\frac{1}{\tau_r} \xi + k_r r.$$

Discrete-time:

$$\xi_k = \xi_{k-1} + T_s \left(-\frac{1}{\tau_r} \xi_{k-1} + k_r r_{k-1} \right)$$

$$\delta_{rk} = -\frac{1}{\tau_r} \xi_k + k_r r_k.$$

Yaw Damper Implementation Alternative

Discrete-time:

$$\xi_k = \xi_{k-1} + T_s \left(-\frac{1}{\tau_r} \xi_{k-1} + k_r r_{k-1} \right)$$
$$\delta_{rk} = -\frac{1}{\tau_r} \xi_k + k_r r_k.$$

```
class yawDamper:  
    def __init__(self, k_r, tau_r, Ts):  
        # set initial conditions  
        self.xi = 0.  
        self.Ts = Ts  
        self.k_r = k_r  
        self.tau_r = tau_r  
  
    def update(self, u):  
        self.xi = self.xi  
            + self.Ts * (-1 / self.tau_r * self.xi + self.k_r * r)  
        delta_r = -1 / self.tau_r * self.xi + self.k_r * r  
        return delta_r
```

Simulation Project

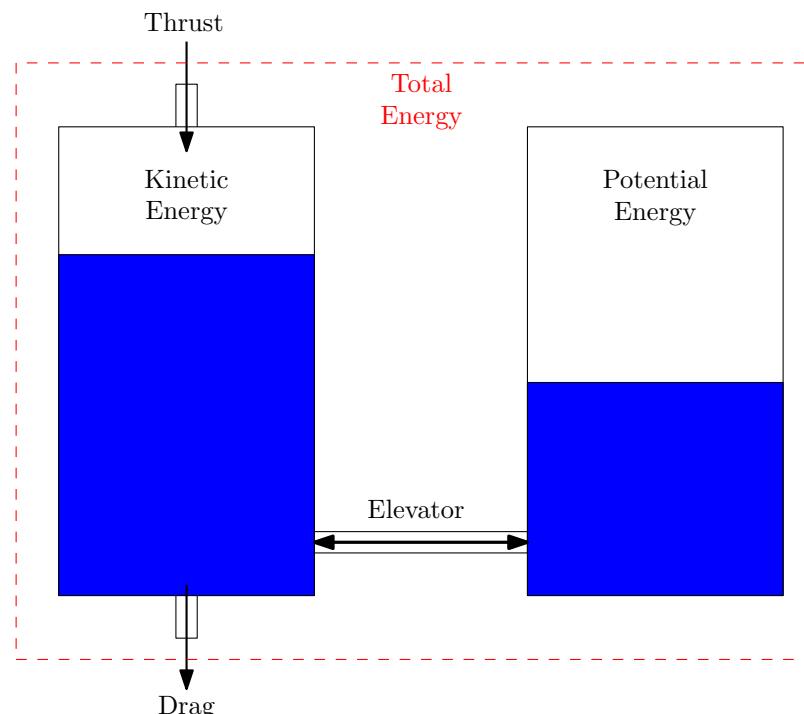
- Step 1.** Roll attitude loop. For Aerosonde model use $V_a = 25 \text{ m/s}$. Put aircraft in trim, and tune step response on roll.
- Step 2.** Tune the yaw damper to keep $\beta \approx 0$.
- Step 3.** Course attitude loop. Tune step response in χ .
- Step 4.** Pitch attitude loop. Tune step response in pitch angle.
- Step 5.** Tune step response in airspeed.
- Step 6.** Tune step response in altitude (after saturation).
- Step 7.** Stress test simulation from take-off. Command steps in alititude, airspeed, course.

Outline

- Successive Loop Closure
- Total Energy Control
- LQR Control

Total Energy Control

- Developed in the 1980's by Antonius Lambregts
- Based on energy manipulation techniques from the 1950's
- Control the energy of the system instead of the altitude and airspeed



Total Energy Control

Kinetic Energy:

$$E_K \triangleq \frac{1}{2}mV_a^2$$

Potential Energy:

$$E_P \triangleq mgh$$

Total Energy:

$$E_T \triangleq E_P + E_K$$

Energy Difference:

$$E_D \triangleq E_P - E_K$$

Error Definition:

$$\tilde{E}_K = \frac{1}{2}m \left((V_a^d)^2 - V_a^2 \right)$$

$$\tilde{E}_P = mg (h^d - h)$$

$$\tilde{E}_T = \tilde{E}_P + \tilde{E}_K$$

$$\tilde{E}_D = \tilde{E}_P - \tilde{E}_K$$

Total Energy Control

Original TECS proposed by Lambregts is based on energy rates:

- $T^c = T_D + k_{p,t} \dot{E}_t + k_{i,t} \int_{t_0}^t \dot{\tilde{E}}_t \delta_\tau$
 - T_D is thrust needed to counteract drag
 - PI controller based on total energy rate
- $\theta^c = k_{p,\theta} \dot{E}_d + k_{i,\theta} \int_{t_0}^t \dot{\tilde{E}}_d \delta_\tau$
 - PI controller based on energy distribution rate
- Stability shown for linear systems

We will show that the performance of this scheme is less than desirable.

Total Energy Control

If the thrust needed to counteract drag is unknown, then one possibility is to use an integrator to find T_D :

- $T^c = k_{p,t} \tilde{E}_t + k_{i,t} \int \tilde{E}_t d\tau + k_{d,t} \dot{E}_t$
 - PID controller based on total energy (not energy rate)
- $\theta^c = k_{p,\theta} \tilde{E}_d + k_{i,\theta} \int \tilde{E}_d d\tau + k_{d,\theta} \dot{E}_d$
 - PID controller based on energy distribution (not rate)

Total Energy Control

Nonlinear re-derivation:

- Error Definitions

$$\begin{aligned}\tilde{E}_K &= \frac{1}{2}m \left((V_a^d)^2 - V_a^2 \right) \\ \tilde{E}_P &= mg (h^d - h)\end{aligned}$$

- Lyapunov Function

$$V = \frac{1}{2}\tilde{E}_T^2 + \frac{1}{2}\tilde{E}_D^2$$

- Controller

$$\begin{aligned}T^c &= D + \frac{\tilde{E}_T^d}{V_a} + k_T \frac{\tilde{E}_T}{V_a} \\ \gamma^c &= \sin^{-1} \left(\frac{\dot{h}^d}{V_a} + \frac{1}{2mgV_a} \left(-k_1 \tilde{E}_K + k_2 \tilde{E}_P \right) \right)\end{aligned}$$

Total Energy Control

- Original:
$$T^c = D + k_{p,t} \frac{\dot{E}_T}{mgV_a} + k_{i,t} \frac{\tilde{E}_T}{mgV_a}$$
- Nonlinear:
$$T^c = D + \frac{\dot{E}_T^d}{V_a} + k_T \frac{\tilde{E}_T}{V_a}$$

Similar if $k_{p,T} = mg$ and $k_{i,T} = mgk_T$.

The nonlinear controller uses the desired energy rate.

Total Energy Control

- Modified Original (Ardupilot):

$$\theta^c = \frac{k_{p,\theta}}{V_a mg} \left((2 - k) \dot{E}_P - k \dot{E}_K \right) + \frac{k_{i,\theta}}{V_a mg} \tilde{E}_D$$
$$k \in [0, 2]$$

- Nonlinear:

$$\gamma^c = \sin^{-1} \left(\frac{\dot{h}^d}{V_a} + \frac{1}{2mgV_a} \left(-k_1 \tilde{E}_K + k_2 \tilde{E}_P \right) \right)$$

$$k_1 \triangleq |k_T - k_D|$$

$$k_2 \triangleq k_T + k_D$$

$$0 < k_T \leq k_D$$

Lyapunov derivation suggests potential energy error should be weighted more than kinetic energy

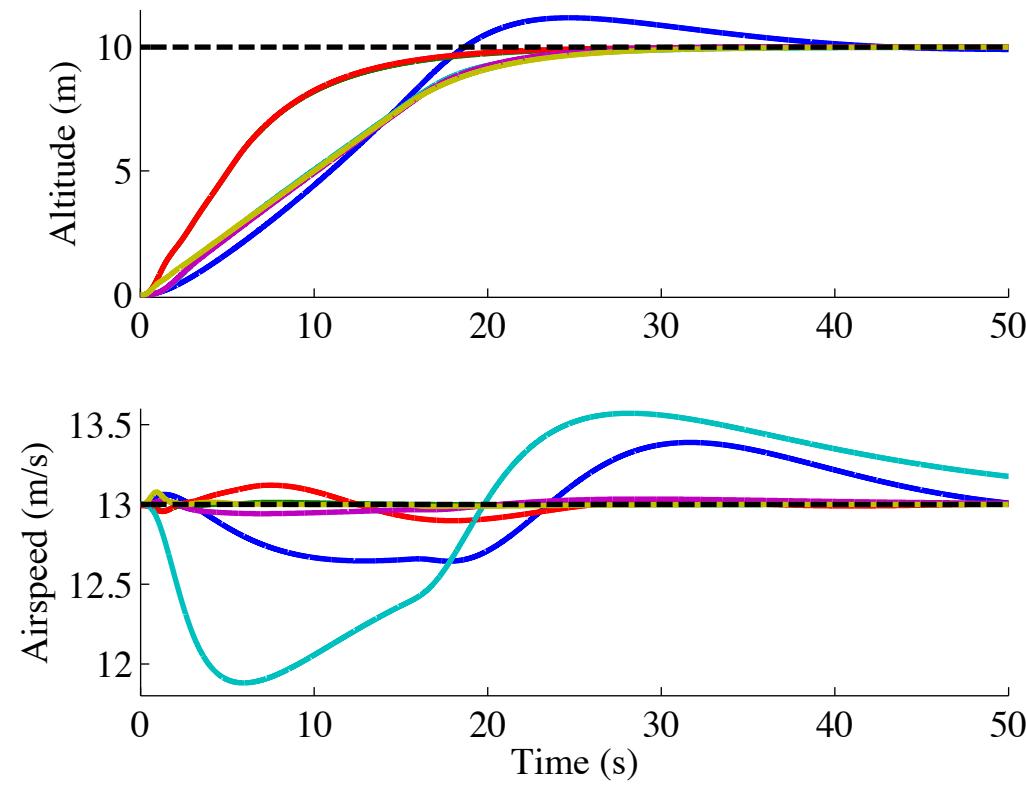
Total Energy Control

If the drag is unknown, then we can add an adaptive estimate:

$$\begin{aligned} T^c &= \hat{D} + \Phi^\top \hat{\Psi} + \frac{\dot{E}_T^d}{V_a} + k_T \frac{\tilde{E}_T}{V_a} \\ \gamma^c &= \sin^{-1} \left(\frac{\dot{h}^d}{V_a} + \frac{1}{2mgV_a} \left(-k_1 \tilde{E}_K + k_2 \tilde{E}_P \right) \right) \\ \dot{\hat{\Psi}} &= \left(\Gamma_T \tilde{E}_T - \Gamma_D \tilde{E}_D \right) \Phi V_a \end{aligned}$$

Total Energy Control

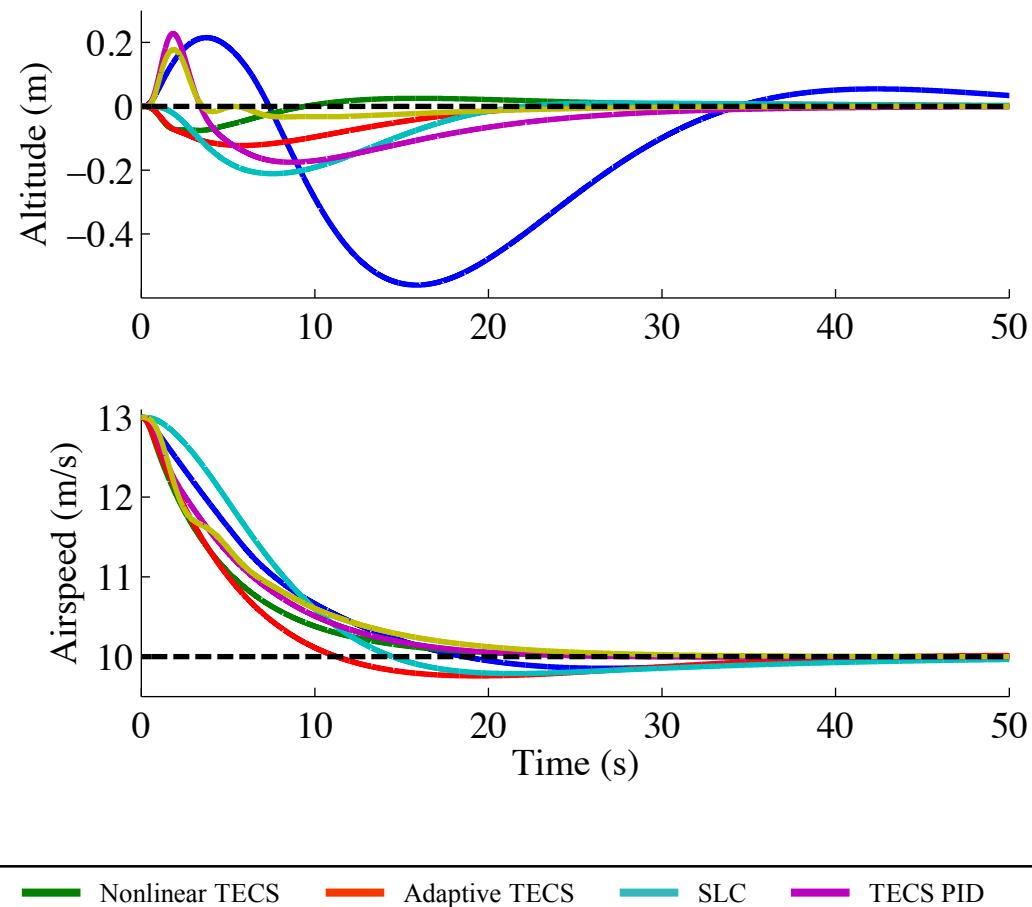
Step in Altitude, Constant Airspeed



Original TECS Nonlinear TECS Adaptive TECS SLC TECS PID ArduPilot PID

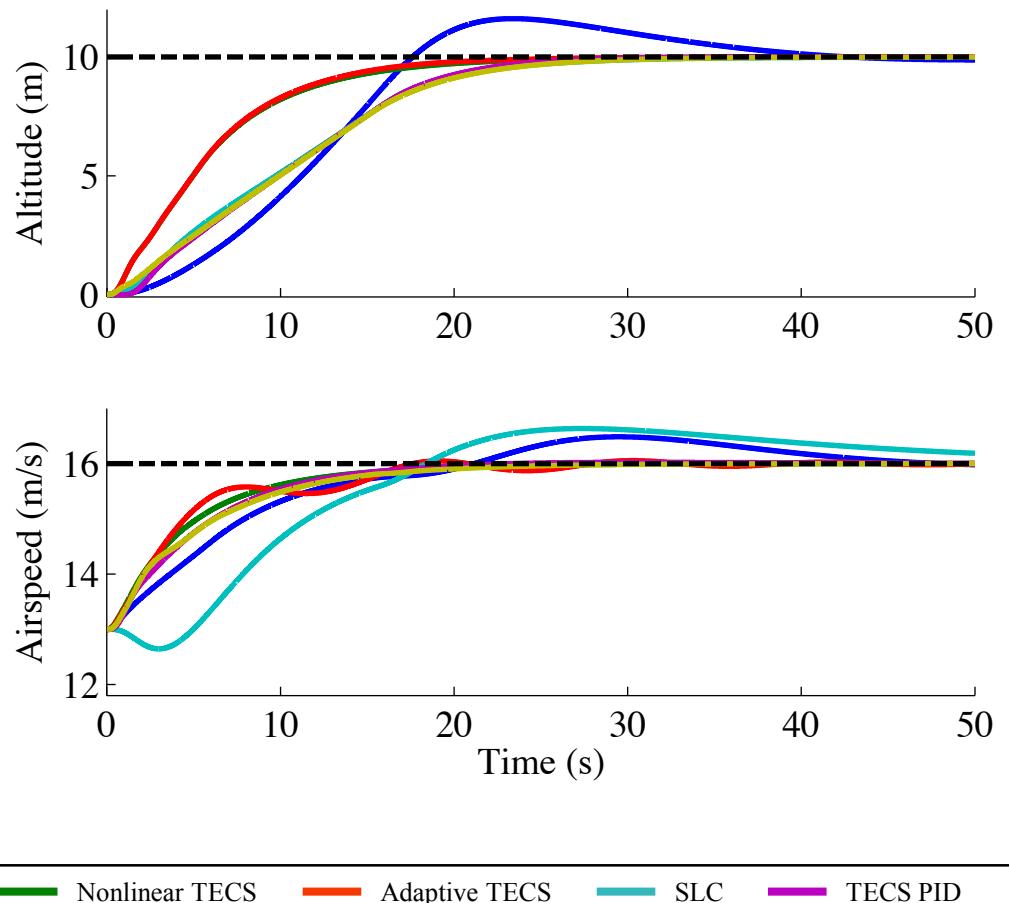
Total Energy Control

Step in Airspeed, Constant Altitude



Total Energy Control

Step in Altitude and Airspeed



Total Energy Control

- Observations
 - TECS seems to work better than successive loop closure.
 - Removes needs for different flight modes.
 - Nonlinear TECS seems to better, but the Ardupilot controller works very well.

Outline

- Successive Loop Closure
- Total Energy Control
- LQR Control

LQR Control

Augment the States with an Integrator.

Given the state space system

$$\dot{x} = Ax + Bu$$

$$z = Hx$$

where z represents the controlled output. Suppose that the objective is to drive z to a reference signal z_r . Augment the state with the integrator

$$x_I = \int_{-\infty}^t (z(\tau) - z_r) d\tau.$$

Therefore

$$\dot{x}_I = Hx - z_r = H(x - x_r)$$

LQR Control

Augment the States with an Integrator. (cont)

Defining the augmented state as $\xi = (x^\top, x_I^\top)^\top$, results in the augmented state space equations

$$\dot{\xi} = \bar{A}\xi + \bar{B}u,$$

where

$$\bar{A} = \begin{pmatrix} A & 0 \\ H & 0 \end{pmatrix} \quad \bar{B} = \begin{pmatrix} B \\ 0 \end{pmatrix}.$$

LQR Control

Linear Quadratic Regulator Theory

Given the state space equation

$$\dot{x} = Ax + Bu$$

and the symmetric positive semi-definite matrix Q , and the symmetric positive definite matrix R , the LQR problem is to minimize the cost index

$$J(x_0) = \min_{u(t)} \int_0^\infty x^\top(\tau) Q x(\tau) + u^\top(\tau) R u(\tau) d\tau.$$

If (A, B) is controllable, and $(A, Q^{1/2})$ is observable, then a unique optimal control exists and is given in linear feedback form as

$$u^*(t) = -K_{lqr}x(t).$$

LQR Control

Linear Quadratic Regulator Theory (cont)

The LQR gain is given by

$$K_{lqr} = R^{-1}B^\top P,$$

where P is the symmetric positive definite solution of the Algebraic Riccati Equation

$$PA + A^\top P + Q - PBR^{-1}B^\top P = 0.$$

LQR Control

Linear Quadratic Regulator Theory (cont)

It should be noted that K_{lqr} is the optimal feedback gains given Q and R . The controller is tuned by changing Q and R .

Typically we choose Q and R to be diagonal matrices

$$Q = \begin{pmatrix} q_1 & 0 & \dots & 0 \\ 0 & q_2 & \dots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & q_n \end{pmatrix} \quad R = \begin{pmatrix} r_1 & 0 & \dots & 0 \\ 0 & r_2 & \dots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & r_m \end{pmatrix},$$

where n is the number of states, m is the number of inputs, and $q_i \geq 0$ ensures Q is positive semi-definite, and $r_i > 0$ ensure R is positive definite.

LQR Control

Lateral Autopilot

As derived in Chapter 5, the state space equations for the lateral equation of motion are given by

$$\dot{x}_{lat} = A_{lat}x_{lat} + B_{lat}u_{lat},$$

where $x_{lat} = (v, p, r, \phi, \psi)^\top$ and $u_{lat} = (\delta_a, \delta_r)^\top$.

The objective of the lateral autopilot is to drive course χ to commanded course χ_c . Therefore, we augment the state with

$$x_I = \int (\chi - \chi_c) dt.$$

Since $\chi \approx \psi$, we approximate x_I as

$$x_I = \int (H_{lat}x_{lat} - \chi_c) dt,$$

where $H_{lat} = (0, 0, 0, 0, 1)$.

LQR Control

Lateral Autopilot (cont)

The augmented lateral state equations are therefore

$$\dot{\xi}_{lat} = \bar{A}_{lat}\xi_{lat} + \bar{B}_{lat}u_{lat},$$

where

$$\xi_{lat} = (\tilde{v}, p, r, \phi, \tilde{\chi}, \int \tilde{\chi})^\top$$

$$\bar{A}_{lat} = \begin{pmatrix} A_{lat} & 0 \\ H_{lat} & 0 \end{pmatrix} \quad \bar{B}_{lat} = \begin{pmatrix} B_{lat} \\ 0 \end{pmatrix}$$

where

$$\tilde{v} = (V_a - V_a^c) * \sin \beta, \quad \tilde{\chi} = \chi - \chi^c$$

The LQR controller designed using

$$Q = \text{diag}([q_v, q_p, q_r, q_\phi, q_\chi, q_I])$$

$$R = \text{diag}([r_{\delta_a}, r_{\delta_r}]).$$

Hints: Since the goal is to drive $\tilde{\chi}$ and $\int \tilde{\chi}$ to zero, it is best to place low weights on v , p , and r relative to the other variables.

LQR Control

Longitudinal Autopilot

As derived in Chapter 5, the state space equations for the longitudinal equations of motion are given by

$$\dot{x}_{lon} = A_{lon}x_{lon} + B_{lon}u_{lon},$$

where $x_{lon} = (u, w, q, \theta, h)^\top$ and $u_{lat} = (\delta_e, \delta_t)^\top$.

The objective of the longitudinal autopilot is to drive altitude h to commanded altitude h^c , and airspeed V_a to commanded airspeed V_a^c . Therefore, we augment the state with

$$x_I = \begin{pmatrix} \int(h - h^c)dt \\ \int(V_a - V_a^c)dt \end{pmatrix} = \int \left(H_{lon}x_{lon} - \begin{pmatrix} h^c \\ V_a^c \end{pmatrix} \right) dt.$$

Therefore

$$\dot{x}_I = H_{lon}x_{lon} - \begin{pmatrix} h^c \\ V_a^c \end{pmatrix} = H_{lon}(x_{lon} - x^c)$$

where

$$H_{lon} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ \frac{1}{V_a} & \frac{1}{V_a} & 0 & 0 & 0 \end{pmatrix}.$$

LQR Control

Longitudinal Autopilot (cont)

The augmented longitudinal state equations are therefore

$$\dot{\xi}_{lon} = \bar{A}_{lon}\xi_{lon} + \bar{B}_{lon}u_{lon},$$

where

$$\xi_{lon} = (\tilde{u}, \tilde{w}, q, \theta, \tilde{h}, \int \tilde{h}, \int \tilde{V}_a)^\top$$

$$\bar{A}_{lon} = \begin{pmatrix} A_{lon} & 0 \\ H_{lon} & 0 \end{pmatrix} \quad \bar{B}_{lon} = \begin{pmatrix} B_{lon} \\ 0 \end{pmatrix}$$

where

$$\tilde{u} = (V_a - V_a^c) \cos \alpha, \quad \tilde{w} = (V_a - V_a^c) \sin \alpha, \quad \tilde{h} = h - h^c, \quad \tilde{V}_a = V_a - V_a^c$$

The LQR controller designed using

$$Q = \text{diag}([q_u, q_w, q_q, q_\theta, q_h, q_{Ih}, q_{IV_a}])$$

$$R = \text{diag}([r_{\delta_e}, r_{\delta_t}]).$$

Hints: Since θ need to be allowed to be non-zero, and since pitch rate does not necessarily need to be small, the weights q_θ and q_q should be small.

Python Code

```

import sys
from numpy import array, sin, cos, radians, concatenate, zeros, diag
from scipy.linalg import solve_continuous_are, inv
sys.path.append('..')
import parameters.control_parameters as AP
#from tools.transfer_function import TransferFunction
from tools.wrap import wrap
import chap5.model_coef as M
from message_types.msg_state import MsgState
from message_types.msg_delta import MsgDelta

class Autopilot:
    def __init__(self, ts_control):
        self.Ts = ts_control
        # initialize integrators and delay variables
        self.integratorCourse = 0
        self.integratorAltitude = 0
        self.integratorAirspeed = 0
        self.errorCourseD1 = 0
        self.errorAltitudeD1 = 0
        self.errorAirspeedD1 = 0
        # compute LQR gains
        CrLat = array([[0, 0, 0, 0, 1.0]])
        AAlat = concatenate((
            concatenate((M.A_lat, zeros((5,1))), axis=1),
            concatenate((CrLat, zeros((1,1))), axis=1)),
            axis=0)
        BBlat = concatenate((M.B_lat, zeros((1,2))), axis=0)
        Qlat = diag([.001, .01, .1, 100, 1, 100]) # v, p, r, phi, chi, intChi
        Rlat = diag([1, 1]) # a, r
        Plat = solve_continuous_are(AAlat, BBlat, Qlat, Rlat)
        self.Klat = inv(Rlat) @ BBlat.T @ Plat
        #CrLon = array([[0, 0, 0, 0, 1.0], [1.0, 0, 0, 0, 0]])
        CrLon = array([[0, 0, 0, 0, 1.0], [1/AP.Va0, 1/AP.Va0, 0, 0, 0]])
        AAlon = concatenate((
            concatenate((M.A_lon, zeros((5,2))), axis=1),
            concatenate((CrLon, zeros((2,2))), axis=1)),
            axis=0)
        BBlon = concatenate((M.B_lon, zeros((2,2))), axis=0)
        Qlon = diag([10, 10, .001, .01, 10, 100, 100]) # u, w, q, theta, h, intH
        Rlon = diag([1, 1]) # e, t
        Plon = solve_continuous_are(AAlon, BBlon, Qlon, Rlon)
        self.Klon = inv(Rlon) @ BBlon.T @ Plon
        self.commanded_state = MsgState()

    def update(self, cmd, state):
        # lateral autopilot
        errorAirspeed = state.Va - cmd.airspeed_command
        chi_c = wrap(cmd.course_command, state.chi)
        errorCourse = saturate(state.chi - chi_c, -radians(15), radians(15))
        self.integratorCourse = self.integratorCourse + (self.Ts/2) * (errorCourse +
        self.errorCourseD1 = errorCourse
        xLat = array([[errorAirspeed * sin(state.beta)], # v
                     [state.p],
                     [state.r],
                     [state.phi],
                     [errorCourse],
                     [self.integratorCourse]])
        tmp = -self.Klat @ xLat
        delta_a = saturate(tmp.item(0), -radians(30), radians(30))
        delta_r = saturate(tmp.item(1), -radians(30), radians(30))

        # longitudinal autopilot
        altitude_c = saturate(cmd.altitude_command,
                               state.altitude - 0.2*AP.altitude_zone,
                               state.altitude + 0.2*AP.altitude_zone)
        errorAltitude = state.altitude - altitude_c
        self.integratorAltitude = self.integratorAltitude \
            + (self.Ts/2) * (errorAltitude + self.errorAltitudeD
        self.errorAltitudeD1 = errorAltitude
        self.integratorAirspeed = self.integratorAirspeed \
            + (self.Ts/2) * (errorAirspeed + self.errorAirspeedD
        self.errorAirspeedD1 = errorAirspeed
        xLon = array([[errorAirspeed * cos(state.alpha)], # u
                     [errorAirspeed * sin(state.alpha)], # w
                     [state.q],
                     [state.theta],
                     [errorAltitude],
                     [self.integratorAltitude],
                     [self.integratorAirspeed]])
        tmp = -self.Klon @ xLon
        delta_e = saturate(tmp.item(0), -radians(30), radians(30))
        delta_t = saturate(tmp.item(1), 0.0, 1.0)

        # construct control outputs and commanded states
        delta = MsgDelta(elevator=delta_e,
                         aileron=delta_a,
                         rudder=delta_r,
                         throttle=delta_t)
        self.commanded_state.altitude = cmd.altitude_command
        self.commanded_state.Va = cmd.airspeed_command

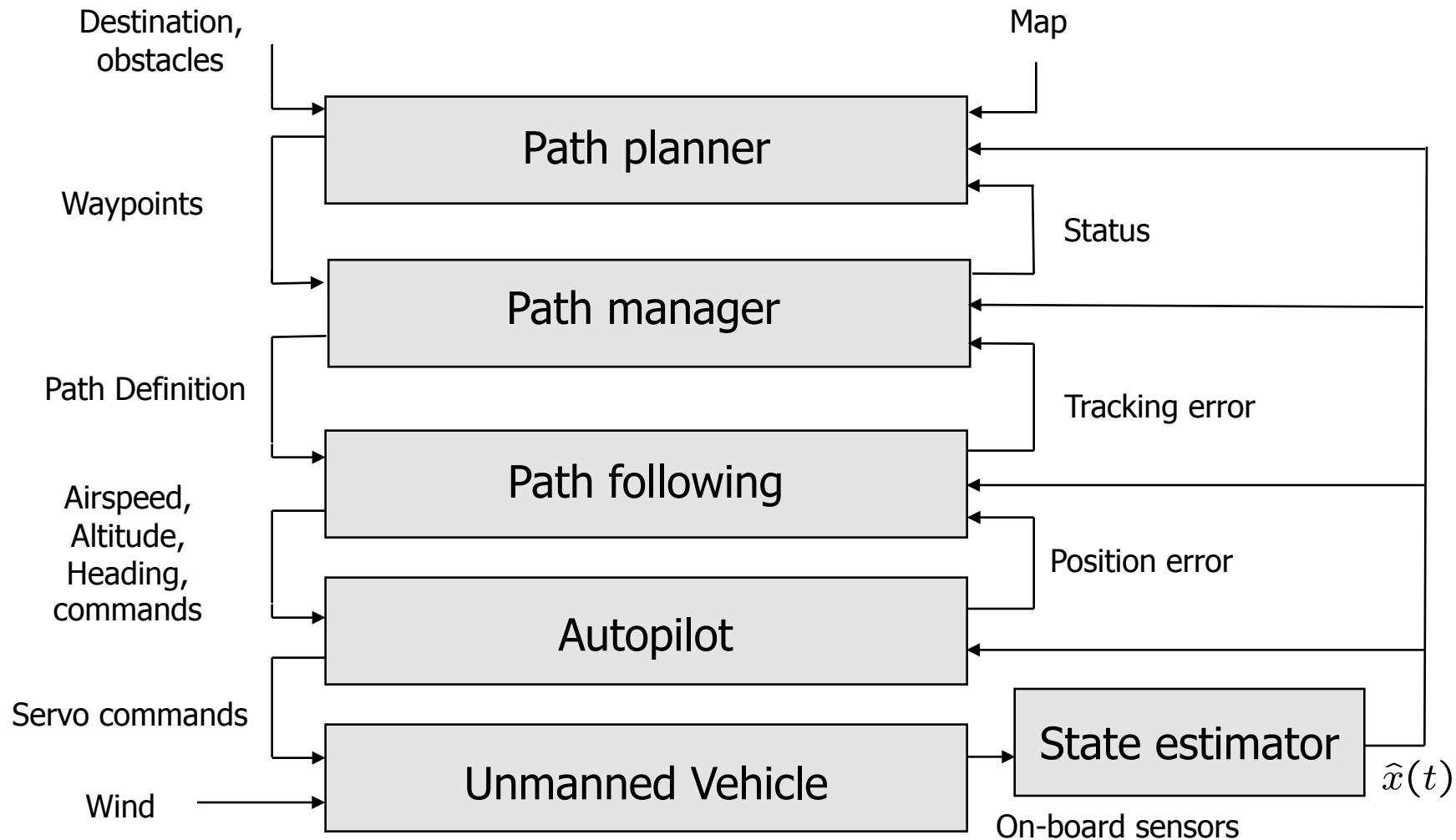
```



Chapter 7

Sensors

Architecture

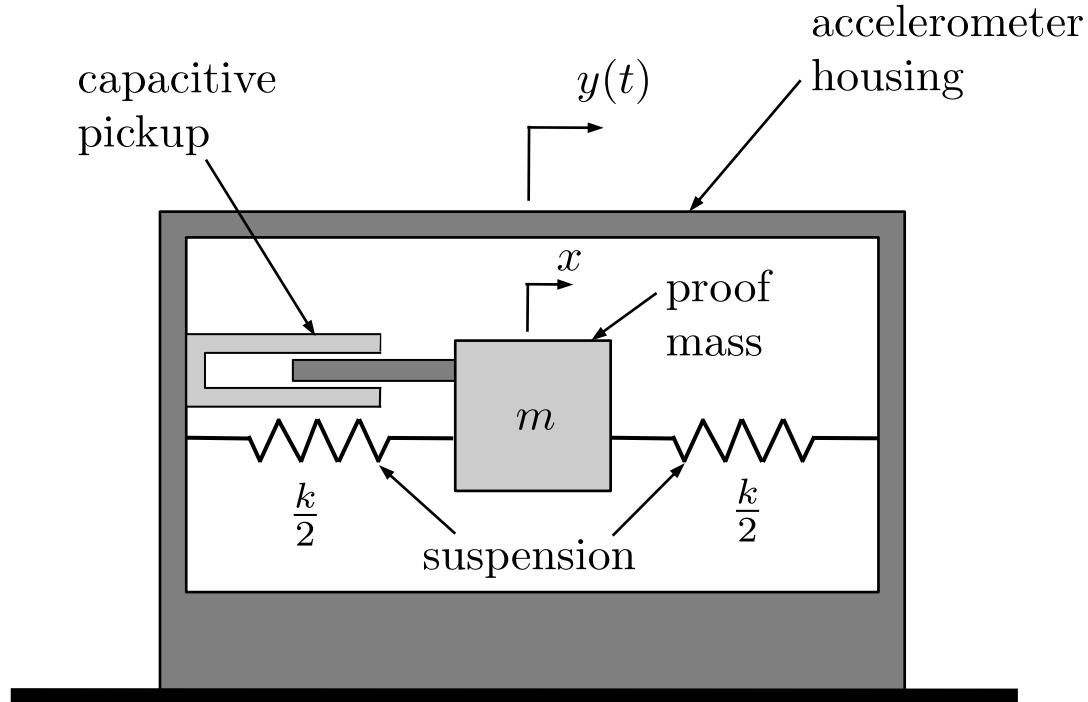


Sensors for MAVs

- The following types of sensors are commonly used for guidance and control of MAVs
 - accelerometers
 - rate gyros
 - pressure sensors
 - magnetometers (digital compasses)
 - GPS

MEMS Accelerometer

Newton's 2nd law gives



$$m\ddot{x} = k(y - x)$$

Note that the acceleration of the proof mass proportional to deflection of the suspension

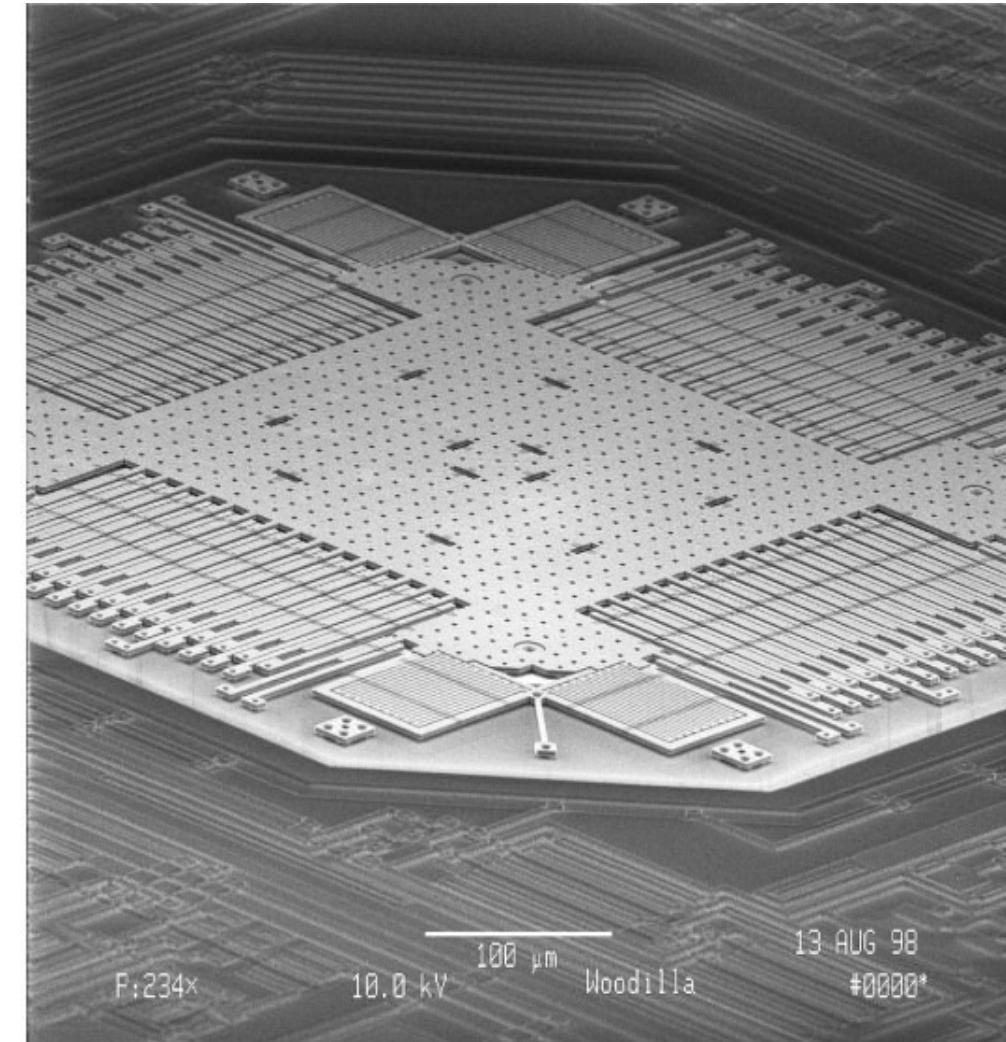
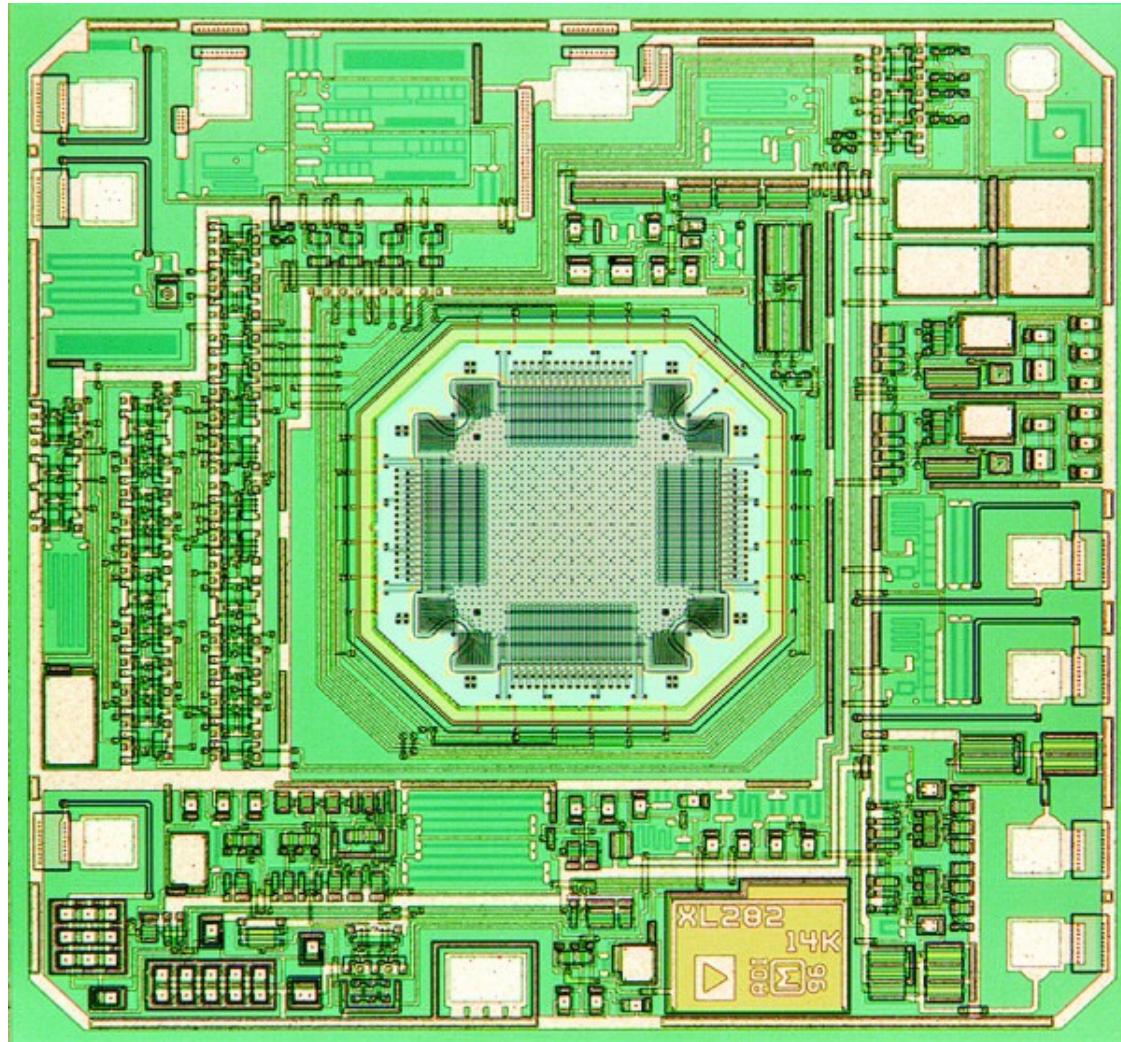
Taking the Laplace transform gives

$$\begin{aligned} X(s) &= \frac{1}{\frac{m}{k}s^2 + 1} Y(s) \\ \implies s^2 X(s) &= \frac{1}{\frac{m}{k}s^2 + 1} (s^2 Y(s)) \\ \implies A_X(s) &= \frac{1}{\frac{m}{k}s^2 + 1} A_Y(s) \end{aligned}$$

Accelerometers also have bias and zero mean Gaussian noise. Therefore, the sensor model is

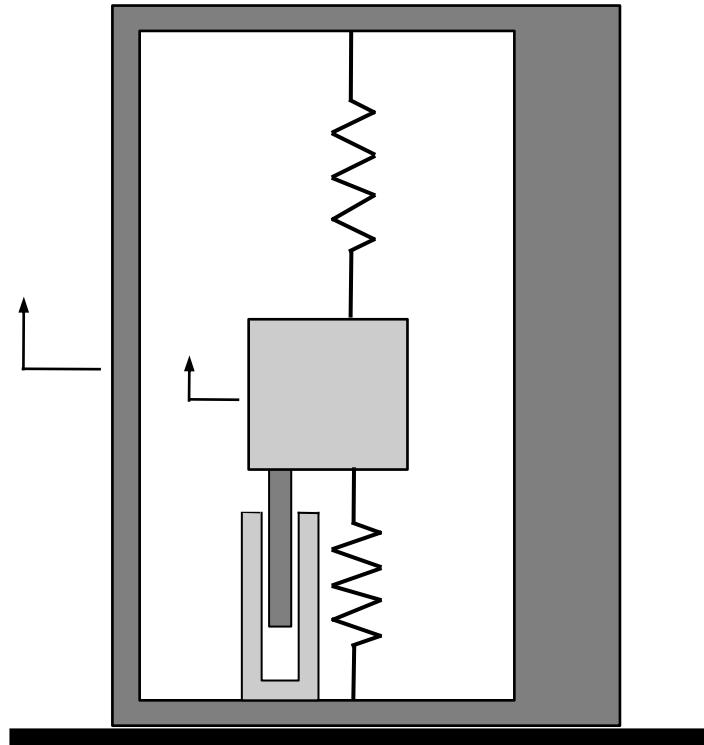
$$y_{\text{accel}} = k_{\text{accel}} a + \beta_{\text{accel}} + \eta'_{\text{accel}}.$$

MEMS Accelerometer



Acceleration Measurement

Tricky concept: Measured acceleration is the total acceleration of the accelerometer casing minus the acceleration of gravity



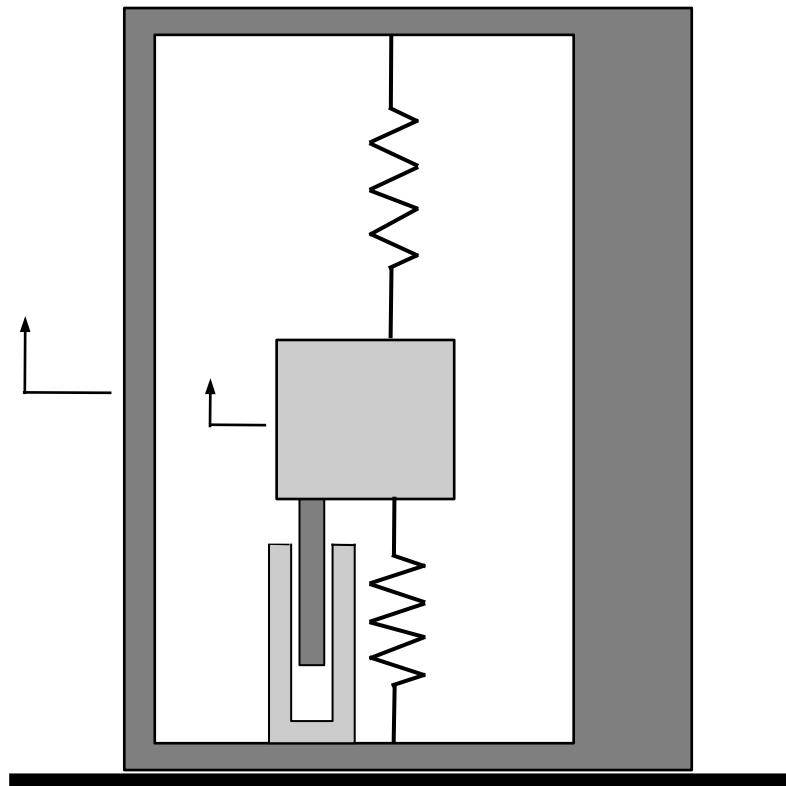
$$\mathbf{a} = \frac{1}{m} (\mathbf{F}_{\text{total}} - \mathbf{F}_{\text{gravity}})$$

Example: Set the accelerometer on a table top.
What does it measure?

Accelerometers measure components of linear, coriolis, and externally applied acceleration. They do not measure gravity, since both the proof mass and the casing are acted on by gravity in exactly the same way

Acceleration Measurement

Said another way, accelerometers measure *specific force*, which is defined as the sum of the non-gravitational forces divided by the mass

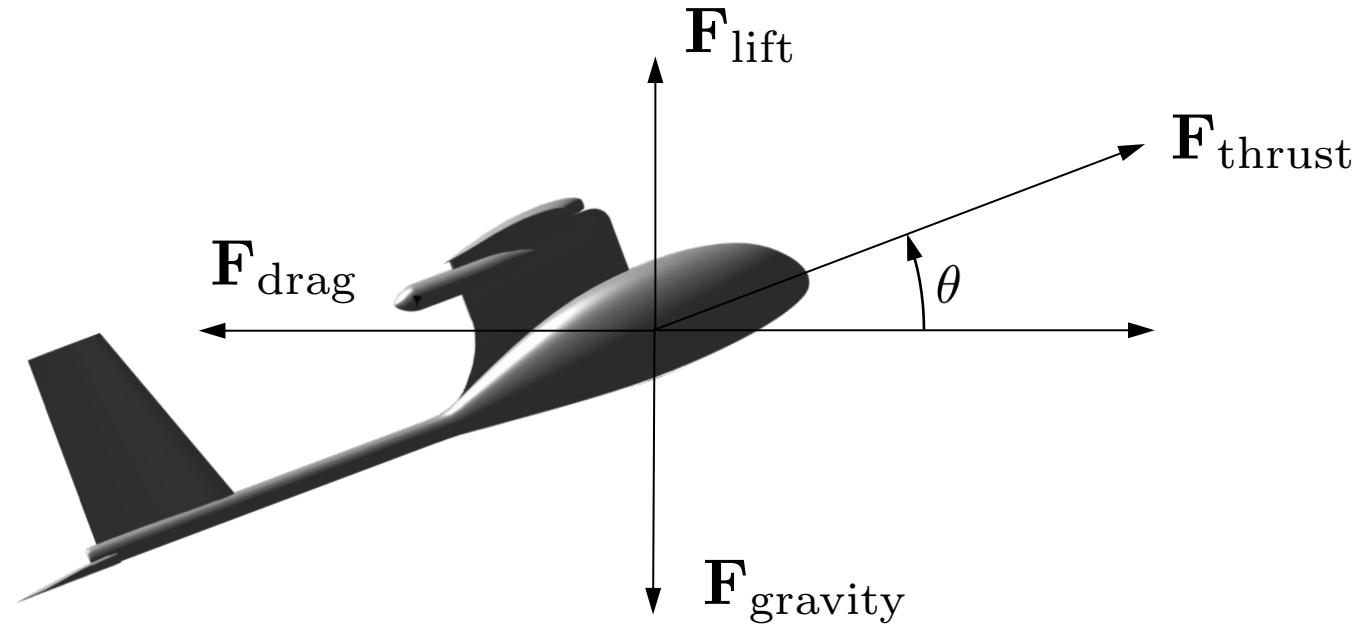


$$\begin{aligned}\mathbf{a}_{\text{measured}} &= \frac{1}{m} \left(\sum \mathbf{F}_{\text{non-gravitational}} \right) \\ &= \frac{1}{m} \left(\sum \mathbf{F} - \mathbf{F}_{\text{gravitational}} \right)\end{aligned}$$

Example: Set the accelerometer on a table top.
What does it measure?

Hint: (Draw FBD of accel housing)

Acceleration on Fixed-Wing Aircraft



$$\begin{aligned} \mathbf{a}_{\text{measured}} &= \frac{1}{m} (\mathbf{F}_{\text{total}} - \mathbf{F}_{\text{gravity}}) \\ &= \frac{1}{m} ((\mathbf{F}_{\text{lift}} + \mathbf{F}_{\text{drag}} + \mathbf{F}_{\text{thrust}} + \mathbf{F}_{\text{gravity}}) - \mathbf{F}_{\text{gravity}}) \\ &= \frac{1}{m} (\mathbf{F}_{\text{lift}} + \mathbf{F}_{\text{drag}} + \mathbf{F}_{\text{thrust}}) \end{aligned}$$

Acceleration on Fixed-Wing Aircraft

Recall from Chapter 3, that

$$m \left(\frac{d\mathbf{v}}{dt_b} + \boldsymbol{\omega}_{b/i} \times \mathbf{v} \right) = \mathbf{F}_{\text{total}}.$$

Using the expression

$$\mathbf{a}_{\text{measured}} = \frac{1}{m} (\mathbf{F}_{\text{total}} - \mathbf{F}_{\text{gravity}}),$$

the output of the accelerometer can be expressed as

$$\mathbf{a}_{\text{measured}} = \frac{d\mathbf{v}}{dt_b} + \boldsymbol{\omega}_{b/i} \times \mathbf{v} - \frac{1}{m} \mathbf{F}_{\text{gravity}}.$$

Expressing this relationship in the body frame gives

$$a_x = \dot{u} + qw - rv + g \sin \theta$$

$$a_y = \dot{v} + ru - pw - g \cos \theta \sin \phi$$

$$a_z = \dot{w} + pv - qu - g \cos \theta \cos \phi$$

Accelerometer Models

$$\begin{aligned}y_{\text{accel},x} &= \dot{u} + qw - rv + g \sin \theta + \eta_{\text{accel},x} \\y_{\text{accel},y} &= \dot{v} + ru - pw - g \cos \theta \sin \phi + \eta_{\text{accel},y} \\y_{\text{accel},z} &= \dot{w} + pv - qu - g \cos \theta \cos \phi + \eta_{\text{accel},z}\end{aligned}$$

or

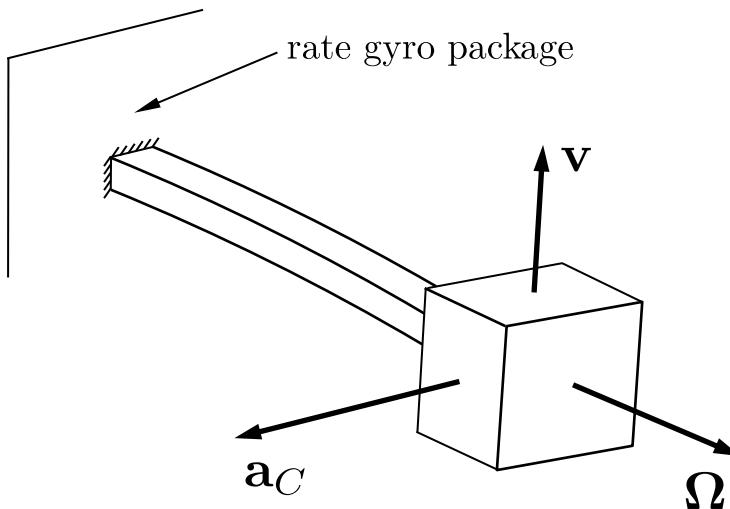
$$\begin{aligned}y_{\text{accel},x} &= \frac{\rho V_a^2 S}{2m} \left[C_X(\alpha) + C_{X_q}(\alpha) \frac{\bar{c}q}{2V_a} + C_{X_{\delta_e}}(\alpha) \delta_e \right] + T_p(\delta_t, V_a) + \eta_{\text{accel},x} \\y_{\text{accel},y} &= \frac{\rho V_a^2 S}{2m} \left[C_{Y_0} + C_{Y_\beta} \beta + C_{Y_p} \frac{bp}{2V_a} + C_{Y_r} \frac{br}{2V_a} + C_{Y_{\delta_a}} \delta_a + C_{Y_{\delta_r}} \delta_r \right] + \eta_{\text{accel},y} \\y_{\text{accel},z} &= \frac{\rho V_a^2 S}{2m} \left[C_Z(\alpha) + C_{Z_q}(\alpha) \frac{\bar{c}q}{2V_a} + C_{Z_{\delta_e}}(\alpha) \delta_e \right] + \eta_{\text{accel},z}.\end{aligned}$$

Accelerometer Models

or

$$\begin{aligned}y_{\text{accel},x} &= \frac{f_x}{m} + g \sin \theta + \eta_{\text{accel},x} \\y_{\text{accel},y} &= \frac{f_y}{m} - g \cos \theta \sin \phi + \eta_{\text{accel},y} \\y_{\text{accel},z} &= \frac{f_z}{m} - g \cos \theta \cos \phi + \eta_{\text{accel},z}\end{aligned}$$

MEMS Rate Gyro



Point translating on a rotating rigid body experiences a coriolis acceleration:

$$\mathbf{a}_C = 2\boldsymbol{\Omega} \times \mathbf{v}$$

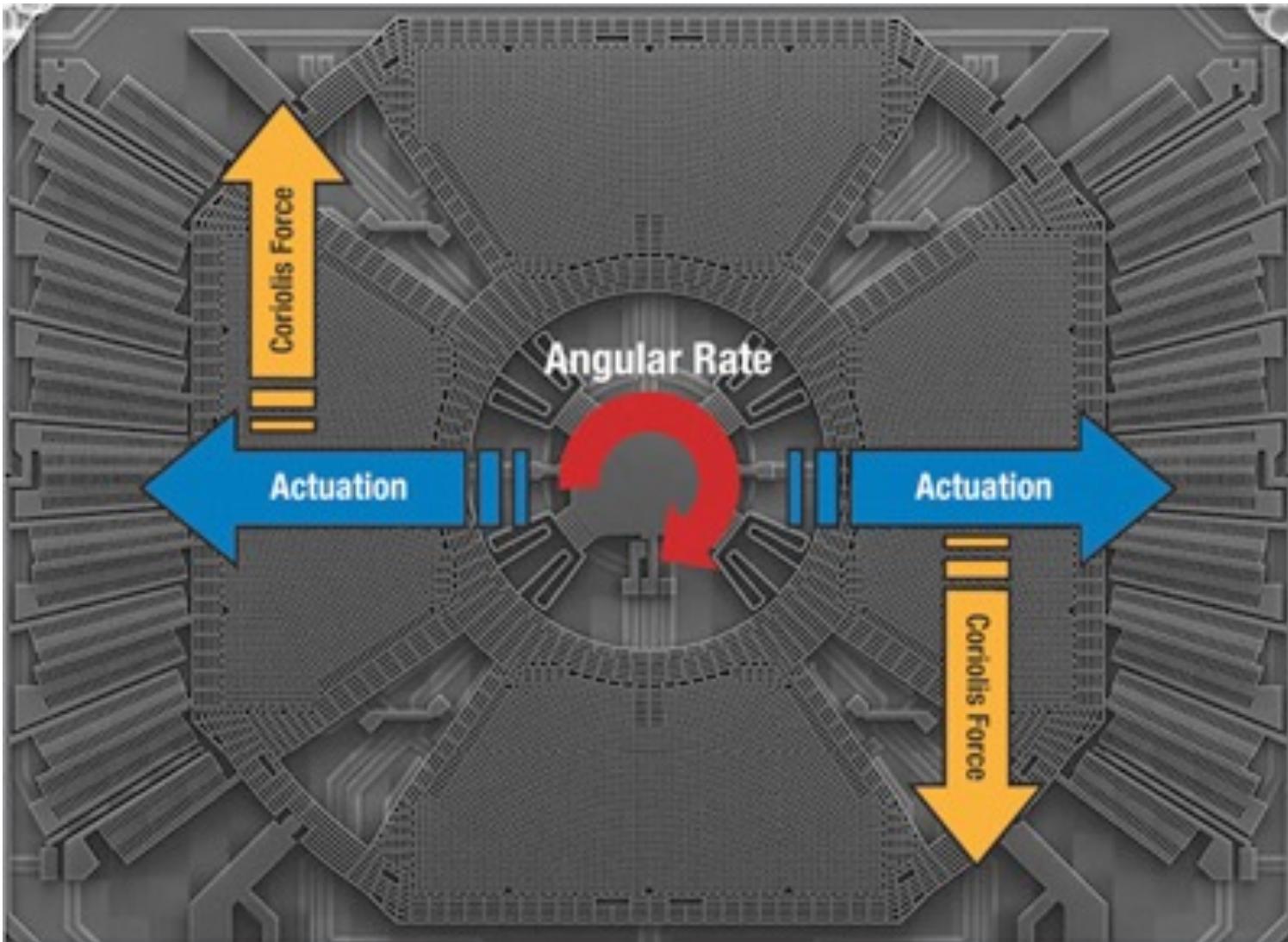
MEMS rate gyro – resonating proof mass:

$$|\mathbf{v}| = A\omega_n \sin(\omega_n t)$$

Sensor measures deflection of proof mass due to coriolis acceleration

$$\begin{aligned} V_{\text{gyro}} &= k_C |\mathbf{a}_C| \\ &= 2k_C |\boldsymbol{\Omega} \times \mathbf{v}| = 2k_C \Omega |\mathbf{v}| \\ &= 2k_C \Omega |A\omega_n \sin(\omega_n t)| = 2k_C A\omega_n \Omega \\ &= k_{\text{gyro}} \Omega, \quad \text{where } k_{\text{gyro}} = 2k_C A\omega_n \end{aligned}$$

MEMS Rate Gyro



Rate Gyro Model

The manufacturing process implies that rate gyros will have a drift term, as well as zero mean Gaussian noise:

$$y_{\text{gyro}} = k_{\text{gyro}} \Omega + \beta_{\text{gyro}} + \eta'_{\text{gyro}},$$

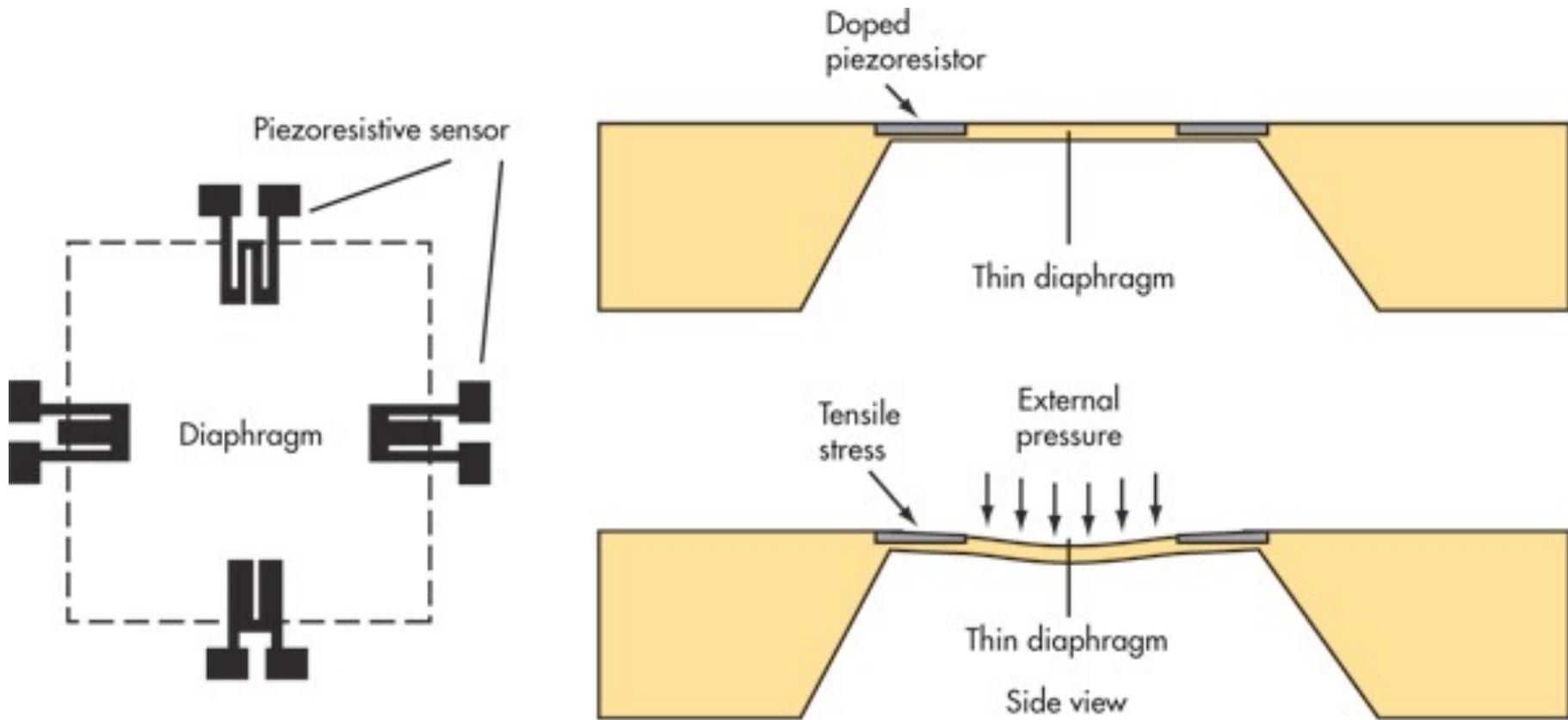
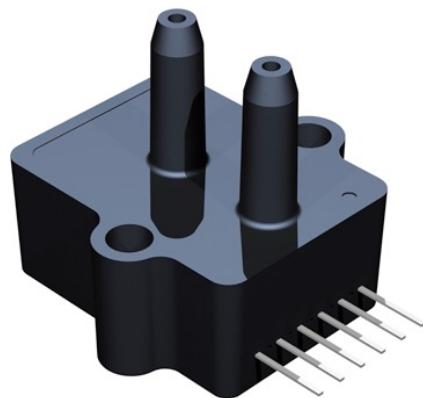
where Υ_{gyro} is in units of voltage. Calibrating to units of rad/s gives

$$y_{\text{gyro},x} = p + \beta_{\text{gyro},x} + \eta_{\text{gyro},x}$$

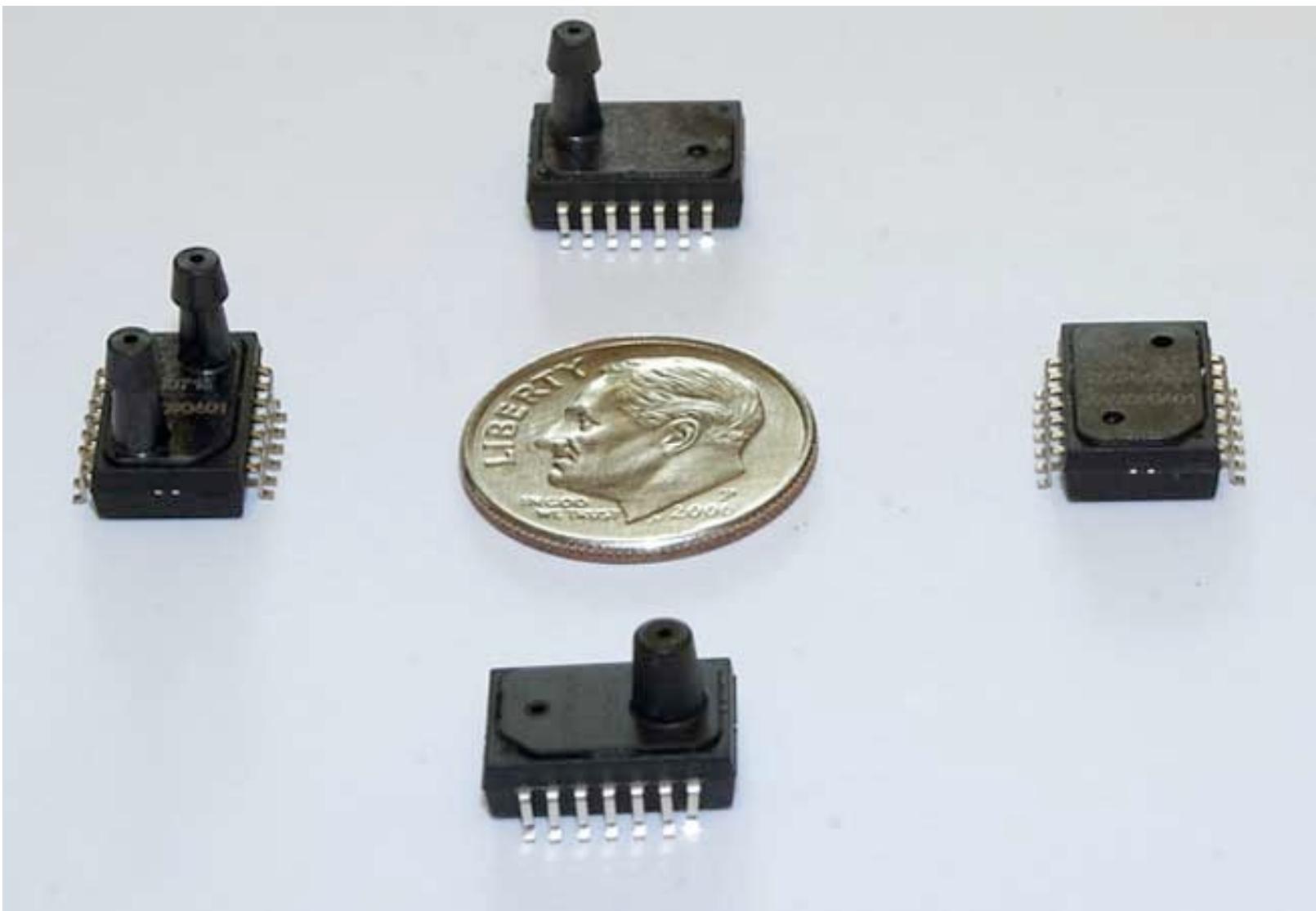
$$y_{\text{gyro},y} = q + \beta_{\text{gyro},y} + \eta_{\text{gyro},y}$$

$$y_{\text{gyro},z} = r + \beta_{\text{gyro},z} + \eta_{\text{gyro},z}$$

Pressure Measurement



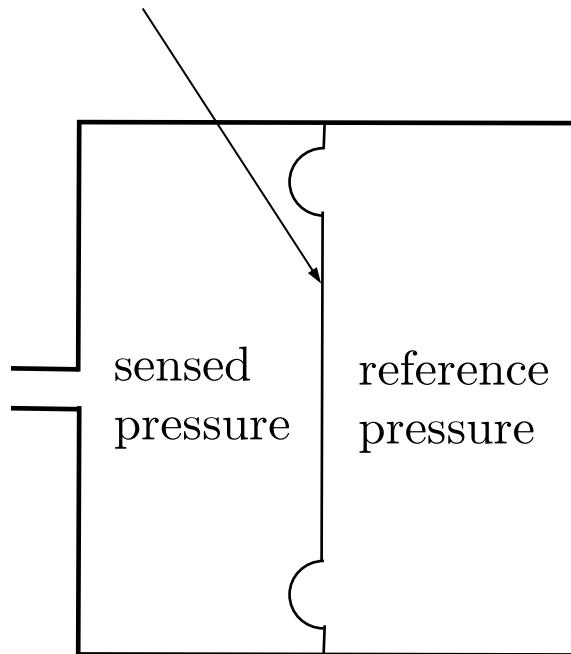
Pressure Measurement



Altitude Measurement

The basic equation of hydrostatics is

strain-sensing
diaphragm



$$P_2 - P_1 = \rho g(z_2 - z_1)$$

Using the ground as reference, and assuming constant air density gives

$$\begin{aligned} P - P_{\text{ground}} &= -\rho g(h - h_{\text{ground}}) \\ &= -\rho g h_{\text{AGL}} \end{aligned}$$

Below 11,000 m, can use barometric formula:

$$P = P_0 \left[\frac{T_0}{T_0 + L_0 h_{\text{ASL}}} \right]^{\frac{g M}{R L_0}},$$

where P_0 : standard pressure at sea level

T_0 : standard temperature at sea level

L_0 : rate of temperature decrease

g : gravitational constant

R : universal gas constant for air

M : standard molar mass of atmospheric air,

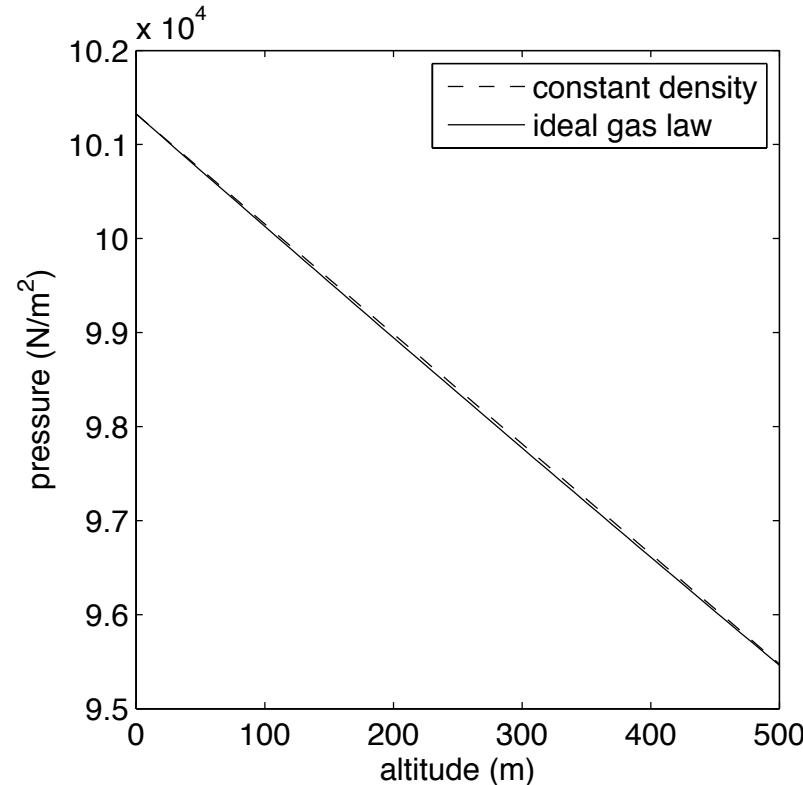
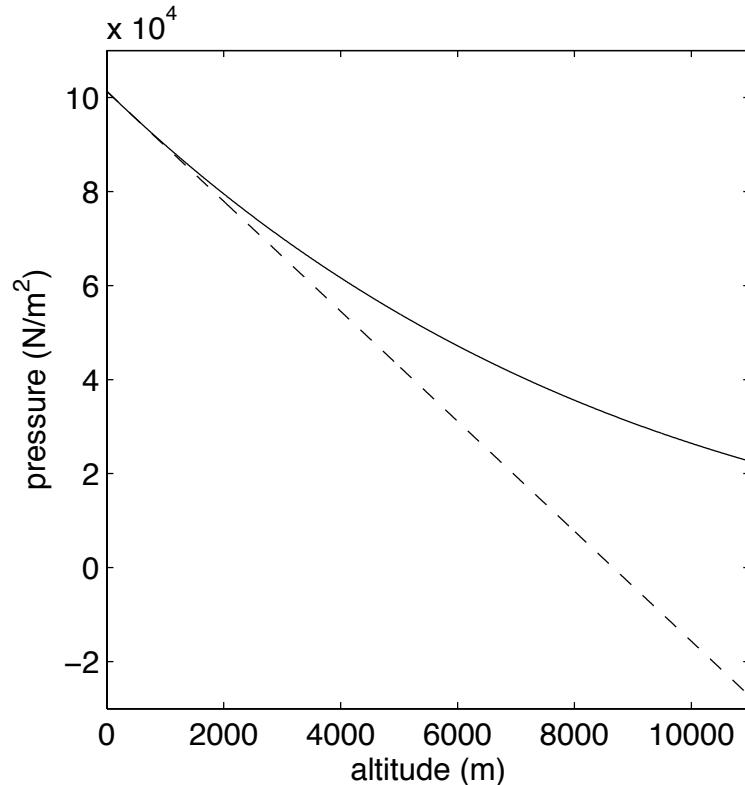
which takes into account change in density with altitude and temperature.

Altitude Measurement

We usually assume density is constant:

$$\begin{aligned}y_{\text{abs pres}} &= (P_{\text{ground}} - P) + \beta_{\text{abs pres}} + \eta_{\text{abs pres}} \\&= \rho gh_{\text{AGL}} + \beta_{\text{abs pres}} + \eta_{\text{abs pres}}\end{aligned}$$

Is this valid?



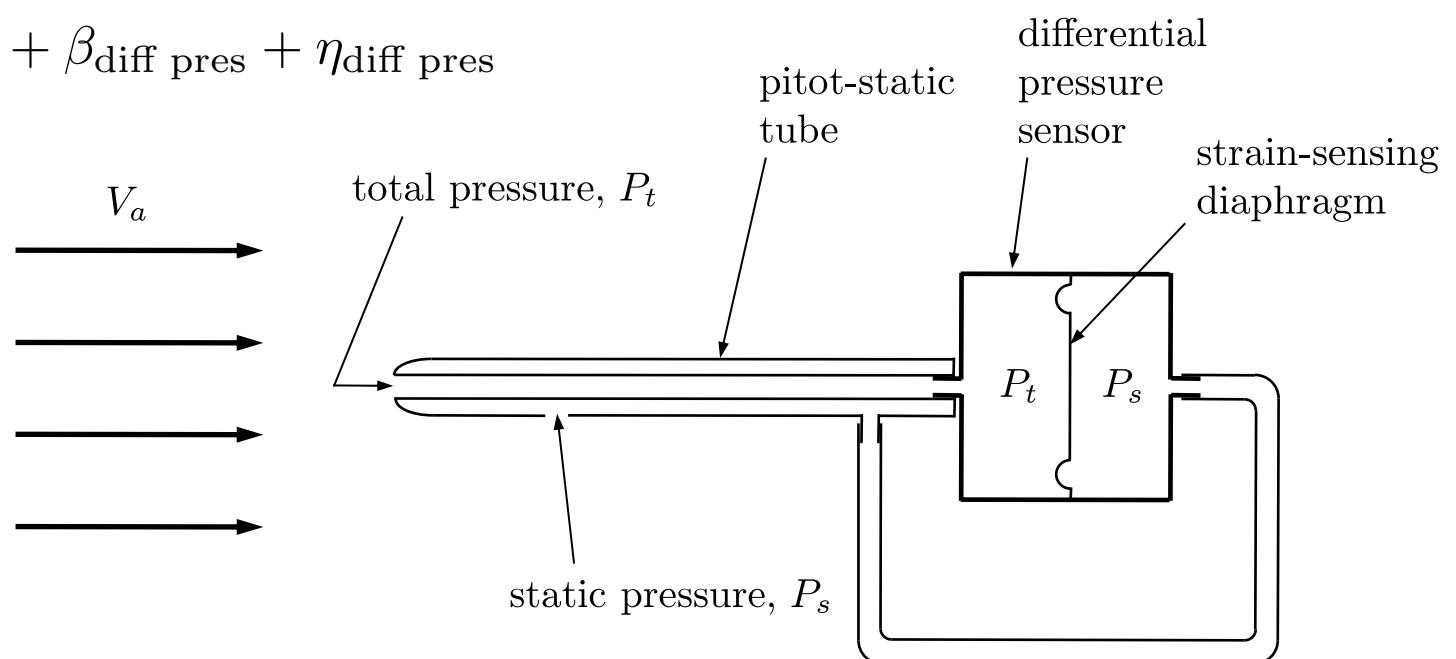
Airspeed Measurement

From Bernoulli's equation:

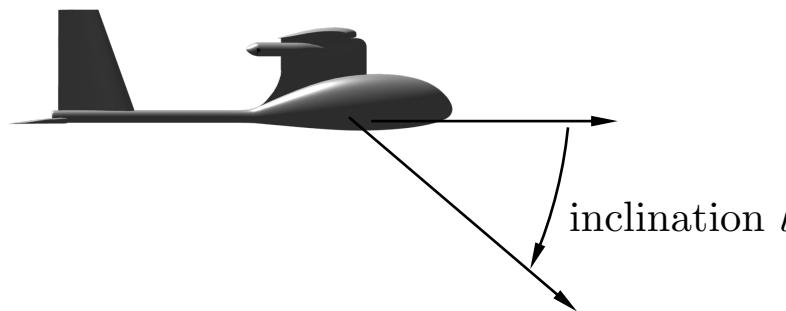
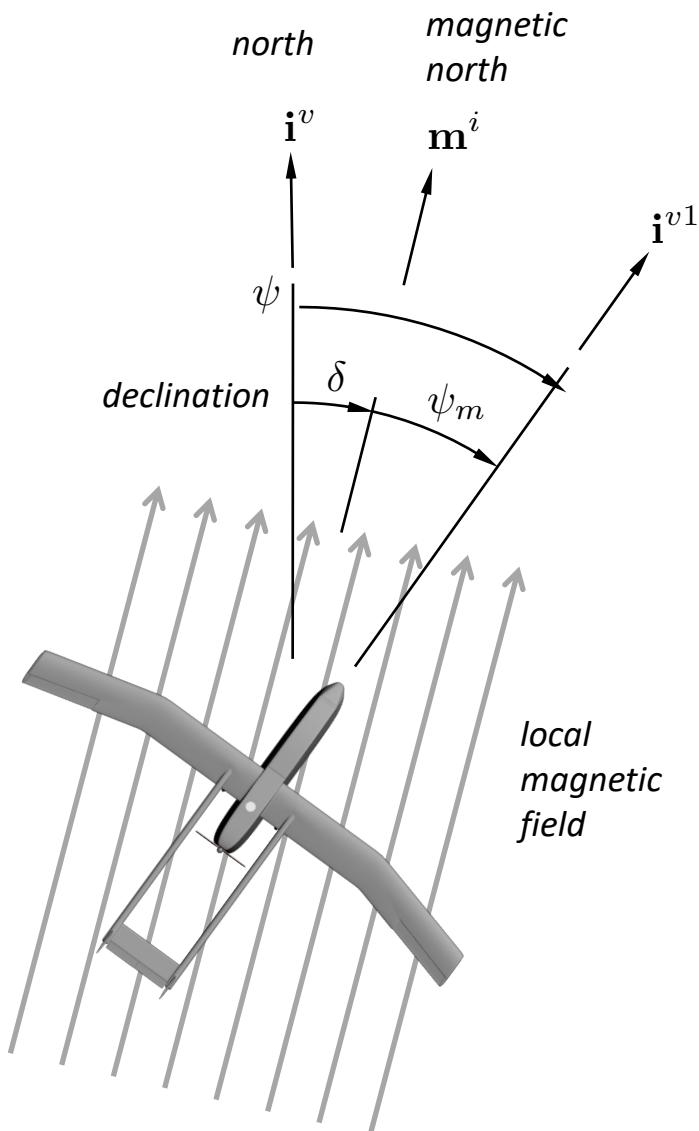
$$P_t = P_s + \frac{\rho V_a^2}{2} \quad \text{or} \quad \frac{\rho V_a^2}{2} = P_t - P_s$$

Pitot-static pressure sensor measures dynamic pressure:

$$y_{\text{diff pres}} = \frac{\rho V_a^2}{2} + \beta_{\text{diff pres}} + \eta_{\text{diff pres}}$$



Magnetometer



Let $\mathbf{e}_1 = (1, 0, 0)^\top$ be a unit vector pointing north.

Let $R(0, -\iota, \delta)$ be the rotation matrix from the magnetic frame to the inertial frame. Then

$$\mathbf{m}^i = R^\top(0, -\iota, \delta)\mathbf{e}_1$$

is the unit vector that points in the direction of the magetic field, resolved in the inertial frame.

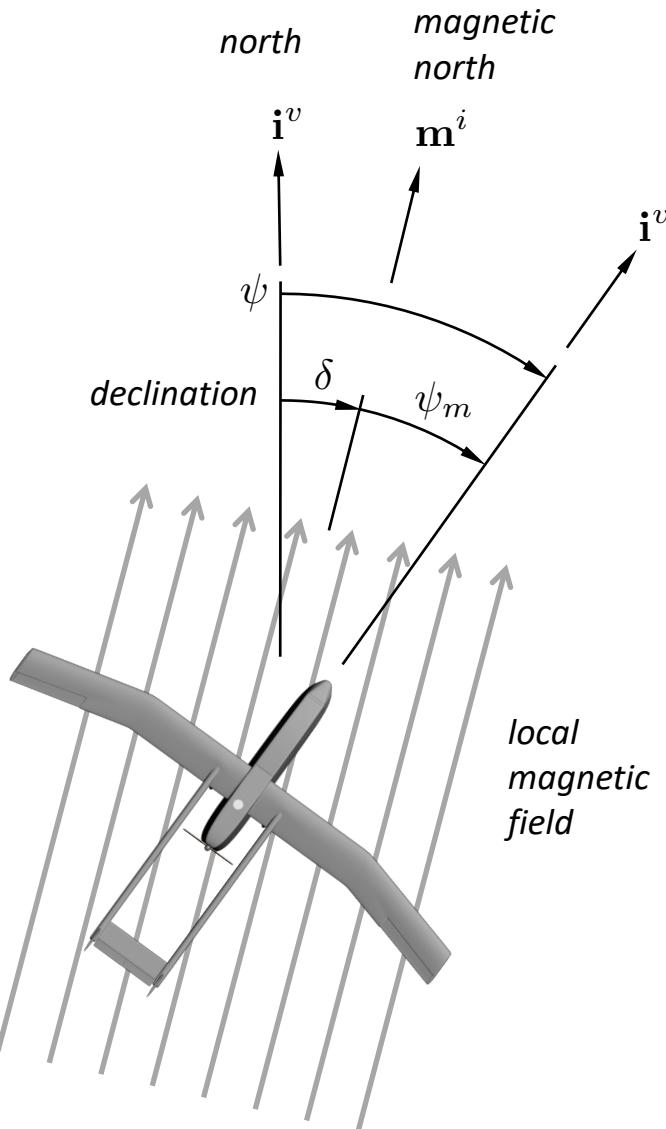
The normalized magnetic field measured in the body frame is

$$\mathbf{m}^b = R_b^{i\top} \mathbf{m}^i.$$

The measurement is given by

$$\mathbf{y}_{mag} = \mathbf{m}^b + \boldsymbol{\eta}.$$

Digital Compass



Heading is sum of magnetic declination angle and magnetic heading

$$\psi = \delta + \psi_m$$

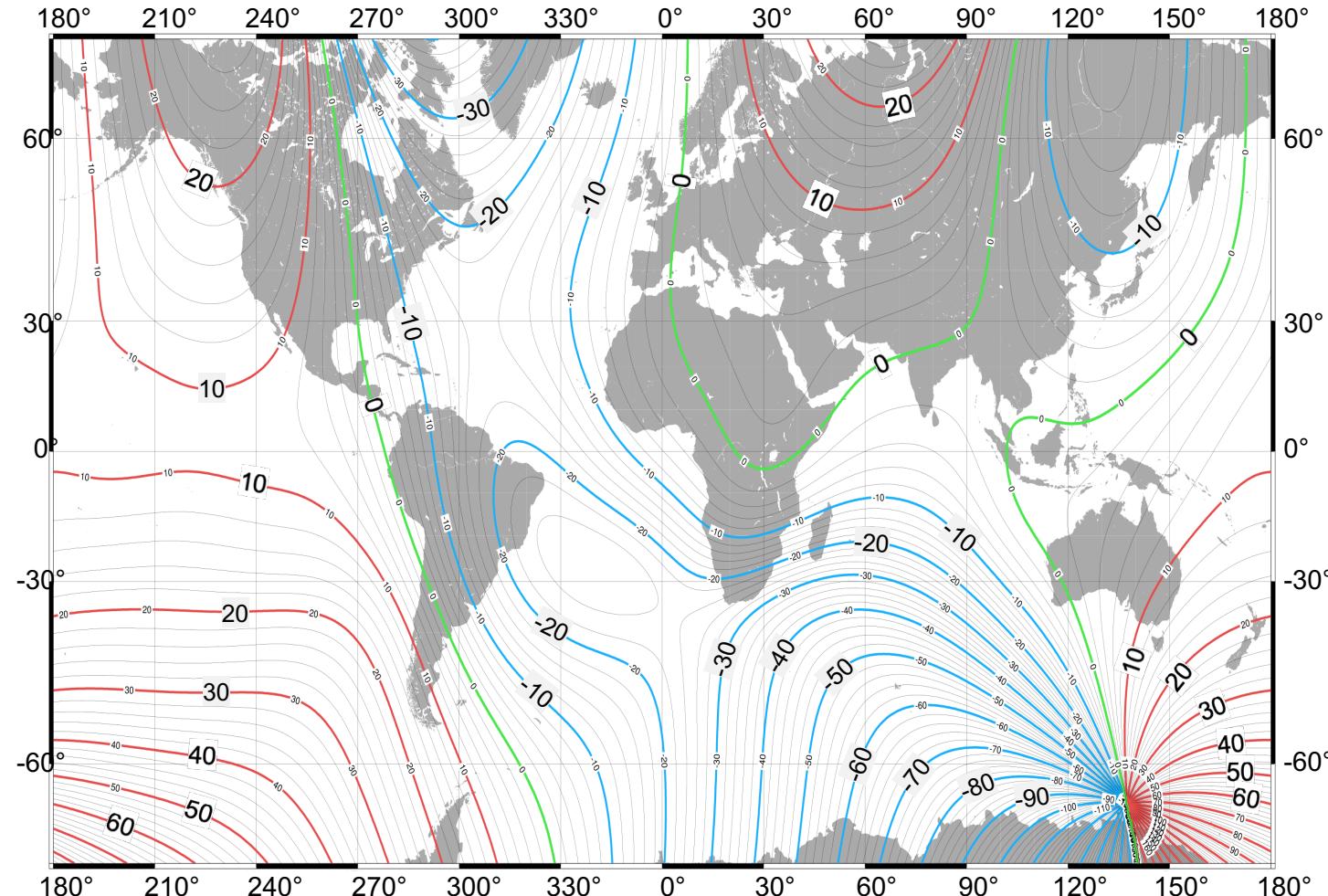
Magnetic heading determined from measurements of body-frame components of magnetic field projected onto horizontal plane

$$\begin{aligned}\mathbf{m}^{v1} &= \begin{pmatrix} m_x^{v1} \\ m_y^{v1} \\ m_z^{v1} \end{pmatrix} = \mathcal{R}_b^{v1}(\phi, \theta) \mathbf{m}^b \\ &= \mathcal{R}_{v2}^{v1}(\theta) \mathcal{R}_b^{v2}(\phi) \mathbf{m}^b \\ \begin{pmatrix} m_x^{v1} \\ m_y^{v1} \\ m_z^{v1} \end{pmatrix} &= \begin{pmatrix} c_\theta & s_\theta s_\phi & s_\theta c_\phi \\ 0 & c_\phi & -s_\phi \\ -s_\theta & c_\theta s_\phi & c_\theta c_\phi \end{pmatrix} \mathbf{m}^b\end{aligned}$$

Solving for heading gives

$$\psi_m = -\text{atan2}(m_y^{v1}, m_x^{v1}).$$

Magnetic Declination Variation

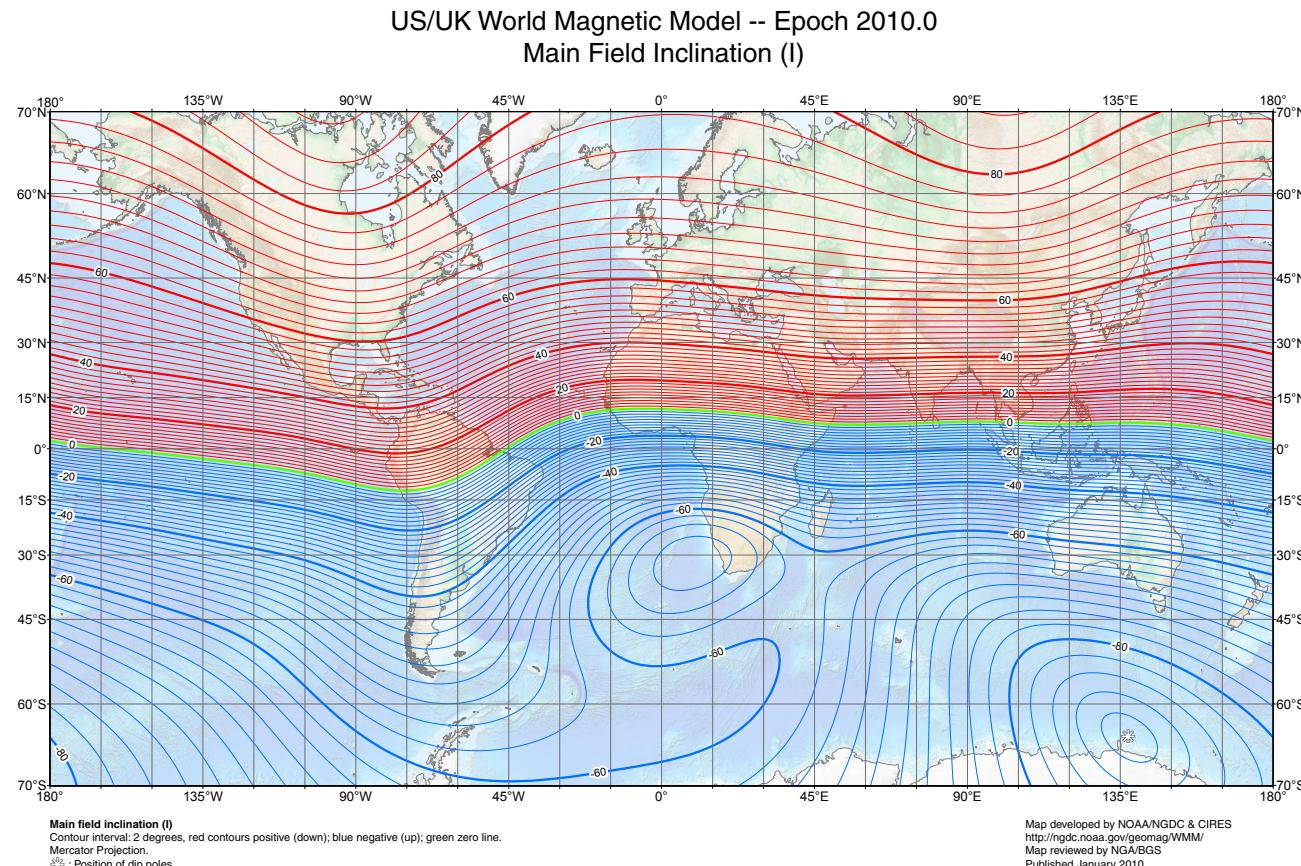


World Magnetic Model, National Geophysical Data Center

Magnetic declination in Provo Utah is 12.5 degrees

Magnetic Inclination

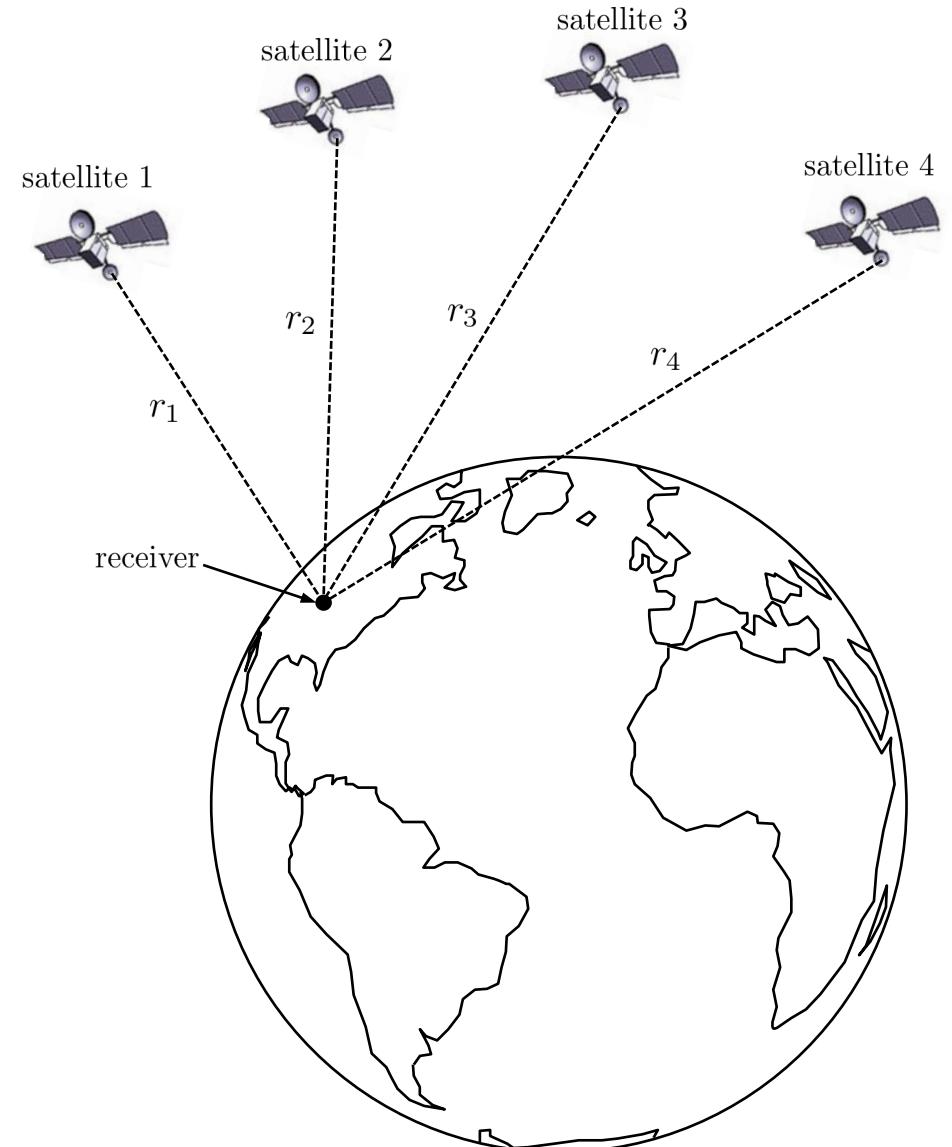
From Wikipedia: "Magnetic dip or magnetic inclination is the angle made by a compass needle with the horizontal at any point on the Earth's surface. Positive values of inclination indicate that the field is pointing downward, into the Earth, at the point of measurement."



Magnetic inclination in Provo Utah is 66 degrees

Global Positioning System

- 24 satellites orbiting the earth
- Altitude 20,180 km
- Any point on Earth's surface can be seen by at least 4 satellites at all times
- Time of flight of radio signal from 4 satellites to receiver used to trilaterate location of receiver in 3 dimensions
- 4 range measurements needed to account for clock offset error
- 4 nonlinear equations in 4 unknowns results:
 - latitude
 - longitude
 - altitude
 - receiver clock time offset



GPS Error Sources

- Time of flight of radio signal from satellite to receiver used to calculate pseudorange
 - Called pseudorange to distinguish it from true range
- Numerous sources of error in time-of-flight measurement:
 - Ephemeris Data – errors in satellite location
 - Satellite Clock – due to clock drift
 - Ionosphere – upper atmosphere, free electrons slow transmission of GPS signal
 - Troposphere – lower atmosphere, weather (temperature and density) affect speed of light, GPS signal transmission
 - Multipath Reception – signals not following direct path
 - Receiver Measurement – limitations in accuracy of receiver timing
- Small timing errors can result in large position errors
 - 10 ns timing error → 3 m pseudorange error

GPS Error Characterization

- Cumulative effect of GPS pseudorange errors is described by user equivalent range error (UERE)
- UERE has two components
 - Bias
 - Random

1- σ , in meters

Error source	Bias	Random	Total
Ephemeris data	2.1	0.0	2.1
Satellite clock	2.0	0.7	2.1
Ionosphere	4.0	0.5	4.0
Troposphere monitoring	0.5	0.5	0.7
Multipath	1.0	1.0	1.4
Receiver measurement	0.5	0.2	0.5
UERE, rms	5.1	1.4	5.3
Filtered UERE, rms	5.1	0.4	5.1

GPS Error Characterization

- Effect of satellite geometry on position calculation is expressed by Dilution of Precision (DOP)
- Satellites close together → high DOP
- Satellites far apart → low DOP
- DOP varies with time
- Horizontal DOP is smaller than vertical DOP
- Nominal HDOP = 1.3
- Nominal VDOP = 1.8

Total GPS Error (RMS)

Standard deviation of RMS error in the north-east plane:

$$\begin{aligned}E_{n-e,\text{rms}} &= \text{HDOP} \times \text{UERE}_{\text{rms}} \\&= (1.3)(5.1 \text{ m}) \\&= 6.6 \text{ m}\end{aligned}$$

Standard deviation of RMS altitude error:

$$\begin{aligned}E_{h,\text{rms}} &= \text{VDOP} \times \text{UERE}_{\text{rms}} \\&= (1.8)(5.1 \text{ m}) \\&= 9.2 \text{ m}\end{aligned}$$

GPS Error Model

- Interested in transient behavior of errors – how does GPS error change with time
- We use Gauss-Markov error model proposed by Rankin

$$\nu[n + 1] = e^{-k_{\text{GPS}}T_s} \nu[n] + \eta_{\text{GPS}}[n]$$

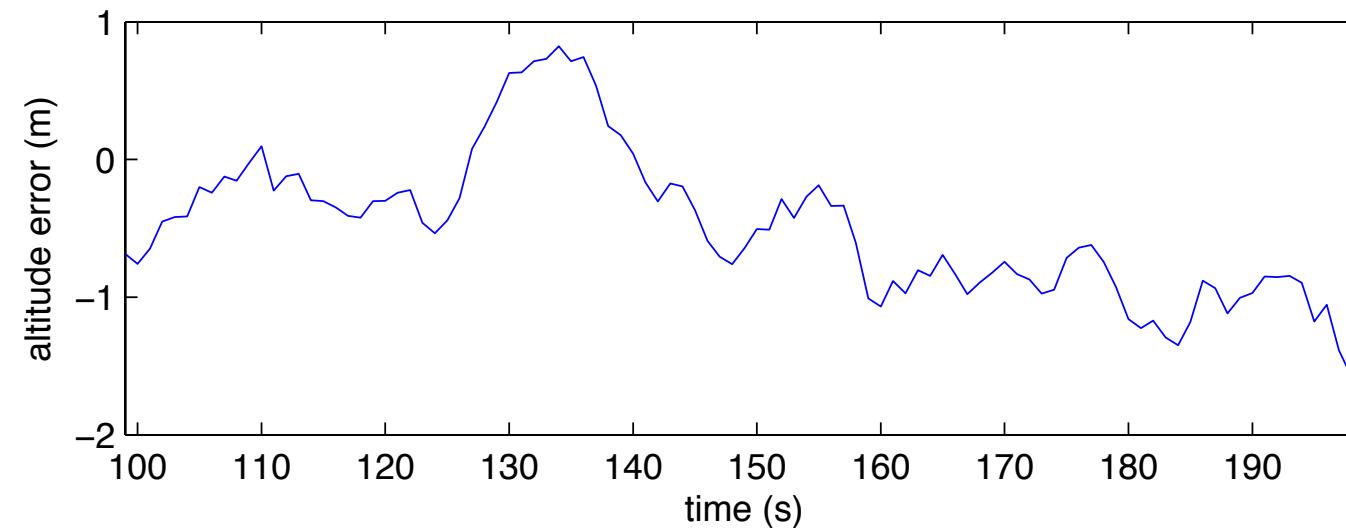
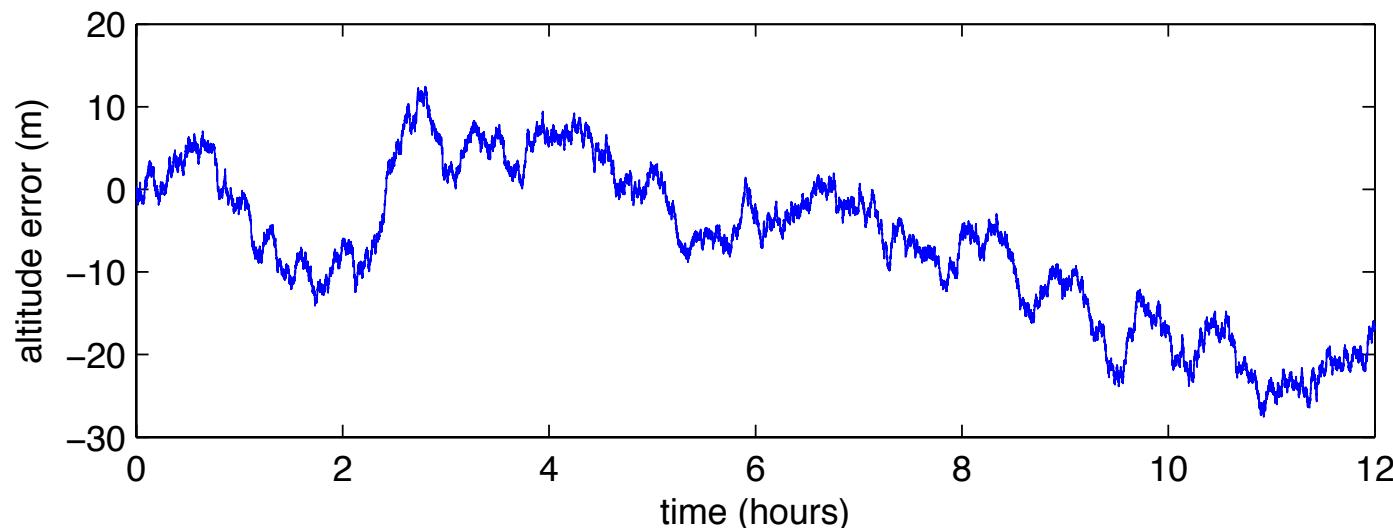
Direction	Nominal 1- σ error (m)		Model Parameters		
	Bias	Random	Std. Dev. η_{GPS} (m)	$1/k_{\text{GPS}}$ (s)	T_s (s)
North	4.7	0.4	0.21	1100	1.0
East	4.7	0.4	0.21	1100	1.0
Altitude	9.2	0.7	0.40	1100	1.0

$$y_{\text{GPS},n}[n] = p_n[n] + \nu_n[n]$$

$$y_{\text{GPS},e}[n] = p_e[n] + \nu_e[n]$$

$$y_{\text{GPS},h}[n] = -p_d[n] + \nu_h[n]$$

GPS Gauss Markov Process Error Model



Project

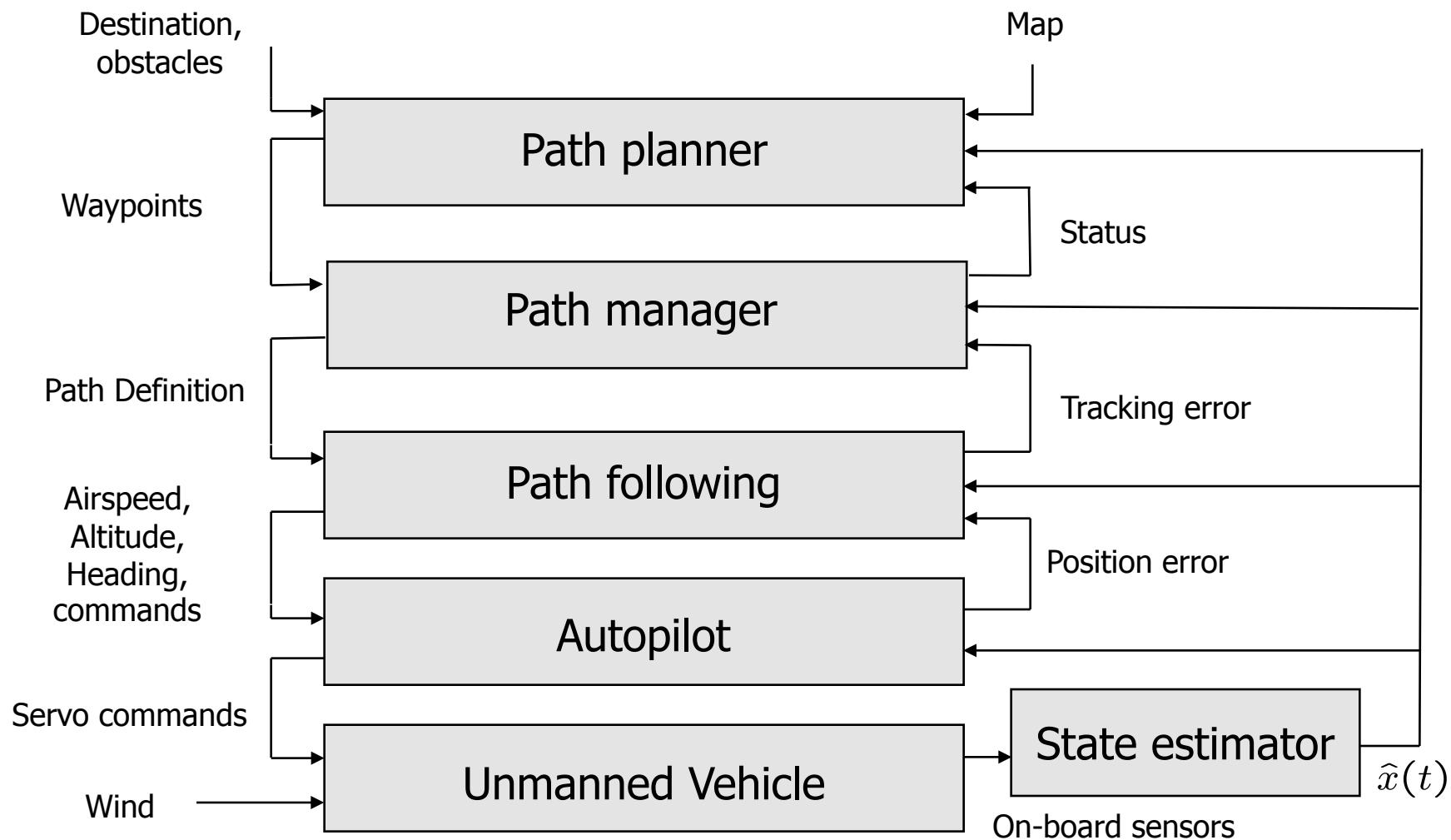
- Add sensor models to the simulation



Chapter 8

State Estimation

Architecture

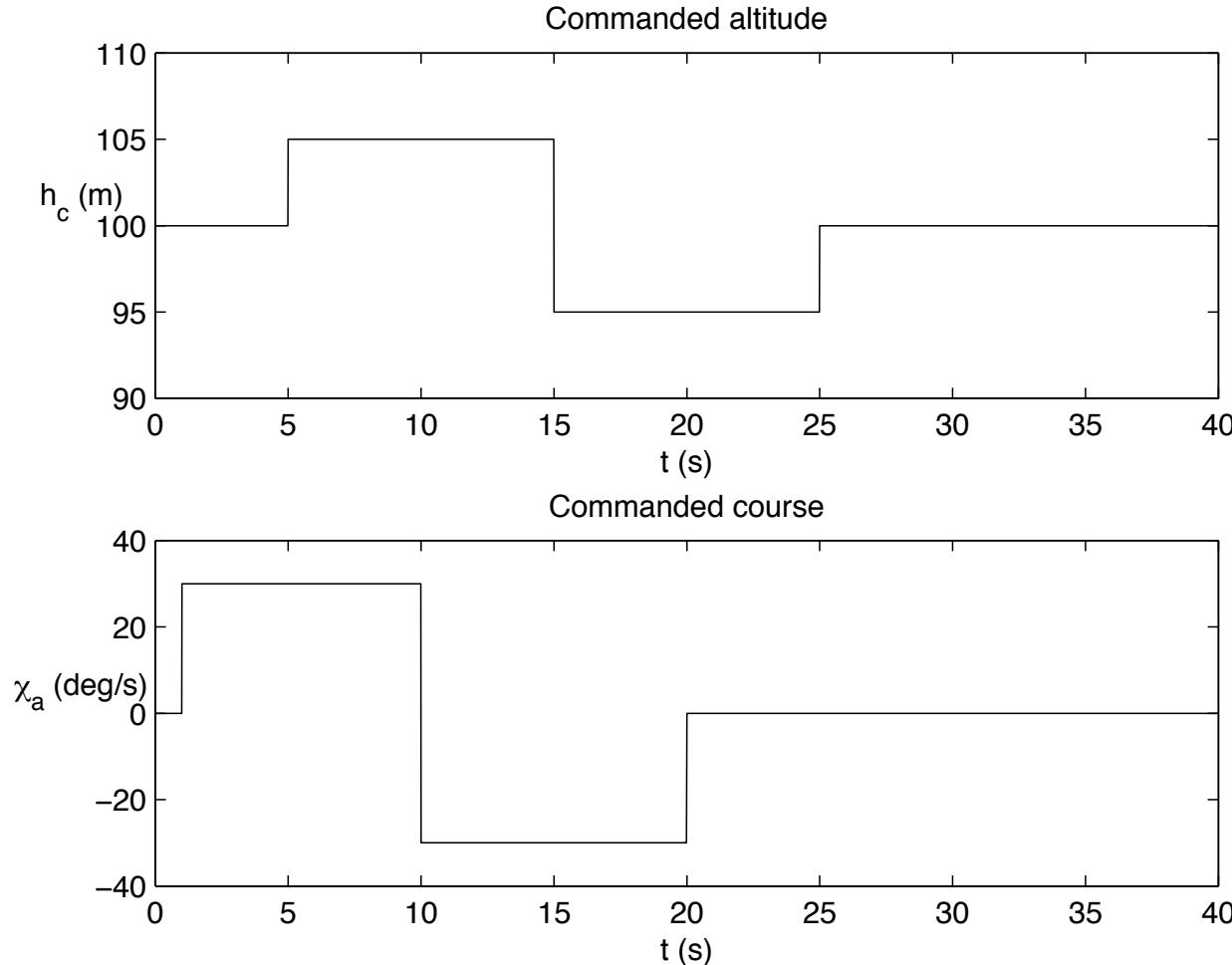


Why estimate states?

- State information needed to control aircraft
- We measure many things
 - accelerations, angular rates, pressure altitude, dynamic pressure (airspeed), magnetic heading (sometimes), GPS position
- Don't have direct measurements of everything we need

State	Measured directly or estimated?
p_n	Measured (GPS) and smoothed
p_e	Measured (GPS) and smoothed
p_d	Measured (absolute pressure, GPS)
u	Estimated with EKF
v	Estimated with EKF
w	Estimated with EKF
ϕ	Estimated with EKF
θ	Estimated with EKF
ψ	Estimated with EKF
p	Measured (rate gyro)
q	Measured (rate gyro)
r	Measured (rate gyro)

Benchmark Maneuver



Inverting Sensor Measurement Model

- Several states can be estimated by inverting sensor measurement model
 - Angular rates, altitude, airspeed
 - LPF denotes low pass filter

Mathematical Model

$$y_{\text{gyro,x}} = p + \beta_p + \eta_p$$

$$y_{\text{gyro,y}} = q + \beta_q + \eta_q$$

$$y_{\text{gyro,z}} = r + \beta_r + \eta_r$$

$$y_{\text{abs pres}} = \rho gh + \beta_{\text{abs}} + \eta_{\text{abs}}$$

$$y_{\text{diff pres}} = \frac{1}{2}\rho V_a^2 + \beta_{\text{diff}} + \eta_{\text{diff}}$$

State Estimate

$$\hat{p} = LPF(y_{\text{gyro,x}})$$

$$\hat{q} = LPF(y_{\text{gyro,y}})$$

$$\hat{r} = LPF(y_{\text{gyro,z}})$$

$$\hat{h} = \frac{LPF(y_{\text{static pres}})}{\rho g}$$

$$\hat{V}_a = \sqrt{\frac{2}{\rho} LPF(y_{\text{diff pres}})}$$

The alpha-filter

Suppose that your sensor gives noisy data as shown on the right.

The objective is to process this data to smooth out the noise.

The standard method is to use an alpha-filter:

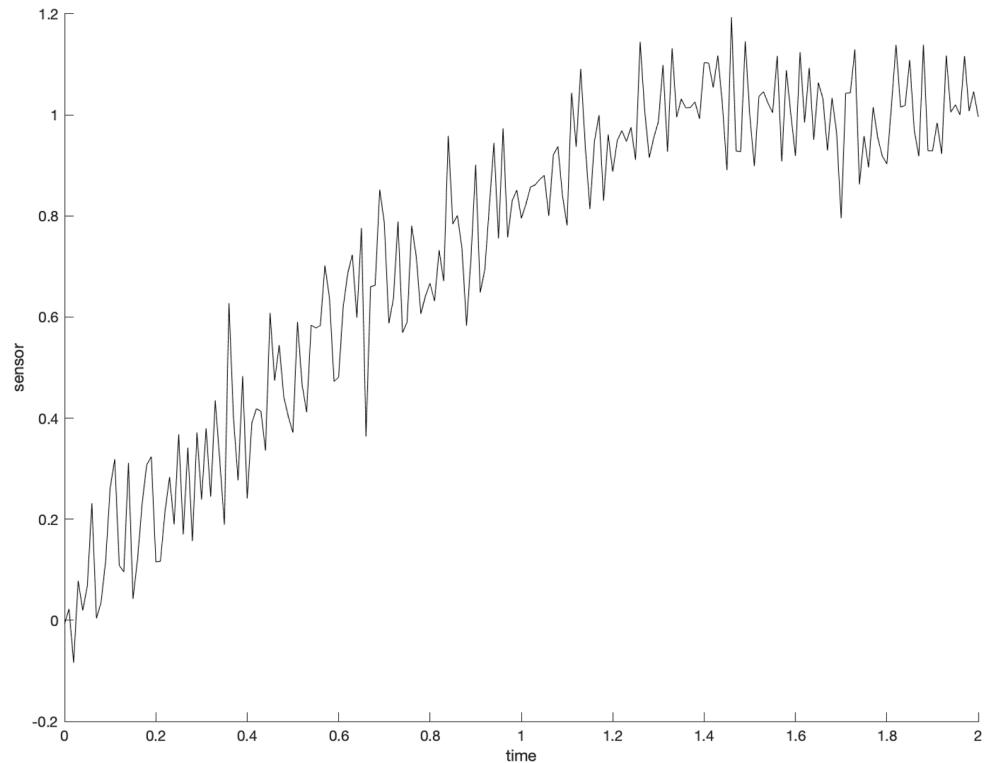
$$z_k = \alpha z_{k-1} + (1 - \alpha)y_k$$

where

$$0 \leq \alpha \leq 1$$

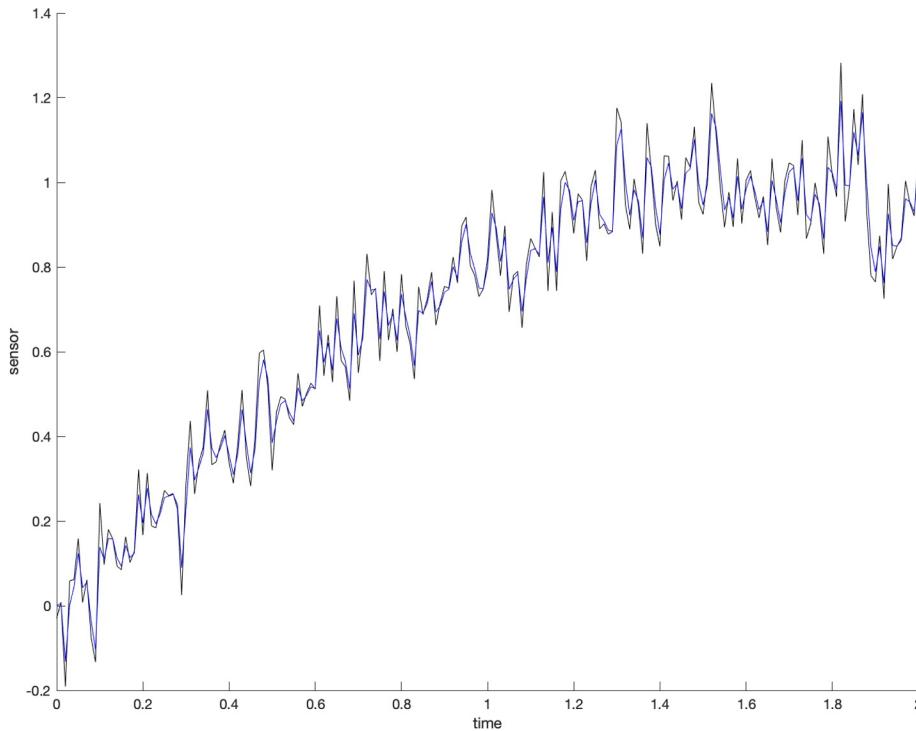
y_k \equiv sensor output at time k

z_k \equiv smoothed sensor output at time k

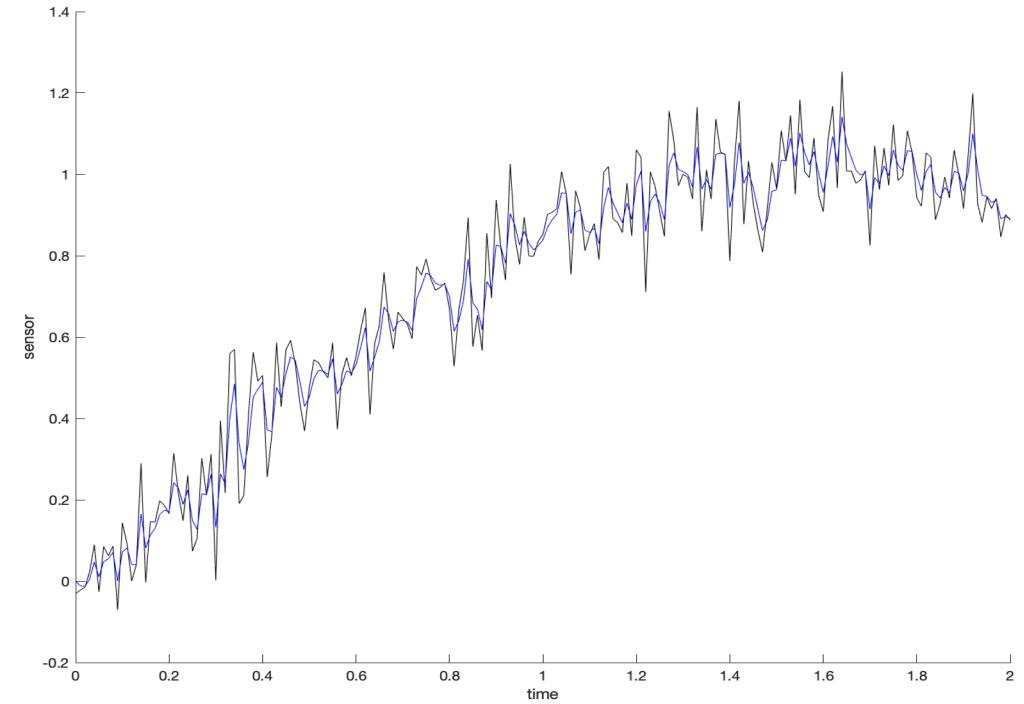


The alpha-filter

$$z_k = \alpha z_{k-1} + (1 - \alpha)y_k$$



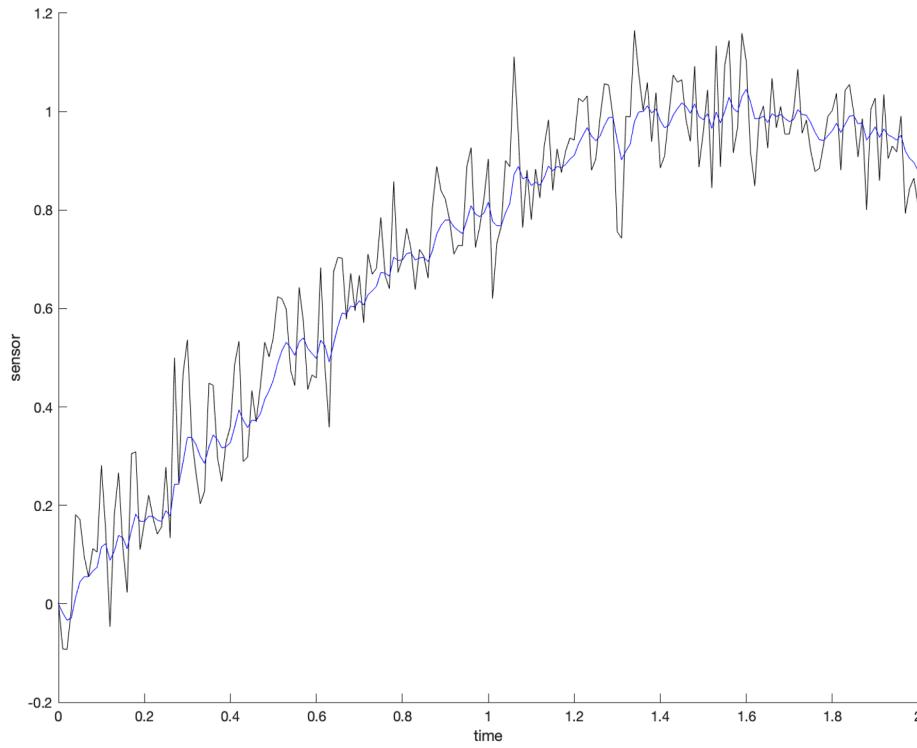
$$\alpha = 0.3$$



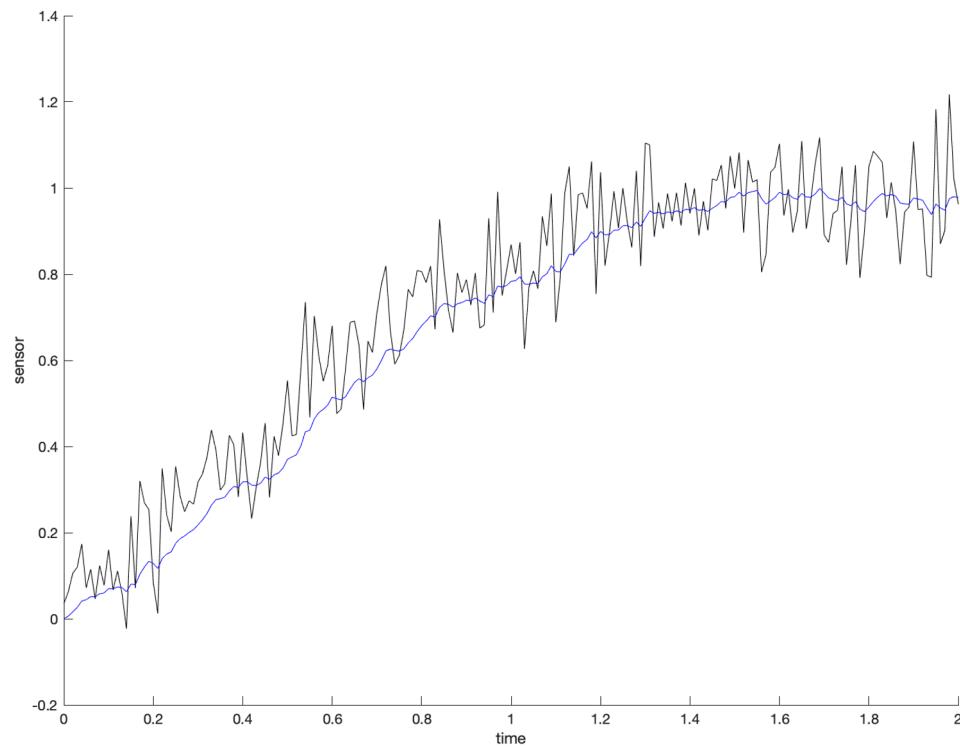
$$\alpha = 0.5$$

The alpha-filter

$$z_k = \alpha z_{k-1} + (1 - \alpha)y_k$$



$$\alpha = 0.8$$



$$\alpha = 0.9$$

The alpha-filter: a deeper look

The traditional low-pass filter is given by

$$Z(s) = \frac{a}{s + a} Y(s),$$

where frequencies above a radians/s are "filtered."

The associated differential equation is

$$\dot{z} = -az + ay.$$

Solving the differential equation gives

$$z(t) = e^{-a(t-t_0)} z(t_0) + \int_{t_0}^t e^{-a(t-\tau)} ay(\tau) d\tau$$

The alpha-filter: a deeper look

Letting $t = t_k$, and $t_0 = t_{k-1}$, and $t_k = t_{k-1} + T_s$, and $\sigma = \tau - t_{k-1}$ gives

$$\begin{aligned} z(t_k) &= e^{-a(t_k - t_{k-1})} z(t_{k-1}) + \int_{t_{k-1}}^{t_k} e^{-a(t_k - \tau)} a y(\tau) d\tau \\ &= e^{-aT_s} z(t_{k-1}) + a \int_0^{T_s} e^{-a(T_s - \sigma)} y(\sigma + t_{k-1}) d\sigma \end{aligned}$$

Define $z_k \equiv z(t_k)$, and approximating $y(t)$ as constant between samples gives

$$\begin{aligned} z_k &= e^{-aT_s} z_{k-1} + a \int_0^{T_s} e^{-a(T_s - \sigma)} d\sigma y_k \\ &= e^{-aT_s} z_{k-1} + (1 - e^{-aT_s}) y_k \\ &= \alpha z_{k-1} + (1 - \alpha) y_k, \end{aligned}$$

where $0 \leq e^{-aT_s} \equiv \alpha \leq 1$.

Therefore the α -filter is just a sampled low-pass filter.

The alpha-filter: Python Implementation

```
class AlphaFilter:  
    # alpha filter implements a simple low pass filter  
    #  $y[k] = \alpha * y[k-1] + (1-\alpha) * u[k]$   
    def __init__(self, alpha=0.5, y0=0.0):  
        self.alpha = alpha # filter parameter  
        self.y = y0 # initial condition  
  
    def update(self, u):  
        self.y = self.alpha * self.y + (1-self.alpha) * u  
        return self.y  
  
# instantiation  
self.lpf_gyro_x = AlphaFilter(alpha=0.7, y0=initial_measurements.gyro_x)  
self.lpf_abs = AlphaFilter(alpha=0.9, y0=initial_measurements.abs_pressure)  
self.lpf_diff = AlphaFilter(alpha=0.7, y0=initial_measurements.diff_pressure)  
  
# update  
# estimates for p, q, r are low pass filter of gyro minus bias estimate  
self.estimated_state.p = self.lpf_gyro_x.update(measurement.gyro_x) \  
                      - self.estimated_state.bx  
abs_pressure = self.lpf_abs.update(measurement.abs_pressure)  
diff_pressure = self.lpf_diff.update(measurement.diff_pressure)
```

Inverting Sensor Measurement Model

- Several states can be estimated by inverting sensor measurement model
 - Angular rates, altitude, airspeed
 - LPF denotes low pass filter

Mathematical Model

$$y_{\text{gyro,x}} = p + \beta_p + \eta_p$$

$$y_{\text{gyro,y}} = q + \beta_q + \eta_q$$

$$y_{\text{gyro,z}} = r + \beta_r + \eta_r$$

$$y_{\text{abs pres}} = \rho gh + \beta_{\text{abs}} + \eta_{\text{abs}}$$

$$y_{\text{diff pres}} = \frac{1}{2}\rho V_a^2 + \beta_{\text{diff}} + \eta_{\text{diff}}$$

State Estimate

$$\hat{p} = LPF(y_{\text{gyro,x}})$$

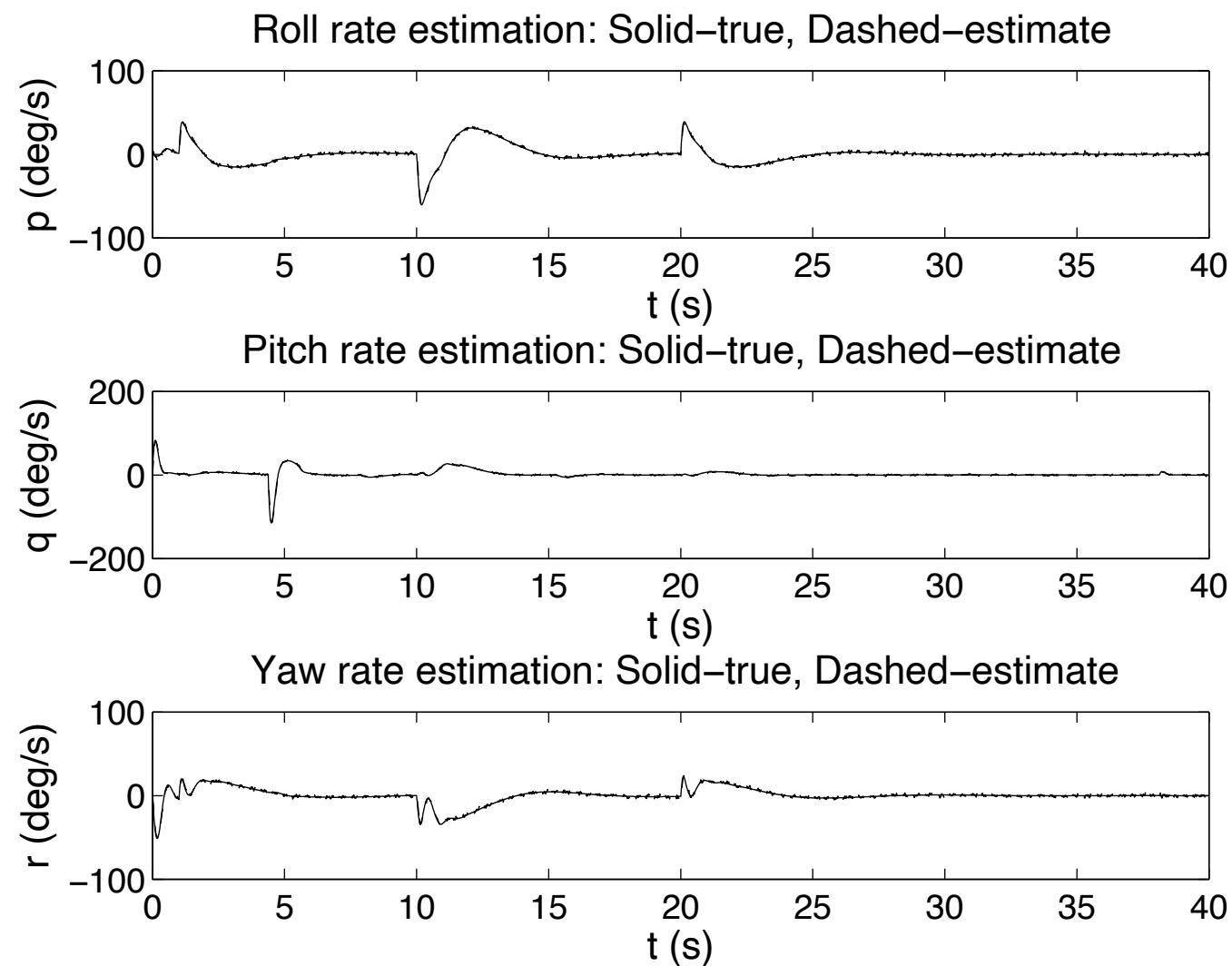
$$\hat{q} = LPF(y_{\text{gyro,y}})$$

$$\hat{r} = LPF(y_{\text{gyro,z}})$$

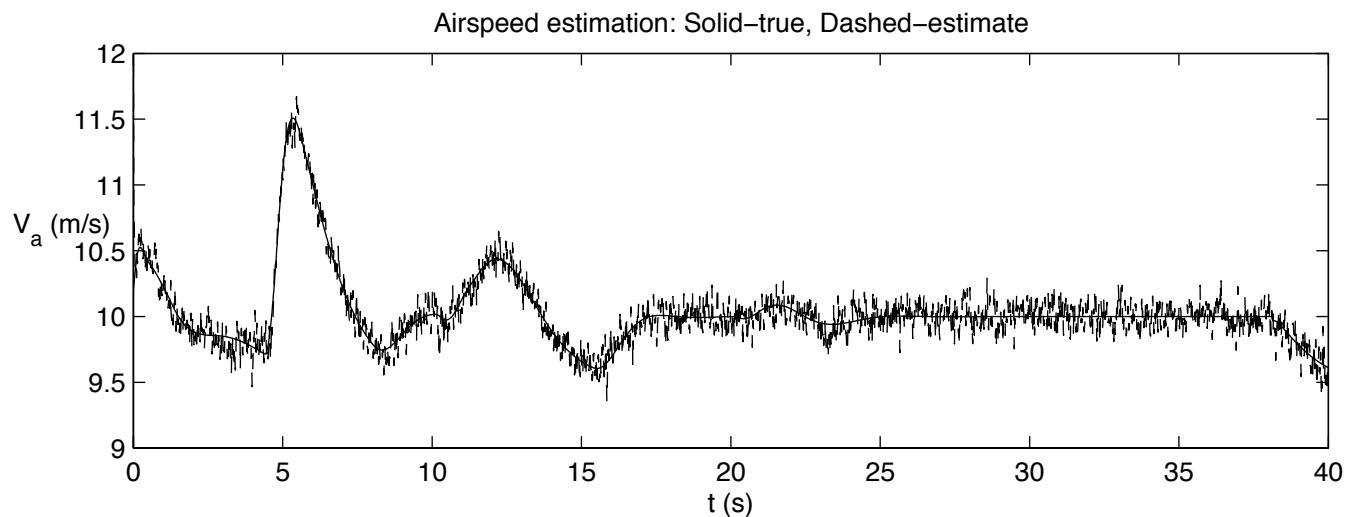
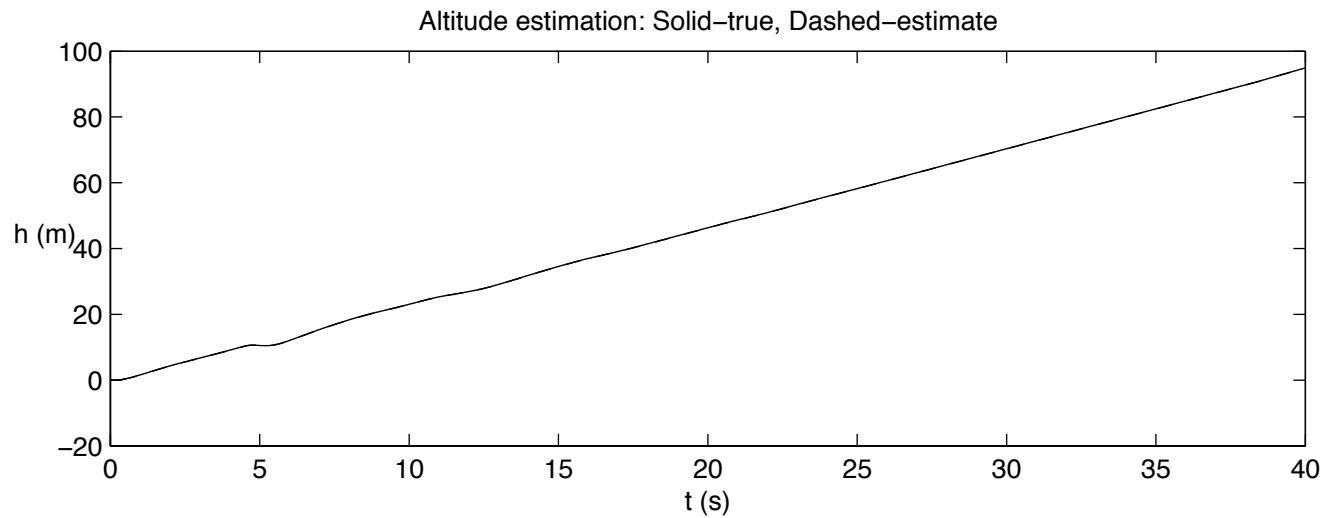
$$\hat{h} = \frac{LPF(y_{\text{static pres}})}{\rho g}$$

$$\hat{V}_a = \sqrt{\frac{2}{\rho} LPF(y_{\text{diff pres}})}$$

Model Inversion - p, q, r (no bias)



Model Inversion - h , V_a



Inverting Sensor Measurement Model

Assuming steady, level flight (no acceleration), we can also invert accelerometer measurements to determine pitch and roll. Accelerometer outputs are

$$y_{\text{accel},x} = \dot{u} + qw - rv + g \sin \theta + \eta_{\text{accel},x}$$

$$y_{\text{accel},y} = \dot{v} + ru - pw - g \cos \theta \sin \phi + \eta_{\text{accel},y}$$

$$y_{\text{accel},z} = \dot{w} + pv - qu - g \cos \theta \cos \phi + \eta_{\text{accel},z}$$

In unaccelerated flight, $\dot{u} = \dot{v} = \dot{w} = p = q = r = 0$, and

$$\text{LPF}(y_{\text{accel},x}) = g \sin \theta$$

$$\text{LPF}(y_{\text{accel},y}) = -g \cos \theta \sin \phi$$

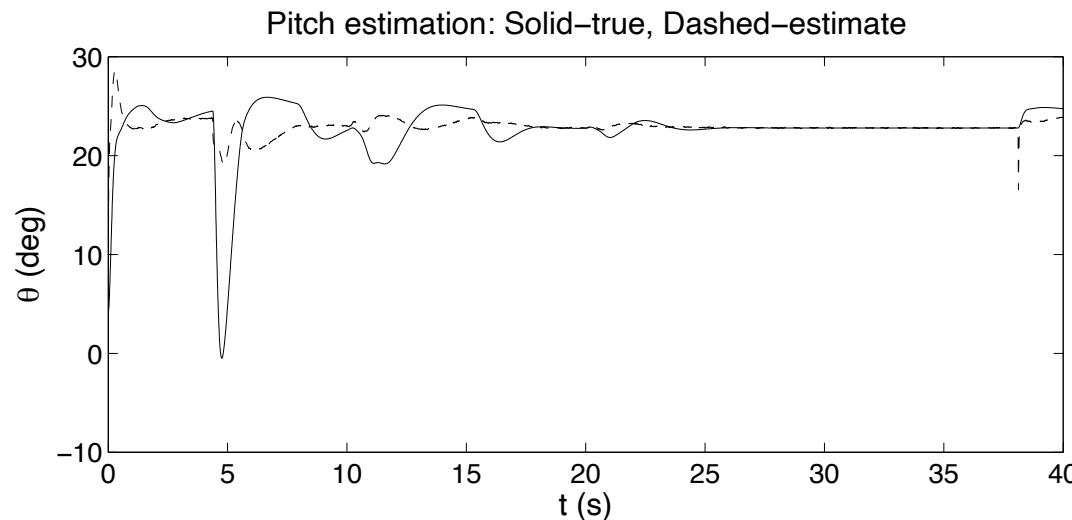
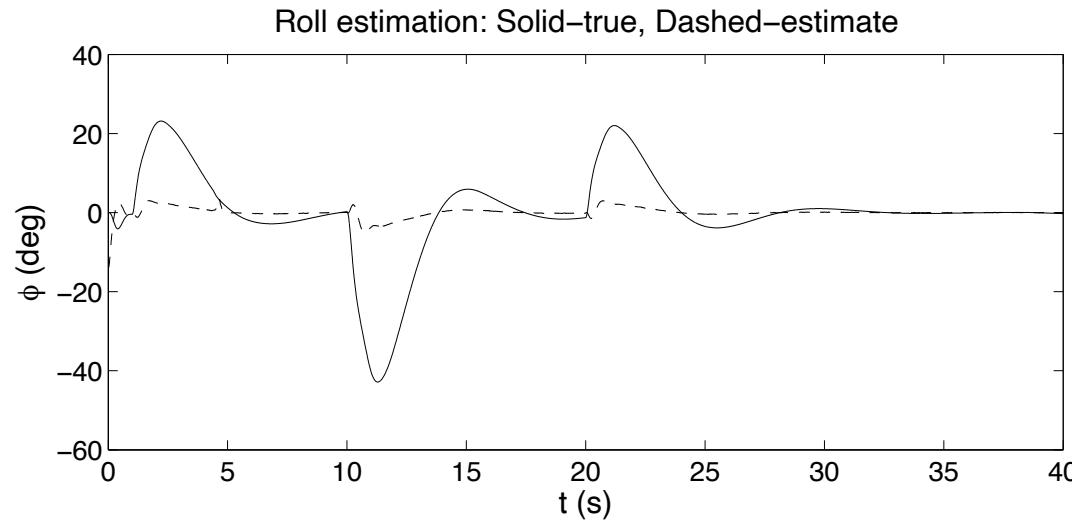
$$\text{LPF}(y_{\text{accel},z}) = -g \cos \theta \cos \phi$$

Solving for ϕ and θ :

$$\hat{\phi}_{\text{accel}} = \tan^{-1} \left(\frac{\text{LPF}(y_{\text{accel},y})}{\text{LPF}(y_{\text{accel},z})} \right)$$

$$\hat{\theta}_{\text{accel}} = \sin^{-1} \left(\frac{\text{LPF}(y_{\text{accel},x})}{g} \right)$$

Model Inversion - ϕ , θ



Inverting Sensor Measurement Model

- Can also invert sensor models for GPS signals:

$$\hat{p}_n = LPF(y_{GPS,n})$$

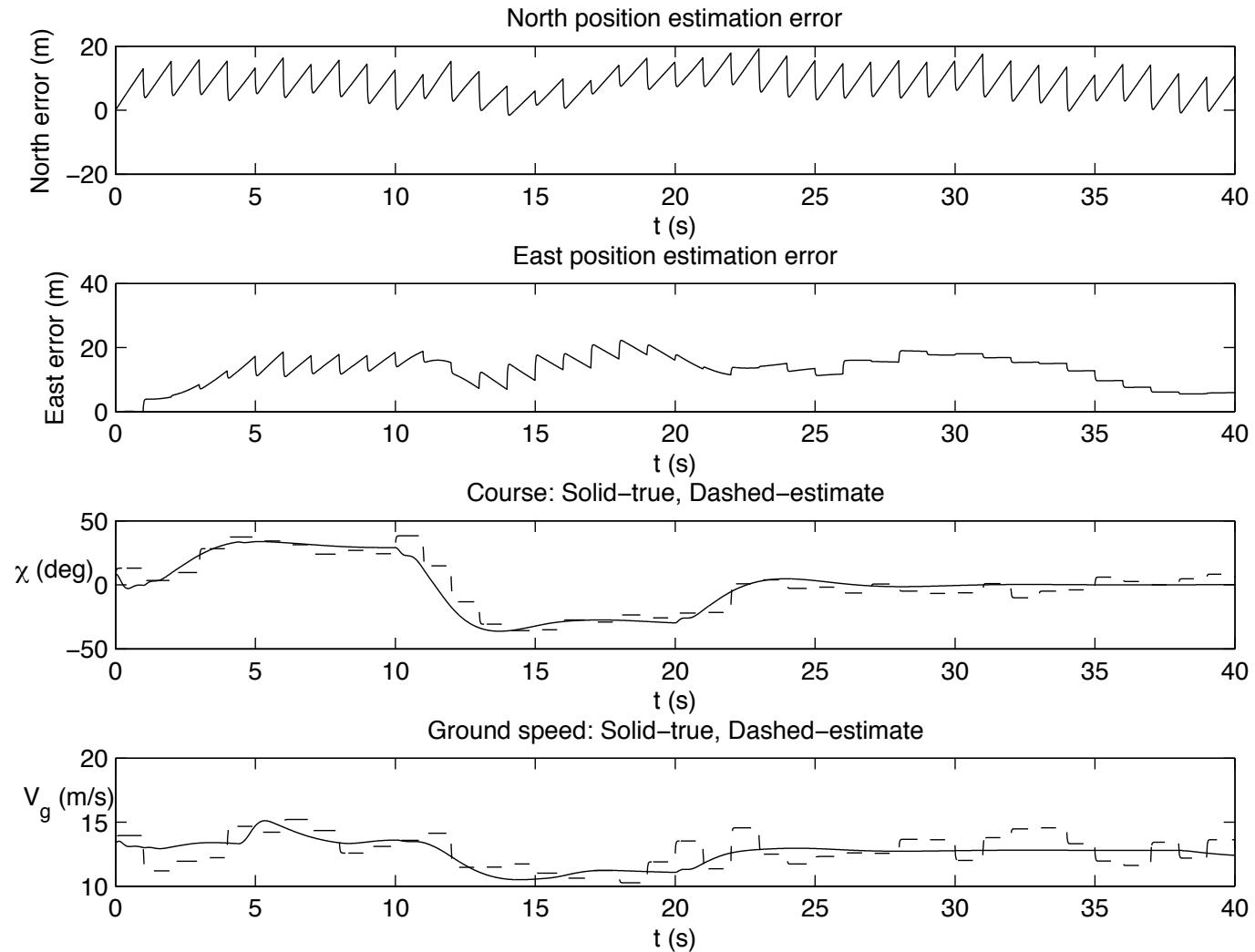
$$\hat{p}_e = LPF(y_{GPS,e})$$

$$\hat{\chi} = LPF(y_{GPS,\chi})$$

$$\hat{V}_g = LPF(y_{GPS,V_g})$$

- Update rate too slow (1 Hz)
- Need to fill in samples between measurement updates

Model Inversion - p_n , p_e , χ , V_g



Dynamic Observer Theory

Given linear time-invariant model

$$\begin{aligned}\dot{x} &= Ax + Bu \\ y &= Cx.\end{aligned}$$

A continuous-time observer for this system is given by

$$\dot{\hat{x}} = \underbrace{A\hat{x} + Bu}_{\text{copy of the model}} + \underbrace{L(y - C\hat{x})}_{\text{correction due to sensor reading}},$$

where \hat{x} is the estimated value of x . Defining the observation error as $\tilde{x} = x - \hat{x}$

$$\dot{\tilde{x}} = (A - LC)\tilde{x}.$$

Therefore, observation error $\rightarrow 0$ if $\text{eig}(A - LC)$ in LHP.

Predictor-Corrector Structure

For linear systems:

Between Measurements (predictor):

$$\dot{\hat{x}} = A\hat{x} + Bu,$$

At a Measurements (corrector):

$$\hat{x}^+ = \hat{x}^- + L(y(t_n) - C\hat{x}^-),$$

where t_n is the instant in time that the measurement is received and \hat{x}^- is the state estimate produced at time t_n .

For nonlinear systems:

Between Measurements (predictor):

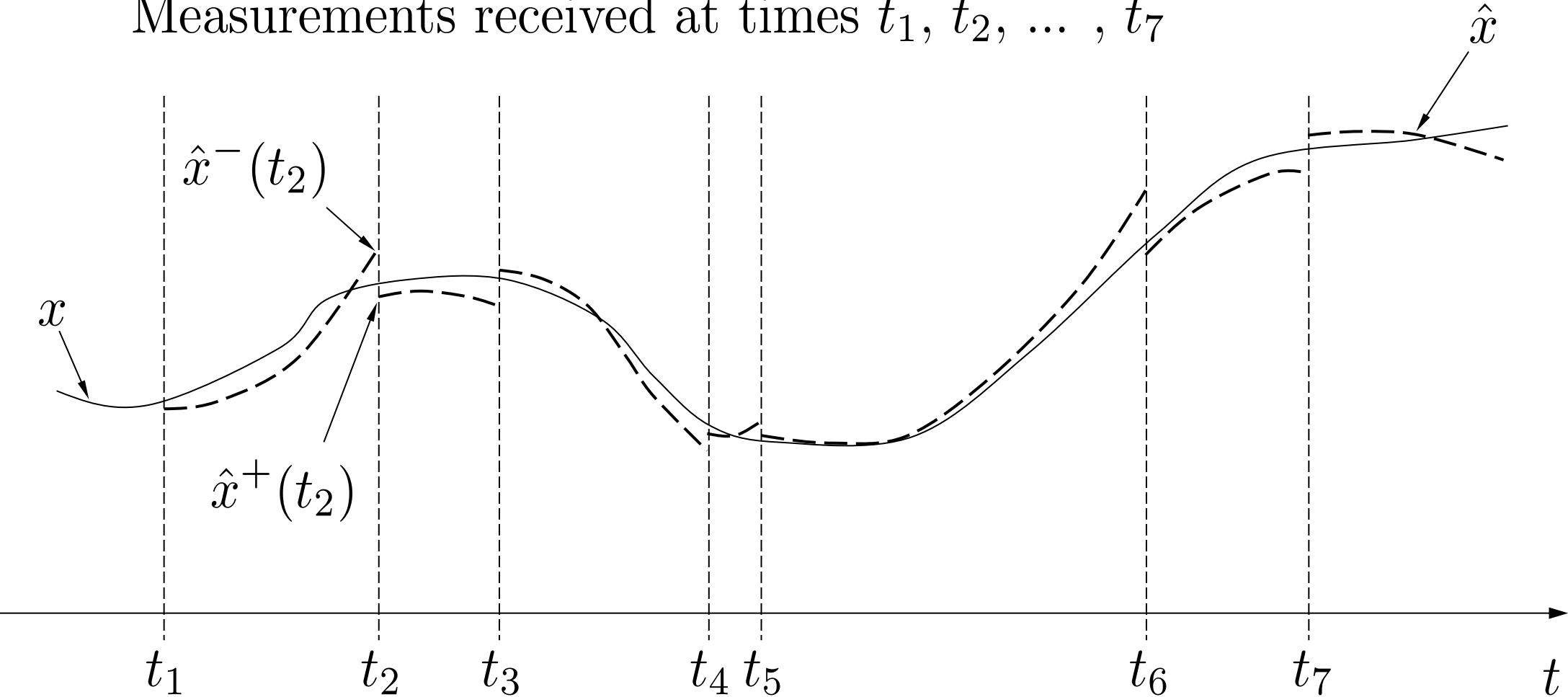
$$\dot{\hat{x}} = f(\hat{x}, u)$$

At a Measurements (corrector):

$$\hat{x}^+ = \hat{x}^- + L(y(t_n) - h(\hat{x}^-)).$$

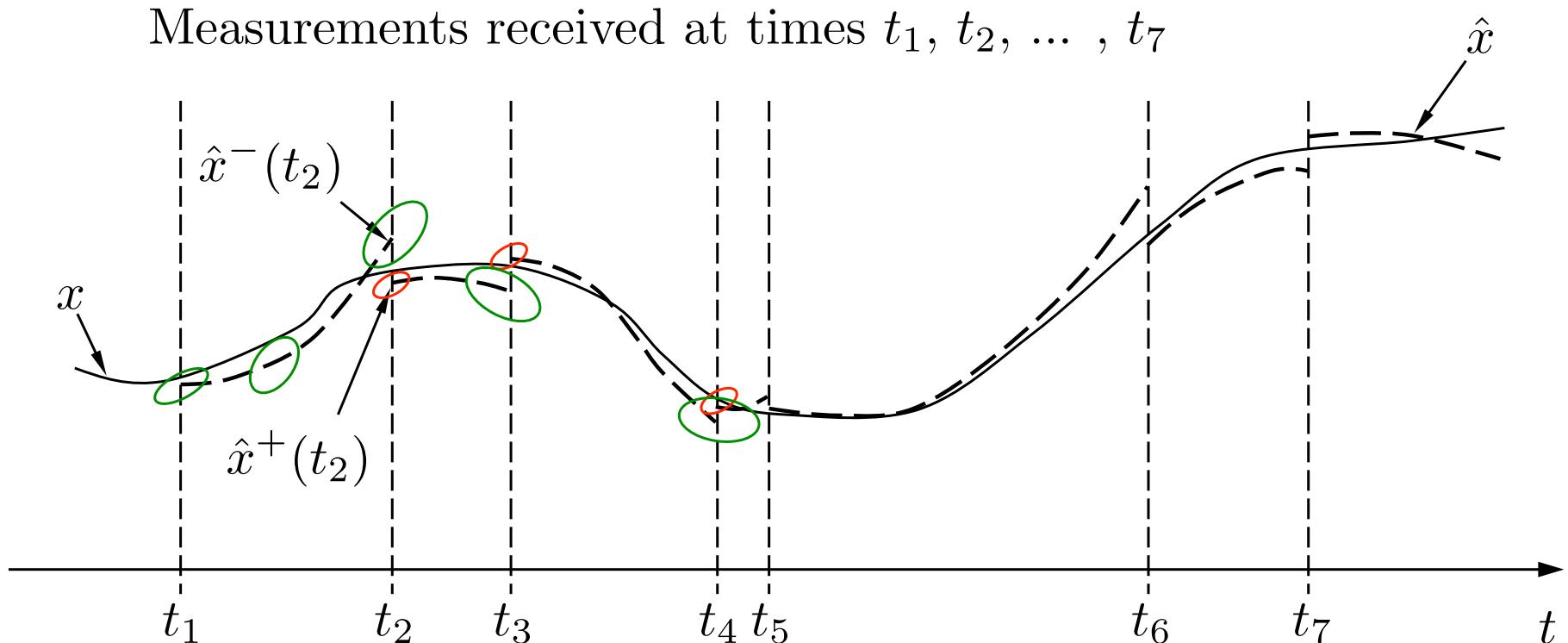
Predictor-Corrector Structure

Measurements received at times t_1, t_2, \dots, t_7



Kalman Filter

The Kalman filter follows this same process, but also attempts to estimate the quality of the estimate at each time step by propagating an error covariance matrix.

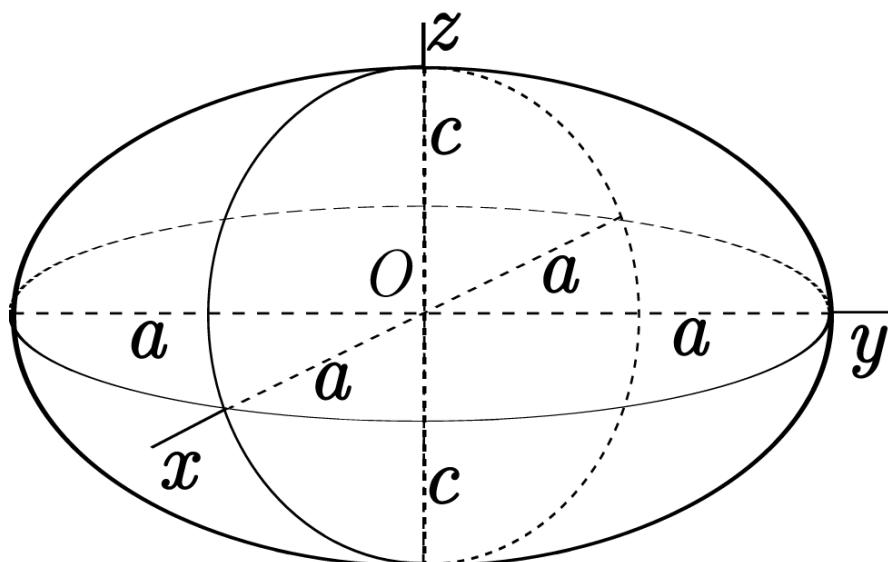


Quadratic Forms

Recall that for a positive definite matrix $P = P^\top > 0$, the set

$$\mathcal{E}(k) = \{y \in \mathbb{R}^n : y^\top P^{-1} y = k^2\}$$

is an ellipsoid whose axes are aligned with the eigenvectors of P , with stretching along each axis given by $k\sqrt{\lambda_i}$ where λ_i is an eigenvalue of P .



(From wikipedia)

The eigenaxes are given by x, y, z .
 a, b , and c correspond to $k\sqrt{\lambda_i}$.

Multivariate Gaussian

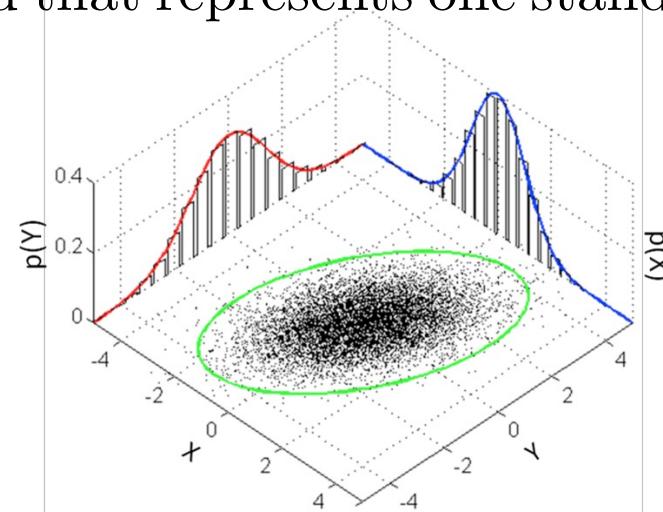
The multivariate Gaussian distribution is given by

$$\mathcal{N}(\mu, P) = \frac{1}{(2\pi)^{k/2} \|P\|^{1/2}} \exp \left(-\frac{1}{2} (x - \mu)^\top P^{-1} (x - \mu) \right),$$

where μ is the mean and P is the covariance matrix. The $k^{th} - \sigma$ ellipsoid is defined as

$$\mathcal{E}(k) = \{x \in \mathbb{R}^k : (x - \mu)^\top P^{-1} (x - \mu) = k^2\}$$

where *one*– σ or $k = 1$ corresponds to the ellipsoid that represents one standard deviation from the mean μ .



Kalman Filter

Assume continuous linear system dynamics, with discrete measurement update

$$\begin{aligned}\dot{x} &= Ax + Bu + \xi \\ y[n] &= Cx[n] + \eta[n],\end{aligned}$$

where $\xi \sim \mathcal{N}(0, Q)$ is the process noise, $\nu \sim \mathcal{N}(0, R)$ is the measurement noise.

The measurement covariance R is usually obtained from sensor calibration. The process covariance Q represents all other uncertainties in the system and is the standard tuning parameter for the Kalman filter.

Kalman Filter Derivation

The continuous-discrete Kalman filter has the form

$$\dot{\hat{x}} = A\hat{x} + Bu \quad (\text{Prediction})$$

$$\hat{x}^+ = \hat{x}^- + L(y(t_n) - C\hat{x}^-). \quad (\text{Correction})$$

Define the estimation error

$$\tilde{x} = x - \hat{x},$$

and the covariance of the estimation error

$$P(t) = E\{\tilde{x}(t)\tilde{x}(t)^\top\},$$

where $P = P^\top \geq 0$.

Estimation Objective: Choose the estimation gain L s.t.

- $E\{\tilde{x}(t)\} = 0$ (unbiased),
- $\text{trace}(P) = \sum \lambda_i$ is minimized.

Kalman Filter Derivation

Between Measurements (prediction):

Differentiate \tilde{x}

$$\begin{aligned}\dot{\tilde{x}} &= \dot{x} - \dot{\hat{x}} \\ &= Ax + Bu + \xi - A\hat{x} - Bu \\ &= A\tilde{x} + \xi.\end{aligned}$$

Solve for $\tilde{x}(t)$:

$$\tilde{x}(t) = e^{At}\tilde{x}_0 + \int_0^t e^{A(t-\tau)}\xi(\tau) d\tau.$$

Note that

$$E\{\tilde{x}(t)\} = e^{At}E\{\tilde{x}_0\} + \int_0^t e^{A(t-\tau)}E\{\xi(\tau)\} d\tau = 0$$

Kalman Filter Derivation

Between Measurements (prediction):

Compute the evolution of the error covariance P :

$$\begin{aligned}\dot{P} &= \frac{d}{dt} E\{\tilde{x}\tilde{x}^\top\} = E\{\dot{\tilde{x}}\tilde{x}^\top + \tilde{x}\dot{\tilde{x}}^\top\} \\ &= E\{A\tilde{x}\tilde{x}^\top + \xi\tilde{x}^\top + \tilde{x}\tilde{x}^\top A^\top + \tilde{x}\xi^\top\} \\ &= AP + PA^\top + E\{\xi\tilde{x}^\top\} + E\{\tilde{x}\xi^\top\},\end{aligned}$$

Where

$$\begin{aligned}E\{\tilde{x}\xi^\top\} &= E\{e^{At}\tilde{x}_0\xi^\top(t) + \int_0^t e^{A(t-\tau)}\xi(\tau)\xi^\top(t) d\tau\} \\ &= \int_0^t e^{A(t-\tau)}Q\delta(t-\tau) d\tau \\ &= \frac{1}{2}Q. \quad (\text{only used half of area in delta function})\end{aligned}$$

Therefore

$$\dot{P} = AP + PA^\top + Q.$$

Kalman Filter Derivation

At Measurements (correction):

At a measurement, we have that

$$\begin{aligned}\tilde{x}^+ &= x - \hat{x}^+ \\ &= x - \hat{x}^- - L(Cx + \eta - C\hat{x}^-) \\ &= \tilde{x}^- - LC\tilde{x}^- - L\eta.\end{aligned}$$

We also have that

$$\begin{aligned}P^+ &= E\{\tilde{x}^+\tilde{x}^{+T}\} \\ &= E\left\{(\tilde{x}^- - LC\tilde{x}^- - L\eta)(\tilde{x}^- - LC\tilde{x}^- - L\eta)^T\right\} \\ &= E\left\{\tilde{x}^-\tilde{x}^{-T} - \tilde{x}^-\tilde{x}^{-T}C^T L^T - \tilde{x}^-\eta^T L^T - LC\tilde{x}^-\tilde{x}^{-T} + LC\tilde{x}^-\tilde{x}^{-T}C^T L^T + LC\tilde{x}^-\eta^T L^T\right. \\ &\quad \left.- L\eta\tilde{x}^{-T} + L\eta\tilde{x}^{-T}C^T L^T + L\eta\eta^T L^T\right\} \\ &= P^- - P^-C^T L^T - LCP^- + LCP^-C^T L^T + LRL^T \\ &= (I - LC)P^-(I - LC)^T + LRL^T.\end{aligned}$$

Objective: Pick L to minimize $\text{trace}(P^+)$.

Kalman Filter Derivation

At Measurements (correction):

Using

$$\frac{\partial}{\partial A} \text{trace}(BAD) = B^\top D^\top,$$

$$\frac{\partial}{\partial A} \text{trace}(ABA^\top) = 2AB, \text{ if } B = B^\top,$$

$$\frac{\partial}{\partial A} \text{trace}(AB) = B$$

a necessary condition is

$$\begin{aligned}\frac{\partial}{\partial L} \text{tr}(P^+) &= \frac{\partial}{\partial L} \text{tr}\left((I - LC)P^-(I - LC)^\top + LRL^\top\right) = -P^-C^\top - P^-C^\top + 2LCP^-C^\top + 2LR = 0 \\ &\implies 2L^*(R + CP^-C^\top) = 2P^-C^\top \\ &\implies L^* = P^-C^\top(R + CP^-C^\top)^{-1}.\end{aligned}$$

Substituting into prior equation for P^+ gives (after algebra)

$$P^+ = (I - L^*C)P^-(I - L^*C)^\top + L^*RL^{*\top}.$$

Kalman Filter Derivation

Between Measurements (prediction):

$$\dot{\hat{x}} = A\hat{x} + Bu$$

$$\dot{P} = AP + PA^\top + Q,$$

At the i^{th} Measurement (correction):

$$L_i = P^- C_i^\top (R_i + C_i P^- C_i^\top)^{-1}$$

$$\hat{x}^+ = \hat{x}^- + L_i(y_i - C_i \hat{x}^-),$$

$$P^+ = (I - L_i C_i) P^- (I - L_i C_i)^\top + L_i R_i L_i^\top$$

where L_i is called the Kalman gain for sensor i .

Covariance Update

Between measurements, the covariance evolves according to the equation

$$\dot{P} = AP + PA^\top + Q.$$

To implement, could use Euler approximation:

$$P_{k+1} = P_k + T_s (AP_k + P_k A^\top + Q).$$

Disadvantage: Due to numerical error, P may not remain positive definite (i.e., won't represent a covariance matrix).

Covariance Update

Recall that

$$\tilde{x}(t) = e^{A(t-t_0)}\tilde{x}(t_0) + \int_{t_0}^t e^{A(t-\tau)}\xi(\tau)d\tau.$$

Let $t = kT_s$ and assuming that $\xi(\tau)$ is constant over each time interval, then

$$\tilde{x}_{k+1} = e^{AT_s}\tilde{x}_k + \left(\int_0^{T_s} e^{A\tau} d\tau \right) \xi_k.$$

Using the approximation

$$A_d = e^{AT_s} \approx I + AT_s + A^2 \frac{T_s^2}{2}$$
$$B_d = \left(\int_0^{T_s} e^{A\tau} d\tau \right) \approx \int_0^{T_s} Id\tau = T_s I,$$

gives

$$\tilde{x}_{k+1} = A_d \tilde{x}_k + T_s \xi_k.$$

Covariance Update

Therefore, the covariance update becomes

$$\begin{aligned} P_{k+1} &= E\{\tilde{x}_{k+1}\tilde{x}_{k+1}^\top\} \\ &= E\{(A_d\tilde{x}_k + T_s\xi_k)(A_d\tilde{x}_k + T_s\xi_k)^\top\} \\ &= E\{A_d\tilde{x}_k\tilde{x}_k^\top A_d + T_s\xi_k\tilde{x}_k^\top A_d^\top + A_d\tilde{x}_k + T_s\xi_k\xi_k^\top + T_s^2\xi_k\xi_k^\top\} \\ &= A_d E\{\tilde{x}_k\tilde{x}_k^\top\} A_d^\top + T_s E\{\xi_k\tilde{x}_k^\top\} A_d^\top + T_s A_d E\{\tilde{x}_k\xi_k^\top\} + T_s^2 E\{\xi_k\xi_k^\top\} \\ &= A_d P_k A_d^\top + T_s^2 Q. \end{aligned}$$

Note that the last line ensures that the error covariance remains positive definite since the sum of positive definite matrices is positive definite.

Kalman Filter Derivation

Between Measurements (prediction):

$$\dot{\hat{x}} = A\hat{x} + Bu$$

$$A_d = I + AT_s + A^2 \frac{T_s^2}{2}$$

$$P_{k+1} = A_d P_k A_d^\top + T_s^2 Q.$$

At the i^{th} Measurement (correction):

$$L_i = P^- C_i^\top (R_i + C_i P^- C_i^\top)^{-1}$$

$$\hat{x}^+ = \hat{x}^- + L_i(y_i - C_i \hat{x}^-),$$

$$P^+ = (I - L_i C_i) P^- (I - L_i C_i)^\top + L_i R_i L_i^\top$$

where L_i is called the Kalman gain for sensor i .

Extended Kalman Filter

Basic idea of Extended Kalman Filter:

- Propagate multivariate Gaussian distribution that captures probability distribution associated with belief about state
- Mean of distribution is \hat{x} — estimated state
- Covariance quantifies associated estimation error
- Kalman gain minimizes covariance of estimation error given uncertainty in model and measurements

System model given by:

$$\begin{aligned}\dot{x} &= f(x, u) + \xi \\ y_i[n] &= h_i(x[n], u[n]) + \eta_i[n],\end{aligned}$$

where

$$\begin{aligned}\xi &\sim \mathcal{N}(0, Q) \\ \eta_i &\sim \mathcal{N}(0, R_i)\end{aligned}$$

Extended Kalman Filter, cont.

Between Measurements (prediction):

$$\dot{\hat{x}} = f(\hat{x}, u)$$

$$A = \frac{\partial f}{\partial x}(\hat{x}, u)$$

$$A_d = I + AT_s + A^2 \frac{T_s^2}{2}$$

$$P_{k+1} = A_d P_k A_d^\top + T_s^2 Q.$$

At the i^{th} Measurements (correction):

$$C_i = \frac{\partial h_i}{\partial x}(\hat{x}^-)$$

$$L_i = P^- C_i^\top (R_i + C_i P^- C_i^\top)^{-1}$$

$$\hat{x}^+ = \hat{x}^- + L_i(y_i(t_n) - h_i(\hat{x}^-)),$$

$$P^+ = (I - L_i C_i) P^- (I - L_i C_i)^\top + L_i R_i L_i^\top$$

where L_i is called the Kalman gain for sensor i .

EKF Algorithm

Algorithm 1 Continuous-Discrete Extended Kalman Filter

- 1: Initialize: $\hat{x} = 0$.
- 2: Pick an output sample rate T_{out} which is much less than the sample rates of the sensors.
- 3: At each sample time T_{out} :
- 4: **for** $i = 1$ to N **do** {Prediction}
 - 5: $T_p = T_{out}/N$
 - 6: $\hat{x} \leftarrow \hat{x} + T_p f(\hat{x}, u)$
 - 7: $A = \frac{\partial f}{\partial x}(\hat{x}, u)$
 - 8: $A_d = I + AT_p + A^2T_p^2$
 - 9: $P \leftarrow A_d P A_d^\top + T_p^2 Q$
- 10: **end for**
- 11: **if** Measurement has been received from sensor i **then** {Correction}
 - 12: $C_i = \frac{\partial h_i}{\partial x}(\hat{x}, u[n])$
 - 13: $L_i = P C_i^\top (R_i + C_i P C_i^\top)^{-1}$
 - 14: $P \leftarrow (I - L_i C_i) P (I - L_i C_i)^\top + L_i R_i L_i^\top$
 - 15: $\hat{x} \leftarrow \hat{x} + L_i (y_i[n] - h(\hat{x}, u[n]))$.
- 16: **end if**

If R is diagonal (uncorrelated sensors), then update one measurement at a time.

Attitude Estimation Using EKF

Use the nonlinear propagation model

$$\begin{aligned}\dot{\phi} &= p + q \sin \phi \tan \theta + r \cos \phi \tan \theta + \xi_{\phi} \\ \dot{\theta} &= q \cos \phi - r \sin \phi + \xi_{\theta}\end{aligned}$$

where

$$\xi_{\phi} \sim \mathcal{N}(0, Q_{\phi}) \quad \text{and} \quad \xi_{\theta} \sim \mathcal{N}(0, Q_{\theta})$$

Use accelerometers as measured outputs:

$$y_{\text{accel}} = \begin{pmatrix} \dot{u} + qw - rv + g \sin \theta \\ \dot{v} + ru - pw - g \cos \theta \sin \phi \\ \dot{w} + pv - qu - g \cos \theta \cos \phi \end{pmatrix} + \eta_{\text{accel}}.$$

Problem: Do not have method for directly measuring \dot{u} , \dot{v} , \dot{w} , u , v , and w .

What do we do?

Attitude Estimation Using EKF

Assume that $\dot{u} = \dot{v} = \dot{w} \approx 0$

Recall that

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} \approx V_a \begin{pmatrix} \cos \alpha \cos \beta \\ \sin \beta \\ \sin \alpha \cos \beta \end{pmatrix}.$$

Assuming that $\alpha \approx \theta$ and $\beta \approx 0$ gives

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} \approx V_a \begin{pmatrix} \cos \theta \\ 0 \\ \sin \theta \end{pmatrix}$$

Substituting into original output equation

$$y_{\text{accel}} = \begin{pmatrix} \dot{u} + qw - rv + g \sin \theta \\ \dot{v} + ru - pw - g \cos \theta \sin \phi \\ \dot{w} + pv - qu - g \cos \theta \cos \phi \end{pmatrix} + \eta_{\text{accel}}$$

gives

$$y_{\text{accel}} = \begin{pmatrix} qV_a \sin \theta + g \sin \theta \\ rV_a \cos \theta - pV_a \sin \theta - g \cos \theta \sin \phi \\ -qV_a \cos \theta - g \cos \theta \cos \phi \end{pmatrix} + \eta_{\text{accel}}$$

Attitude Estimation Using EKF

Defining $x = (\phi, \theta)^\top$, $u = (p, q, r, V_a)^\top$, $\xi = (\xi_\phi, \xi_\theta)^\top$, and $\eta = (\eta_\phi, \eta_\theta)^\top$, and noting that there is noise on the gyros and pressure sensor, i.e., $u_{actual} = u + \xi_u$ where $\xi_u = (\xi_p, \xi_q, \xi_r, \xi_{V_a})^\top$, gives

$$\begin{aligned}\dot{x} &= f(x, u) + G(x)\xi_u + \xi \\ y &= h(x, u) + \eta,\end{aligned}$$

where

$$\begin{aligned}f(x, u) &= \begin{pmatrix} p + q \sin \phi \tan \theta + r \cos \phi \tan \theta \\ q \cos \phi - r \sin \phi \end{pmatrix} \\ h(x, u) &= \begin{pmatrix} qV_a \sin \theta + g \sin \theta \\ rV_a \cos \theta - pV_a \sin \theta - g \cos \theta \sin \phi \\ -qV_a \cos \theta - g \cos \theta \cos \phi \end{pmatrix} \\ G(x) &= \begin{pmatrix} 1 & \sin \phi \tan \theta & \cos \phi \tan \theta & 0 \\ 0 & \cos \phi & -\sin \phi & 0 \end{pmatrix}.\end{aligned}$$

Note that

$$E\{(G\xi_u + \xi)(G\xi_u + \xi)^\top\} = GE\{\xi_u\xi_u^\top\} + E\{\xi\xi^\top\} = GQ_uG^\top + Q$$

where we have assumed that $E\{\xi_u\xi_u^\top\} = Q_u$.

Attitude Estimation Using EKF

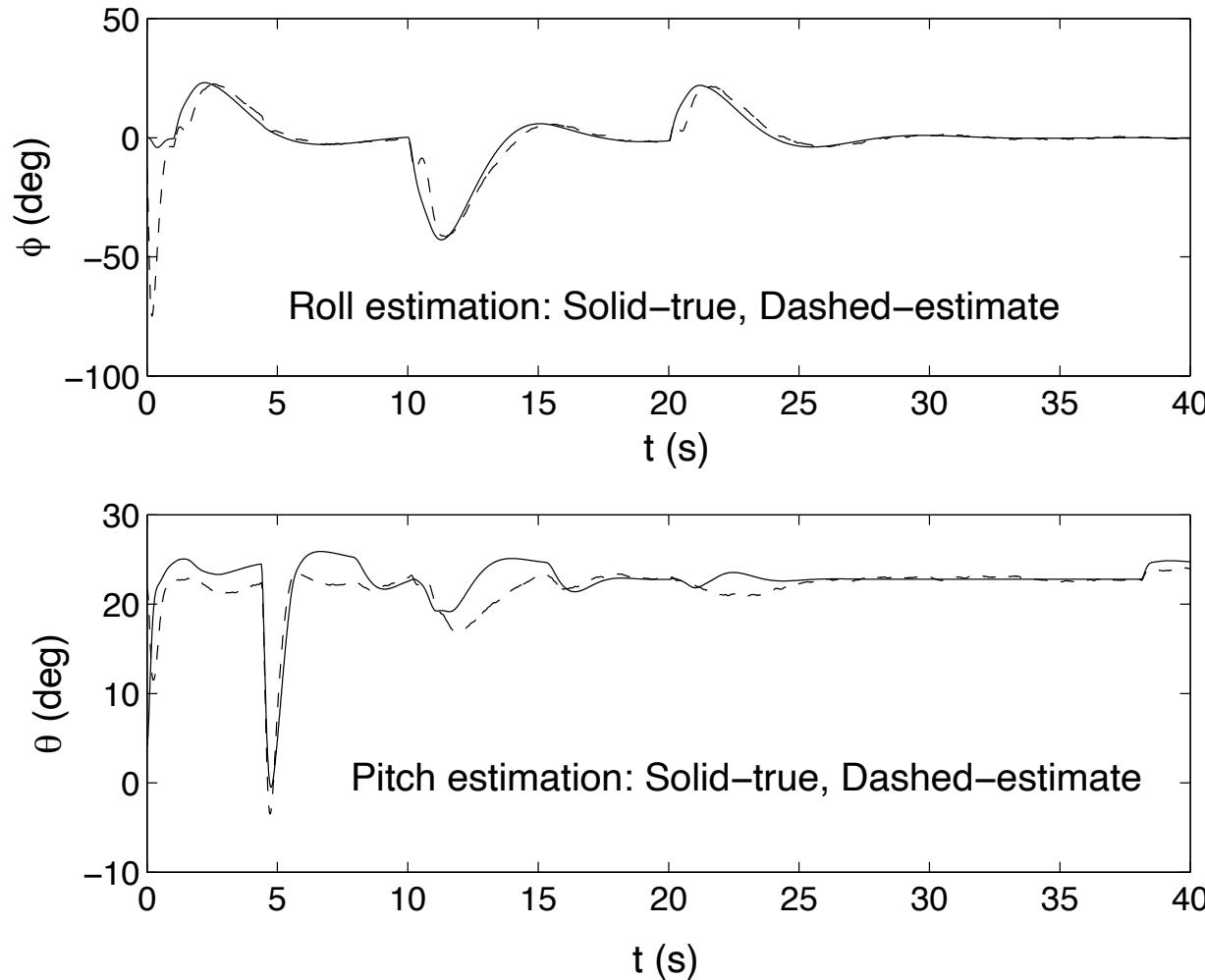
Implementation of Kalman filter requires Jacobians $\frac{\partial f}{\partial x}$ and $\frac{\partial h}{\partial x}$. Accordingly,

$$\frac{\partial f}{\partial x} = \begin{pmatrix} q \cos \phi \tan \theta - r \sin \phi \tan \theta & \frac{q \sin \phi - r \cos \phi}{\cos^2 \theta} \\ -q \sin \phi - r \cos \phi & 0 \end{pmatrix}$$
$$\frac{\partial h}{\partial x} = \begin{pmatrix} 0 & qV_a \cos \theta + g \cos \theta \\ -g \cos \phi \cos \theta & -rV_a \sin \theta - pV_a \cos \theta + g \sin \phi \sin \theta \\ g \sin \phi \cos \theta & (qV_a + g \cos \phi) \sin \theta \end{pmatrix}.$$

At this point, we have everything we need to implement the continuous-discrete EKF.

How well does it work?

Attitude Estimation Results



Not perfect, but significantly better!

GPS Smoothing

- Want to fill in estimates between GPS measurements
- Want to estimate wind

Assuming level flight, evolution of position:

$$\begin{aligned}\dot{p}_n &= V_g \cos \chi \\ \dot{p}_e &= V_g \sin \chi\end{aligned}$$

Evolution of the ground speed:

$$\begin{aligned}\dot{V}_g &= \frac{d}{dt} \sqrt{(V_a \cos \psi + w_n)^2 + (V_a \sin \psi + w_e)^2} \\ &= \frac{(V_a \cos \psi + w_n)(\dot{V}_a \cos \psi - V_a \dot{\psi} \sin \psi + \dot{w}_n) + (V_a \sin \psi + w_e)(\dot{V}_a \sin \psi + V_a \dot{\psi} \cos \psi + \dot{w}_e)}{V_g}\end{aligned}$$

Assuming that wind and airspeed are constant:

$$\dot{V}_g = \frac{(V_a \cos \psi + w_n)(-V_a \dot{\psi} \sin \psi) + (V_a \sin \psi + w_e)(V_a \dot{\psi} \cos \psi)}{V_g}$$

GPS Smoothing

Evolution of χ :

$$\dot{\chi} = \frac{g}{V_g} \tan \phi \cos(\chi - \psi)$$

Assuming that wind is constant:

$$\dot{w}_n = 0$$

$$\dot{w}_e = 0$$

From kinematics, evolution of ψ :

$$\dot{\psi} = q \frac{\sin \phi}{\cos \theta} + r \frac{\cos \phi}{\cos \theta}$$

GPS Smoothing

Define:

$$x = (p_n, p_e, V_g, \chi, w_n, w_e, \psi)^\top$$

$$u = (V_a, q, r, \phi, \theta)^\top:$$

$$f(x, u) \triangleq \begin{pmatrix} V_g \cos \chi \\ V_g \sin \chi \\ \frac{(V_a \cos \psi + w_n)(-V_a \dot{\psi} \sin \psi) + (V_a \sin \psi + w_e)(V_a \dot{\psi} \cos \psi)}{V_g} \\ \frac{g}{V_g} \tan \phi \cos(\chi - \psi) \\ 0 \\ 0 \\ q \frac{\sin \phi}{\cos \theta} + r \frac{\cos \phi}{\cos \theta} \end{pmatrix}$$

GPS Smoothing

Define:

$$x = (p_n, p_e, V_g, \chi, w_n, w_e, \psi)^\top$$

$$u = (V_a, q, r, \phi, \theta)^\top$$

Jacobian of f :

$$\frac{\partial f}{\partial x} = \begin{pmatrix} 0 & 0 & \cos \chi & -V_g \sin \chi & 0 & 0 & 0 \\ 0 & 0 & \sin \chi & V_g \cos \chi & 0 & 0 & 0 \\ 0 & 0 & -\frac{\dot{V}_g}{V_g} & 0 & -\dot{\psi} V_a \sin \psi & \dot{\psi} V_a \cos \psi & \frac{\partial \dot{V}_g}{\partial \psi} \\ 0 & 0 & \frac{\partial \dot{\chi}}{\partial V_g} & \frac{\partial \dot{\chi}}{\partial \chi} & 0 & 0 & \frac{\partial \dot{\chi}}{\partial \psi} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

where

$$\frac{\partial \dot{V}_g}{\partial \psi} = \frac{-\dot{\psi} V_a (w_n \cos \psi + w_e \sin \psi)}{V_g}$$

$$\frac{\partial \dot{\chi}}{\partial V_g} = -\frac{g}{V_g^2} \tan \phi \cos(\chi - \psi)$$

$$\frac{\partial \dot{\chi}}{\partial \chi} = -\frac{g}{V_g} \tan \phi \sin(\chi - \psi)$$

$$\frac{\partial \dot{\chi}}{\partial \psi} = \frac{g}{V_g} \tan \phi \sin(\chi - \psi)$$

GPS Smoothing

Define:

$$x = (p_n, p_e, V_g, \chi, w_n, w_e, \psi)^\top$$

$$u = (V_a, q, r, \phi, \theta)^\top$$

For measurements, use GPS signals for north and east position (p_n, p_e), ground speed (V_g), and course (χ)

Notice that states are not independent – related by wind triangle

Use wind triangle relations to introduce two pseudo measurements.

Assume $\gamma = \gamma_a = 0$:

$$V_a \cos \psi + w_n = V_g \cos \chi$$

$$V_a \sin \psi + w_e = V_g \sin \chi$$

From these expressions, define the pseudo measurements

$$y_{\text{windtri},n} = V_a \cos \psi + w_n - V_g \cos \chi$$

$$y_{\text{windtri},e} = V_a \sin \psi + w_e - V_g \sin \chi$$

where (pseudo) measurement values are equal to zero.

GPS Smoothing

The resulting measurement model is given by

$$y_{GPS} = h(x, u) + \eta_{GPS}$$

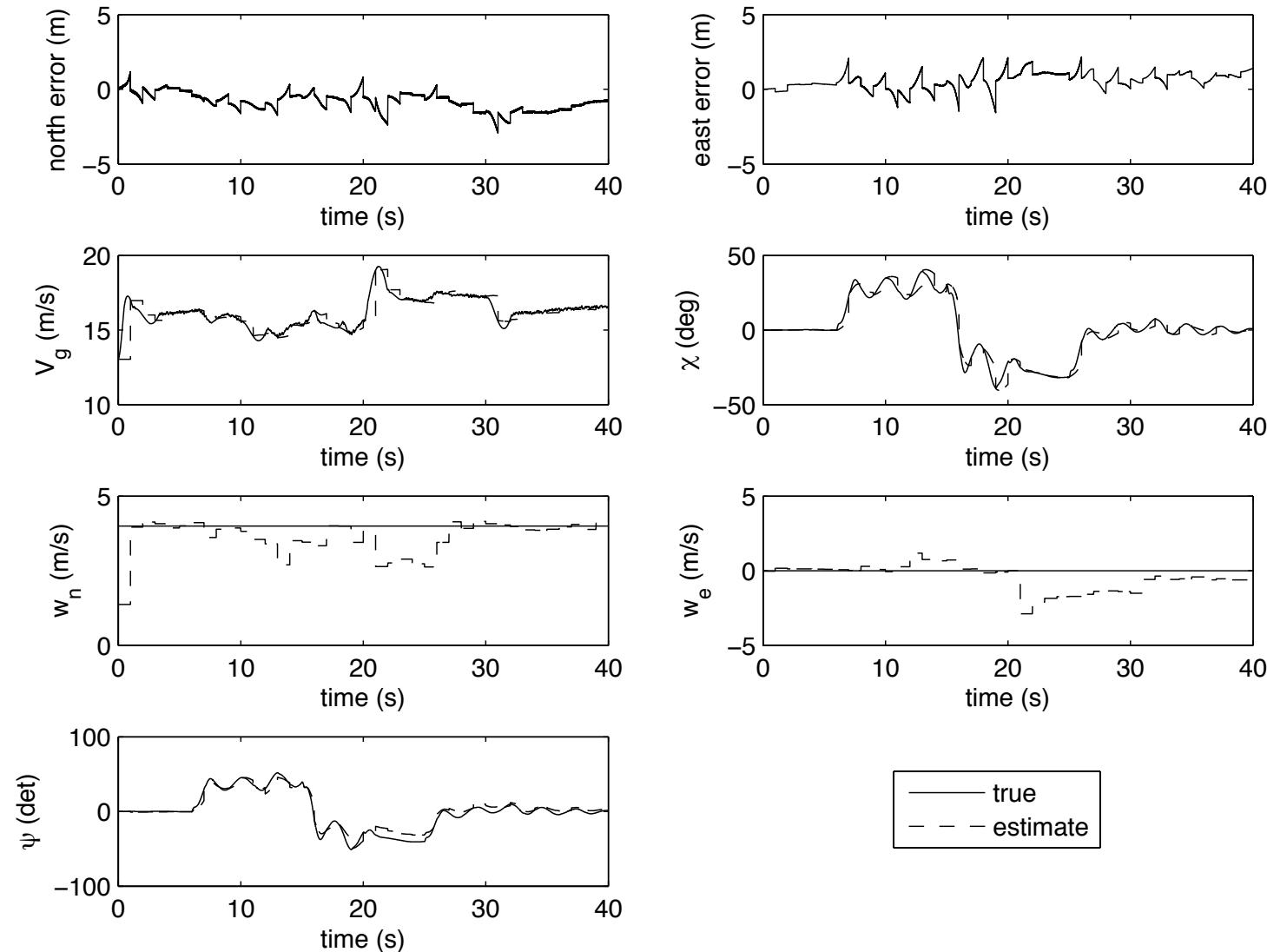
where $y_{GPS} = (y_{GPS,n}, y_{GPS,e}, y_{GPS,V_g}, y_{GPS,\chi}, y_{wind,n}, y_{wind,e})$, and

$$h(x, u) = \begin{pmatrix} p_n \\ p_e \\ V_g \\ \chi \\ V_a \cos \psi + w_n - V_g \cos \chi \\ V_a \sin \psi + w_e - V_g \sin \chi \end{pmatrix}$$

and where the Jacobian is given by

$$\frac{\partial h}{\partial x}(\hat{x}, u) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -\cos \chi & V_g \sin \chi & 1 & 0 & -V_a \sin \psi \\ 0 & 0 & -\sin \chi & -V_g \cos \chi & 0 & 1 & V_a \cos \psi \end{pmatrix}$$

GPS Smoothing Results



Measurement Gating

Similar to least squares, measurement outliers can negatively impact performance of the Kalman Filter.

However, the KF/EKF provides a way to detect outliers.

Define the innovation sequence:

$$v_k = y_k - C\hat{x}_k^- = Cx_k + \eta_k - C\hat{x}_k^- = C\tilde{x}_k^- + \eta_k.$$

The covariance of the innovation sequence is

$$S_k = E\{v_k v_k^\top\} = E\{(\eta_k + C\tilde{x}_k^-)(\eta_k + C\tilde{x}_k^-)^\top\} = R + CP_k^-C^\top.$$

Measurement Gating

The random variable

$$z_k = (y_k - h(x_k))^T S_k^{-1} (y_k - h(x_k)),$$

is therefore a χ^2 (chi-squared) random variable with m degrees-of-freedom.

Therefore, compute the likelihood that z_k is drawn from a $\chi^2(m)$ distribution, and do a measurement update if the likelihood is about a threshold.

```
import numpy as np
from scipy import stats
self.accel_threshold = stats.chi2.isf(q=0.01, df=3)
def measurement_update(self, measurement, state):
    # measurement updates
    h = self.h(self.xhat, measurement, state)
    C = jacobian(self.h, self.xhat, measurement, state)
    y = np.array([[measurement.accel_x,
                  measurement.accel_y,
                  measurement.accel_z]]).T
    S_inv = np.linalg.inv(self.R_accel + C @ self.P @ C.T)
    if (y - h).T @ S_inv @ (y - h) < self.accel_threshold:
        L = self.P @ C.T @ S_inv
        tmp = np.eye(2) - L @ C
        self.P = tmp @ self.P @ tmp.T + L @ self.R_accel @ L.T
        self.xhat = self.xhat + L @ (y - h)
```

Suggestions for Tuning Your EKF

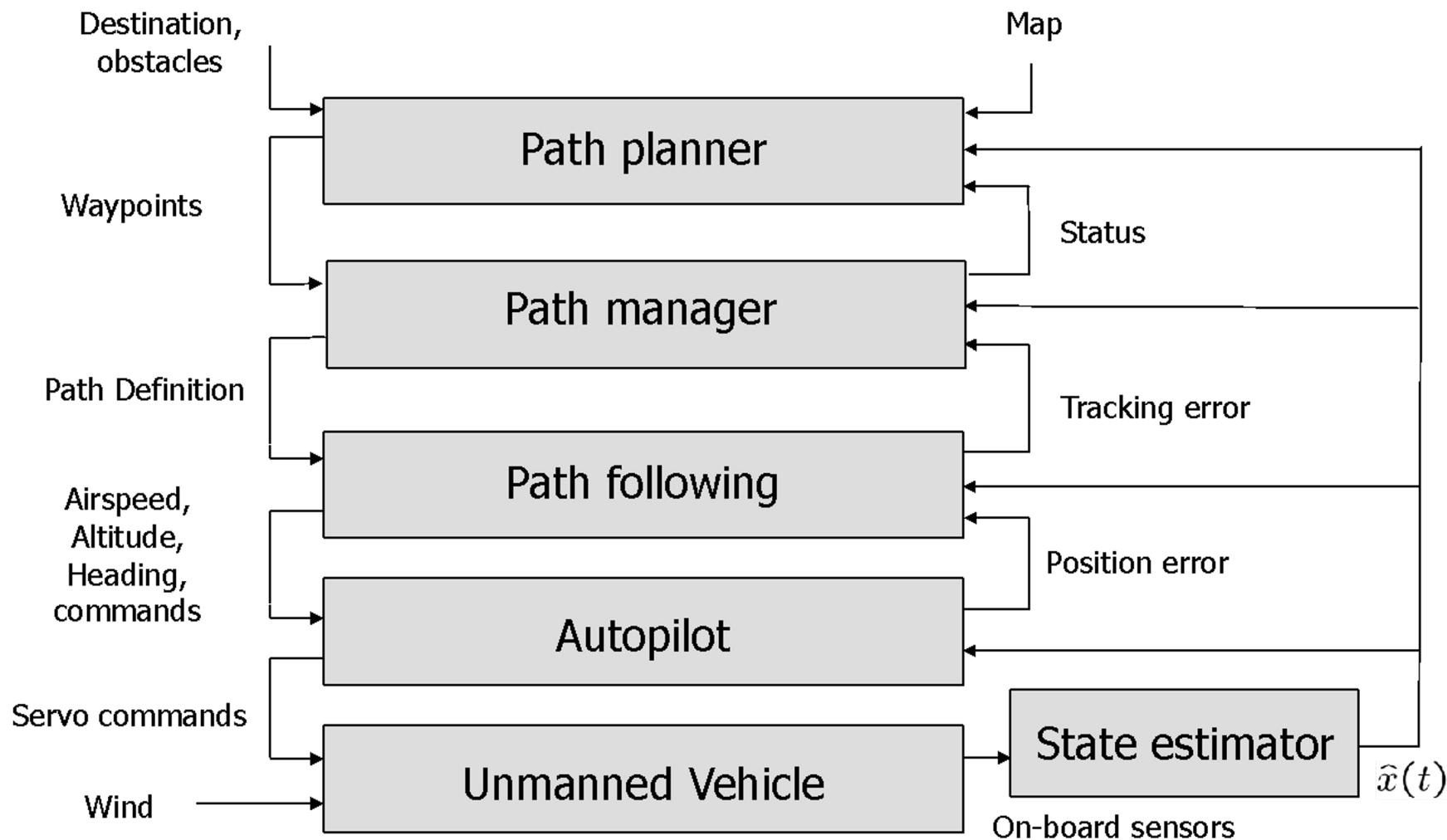
- Implement EKF in steps
 - Attitude estimation
 - GPS smoother
- Test and tune each component independently and thoroughly
 - Attitude estimator first
 - GPS smoother second
- As a first step, make sure your filter estimates track the states when sensors are perfect (no sensor error). This will expose coding errors and show the limits of performance of your filter.
- Keep the wind set to zero until you are confident that your filter is well tuned.
- We know R pretty well typically. Tune filter by changing Q.
- Tune filter by focusing on individual states. Give inputs to those states while keeping other states “quiet”. Adjust Q value corresponding to state of interest. Tune by trial and error (use your brain to make a good guess!).
- Don’t put extreme inputs into the system when tuning the filter (e.g., large steps). Your filter will have its own dynamic limits – don’t make the problem impossibly difficult.
- Tune your controllers so that the response of the aircraft to typical inputs is smooth and graceful. Abrupt and jerky state dynamics are difficult to estimate.
- Check your equations AGAIN!



Chapter 9

Nonlinear Design Models

Architecture



Simplifying Dynamic Models

- When designing higher level autopilot functions, we need models that are easier to analyze and simulate
- Models must capture the essential behavior of system
- We will derive reduced-order, reduced-complexity models suitable for design of higher-level guidance strategies
- Two types of guidance models:
 - Kinematic – utilize kinematic relationships, do not consider aerodynamics, forces directly
 - Dynamic – apply force balance relations to point-mass models

Autopilot Models (transfer function)

Airspeed hold and roll hold loops can be modeled as:

$$V_a(s) = \frac{b_{V_a}}{s + b_{V_a}} V_a^c(s) \quad \phi(s) = \frac{b_\phi}{s + b_\phi} \phi^c(s)$$

Altitude and course hold loops can be modeled as:

$$h(s) = \frac{b_h s + b_h}{s^2 + b_{\dot{h}} s + b_h} h^c(s) \quad \chi(s) = \frac{b_{\dot{\chi}} s + b_\chi}{s^2 + b_{\dot{\chi}} s + b_\chi} \chi^c(s)$$

Alternatively, the heading hold loop can be modeled as:

$$\psi(s) = \frac{b_\psi s + b_\psi}{s^2 + b_{\dot{\psi}} s + b_\psi} \psi^c(s)$$

Flight-path angle and load factor loops can be modeled as:

$$\gamma(s) = \frac{b_\gamma}{s + b_\gamma} \gamma^c(s) \quad n_{lf}(s) = \frac{b_n}{s + b_n} n_{lf}^c(s)$$

Autopilot Models

Airspeed hold and roll hold loops can be modeled as:

$$\begin{aligned}\dot{V}_a &= b_{V_a}(V_a^c - V_a) \\ \dot{\phi} &= b_\phi(\phi^c - \phi)\end{aligned}$$

Altitude and course hold loops can be modeled as:

$$\begin{aligned}\ddot{h} &= b_{\dot{h}}(\dot{h}^c - \dot{h}) + b_h(h^c - h) \\ \ddot{\chi} &= b_{\dot{\chi}}(\dot{\chi}^c - \dot{\chi}) + b_\chi(\chi^c - \chi)\end{aligned}$$

Alternatively, the heading hold loop can be modeled as:

$$\ddot{\psi} = b_{\dot{\psi}}(\dot{\psi}^c - \dot{\psi}) + b_\psi(\psi^c - \psi)$$

Flight-path angle and load factor loops can be modeled as:

$$\begin{aligned}\dot{\gamma} &= b_\gamma(\gamma^c - \gamma) \\ \dot{n}_{lf} &= b_n(n_{lf}^c - n_{lf})\end{aligned}$$

Kinematic Model of Controlled Flight

The velocity vector can be written as

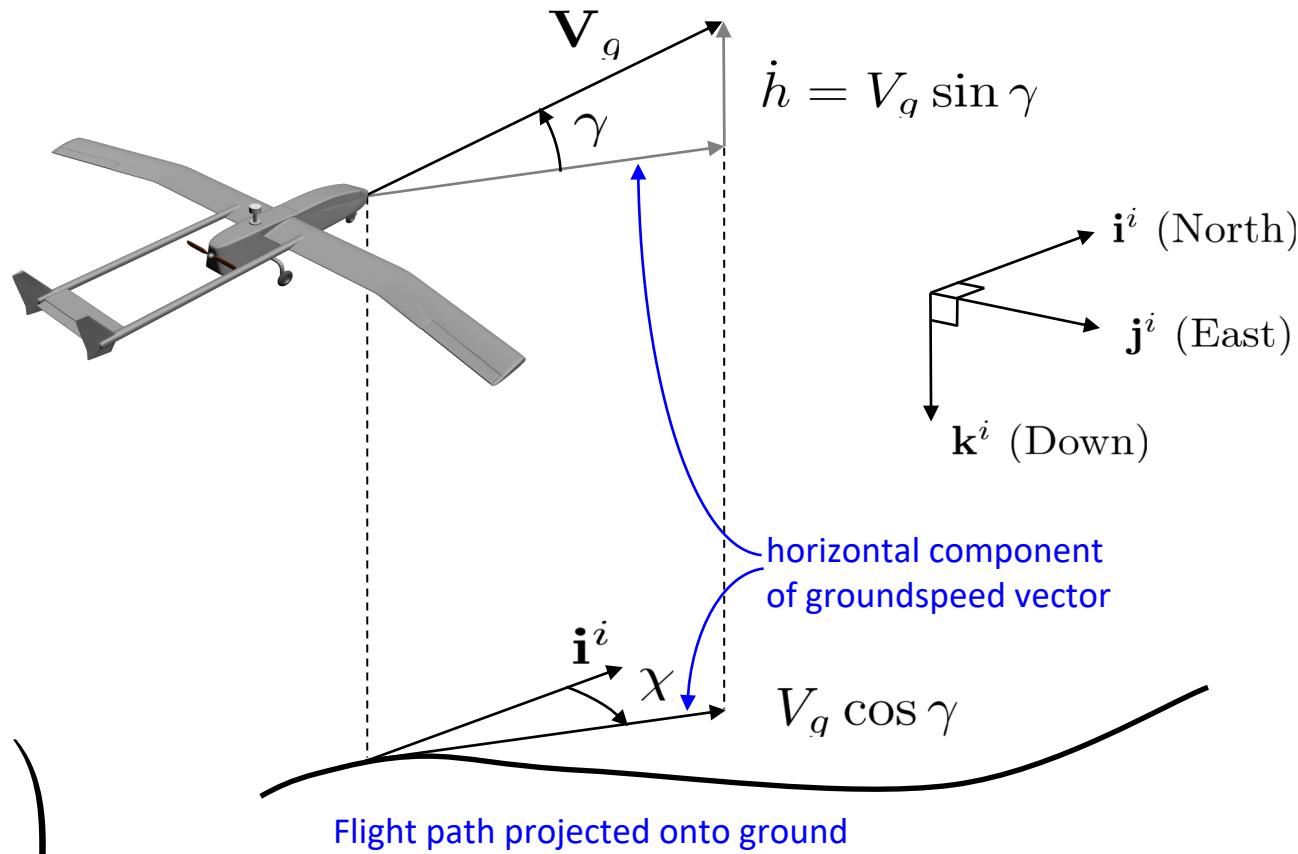
$$\mathbf{V}_g^i = V_g \begin{pmatrix} \cos \chi \cos \gamma \\ \sin \chi \cos \gamma \\ -\sin \gamma \end{pmatrix}$$

which gives:

$$\begin{pmatrix} \dot{p}_n \\ \dot{p}_e \\ \dot{h} \end{pmatrix} = V_g \begin{pmatrix} \cos \chi \cos \gamma \\ \sin \chi \cos \gamma \\ \sin \gamma \end{pmatrix}$$

Alternatively, using the Wind Triangle:

$$\begin{pmatrix} \dot{p}_n \\ \dot{p}_e \\ \dot{h} \end{pmatrix} = V_a \begin{pmatrix} \cos \psi \cos \gamma_a \\ \sin \psi \cos \gamma_a \\ \sin \gamma_a \end{pmatrix} + \begin{pmatrix} w_n \\ w_e \\ -w_d \end{pmatrix}$$



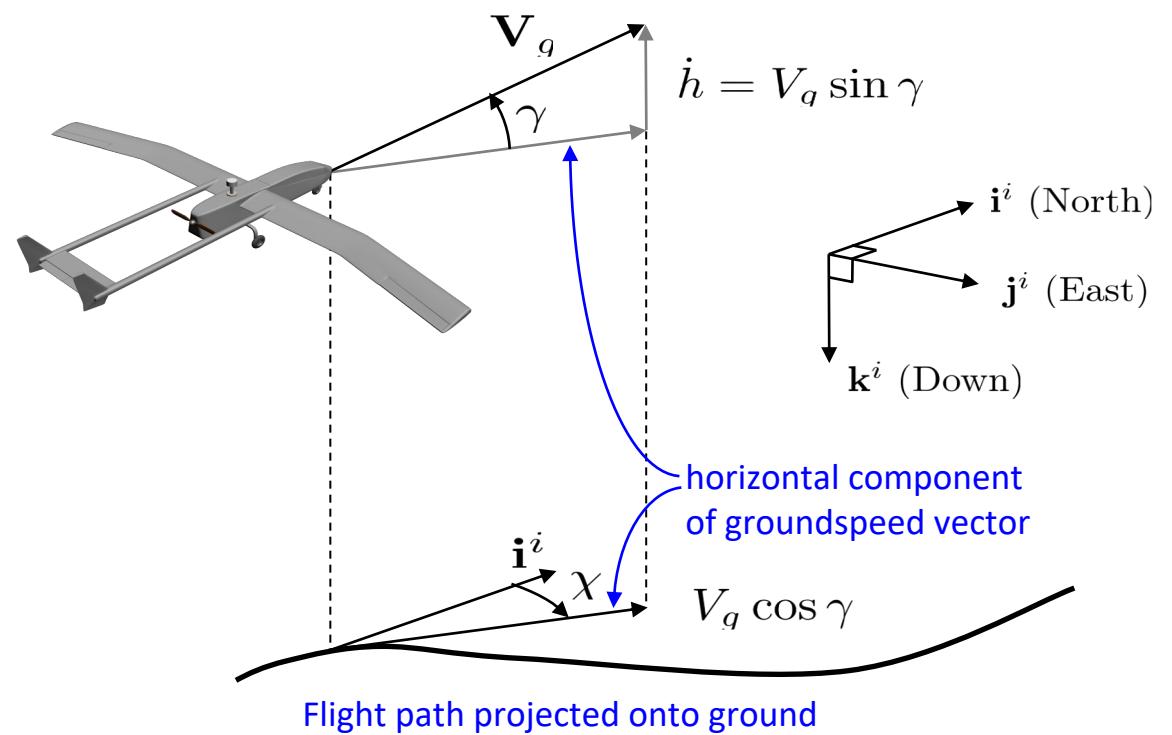
Kinematic Model of Controlled Flight

Beginning with

$$\begin{pmatrix} \dot{p}_n \\ \dot{p}_e \\ \dot{h} \end{pmatrix} = V_a \begin{pmatrix} \cos \psi \cos \gamma_a \\ \sin \psi \cos \gamma_a \\ \sin \gamma_a \end{pmatrix} + \begin{pmatrix} w_n \\ w_e \\ -w_d \end{pmatrix}$$

and assuming level flight, i.e., $\gamma_a = 0$,
and no down component of wind,
i.e., $w_d = 0$:

$$\begin{pmatrix} \dot{p}_n \\ \dot{p}_e \\ \dot{h} \end{pmatrix} = V_a \begin{pmatrix} \cos \psi \\ \sin \psi \\ 0 \end{pmatrix} + \begin{pmatrix} w_n \\ w_e \\ 0 \end{pmatrix}$$



This equation is typically called the Dubin's car model.

Coordinated Turn

From Chapter 2 we have

$$\dot{\chi} = \frac{g}{V_g} \tan \phi \cos(\chi - \psi)$$

For constant-altitude flight with no down component of wind:

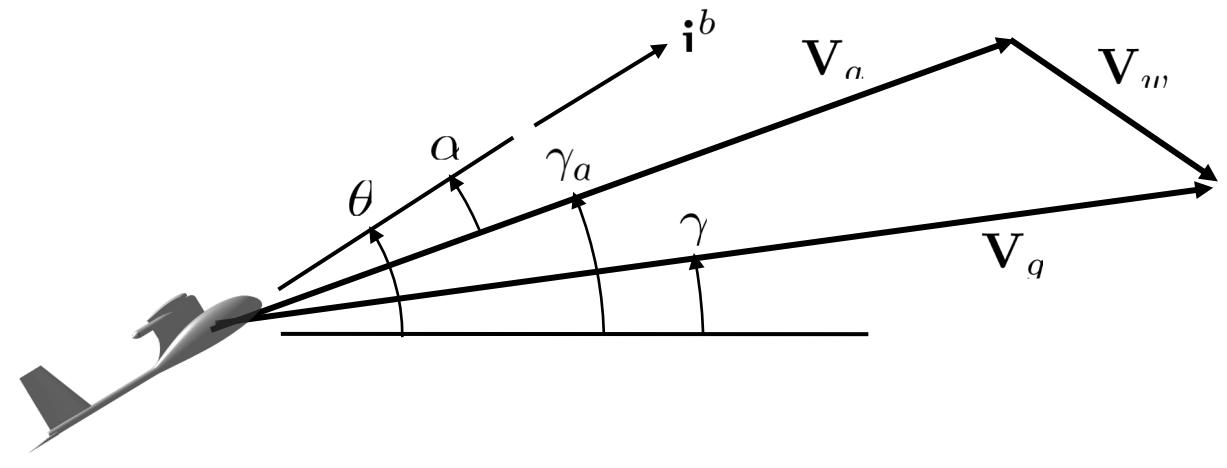
$$\dot{V}_g = \frac{\dot{V}_a}{\cos(\chi - \psi)} + V_g \dot{\chi} \tan(\chi - \psi)$$

$$\dot{\psi} = \frac{\dot{V}_a}{V_a} \tan(\chi - \psi) + \frac{V_g \dot{\chi}}{V_a \cos(\chi - \psi)}$$

If airspeed is constant, then we get the standard coordinated turn condition

$$\dot{\psi} = \frac{g}{V_a} \tan \phi$$

which is true even if when $w_n, w_e \neq 0$.



Differentiate both sides of vector wind-triangle equation (2.9). Solve resulting messy matrix equation.

Accelerating Climb

Summing forces gives

$$F_{\text{lift}} \cos \phi = m V_g \dot{\gamma} + m g \cos \gamma$$

Solving for $\dot{\gamma}$:

$$\dot{\gamma} = \frac{g}{V_g} \left(\frac{F_{\text{lift}}}{m g} \cos \phi - \cos \gamma \right)$$

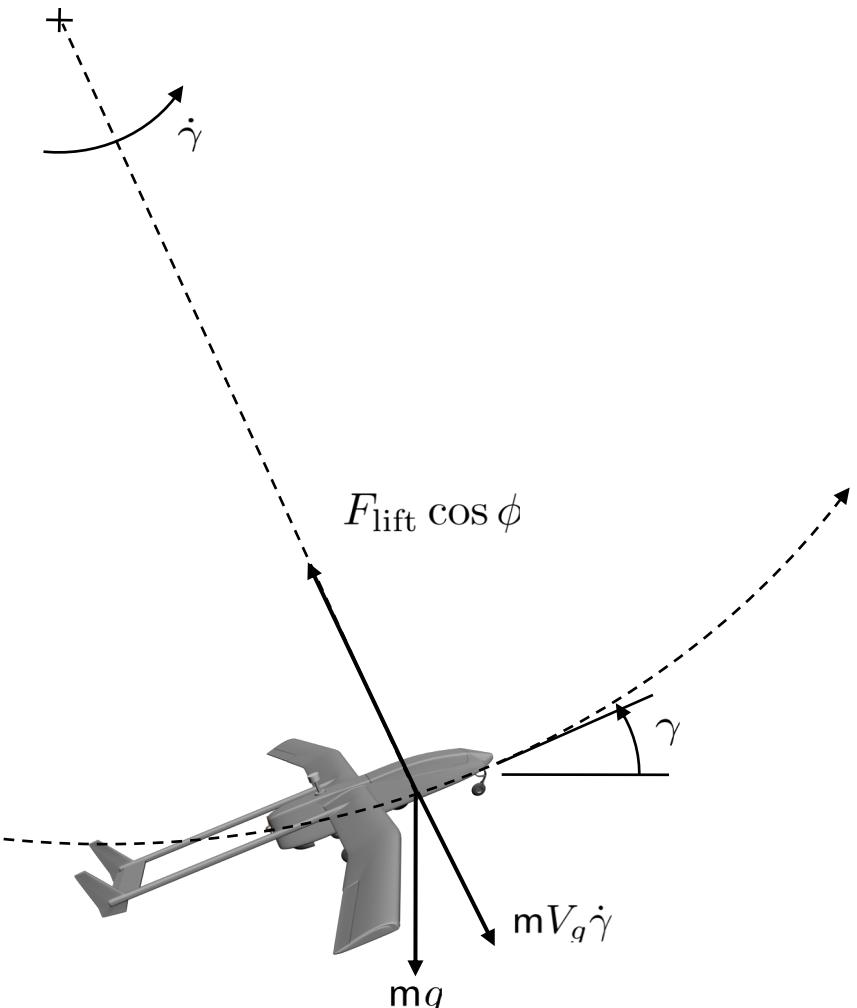
The load factor (often expressed in g 's) is defined as

$$n_{lf} \triangleq F_{\text{lift}} / m g.$$

Note that for wings level, horizontal flight $n_{lf} = 1$. Therefore

$$\dot{\gamma} = \frac{g}{V_g} (n_{lf} \cos \phi - \cos \gamma)$$

In a constant climb where $\dot{\gamma} = 0$, we have $n_{lf} = \frac{\cos \gamma}{\cos \phi}$.



Kinematic Guidance Models

- Several guidance models can be derived, with varying levels of fidelity
- Choice of model depends on application

Kinematic Guidance Model - #1a

The simplest model (and the one used in Chapters 10, 11, 12) is given by

$$\dot{p}_n = V_a \cos \psi + w_n$$

$$\dot{p}_e = V_a \sin \psi + w_e$$

$$\ddot{\chi} = b_{\dot{\chi}}(\dot{\chi}^c - \dot{\chi}) + b_\chi(\chi^c - \chi)$$

$$\ddot{h} = b_{\dot{h}}(\dot{h}^c - \dot{h}) + b_h(h^c - h)$$

$$\dot{V}_a = b_{V_a}(\mathcal{V}_{\mathbf{a}}^c - V_a)$$

where ψ is given by (equation (2.12))

$$\psi = \chi - \sin^{-1} \left(\frac{1}{V_a} \begin{pmatrix} w_n \\ w_e \end{pmatrix}^\top \begin{pmatrix} -\sin \chi \\ \cos \chi \end{pmatrix} \right)$$

Kinematic Guidance Model - #1b

If the autopilot is designed to command heading instead of course, then the model becomes

$$\dot{p}_n = V_a \cos \psi + \textcolor{blue}{w}_n$$

$$\dot{p}_e = V_a \sin \psi + \textcolor{blue}{w}_e$$

$$\ddot{\psi} = b_{\dot{\psi}}(\dot{\psi}^c - \dot{\psi}) + b_\psi(\textcolor{blue}{\psi^c} - \psi)$$

$$\ddot{h} = b_{\dot{h}}(\dot{h}^c - \dot{h}) + b_h(\textcolor{blue}{h^c} - h)$$

$$\dot{V}_a = b_{V_a}(\textcolor{blue}{V_a^c} - V_a)$$

Kinematic Guidance Models - #2a

A more accurate model is to command the roll angle and use the coordinated turn model for χ :

$$\dot{p}_n = V_a \cos \psi + w_n$$

$$\dot{p}_e = V_a \sin \psi + w_e$$

$$\dot{\chi} = \frac{g}{V_g} \tan \phi \cos(\chi - \psi)$$

$$\ddot{h} = b_{\dot{h}}(\dot{h}^c - \dot{h}) + b_h(h^c - h)$$

$$\dot{V}_a = b_{V_a}(V_a^c - V_a)$$

$$\dot{\phi} = b_\phi(\phi^c - \phi)$$

where ψ and V_g are given by (equations (2.10) and (2.12))

$$\psi = \chi - \sin^{-1} \left(\frac{1}{V_a} \begin{pmatrix} w_n \\ w_e \end{pmatrix}^\top \begin{pmatrix} -\sin \chi \\ \cos \chi \end{pmatrix} \right)$$

$$V_g = w_n \cos \chi + w_e \sin \chi + \sqrt{(w_n \cos \chi + w_e \sin \chi)^2 + V_a^2 - w_n^2 - w_e^2}$$

Kinematic Guidance Models - #2b

Or, in terms of heading we have

$$\dot{p}_n = V_a \cos \psi + w_n$$

$$\dot{p}_e = V_a \sin \psi + w_e$$

$$\dot{\psi} = \frac{g}{V_a} \tan \phi$$

$$\ddot{h} = b_h(\dot{h}^c - \dot{h}) + b_h(h^c - h)$$

$$\dot{V}_a = b_{V_a}(V_a^c - V_a)$$

$$\dot{\phi} = b_\phi(\phi^c - \phi)$$

where χ and V_g are computed from the wind triangle if needed by the autopilot

Kinematic Guidance Models - #3

More accurate still is to command the flight path angle

$$\dot{p}_n = V_a \cos \psi \cos \gamma_a + w_n$$

$$\dot{p}_e = V_a \sin \psi \cos \gamma_a + w_e$$

$$\dot{h} = V_a \sin \gamma_a - w_d$$

$$\dot{\psi} = \frac{g}{V_a} \tan \phi$$

$$\dot{\gamma} = b_\gamma (\gamma^c - \gamma)$$

$$\dot{V}_a = b_{V_a} (V_a^c - V_a)$$

$$\dot{\phi} = b_\phi (\phi^c - \phi)$$

where γ_a is given by equation (2.11)

$$\gamma_a = \sin^{-1} \left(\frac{V_g \sin \gamma + w_d}{V_a} \right)$$

and χ and V_g are computed from the wind triangle if needed by the autopilot

Kinematic Guidance Models - #4

More accurate still is to command the load factor

$$\dot{p}_n = V_a \cos \psi \cos \gamma_a + w_n$$

$$\dot{p}_e = V_a \sin \psi \cos \gamma_a + w_e$$

$$\dot{h} = V_a \sin \gamma_a - w_d$$

$$\dot{\psi} = \frac{g}{V_a} \tan \phi$$

$$\dot{\gamma} = \frac{g}{V_g} (n_{lf} \cos \phi - \cos \gamma)$$

$$\dot{V}_a = b_{V_a} (V_a^c - V_a)$$

$$\dot{\phi} = b_\phi (\phi^c - \phi)$$

$$\dot{n}_{lf} = b_n (n_{lf}^c - n_{lf})$$

where χ , V_g , and γ_a are computed from the wind triangle

Dubins Airplane Model

The Dubin's airplane model is a particularly simple model often used in path planning:

$$\dot{r}_n = V \cos \psi \cos \gamma^c$$

$$\dot{r}_e = V \sin \psi \cos \gamma^c$$

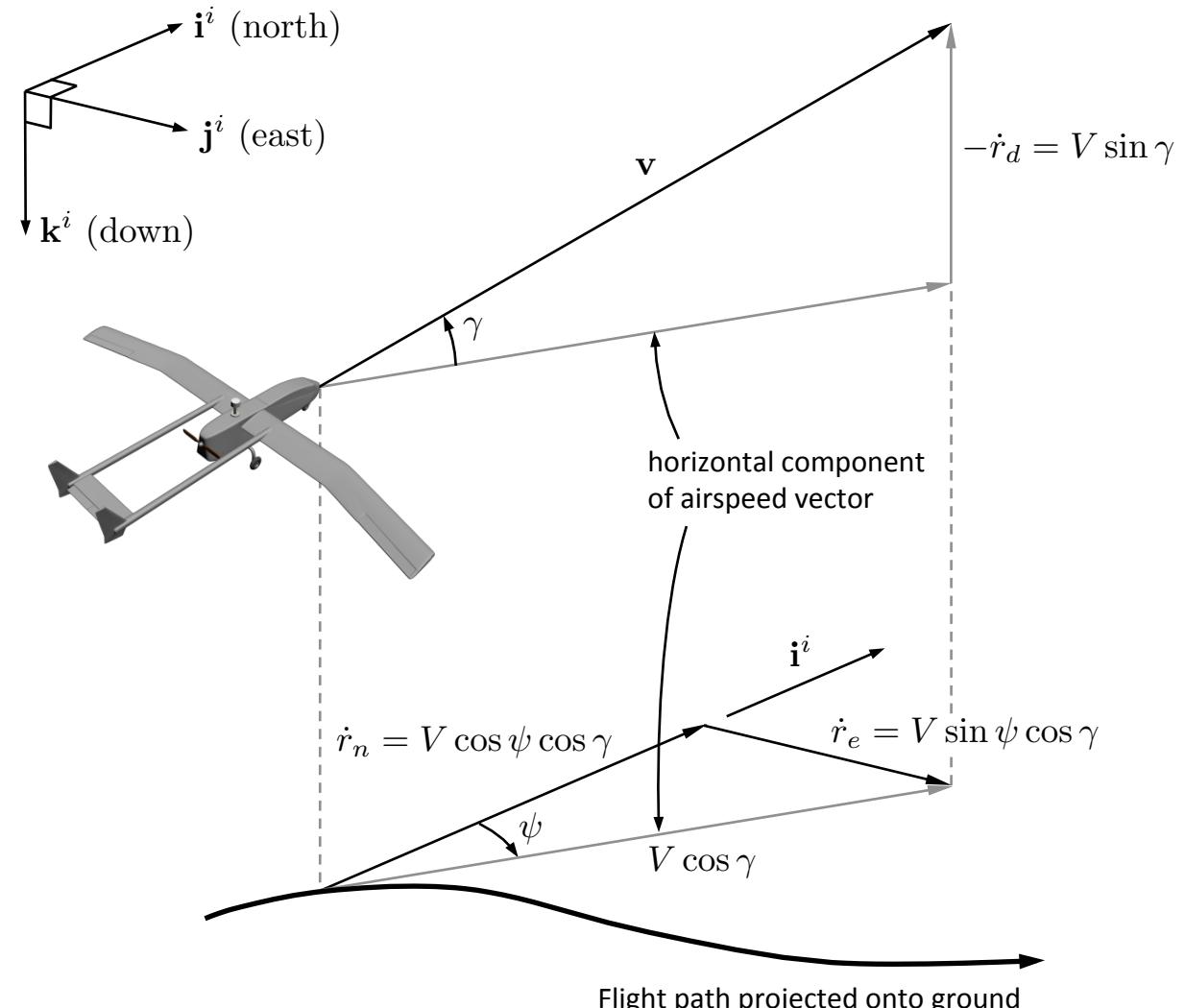
$$\dot{r}_d = -V \sin \gamma^c$$

$$\dot{\psi} = \frac{g}{V} \tan \phi^c.$$

Assumes constant velocity V , and subject the constraints:

$$|\phi^c| \leq \bar{\phi}$$

$$|\gamma^c| \leq \bar{\gamma}.$$



Dubins Airplane Model (simplified)

Often simplified as:

$$\dot{r}_n = V \cos \psi$$

$$\dot{r}_e = V \sin \psi$$

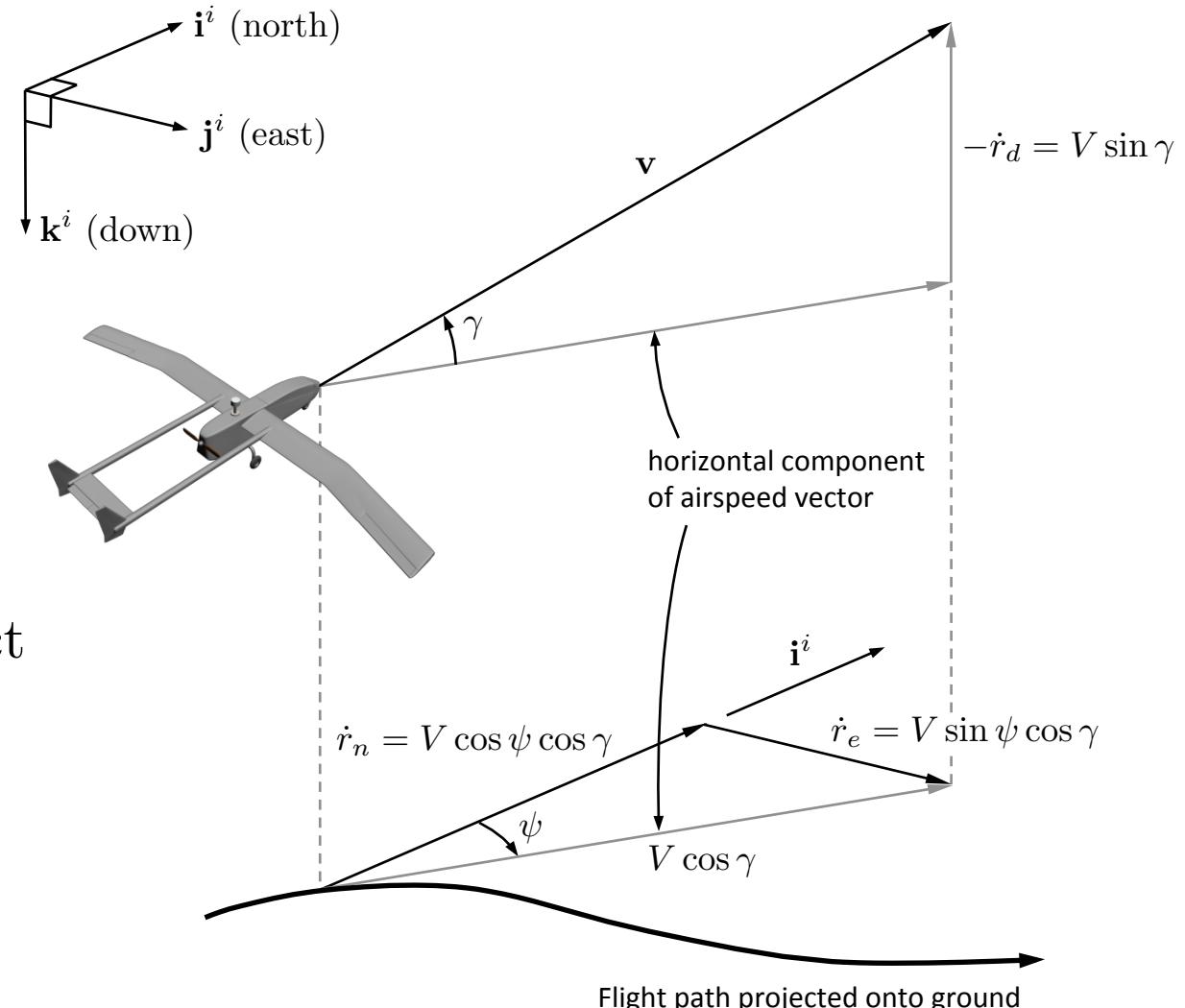
$$\dot{r}_d = -V \sin \gamma^c$$

$$\dot{\psi} = \frac{g}{V} \tan \phi^c.$$

Assumes constant velocity V , and subject
the constraints:

$$|\phi^c| \leq \bar{\phi}$$

$$|\gamma^c| \leq \bar{\gamma}.$$



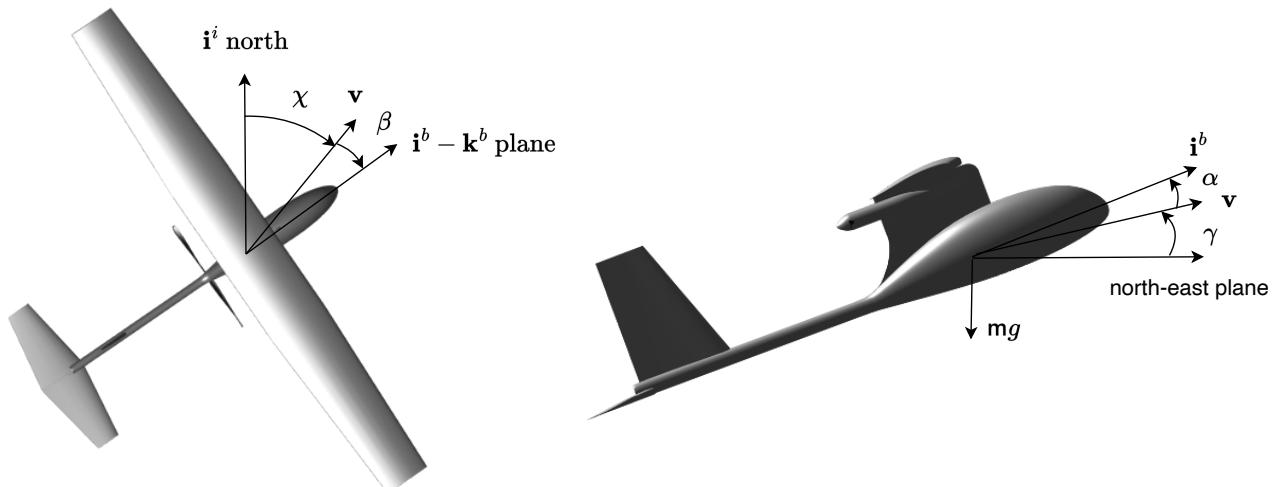
Dynamic Guidance Models

Standard kinematic expression
for evolution of position:

$$\dot{p}_n = V_g \cos \chi \cos \gamma$$

$$\dot{p}_e = V_g \sin \chi \cos \gamma$$

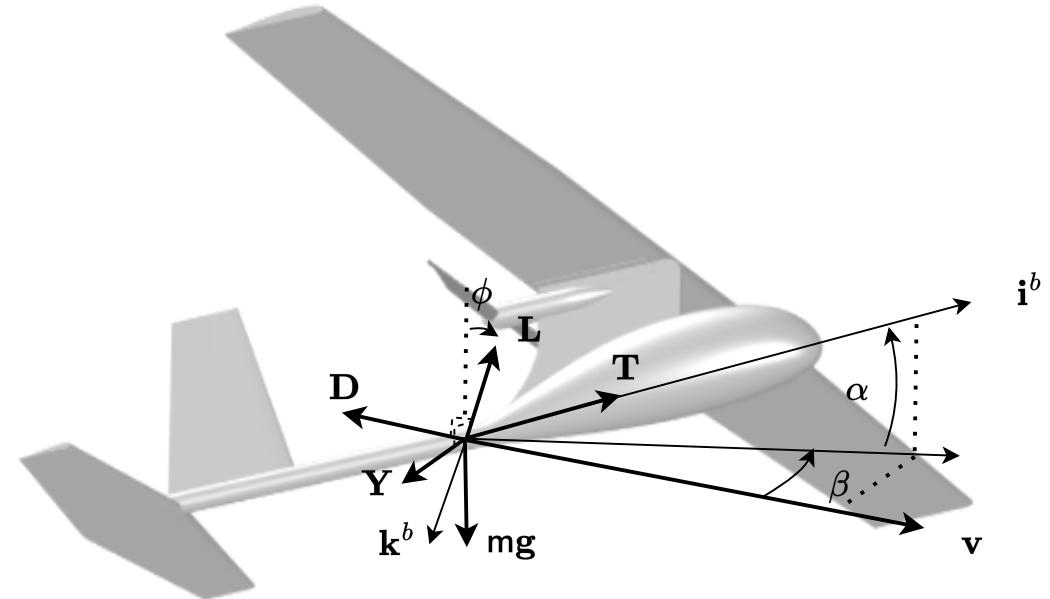
$$\dot{p}_d = -V_g \sin \gamma.$$



Dynamic Guidance Models

Thrust in the wind frame:

$$\begin{aligned}\mathbf{T}^w &= R_b^w \mathbf{T}^b \\ &= \begin{pmatrix} \cos \alpha \cos \beta & \sin \beta & \sin \alpha \cos \beta \\ -\cos \alpha \sin \beta & \cos \beta & -\sin \alpha \sin \beta \\ -\sin \alpha & 0 & \cos \alpha \end{pmatrix} \begin{pmatrix} T \\ 0 \\ 0 \end{pmatrix} \\ &= \begin{pmatrix} T \cos \alpha \cos \beta \\ -T \cos \alpha \sin \beta \\ -T \sin \alpha \end{pmatrix}.\end{aligned}$$



Aerodynamic forces in the wind frame:

$$\mathbf{F}_{aero}^w = \begin{pmatrix} -D \\ -Y \\ -L \end{pmatrix}$$

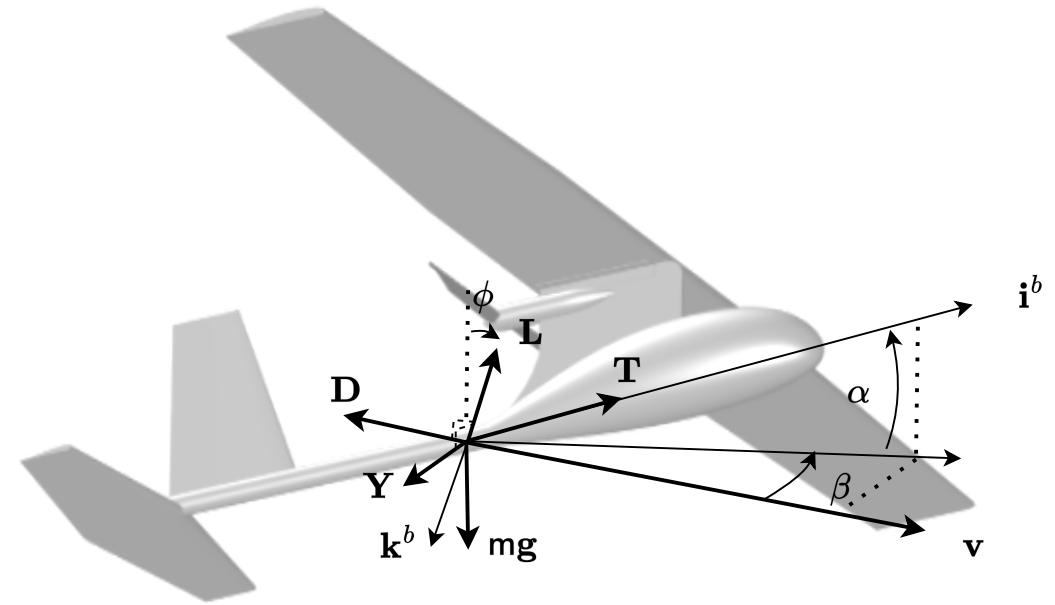
Dynamic Guidance Models

Total forces in the wind frame:

$$\mathbf{F}^w = \begin{pmatrix} -D + T \cos \alpha \cos \beta \\ -Y - T \cos \alpha \sin \beta \\ -L - T \sin \alpha \end{pmatrix}.$$

Total forces in the vehicle-2 (unrolled) frame:

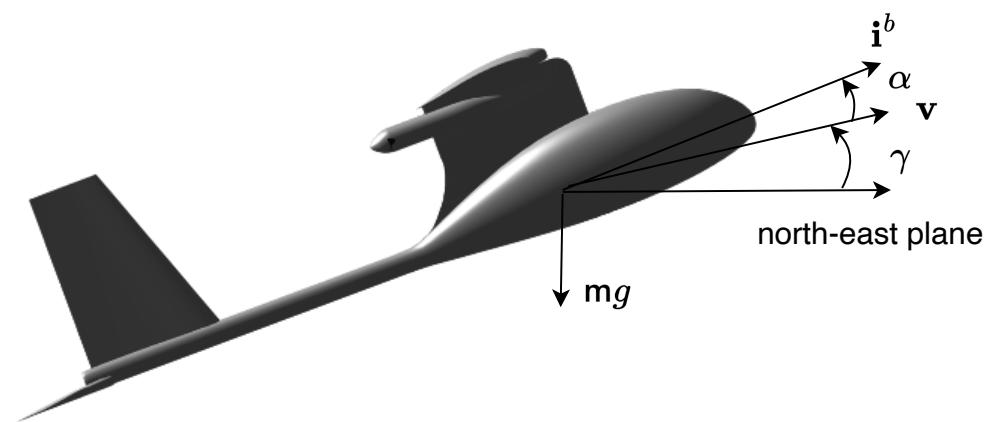
$$\begin{aligned} \mathbf{F}^{v2} &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \phi & -\sin \phi \\ 0 & -\sin \phi & \cos \phi \end{pmatrix} \begin{pmatrix} -D + T \cos \alpha \cos \beta \\ -Y - T \cos \alpha \sin \beta \\ -L - T \sin \alpha \end{pmatrix} \\ &= \begin{pmatrix} -D + T \cos \alpha \cos \beta \\ -(Y + T \cos \alpha \sin \beta) \cos \phi - (L + T \sin \alpha) \sin \phi \\ (Y + T \cos \alpha \sin \beta) \sin \phi - (L + T \sin \alpha) \cos \phi \end{pmatrix}. \end{aligned}$$



Dynamic Guidance Models

Along the \mathbf{i}^{v^2} axis, the magnitude of the velocity vector evolves as:

$$\begin{aligned} m\dot{V}_g &= T \cos \alpha \cos \beta - D - mg \sin \gamma \\ \implies \dot{V}_g &= \frac{T}{m} \cos \alpha \cos \beta - \frac{D}{m} - g \sin \gamma. \end{aligned}$$



Dynamic Guidance Models

Given:

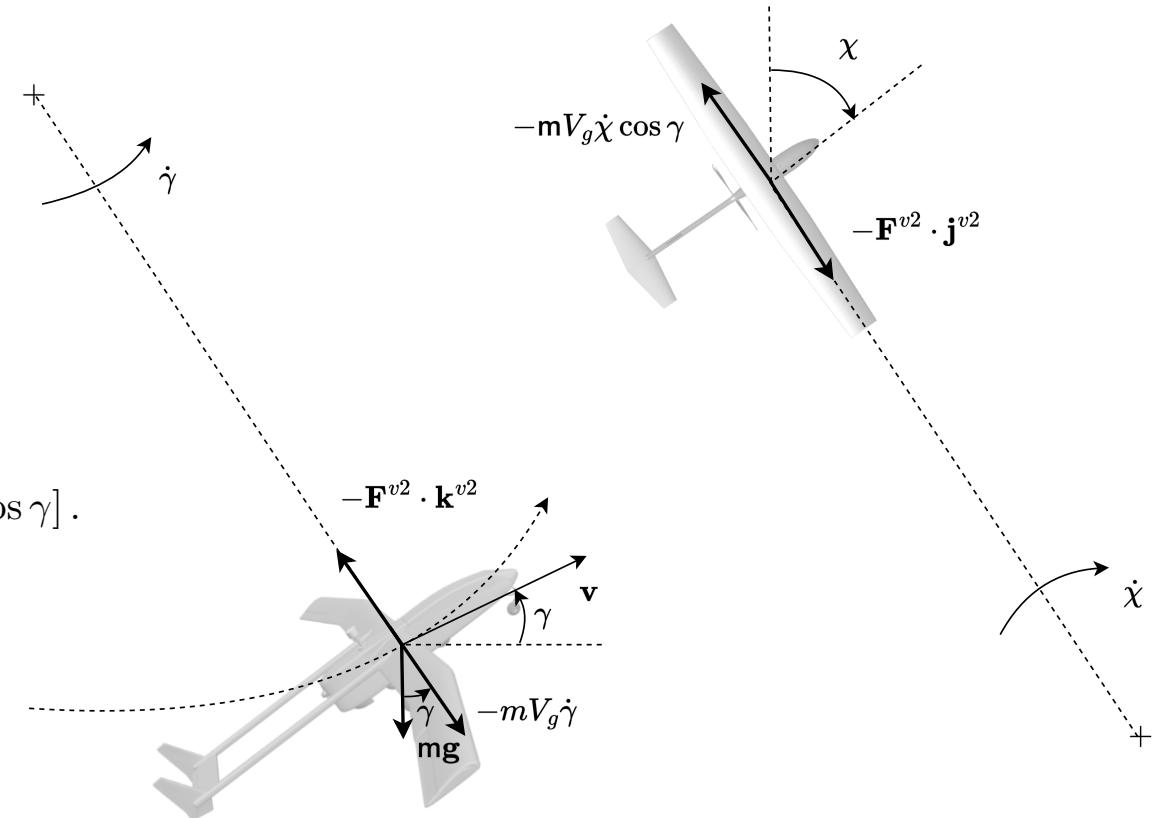
$$\mathbf{F}^{v2} = \begin{pmatrix} -D + T \cos \alpha \cos \beta \\ -(Y + T \cos \alpha \sin \beta) \cos \phi - (L + T \sin \alpha) \sin \phi \\ (Y + T \cos \alpha \sin \beta) \sin \phi - (L + T \sin \alpha) \cos \phi \end{pmatrix}.$$

Longitudinal direction:

$$\begin{aligned} \mathbf{m}V_g\dot{\gamma} &= -(Y + T \cos \alpha \sin \beta) \sin \phi + (L + T \sin \alpha) \cos \phi - \mathbf{m}g \cos \gamma \\ \implies \dot{\gamma} &= \frac{1}{\mathbf{m}V} [(L + T \sin \alpha) \cos \phi - (Y + T \cos \alpha \sin \beta) \sin \phi - \mathbf{m}g \cos \gamma]. \end{aligned}$$

Lateral direction:

$$\begin{aligned} \mathbf{m}V_g\dot{\chi}\cos\gamma &= (Y + T\cos\alpha\sin\beta)\cos\phi + (L + T\sin\alpha)\sin\phi \\ \implies \dot{\chi} &= \frac{1}{\mathbf{m}V_g\cos\gamma} [(Y + T\cos\alpha\sin\beta)\cos\phi + (L + T\sin\alpha)\sin\phi]. \end{aligned}$$



Dynamic Guidance Models: Summary

$$\dot{p}_n = V_g \cos \chi \cos \gamma$$

$$\dot{p}_e = V_g \sin \chi \cos \gamma$$

$$\dot{h} = V_g \sin \gamma$$

$$\dot{V}_g = \frac{\textcolor{blue}{T}}{m} \cos \alpha \cos \beta - \frac{D}{m} - g \sin \gamma$$

$$\dot{\chi} = \frac{1}{m V_g \cos \gamma} [(Y + \textcolor{blue}{T} \cos \alpha \sin \beta) \cos \phi + (L + \textcolor{blue}{T} \sin \alpha) \sin \phi]$$

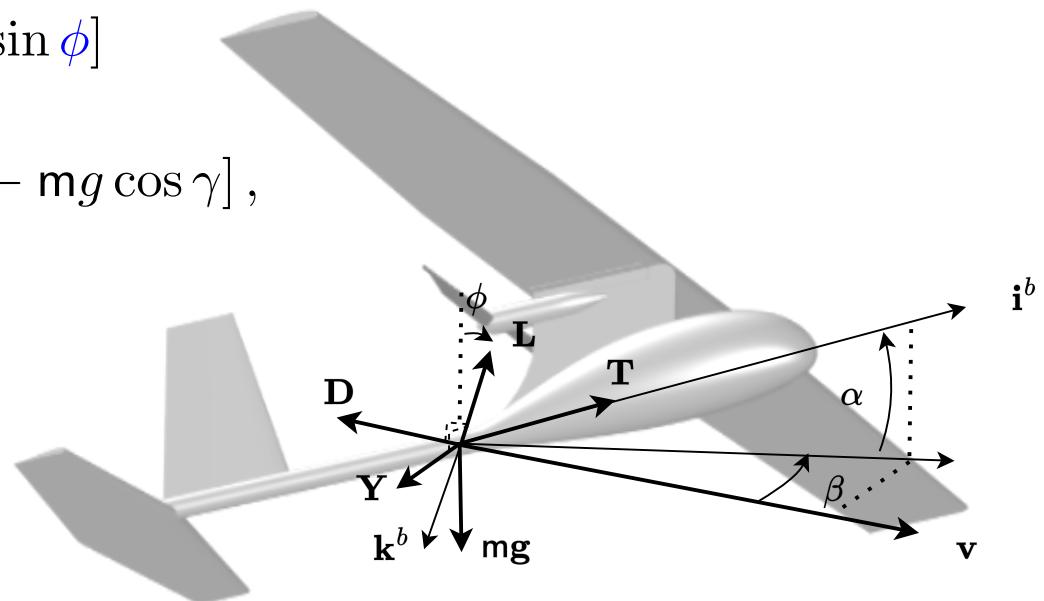
$$\dot{\gamma} = \frac{1}{m V_g} [(L + \textcolor{blue}{T} \sin \alpha) \cos \phi - (Y + \textcolor{blue}{T} \cos \alpha \sin \beta) \sin \phi - mg \cos \gamma],$$

where

$$L(\alpha) = \frac{1}{2} \rho V_a^2 S (C_{L_0} + C_{L_\alpha} \alpha)$$

$$D(\alpha) = \frac{1}{2} \rho V_a^2 S (C_{D_0} + C_{D_\alpha} \alpha + C_{D_{\alpha^2}} \alpha^2)$$

$$Y(\beta) = \frac{1}{2} \rho V_a^2 S C_{Y_\beta} \beta,$$



Dynamic Guidance Models: Summary (simplified)

When sideslip is regulated to zeros:

$$\dot{p}_n = V_g \cos \chi \cos \gamma$$

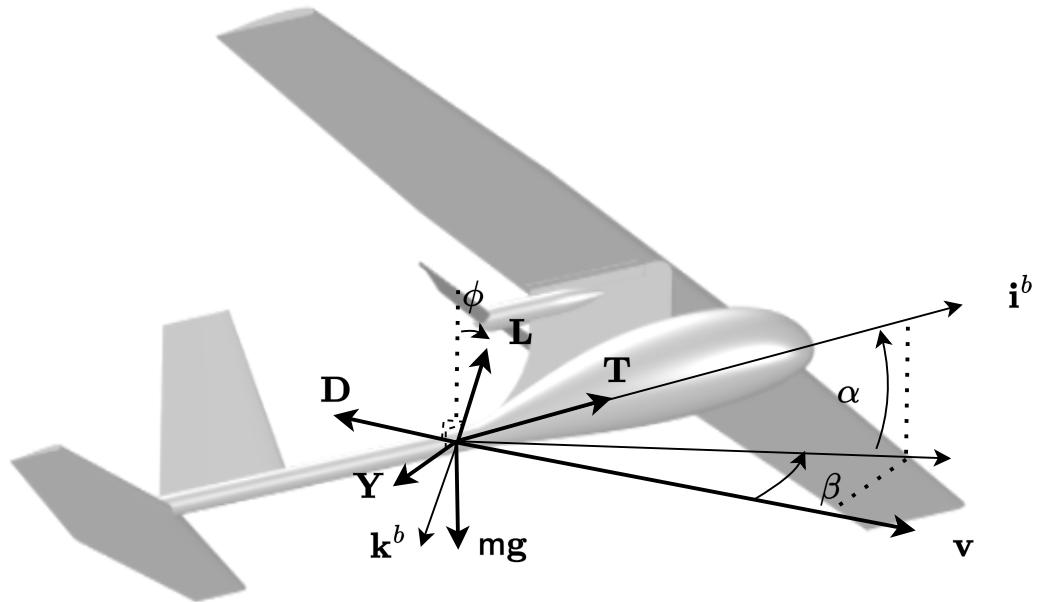
$$\dot{p}_e = V_g \sin \chi \cos \gamma$$

$$\dot{h} = V_g \sin \gamma$$

$$\dot{V}_g = \frac{\textcolor{blue}{T}}{m} \cos \alpha - \frac{D}{m} - g \sin \gamma$$

$$\dot{\chi} = \frac{1}{m V_g \cos \gamma} (L + \textcolor{blue}{T} \sin \alpha) \sin \phi$$

$$\dot{\gamma} = \frac{1}{m V_g} [(L + \textcolor{blue}{T} \sin \alpha) \cos \phi - mg \cos \gamma],$$



where

$$L(\alpha) = \frac{1}{2} \rho V_a^2 S (C_{L_0} + C_{L_\alpha} \alpha)$$

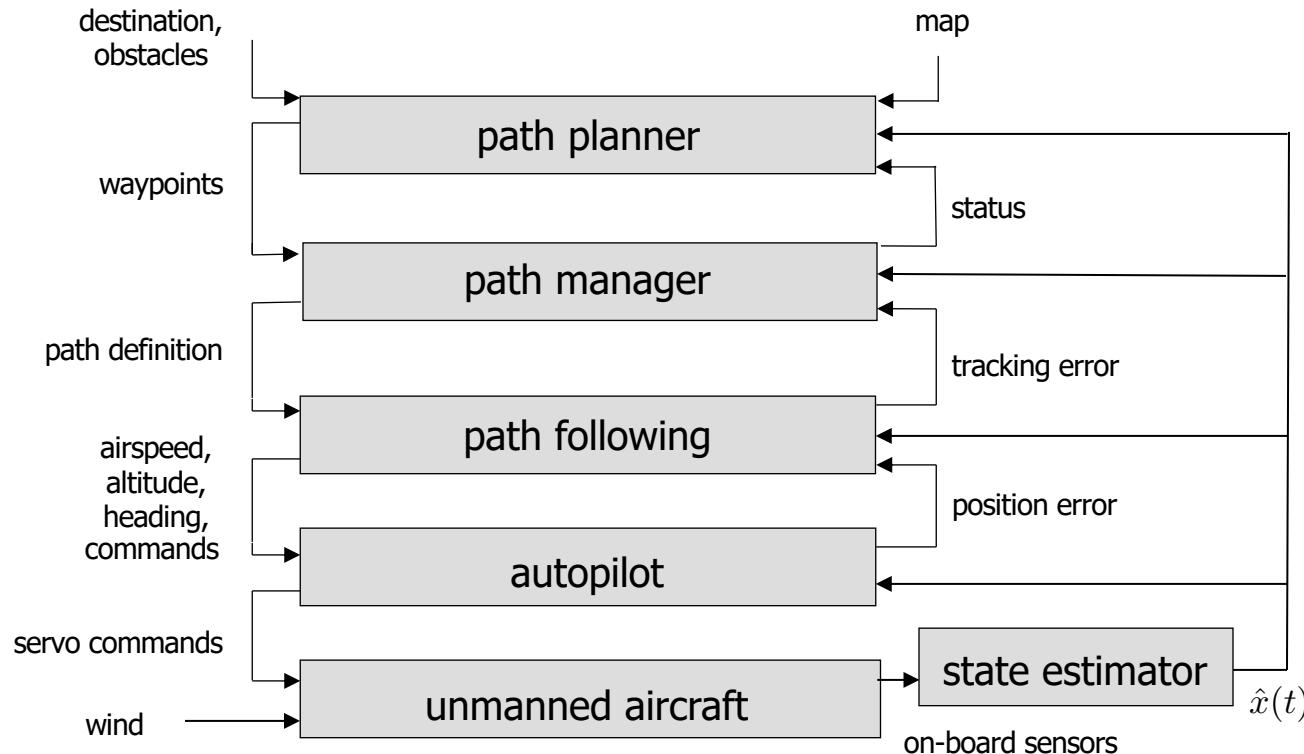
$$D(\alpha) = \frac{1}{2} \rho V_a^2 S (C_{D_0} + C_{D_\alpha} \alpha + C_{D_{\alpha^2}} \alpha^2)$$



Chapter 10

Path Following

Control Architecture



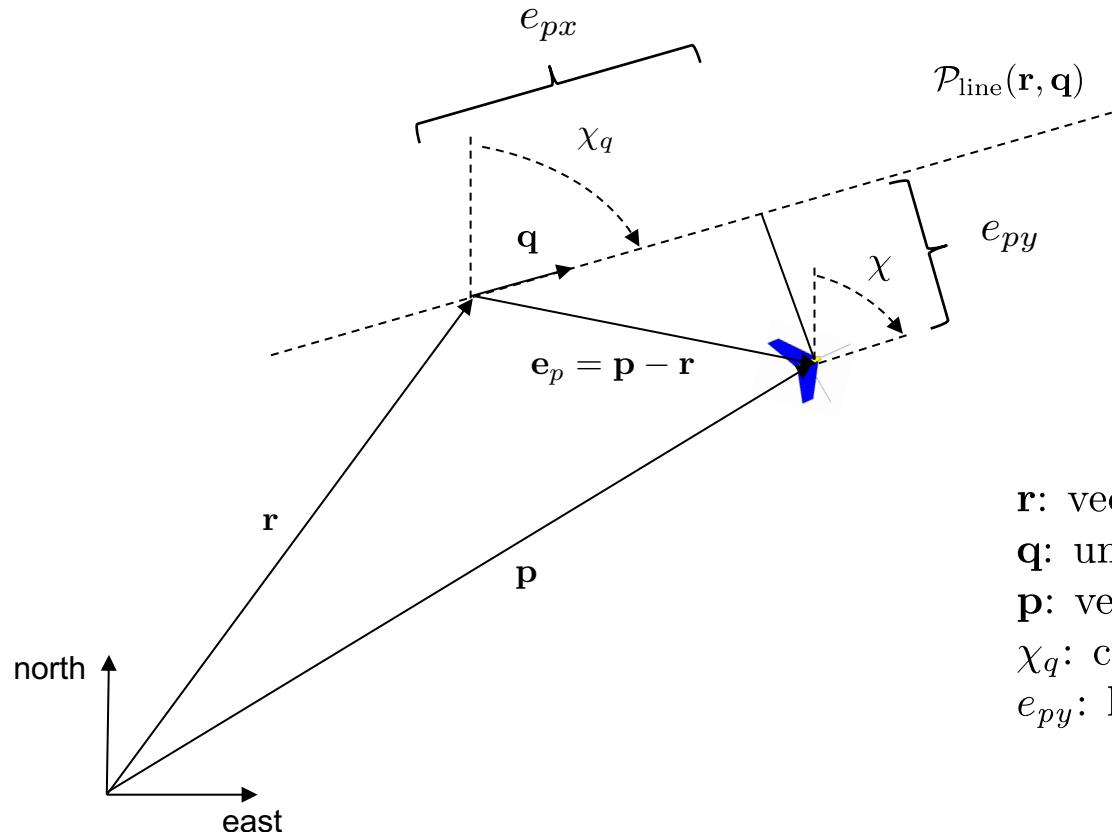
Path Following

- For small UAVs, a major issue is wind
 - Always present to some degree
 - Usually significant with respect to commanded airspeed
- Wind makes traditional trajectory tracking approaches difficult, if not infeasible
 - Have to know the wind precisely at every instant to determine desired airspeed
- Better approach: path following
- Rather than “follow this trajectory”, we control UAV to “stay on this path”

Path Types

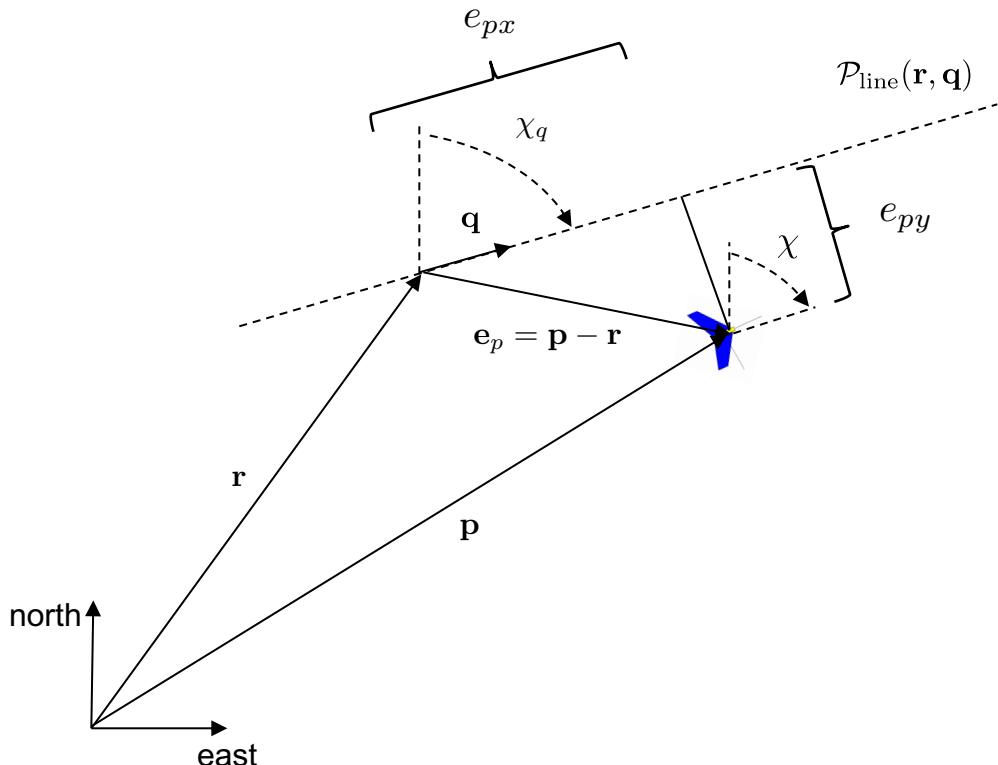
- We will focus on two types of paths to follow:
 - Straight lines between two points in 3-D
 - Inclination of path within climb capabilities of UAV
 - Circular orbits or arcs in the horizontal plane
- Paths for common applications can be built up from these path primitives
 - Methods for following other types of paths found in literature

Straight Line Path Description



- \mathbf{r} : vector defining initiation of path
- \mathbf{q} : unit vector defining direction of path
- \mathbf{p} : vector defining location of MAV
- χ_q : course direction of path
- e_{py} : lateral tracking error

Lateral Tracking Problem



Path error:

$$\mathbf{e}_p = \begin{pmatrix} e_{px} \\ e_{py} \\ e_{pz} \end{pmatrix} \triangleq \mathcal{R}_i^{\mathcal{P}} (\mathbf{p}^i - \mathbf{r}^i)$$

where the transformation from inertial frame to path frame is

$$\mathcal{R}_i^{\mathcal{P}} \triangleq \begin{pmatrix} \cos \chi_q & \sin \chi_q & 0 \\ -\sin \chi_q & \cos \chi_q & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Lateral Tracking Problem

Relative error dynamics in path frame:

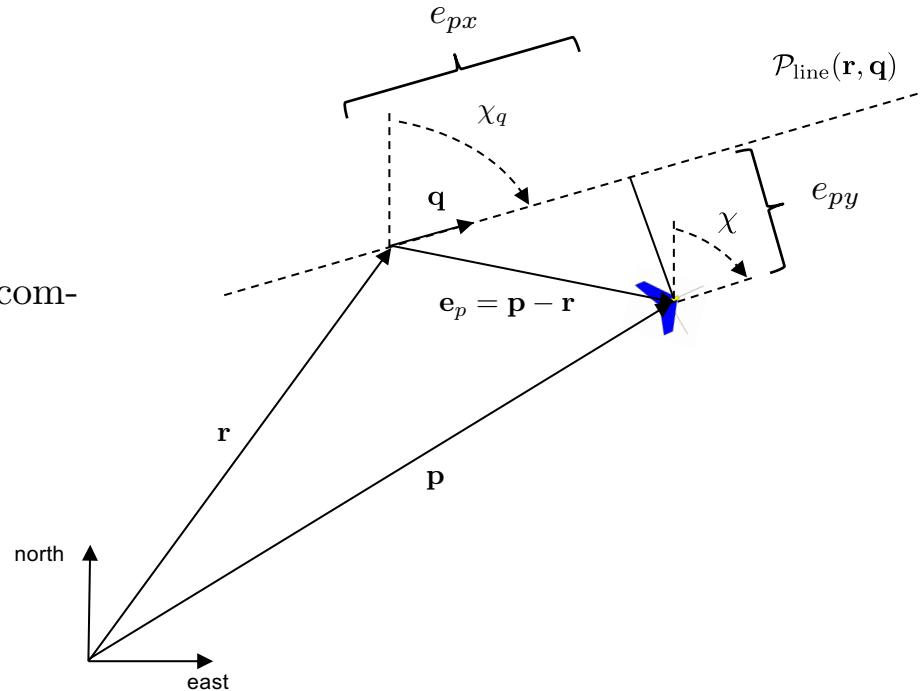
$$\begin{pmatrix} \dot{e}_{px} \\ \dot{e}_{py} \end{pmatrix} = \begin{pmatrix} \cos \chi_q & \sin \chi_q \\ -\sin \chi_q & \cos \chi_q \end{pmatrix} \begin{pmatrix} V_g \cos \chi \\ V_g \sin \chi \end{pmatrix}$$
$$= V_g \begin{pmatrix} \cos(\chi - \chi_q) \\ \sin(\chi - \chi_q) \end{pmatrix}$$

Regulate the cross-track error e_{py} to zero by commanding the course angle:

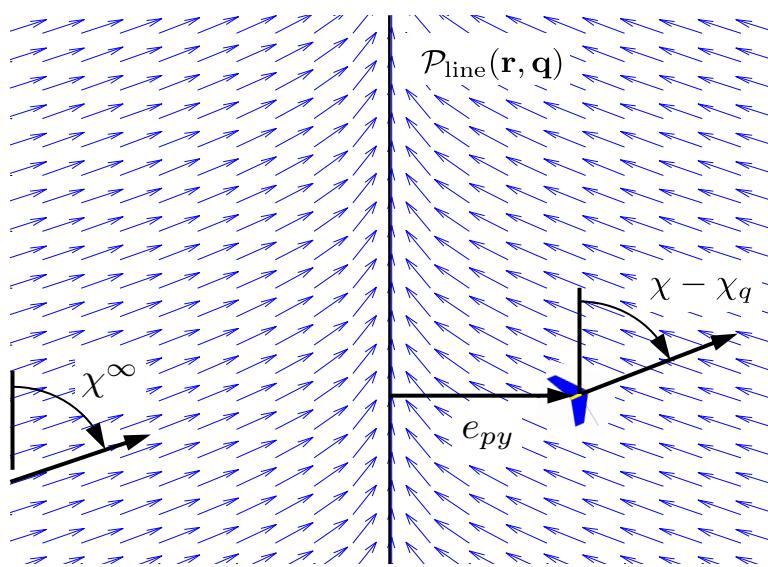
$$\dot{e}_{py} = V_g \sin(\chi - \chi_q)$$

$$\ddot{\chi} = b_{\dot{\chi}}(\dot{\chi}^c - \dot{\chi}) + b_\chi(\chi^c - \chi)$$

Select χ^c so that $e_{py} \rightarrow 0$



Lateral Tracking - Vector Field Concept

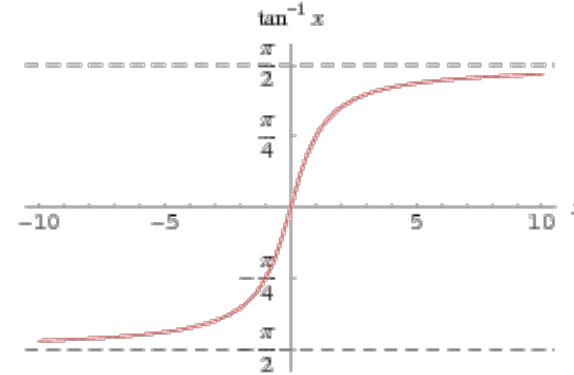


For a general line, the commanded course is

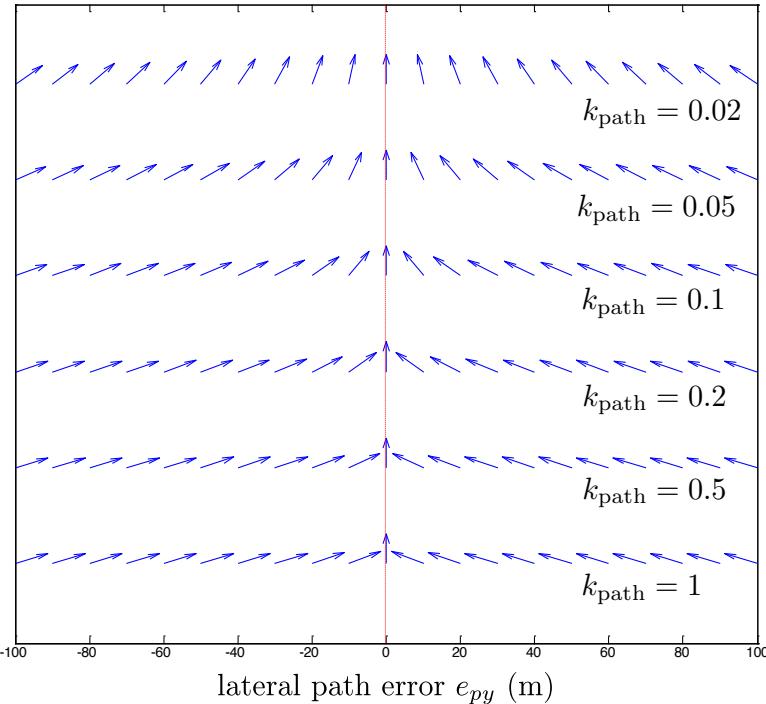
$$\chi^c = \chi_q - \chi^\infty \frac{2}{\pi} \tan^{-1}(k_{\text{path}} e_{py})$$

Desired course based on cross-track error:

$$\chi_d(e_{py}) = -\chi^\infty \frac{2}{\pi} \tan^{-1}(k_{\text{path}} e_{py})$$



Vector Field Tuning



- k_{path} is a positive constant that affects the rate of transition of the desired course
- $k_{\text{path}}\text{-large} \rightarrow$ short, abrupt transition
- $k_{\text{path}}\text{-small} \rightarrow$ long, gradual transition

Rule of thumb:

$$k_{\text{path}} \approx \frac{1}{R_{\min}},$$

where R_{\min} is the minimum turn radius of the aircraft.

Lyapunov's 2nd Method

For a system having a state vector x , consider an energy-like (Lyapunov) function $V(x) : \mathbb{R}^n \mapsto \mathbb{R}$ such that

$$V(x) > 0, \forall x \neq 0 \text{ (positive definite)}$$

$$V(0) = 0$$

and

$$\dot{V}(x) < 0, \forall x \neq 0 \text{ (negative definite)}$$

$$\dot{V}(0) = 0.$$

If such a function $V(x)$ can be defined, then x goes to zero asymptotically and the system is stable.

Lateral Tracking Stability Analysis

Define the Lyapunov function $W(e_{py}) = \frac{1}{2}e_{py}^2$

Assume that course controller works and $\chi = \chi_q + \chi^d(e_{py})$

Since

$$\begin{aligned}\dot{W} &= e_{py} \dot{e}_{py} \\ &= -V_a e_{py} \sin\left(\chi^\infty \frac{2}{\pi} \tan^{-1}(k_{\text{path}} e_{py})\right) \\ &< 0\end{aligned}$$

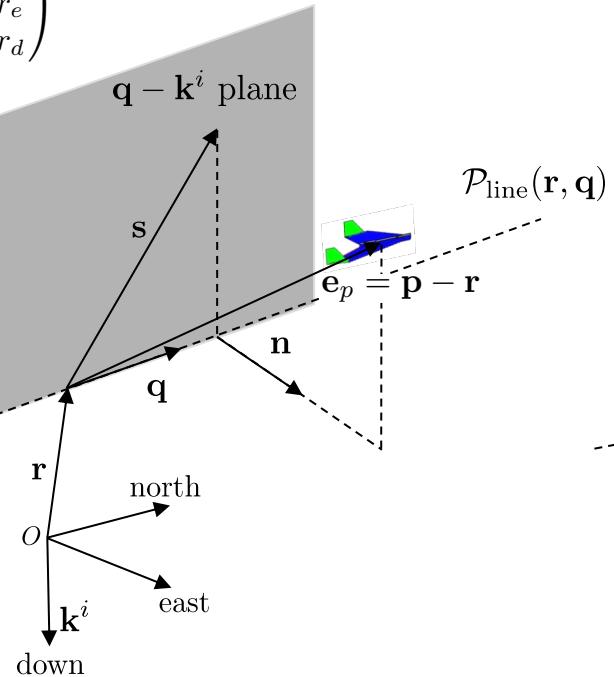
for $e_{py} \neq 0$, then $e_{py} \rightarrow 0$ asymptotically

Longitudinal Tracking Problem

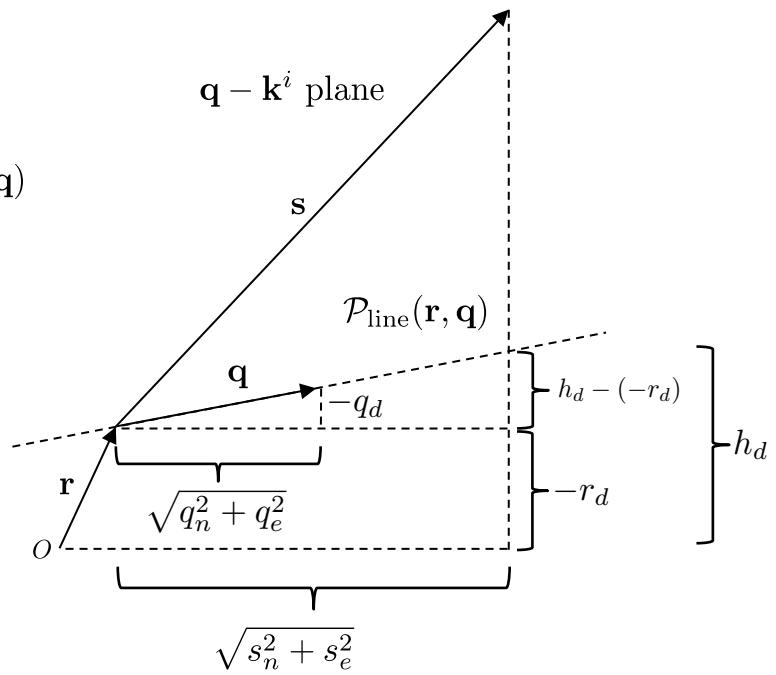
$$\mathbf{e}_p^i = \begin{pmatrix} e_{pn} \\ e_{pe} \\ e_{pd} \end{pmatrix} \triangleq \mathbf{p}^i - \mathbf{r}^i = \begin{pmatrix} p_n - r_n \\ p_e - r_e \\ p_d - r_d \end{pmatrix}$$

$$\mathbf{n} = \frac{\mathbf{k}^i \times \mathbf{q}}{\|\mathbf{k}^i \times \mathbf{q}\|}$$

$$\begin{aligned} \mathbf{s}^i &= \begin{pmatrix} s_n \\ s_e \\ s_d \end{pmatrix} \\ &= \mathbf{e}_p^i - (\mathbf{e}_p^i \cdot \mathbf{n})\mathbf{n} \end{aligned}$$

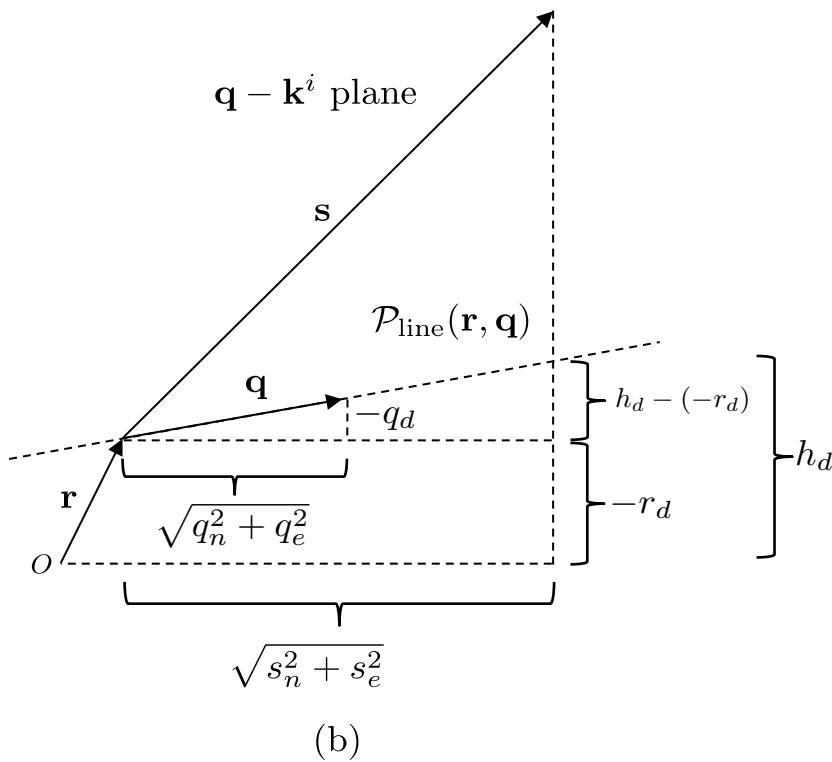


(a)



(b)

Longitudinal Tracking Problem



By similar triangles

$$\frac{(h_d + r_d)}{\sqrt{s_n^2 + s_e^2}} = \frac{-q_d}{\sqrt{q_n^2 + q_e^2}}$$

Desired altitude based on current location

$$h_d(\mathbf{r}, \mathbf{p}, \mathbf{q}) = -r_d - \sqrt{s_n^2 + s_e^2} \left(\frac{q_d}{\sqrt{q_n^2 + q_e^2}} \right)$$

Select h^c so that $h \rightarrow h_d(\mathbf{r}, \mathbf{p}, \mathbf{q})$

Longitudinal Guidance Strategy

Use altitude state machine from Ch. 6.

Closed-loop altitude dynamics:

$$h(s) = \left(\frac{b_h s + b_h}{s^2 + b_h s + b_h} \right) h^c(s).$$

Altitude error:

$$e_h \doteq h_d(\mathbf{r}, \mathbf{p}, \mathbf{q}) - h = h^c - h$$

Error dynamics:

$$\begin{aligned} e_h(s) &= (1 - h(s)) h^c(s) \\ &= \left(\frac{s^2 + b_h s + b_h}{s^2 + b_h s + b_h} - \frac{b_h s + b_h}{s^2 + b_h s + b_h} \right) h^c(s) \\ &= \left(\frac{s^2}{s^2 + b_h s + b_h} \right) h^c(s) \end{aligned}$$

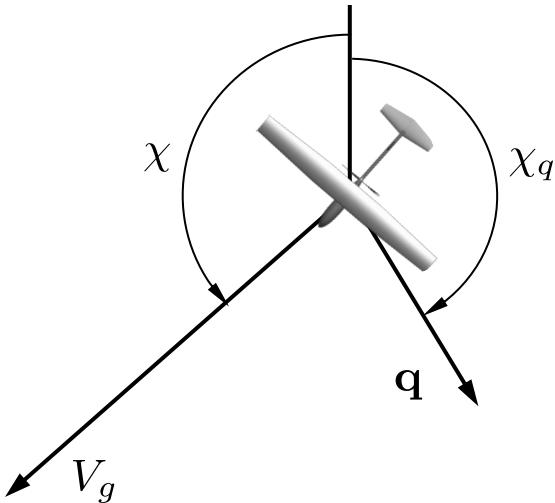
Applying FVT:

$$\begin{aligned} e_{h,ss} &= \lim_{s \rightarrow 0} s \frac{s^2}{s^2 + b_h s + b_h} h^c \\ &= 0, \quad \text{for } h^c = \frac{H_0}{s}, \frac{H_0}{s^2} \end{aligned}$$

Smallest Angle Turn Logic

$$\chi_q = \text{atan}2(q_e, q_n) + 2\pi m$$

$m \in \mathcal{N}$ is selected so that $-\pi \leq \chi_q - \chi \leq \pi$



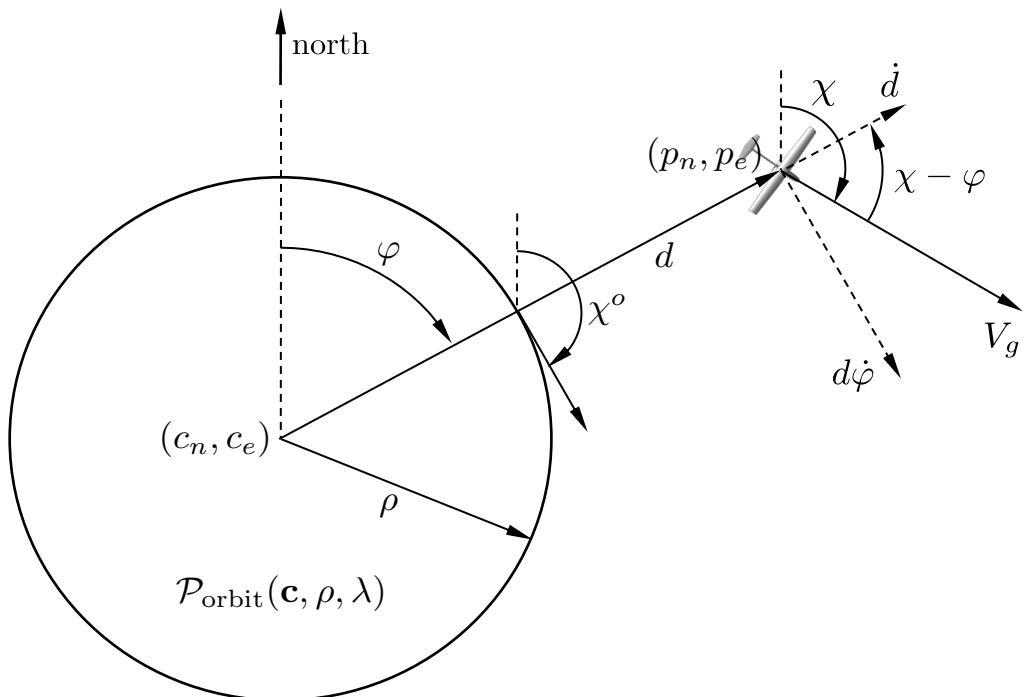
Algorithm 3 Straight-line Following: $[h^c, \chi^c] = \text{followStraightLine}(\mathbf{r}, \mathbf{q}, \mathbf{p}, \chi)$

Input: Path definition $\mathbf{r} = (r_n, r_e, r_d)^\top$ and $\mathbf{q} = (q_n, q_e, q_d)^\top$, MAV position $\mathbf{p} = (p_n, p_e, p_d)^\top$, course χ , gains $\chi_\infty, k_{\text{path}}$, sample rate T_s .

- 1: Compute commanded altitude using equation (10.5).
 - 2: $\chi_q \leftarrow \text{atan}2(q_e, q_n)$
 - 3: **while** $\chi_q - \chi < -\pi$ **do**
 - 4: $\chi_q \leftarrow \chi_q + 2\pi$
 - 5: **end while**
 - 6: **while** $\chi_q - \chi > \pi$ **do**
 - 7: $\chi_q \leftarrow \chi_q - 2\pi$
 - 8: **end while**
 - 9: $e_{py} \leftarrow -\sin \chi_q (p_n - r_n) + \cos \chi_q (p_e - r_e)$
 - 10: Compute commanded course angle using equation (10.8).
 - 11: **return** h^c, χ^c
-

Orbit Following

Orbit definition: $\mathcal{P}_{\text{orbit}}(\mathbf{c}, \rho, \lambda) = \left\{ \mathbf{r} \in \mathbb{R}^3 : \mathbf{r} = \mathbf{c} + \lambda \rho \begin{pmatrix} \cos \varphi & \sin \varphi & 0 \end{pmatrix}^\top, \varphi \in [0, 2\pi) \right\}$



Center: $\mathbf{c} \in \mathbb{R}^3$

Radius: $\rho \in \mathbb{R}$

Direction: $\lambda = 1$ (CW) or $\lambda = -1$ (CCW)

In polar coordinates, the position of MAV given by:

$$d \doteq \sqrt{(p_n - c_n)^2 + (p_e - c_e)^2}: \\ \varphi \doteq \tan^{-1} \left(\frac{p_e - c_e}{p_n - c_n} \right):$$

Orbit Following

Easiest to analyze in polar coordinates.

Using

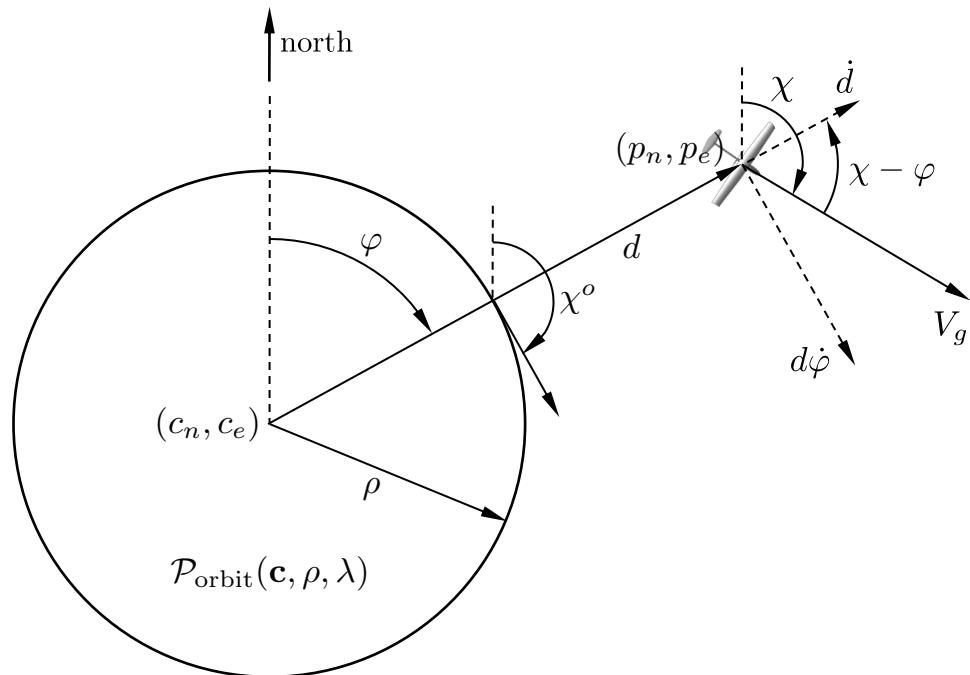
$$\begin{pmatrix} \dot{p}_n \\ \dot{p}_e \end{pmatrix} = \begin{pmatrix} V_g \cos \chi \\ V_g \sin \chi \end{pmatrix}$$

and converting to polar coordinates gives

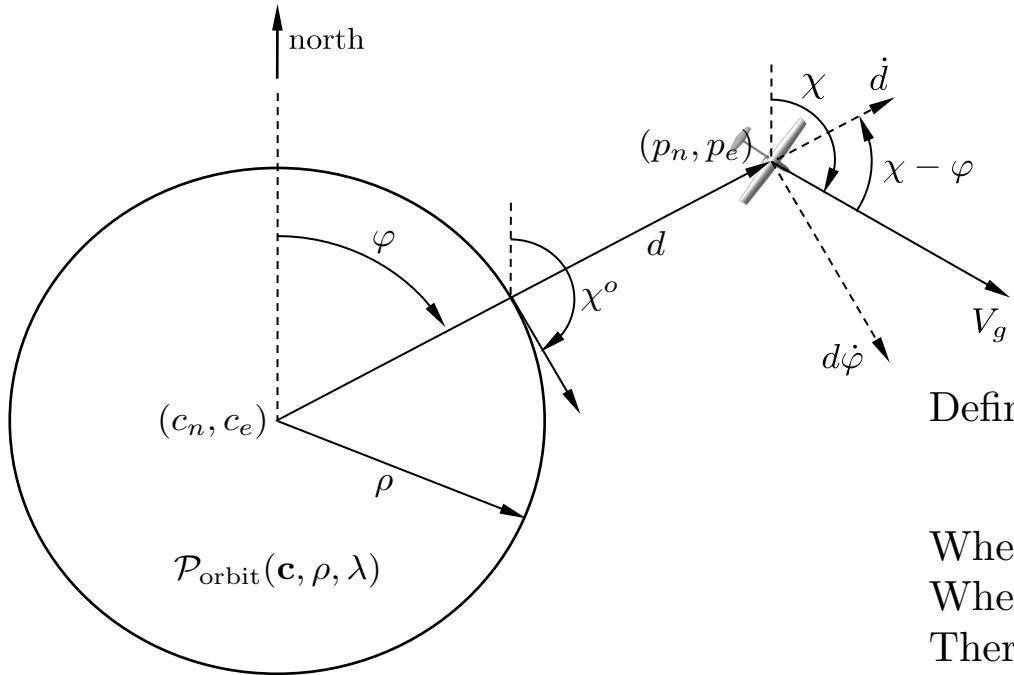
$$\begin{aligned} \begin{pmatrix} \dot{d} \\ d\dot{\varphi} \end{pmatrix} &= \begin{pmatrix} \cos \varphi & \sin \varphi \\ -\sin \varphi & \cos \varphi \end{pmatrix} \begin{pmatrix} \dot{p}_n \\ \dot{p}_e \end{pmatrix} \\ &= \begin{pmatrix} \cos \varphi & \sin \varphi \\ -\sin \varphi & \cos \varphi \end{pmatrix} \begin{pmatrix} V_g \cos \chi \\ V_g \sin \chi \end{pmatrix} \\ &= \begin{pmatrix} V_g \cos(\chi - \varphi) \\ V_g \sin(\chi - \varphi) \end{pmatrix} \end{aligned}$$

The result is

$$\boxed{\begin{aligned} \dot{d} &= V_g \cos(\chi - \varphi) \\ \dot{\varphi} &= \frac{V_g}{d} \sin(\chi - \varphi) \\ \ddot{\chi} &= -b_{\dot{\chi}} \dot{\chi} + b_{\chi} (\chi^c - \chi) \end{aligned}}$$



Orbit Following



Define

$$\chi^o = \varphi + \lambda \frac{\pi}{2}$$

When $d \gg \rho \rightarrow \chi_d \approx \chi^o + \lambda \frac{\pi}{2}$.

When $d = \rho \rightarrow \chi_d = \chi^o$

Therefore, let the desired course angle be

$$\chi_d(d-\rho, \lambda) = \chi^o + \lambda \tan^{-1} \left(k_{\text{orbit}} \left(\frac{d-\rho}{\rho} \right) \right)$$

Orbit Tracking Stability Analysis

Define the Lyapunov function $W = \frac{1}{2}(d - \rho)^2$

Assume that course controller works and $\chi = \chi^d(d - \rho, \lambda)$

Since

$$\begin{aligned}\dot{W} &= (d - \rho)\dot{d} \\ &= (d - \rho)(V_g \cos(\chi - \varphi)) \\ &= -V_g(d - \rho) \sin\left(\tan^{-1}\left(k_{\text{orbit}}\left(\frac{d - \rho}{\rho}\right)\right)\right) \\ &< 0\end{aligned}$$

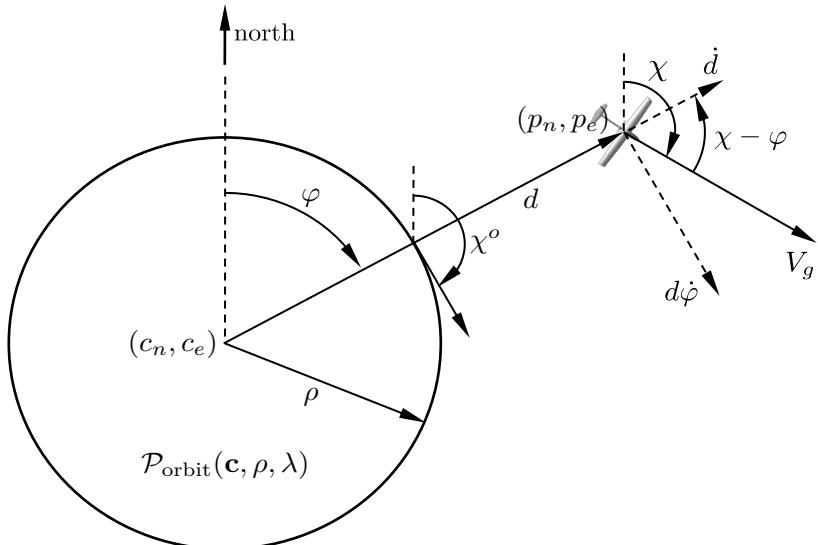
for $d - \rho \neq 0$, then $d - \rho \rightarrow 0$ asymptotically

Orbit Following

The commanded course is:

$$\chi^c(t) = \varphi + \lambda \left[\frac{\pi}{2} + \tan^{-1} \left(k_{\text{orbit}} \left(\frac{d-\rho}{\rho} \right) \right) \right]$$

The orbit angle must be wrapped:

$$\varphi = \text{atan2}(p_e - c_e, p_n - c_n) + 2\pi m$$


Algorithm 4 Circular Orbit Following: $[h^c, \chi^c] = \text{followOrbit}(\mathbf{c}, \rho, \lambda, \mathbf{p}, \chi)$

Input: Orbit center $\mathbf{c} = (c_n, c_e, c_d)^\top$, radius ρ , and direction λ , MAV position $\mathbf{p} = (p_n, p_e, p_d)^\top$, course χ , gains k_{orbit} , sample rate T_s .

- 1: $h^c \leftarrow -c_d$
 - 2: $d \leftarrow \sqrt{(p_n - c_n)^2 + (p_e - c_e)^2}$
 - 3: $\varphi \leftarrow \text{atan2}(p_e - c_e, p_n - c_n)$
 - 4: **while** $\varphi - \chi < -\pi$ **do**
 - 5: $\varphi \leftarrow \varphi + 2\pi$
 - 6: **end while**
 - 7: **while** $\varphi - \chi > \pi$ **do**
 - 8: $\varphi \leftarrow \varphi - 2\pi$
 - 9: **end while**
 - 10: Compute commanded course angle using equation (10.13).
 - 11: **return** h^c, χ^c
-

Roll Feedforward: no wind

For orbit following:

$$\chi^c(t) = \varphi + \lambda \left[\frac{\pi}{2} + \tan^{-1} \left(k_{\text{orbit}} \left(\frac{d - \rho}{\rho} \right) \right) \right].$$

Note:

$$\begin{aligned} d - \rho = 0 &\implies \chi^c = 0 \\ &\implies \phi^c = 0 \\ &\implies \text{UAV will immediately deviate from orbit.} \end{aligned}$$

Problem can be fixed by commanding a roll feedforward for when aircraft is on the orbit.

If on the orbit and no wind, then

$$\dot{\psi}^d = \lambda \frac{V_a}{\rho}.$$

Coordinated turn condition:

$$\dot{\psi} = \frac{g}{V_a} \tan \phi.$$

Equating and solving for roll gives

$$\phi_{ff} = \lambda \tan^{-1} \left(\frac{V_a^2}{g\rho} \right). \quad (1)$$

Roll Feedforward: wind

When wind is present we have

$$\dot{\chi}^d(t) = \lambda \frac{V_g(t)}{\rho},$$

where V_g is the time varying ground speed. The coordinated turn condition in wind is

$$\dot{\chi} = \frac{g}{V_g} \tan \phi \cos(\chi - \psi).$$

Equating these expressions and solving for ϕ gives

$$\phi_{ff} = \lambda \tan^{-1} \left(\frac{V_g^2}{g\rho \cos(\chi - \psi)} \right).$$

Or equivalently

$$\phi_{ff} = \lambda \tan^{-1} \left(\frac{\left((w_n \cos \chi + w_e \sin \chi) + \sqrt{V_a^2 - (w_n \sin \chi - w_e \cos \chi)^2 - w_d^2} \right)^2}{g\rho \sqrt{\frac{V_a^2 - (w_n \sin \chi - w_e \cos \chi)^2 - w_d^2}{V_a^2 - w_d^2}}} \right),$$

Dubins Airplane Model

Adapted from: Mark Owen, Randal W. Beard, Timothy W. McLain, "Implementing Dubins Airplane Paths on Fixed-wing UAVs," *Handbook of Unmanned Aerial Vehicles*, ed. Kimon P. Valavanis, George J. Vachtsevanos, Springer Verlag, Section XII, Chapter 68, p. 1677-1702, 2014.

Dubins Airplane model:

$$\dot{r}_n = V \cos \psi \cos \gamma^c$$

$$\dot{r}_e = V \sin \psi \cos \gamma^c$$

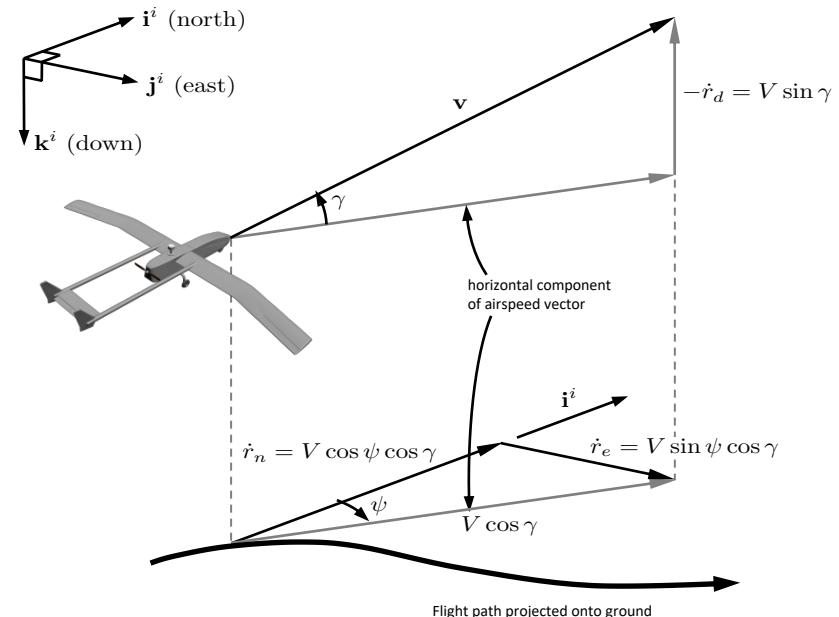
$$\dot{r}_d = -V \sin \gamma^c$$

$$\dot{\psi} = \frac{g}{V} \tan \phi^c$$

Where the commanded flight path angle γ^c and the commanded roll angle ϕ^c are constrained by

$$|\phi^c| \leq \bar{\phi}$$

$$|\gamma^c| \leq \bar{\gamma}.$$



3D Vector Field Path Following

Adapted from: V. M. Goncalves, L. C. A. Pimenta, C. A. Maia, B. C. O. Durtra, G. A. S. Pereira, B. C. O. Dutra, and G. A. S. Pereira, "Vector Fields for Robot Navigation Along Time-Varying Curves in n-Dimensions," IEEE Transactions on Robotics, vol. 26, pp. 647–659, Aug 2010.

The path is specified as the intersection of two 2D manifolds given by

$$\begin{aligned}\alpha_1(\mathbf{r}) &= 0 \\ \alpha_2(\mathbf{r}) &= 0\end{aligned}$$

$\mathbf{r} \in \mathbb{R}^3$. Define the composite function

$$W(\mathbf{r}) = \frac{1}{2}\alpha_1^2(\mathbf{r}) + \frac{1}{2}\alpha_2^2(\mathbf{r}),$$

Note that the gradient

$$\frac{\partial W}{\partial \mathbf{r}} = \alpha_1(\mathbf{r}) \frac{\partial \alpha_1}{\partial \mathbf{r}}(\mathbf{r}) + \alpha_2(\mathbf{r}) \frac{\partial \alpha_2}{\partial \mathbf{r}}(\mathbf{r}).$$

points away from the path.

3D Vector Field Path Following

The desired velocity vector can be chosen as

$$\mathbf{u}' = \underbrace{-K_1 \frac{\partial W}{\partial \mathbf{r}}}_{\text{velocity directed toward the path}} + \underbrace{K_2 \frac{\partial \alpha_1}{\partial \mathbf{r}} \times \frac{\partial \alpha_2}{\partial \mathbf{r}}}_{\text{velocity directed along the path}}$$

where $K_1 > 0$ and K_2 are symmetric tuning matrices, and the definiteness of K_2 determines the direction of travel along the path.

Since \mathbf{u}' may not equal V_a , normalize to get

$$\mathbf{u} = V_a \frac{\mathbf{u}'}{\|\mathbf{u}'\|}.$$

3D Vector Field Path Following

Setting the NED components of the velocity of the Dubins airplane model to $\mathbf{u} = (u_1, u_2, u_3)^\top$ gives

$$V \cos \psi^d \cos \gamma^c = u_1$$

$$V \sin \psi^d \cos \gamma^c = u_2$$

$$-V \sin \gamma^c = u_3.$$

Solving for γ^c , and ψ^d results in

$$\gamma^c = -\text{sat}_{\bar{\gamma}} \left[\sin^{-1} \left(\frac{u_3}{V} \right) \right]$$

$$\psi^d = \text{atan2}(u_2, u_1).$$

Assuming the inner-loop lateral-directional dynamics are accurately modeled by the coordinated-turn equation, the commanded roll angle is

$$\phi^c = \text{sat}_{\bar{\phi}} \left[k_\phi (\psi^d - \psi) \right],$$

where k_ϕ is a positive constant.

3D Vector Field – Straight Line path

The straight line path is given by

$$\mathcal{P}_{\text{line}}(\mathbf{c}_\ell, \psi_\ell, \gamma_\ell) = \left\{ \mathbf{r} \in \mathbb{R}^3 : \mathbf{r} = \mathbf{c}_\ell + \sigma \mathbf{q}_\ell, \sigma \in \mathbb{R} \right\},$$

where

$$\mathbf{q}_\ell = \begin{pmatrix} q_n \\ q_e \\ q_d \end{pmatrix} = \begin{pmatrix} \cos \psi_\ell \cos \gamma_\ell \\ \sin \psi_\ell \cos \gamma_\ell \\ -\sin \gamma_\ell \end{pmatrix}.$$

Define

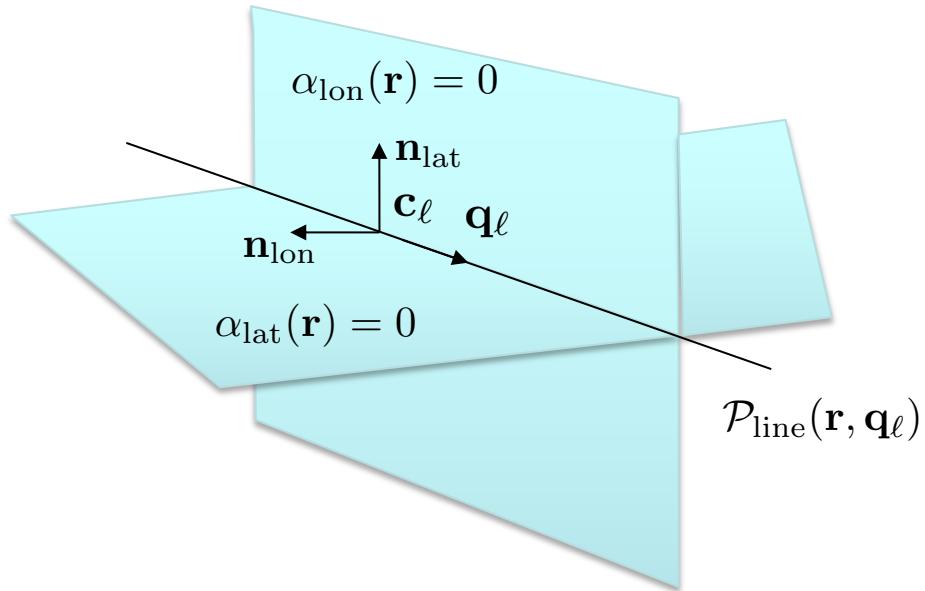
$$\mathbf{n}_{\text{lon}} = \begin{pmatrix} -\sin \psi_\ell \\ \cos \psi_\ell \\ 0 \end{pmatrix}$$

$$\mathbf{n}_{\text{lat}} = \mathbf{n}_{\text{lon}} \times \mathbf{q}_\ell = \begin{pmatrix} -\cos \psi_\ell \sin \gamma_\ell \\ -\sin \psi_\ell \sin \gamma_\ell \\ -\cos \gamma_\ell \end{pmatrix},$$

to get

$$\alpha_{\text{lon}}(\mathbf{r}) = \mathbf{n}_{\text{lon}}^\top (\mathbf{r} - \mathbf{c}_\ell) = 0$$

$$\alpha_{\text{lat}}(\mathbf{r}) = \mathbf{n}_{\text{lat}}^\top (\mathbf{r} - \mathbf{c}_\ell) = 0.$$



3D Vector Field – Helical Path

A helical path is then defined as

$$\mathcal{P}_{\text{helix}}(\mathbf{c}_h, \psi_h, \lambda_h, \rho_h, \gamma_h) = \{\mathbf{r} \in \mathbb{R}^3 : \alpha_{\text{cyl}}(\mathbf{r}) = 0 \text{ and } \alpha_{\text{pl}}(\mathbf{r}) = 0\}.$$

where

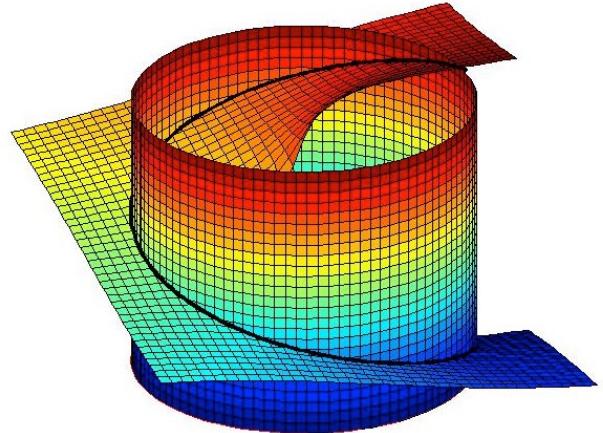
$$\alpha_{\text{cyl}}(\mathbf{r}) = \left(\frac{r_n - c_n}{\rho_h} \right)^2 + \left(\frac{r_e - c_e}{\rho_h} \right)^2 - 1$$

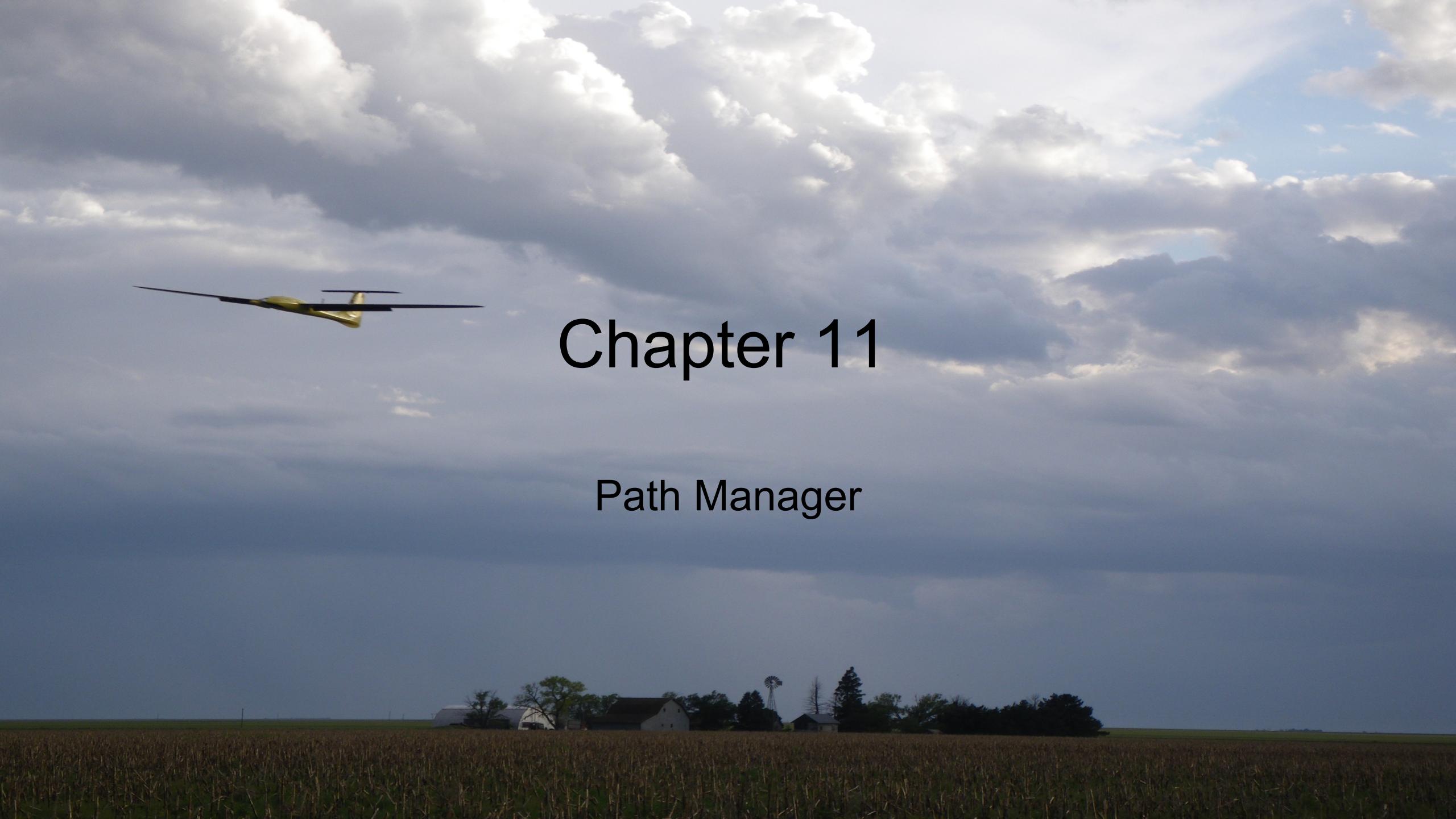
$$\alpha_{\text{pl}}(\mathbf{r}) = \left(\frac{r_d - c_d}{\rho_h} \right) + \frac{\tan \gamma_h}{\lambda_h} \left(\tan^{-1} \left(\frac{r_e - c_e}{r_n - c_n} \right) - \psi_h \right)$$

where the initial position along the helix is

$$\mathbf{r}(0) = \mathbf{c}_h + \begin{pmatrix} \rho_h \cos \psi_h \\ \rho_h \sin \psi_h \\ 0 \end{pmatrix},$$

and where \mathbf{c}_h is the center of the helix, ρ_h is the radius, γ_h is the climb angle.

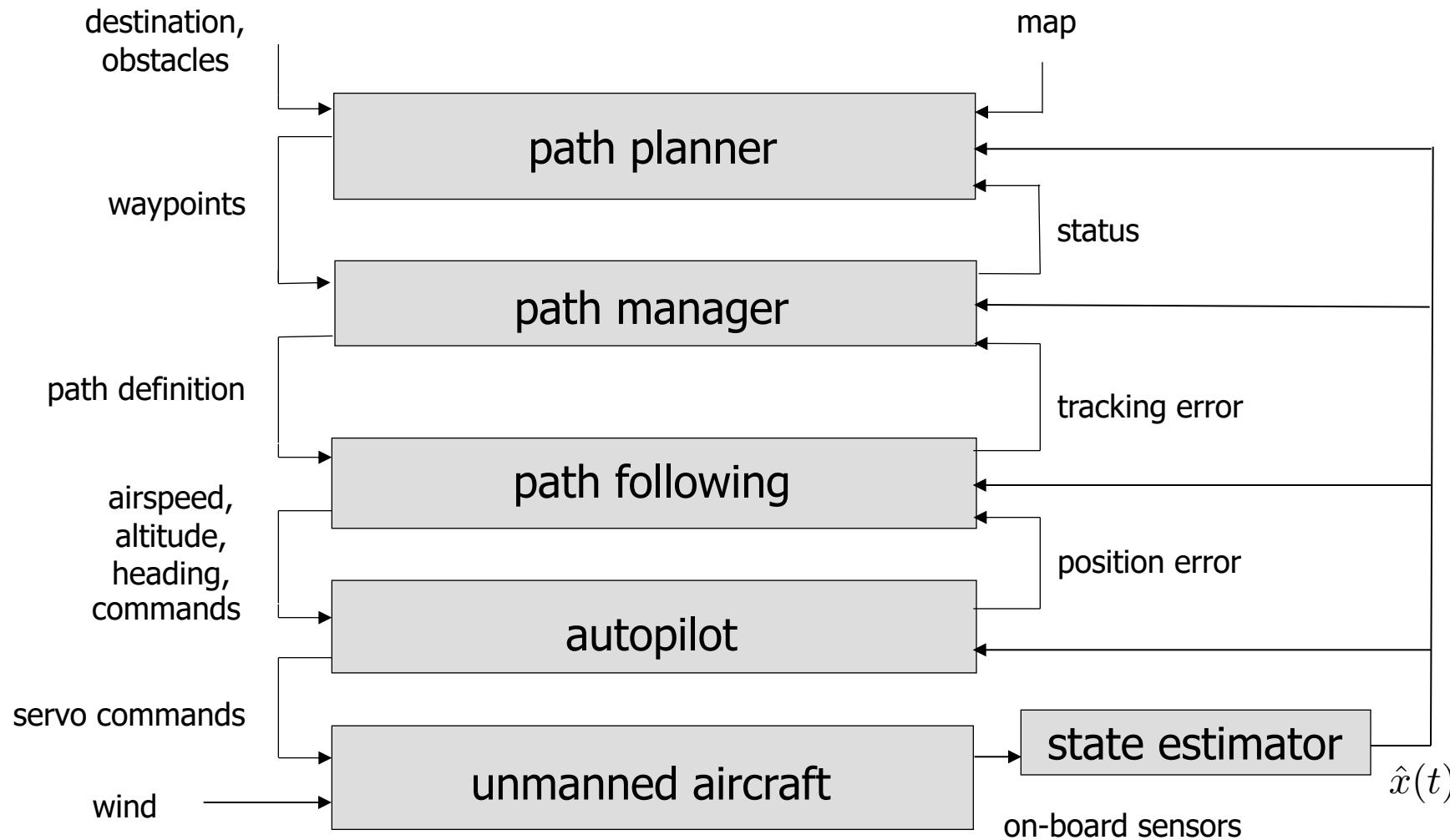


A photograph of a yellow glider plane flying from left to right against a backdrop of a cloudy sky. Below the plane is a vast, open landscape featuring a field of tall grass or crops in the foreground, a line of trees, and a small farm building complex with a windmill in the distance.

Chapter 11

Path Manager

Control Architecture

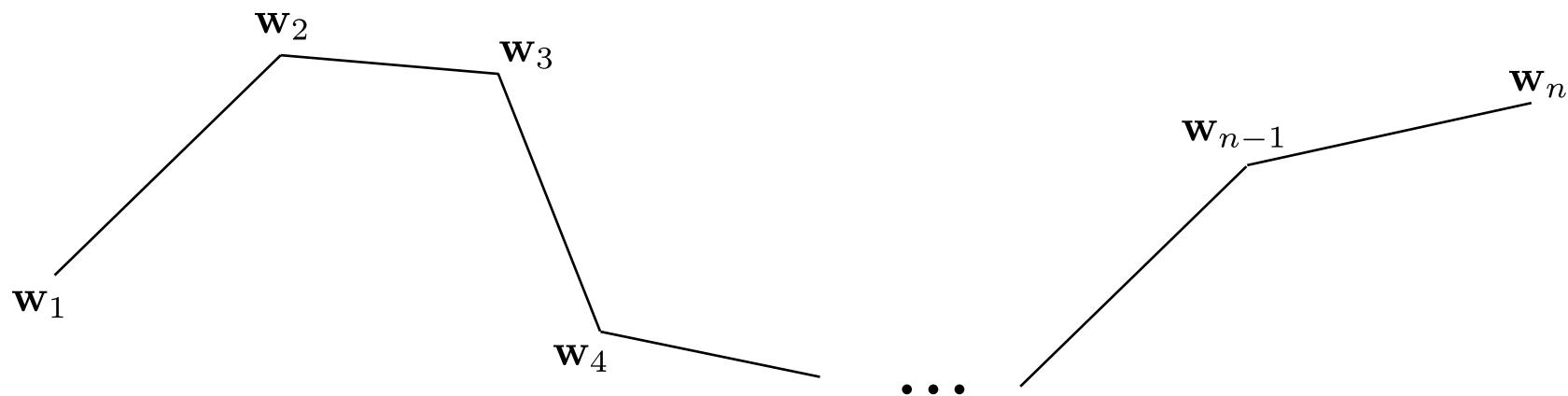


Path Definition

Waypoint path defined as ordered sequence of waypoints

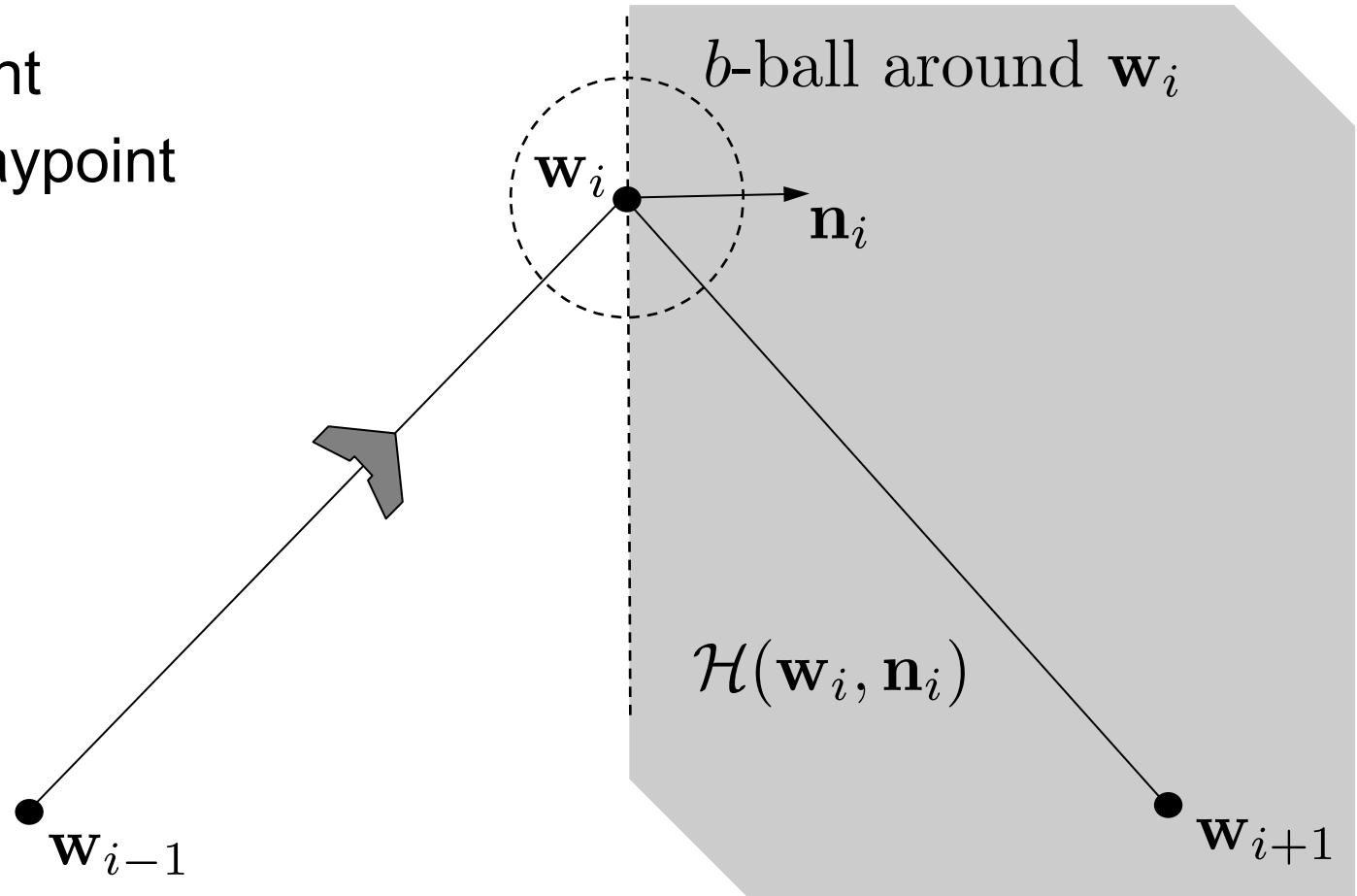
$$\mathcal{W} = \{\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_N\}$$

where $\mathbf{w}_i = (w_{n,i}, w_{e,i}, w_{d,i})^\top \in \mathbb{R}^3$.



Waypoint Switching

- Two methods
 - b -ball around waypoint
 - half plane through waypoint



Waypoint Switching

Given point $\mathbf{r} \in \mathbb{R}^3$ and normal vector $\mathbf{n} \in \mathbb{R}^3$,
define half plane

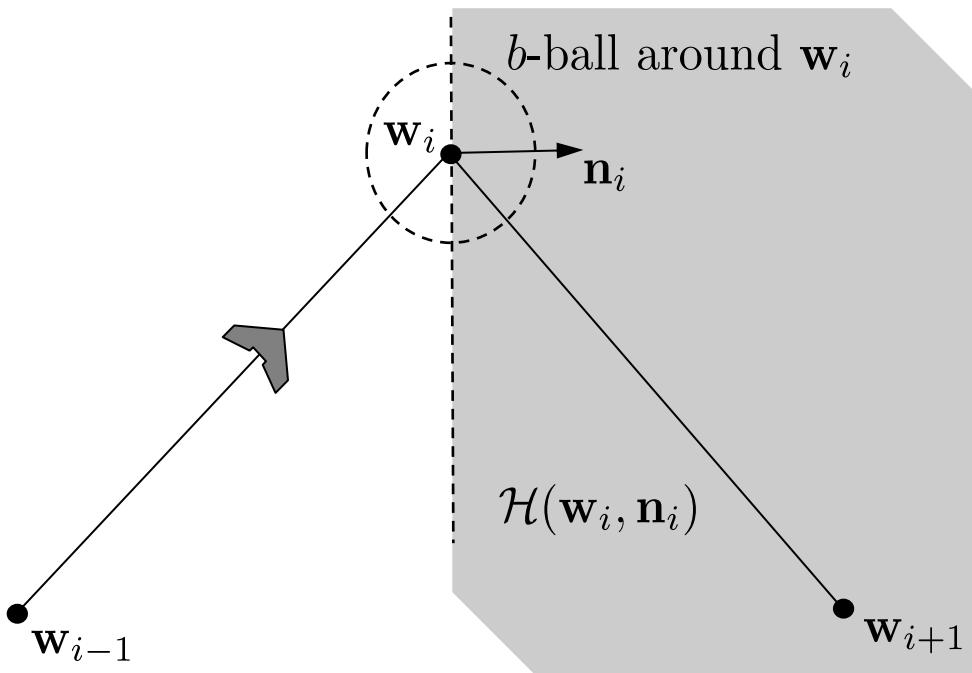
$$\mathcal{H}(\mathbf{r}, \mathbf{n}) \triangleq \{\mathbf{p} \in \mathbb{R}^3 : (\mathbf{p} - \mathbf{r})^\top \mathbf{n} \geq 0\}$$

Define unit vector pointing in direction of line $\overline{\mathbf{w}_i \mathbf{w}_{i+1}}$
as

$$\mathbf{q}_i \triangleq \frac{\mathbf{w}_{i+1} - \mathbf{w}_i}{\|\mathbf{w}_{i+1} - \mathbf{w}_i\|}$$

Unit normal to the 3-D half plane that separates
the line $\overline{\mathbf{w}_{i-1} \mathbf{w}_i}$ from the line $\overline{\mathbf{w}_i \mathbf{w}_{i+1}}$ is given by

$$\mathbf{n}_i \triangleq \frac{\mathbf{q}_{i-1} + \mathbf{q}_i}{\|\mathbf{q}_{i-1} + \mathbf{q}_i\|}$$



MAV tracks straight-line path from \mathbf{w}_{i-1} to \mathbf{w}_i
until it enters $\mathcal{H}(\mathbf{w}_i, \mathbf{n}_i)$, at which point it will
track straight-line path from \mathbf{w}_i to \mathbf{w}_{i+1}

Waypoint Following

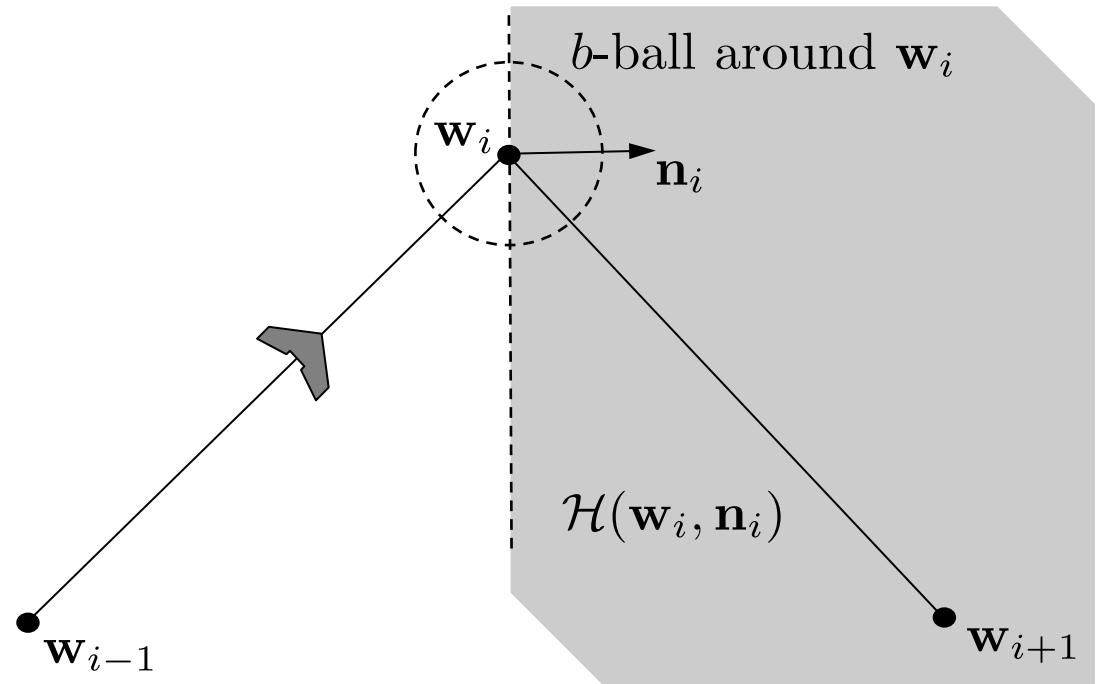
Algorithm 5 Follow Waypoints: $(\mathbf{r}, \mathbf{q}) = \text{followWpp}(\mathcal{W}, \mathbf{p})$

Input: Waypoint path $\mathcal{W} = \{\mathbf{w}_1, \dots, \mathbf{w}_N\}$, MAV position

$$\mathbf{p} = (p_n, p_e, p_d)^\top.$$

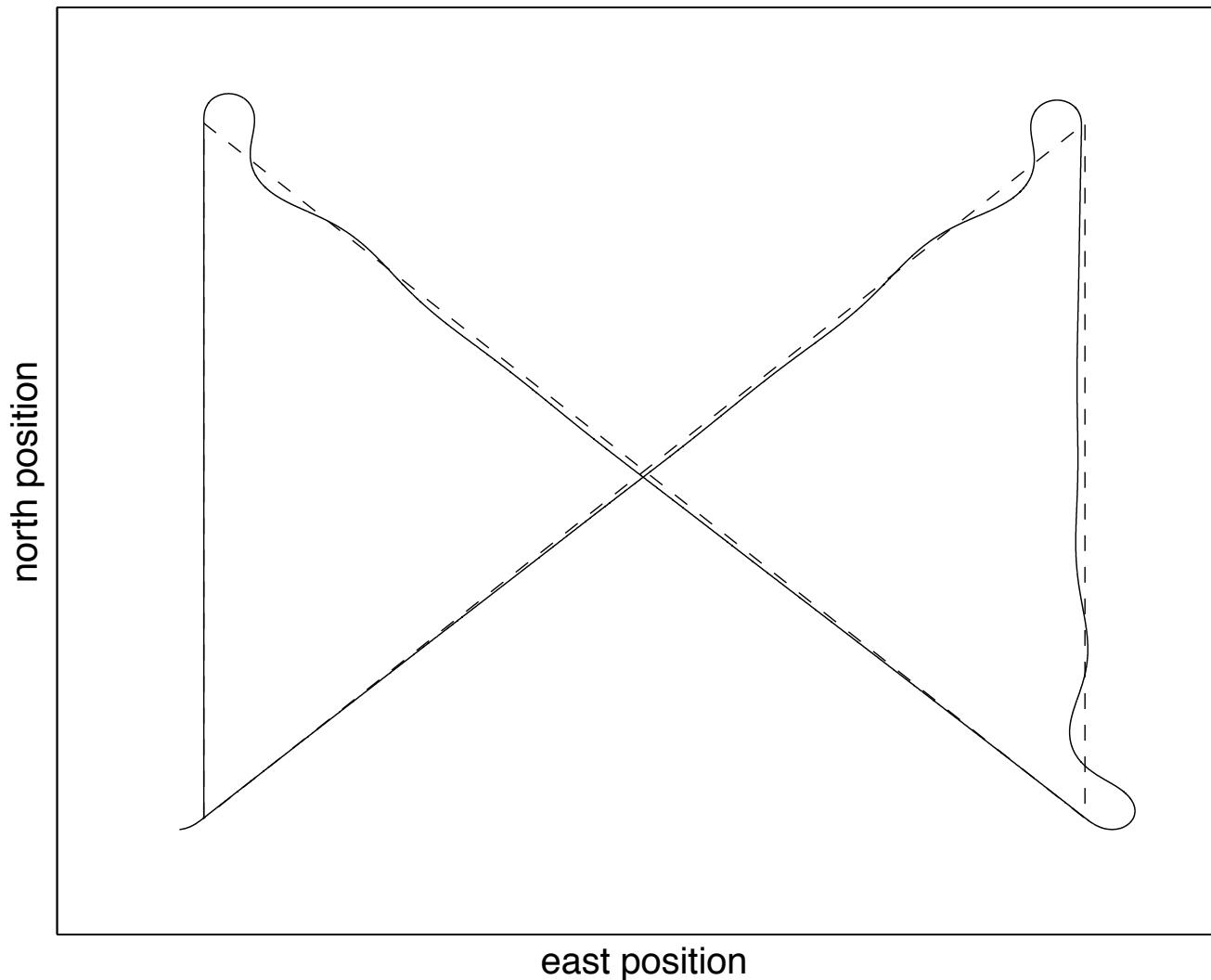
Require: $N \geq 3$

- 1: **if** New waypoint path \mathcal{W} is received **then**
 - 2: Initialize waypoint index: $i \leftarrow 2$
 - 3: **end if**
 - 4: $\mathbf{r} \leftarrow \mathbf{w}_{i-1}$
 - 5: $\mathbf{q}_{i-1} \leftarrow \frac{\mathbf{w}_i - \mathbf{w}_{i-1}}{\|\mathbf{w}_i - \mathbf{w}_{i-1}\|}$
 - 6: $\mathbf{q}_i \leftarrow \frac{\mathbf{w}_{i+1} - \mathbf{w}_i}{\|\mathbf{w}_{i+1} - \mathbf{w}_i\|}$
 - 7: $\mathbf{n}_i \leftarrow \frac{\mathbf{q}_{i-1} + \mathbf{q}_i}{\|\mathbf{q}_{i-1} + \mathbf{q}_i\|}$
 - 8: **if** $\mathbf{p} \in \mathcal{H}(\mathbf{w}_i, \mathbf{n}_i)$ **then**
 - 9: Increment $i \leftarrow (i + 1)$ until $i = N - 1$
 - 10: **end if**
 - 11: **return** $\mathbf{r}, \mathbf{q} = \mathbf{q}_{i-1}$ at each time step
-



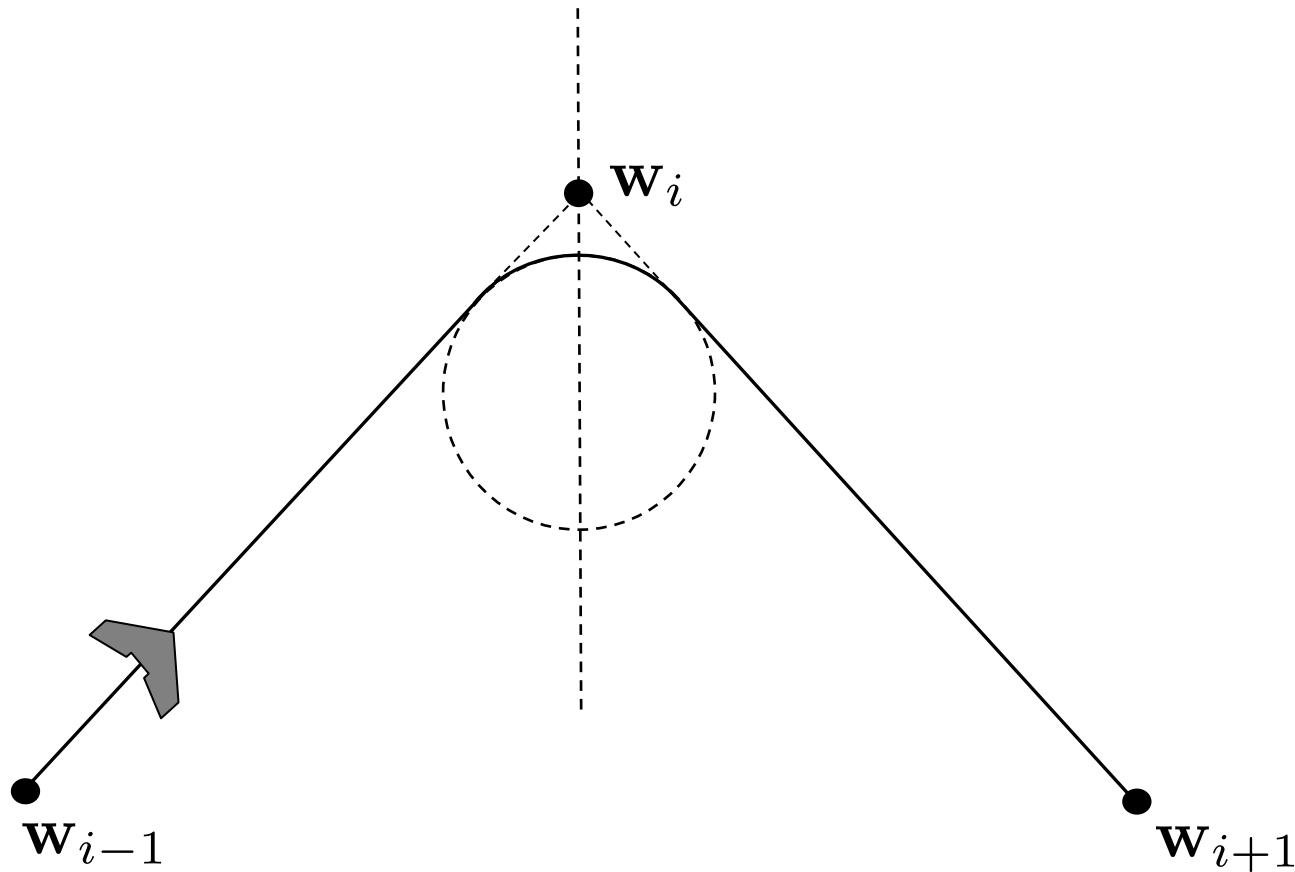
Waypoint Following Results

Path Manager – Straight Line

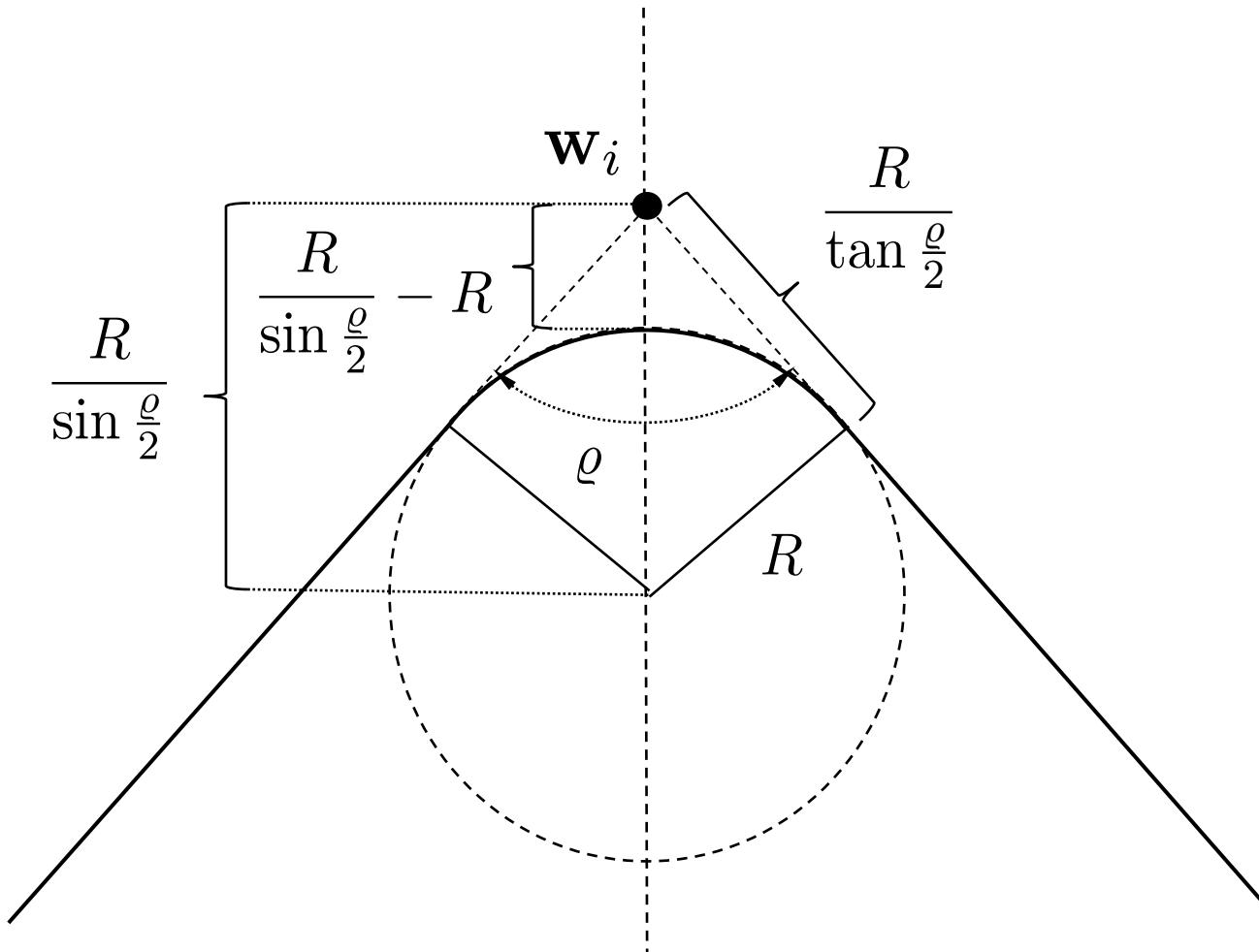


Fillet Transition

Transition between path segments can be smoothed by adding a fillet



Fillet Geometry



Fillet Smoothing Half Planes

Fillet center defined as

$$\mathbf{c} = \mathbf{w}_i - \left(\frac{R}{\sin \frac{\varrho}{2}} \right) \frac{\mathbf{q}_{i-1} - \mathbf{q}_i}{\|\mathbf{q}_{i-1} - \mathbf{q}_i\|}$$

Half plane \mathcal{H}_1 is defined by location

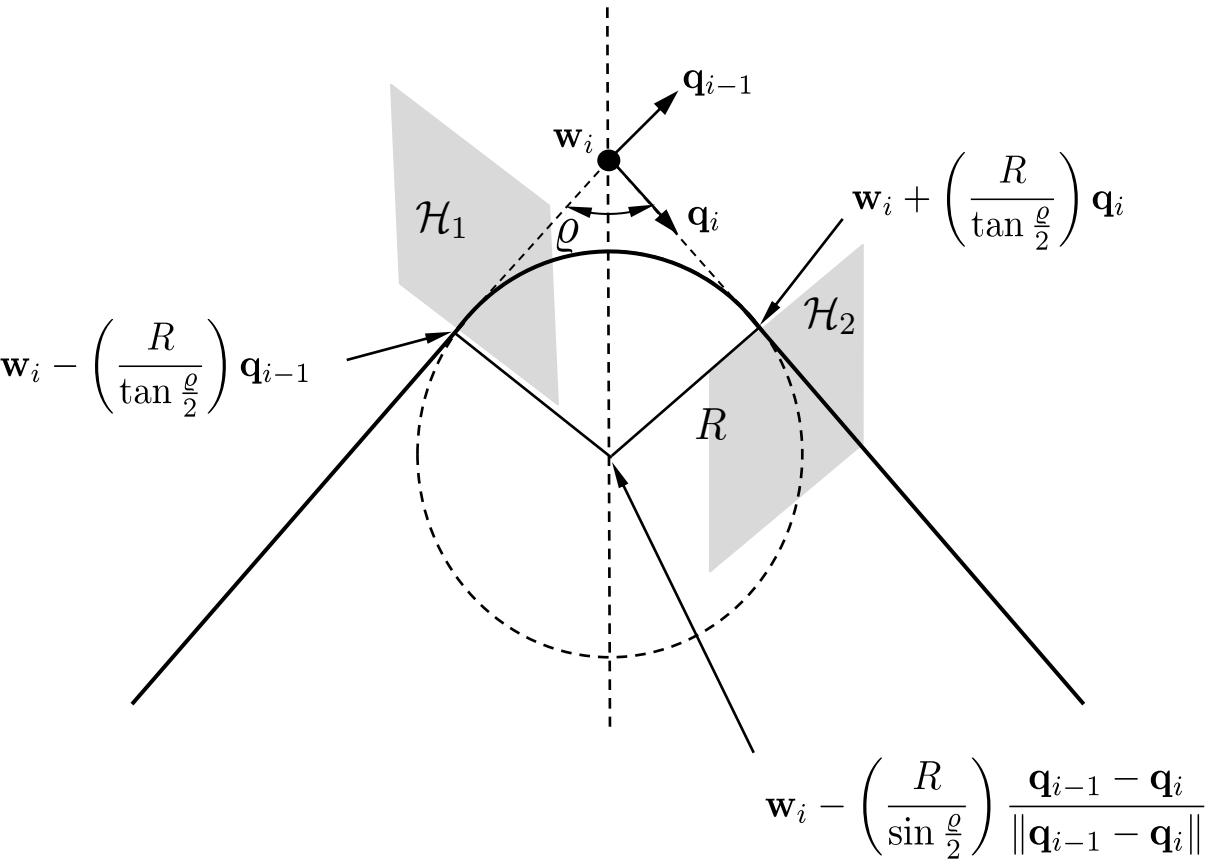
$$\mathbf{r}_1 = \mathbf{w}_i - \left(\frac{R}{\tan \frac{\varrho}{2}} \right) \mathbf{q}_{i-1}$$

and normal vector \mathbf{q}_{i-1}

Half plane \mathcal{H}_2 is defined by location

$$\mathbf{r}_2 = \mathbf{w}_i + \left(\frac{R}{\tan \frac{\varrho}{2}} \right) \mathbf{q}_i$$

and normal vector \mathbf{q}_i



Waypoint Following with Fillets

Algorithm 1 Follow Waypoints with Fillets: $(\text{flag}, \mathbf{r}, \mathbf{q}, \mathbf{c}, \rho, \lambda) = \text{followWppFillet}(\mathcal{W}, \mathbf{p}, R)$

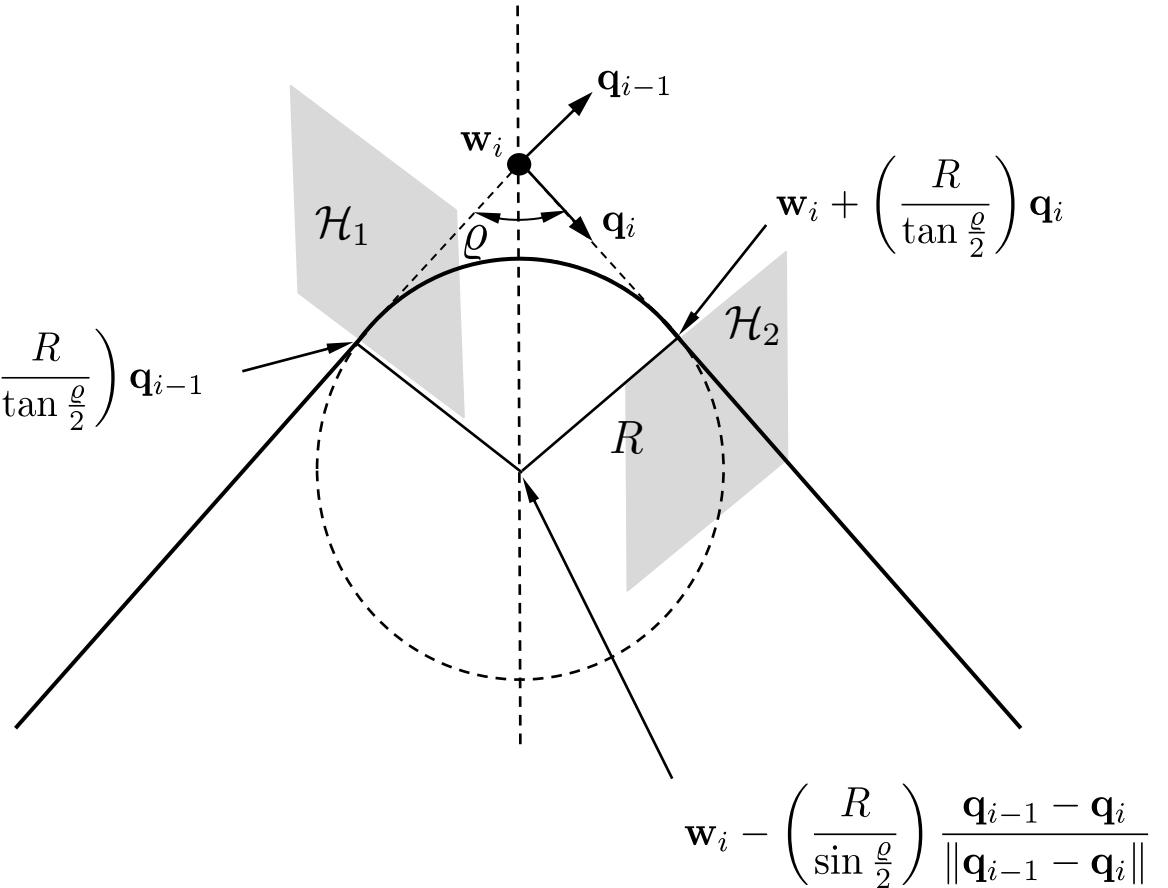
Ensure: Waypoint path $\mathcal{W} = \{\mathbf{w}_1, \dots, \mathbf{w}_N\}$, MAV position $\mathbf{p} = (p_n, p_e, p_d)^\top$, fillet radius R .

Require: $N \geq 3$.

```

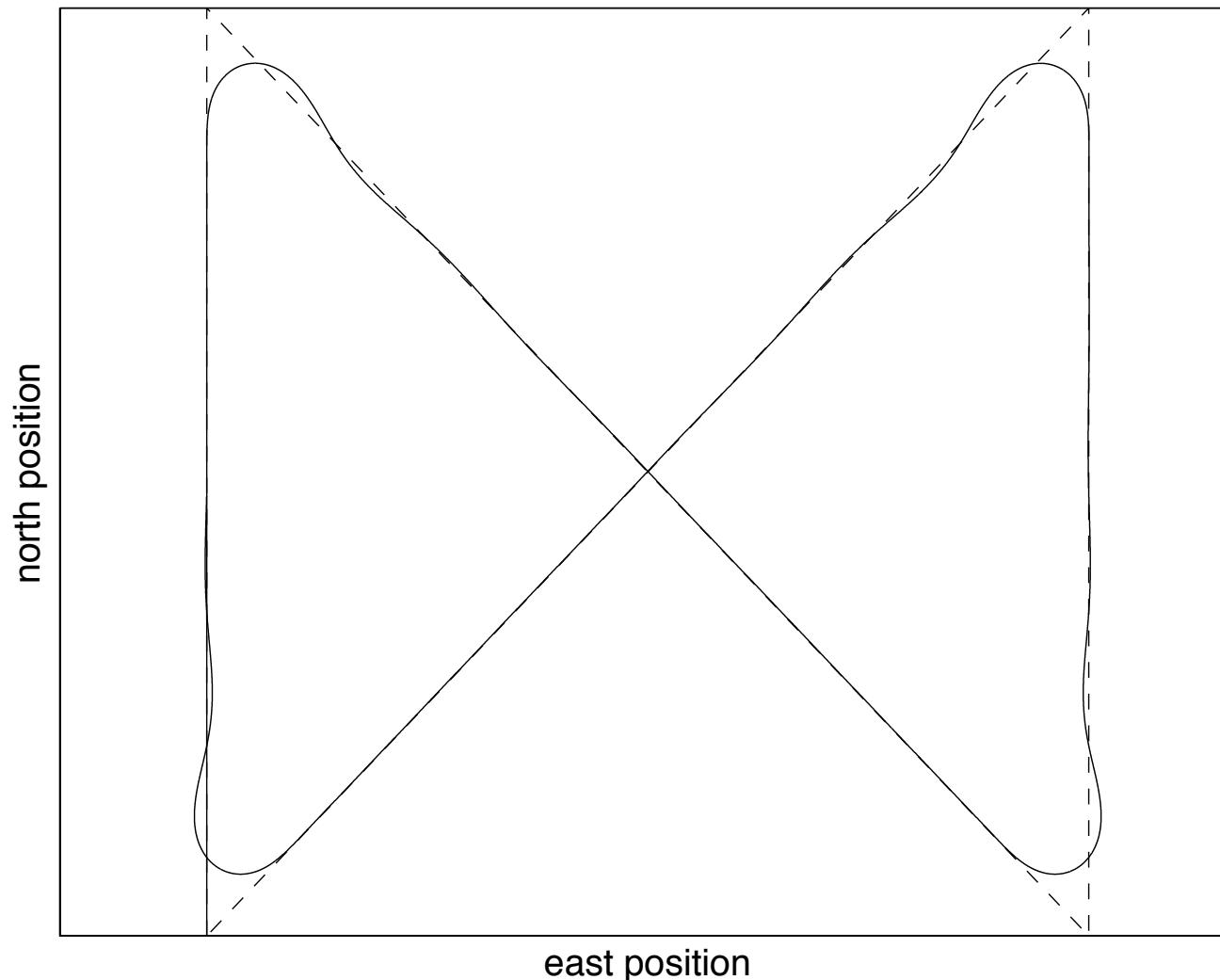
1: if New waypoint path  $\mathcal{W}$  is received then
2:   Initialize waypoint index:  $i \leftarrow 2$ , and state machine: state  $\leftarrow 1$ .
3: end if
4:  $\mathbf{q}_{i-1} \leftarrow \frac{\mathbf{w}_i - \mathbf{w}_{i-1}}{\|\mathbf{w}_i - \mathbf{w}_{i-1}\|}$ .
5:  $\mathbf{q}_i \leftarrow \frac{\mathbf{w}_{i+1} - \mathbf{w}_i}{\|\mathbf{w}_{i+1} - \mathbf{w}_i\|}$ .
6:  $\varrho \leftarrow \cos^{-1}(-\mathbf{q}_{i-1}^\top \mathbf{q}_i)$ .
7: if state = 1 then
8:   flag  $\leftarrow 1$ 
9:    $\mathbf{r} \leftarrow \mathbf{w}_{i-1}$ 
10:   $\mathbf{q} \leftarrow \mathbf{q}_{i-1}$ 
11:   $\mathbf{z} \leftarrow \mathbf{w}_i - \left(\frac{R}{\tan(\varrho/2)}\right) \mathbf{q}_{i-1}$ 
12:  if  $\mathbf{p} \in \mathcal{H}(\mathbf{z}, \mathbf{q}_{i-1})$  then
13:    state  $\leftarrow 2$ 
14:  end if
15: else if state = 2 then
16:   flag  $\leftarrow 2$ 
17:    $\mathbf{c} \leftarrow \mathbf{w}_i - \left(\frac{R}{\sin(\varrho/2)}\right) \frac{\mathbf{q}_{i-1} - \mathbf{q}_i}{\|\mathbf{q}_{i-1} - \mathbf{q}_i\|}$ 
18:    $\rho \leftarrow R$ 
19:    $\lambda \leftarrow \text{sign}(q_{i-1,n} q_{i,e} - q_{i-1,e} q_{i,n})$ .
20:    $\mathbf{z} \leftarrow \mathbf{w}_i + \left(\frac{R}{\tan(\varrho/2)}\right) \mathbf{q}_i$ 
21:   if  $\mathbf{p} \in \mathcal{H}(\mathbf{z}, \mathbf{q}_i)$  then
22:      $i \leftarrow (i+1)$  until  $i = N-1$ .
23:     state  $\leftarrow 1$ 
24:   end if
25: end if
26: return flag,  $\mathbf{r}$ ,  $\mathbf{q}$ ,  $\mathbf{c}$ ,  $\rho$ ,  $\lambda$ .

```



Waypoint Following with Fillets

Path Manager – Fillets



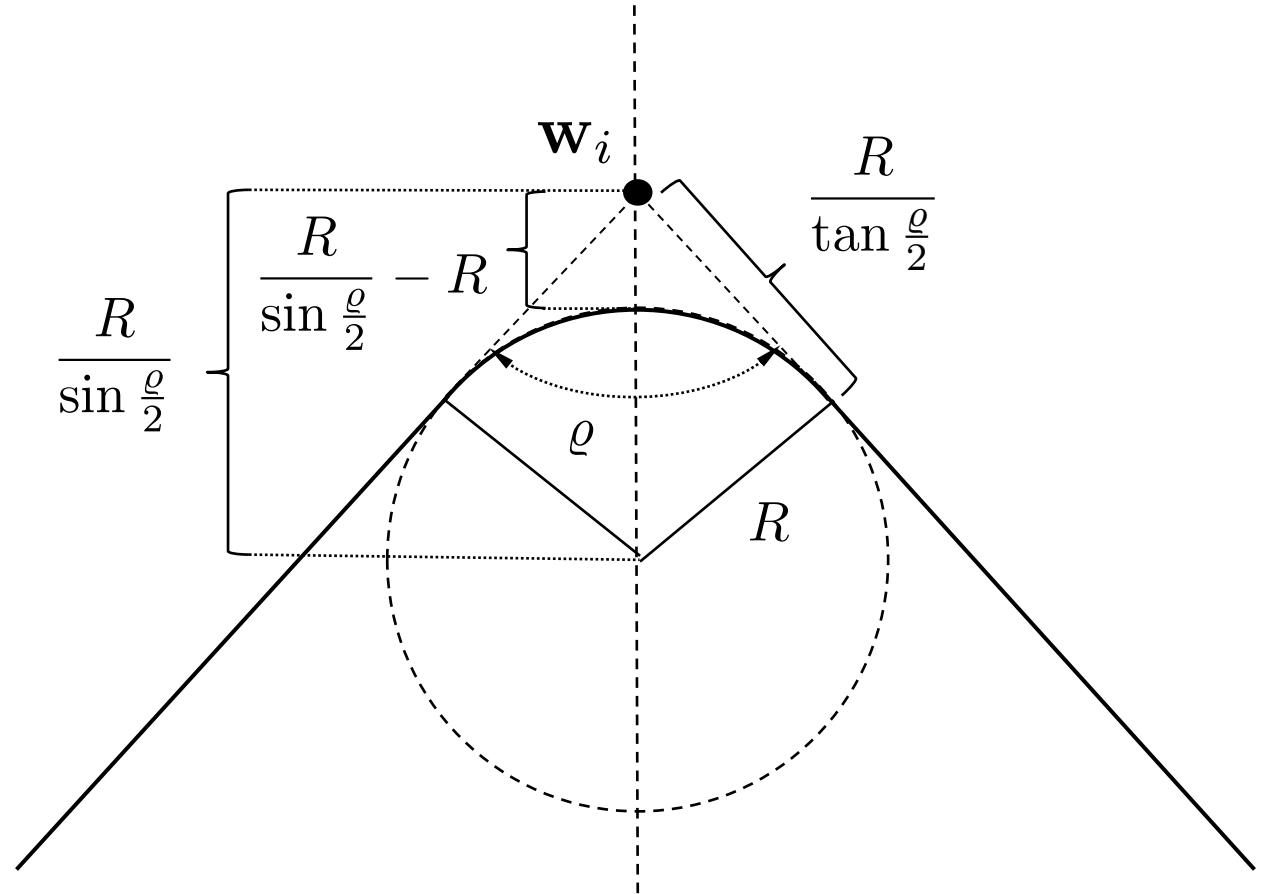
Fillet Path Length

Straight-line path length (no fillets):

$$|\mathcal{W}| \triangleq \sum_{i=2}^N \|\mathbf{w}_i - \mathbf{w}_{i-1}\|.$$

Path length with fillets:

$$|\mathcal{W}|_F = |\mathcal{W}| + \sum_{i=2}^N \left(R\varrho_i - \frac{2R}{\tan \frac{\varrho_i}{2}} \right).$$



Dubins Paths

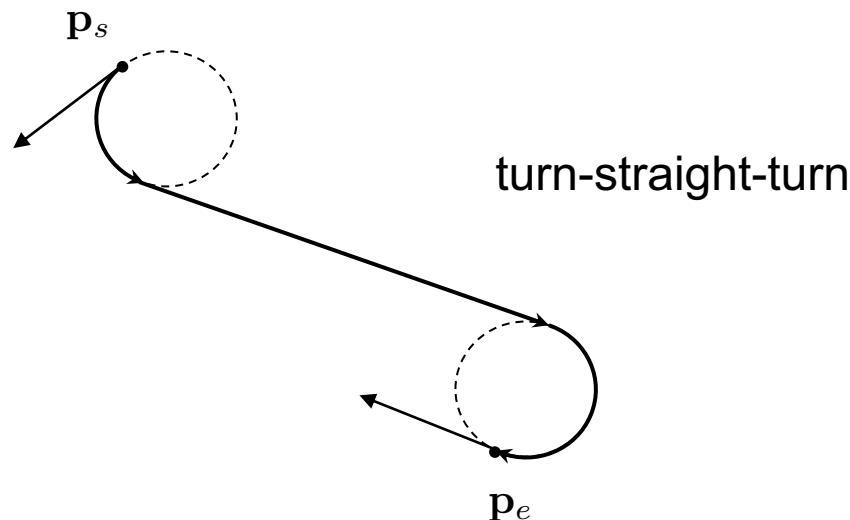
For vehicle with kinematics given by

$$\dot{p}_n = V \cos \vartheta$$

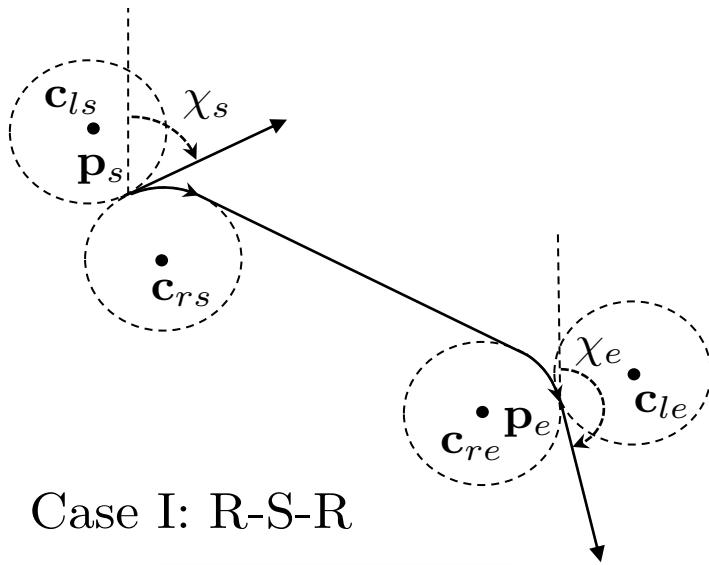
$$\dot{p}_e = V \sin \vartheta$$

$$\dot{\vartheta} = u,$$

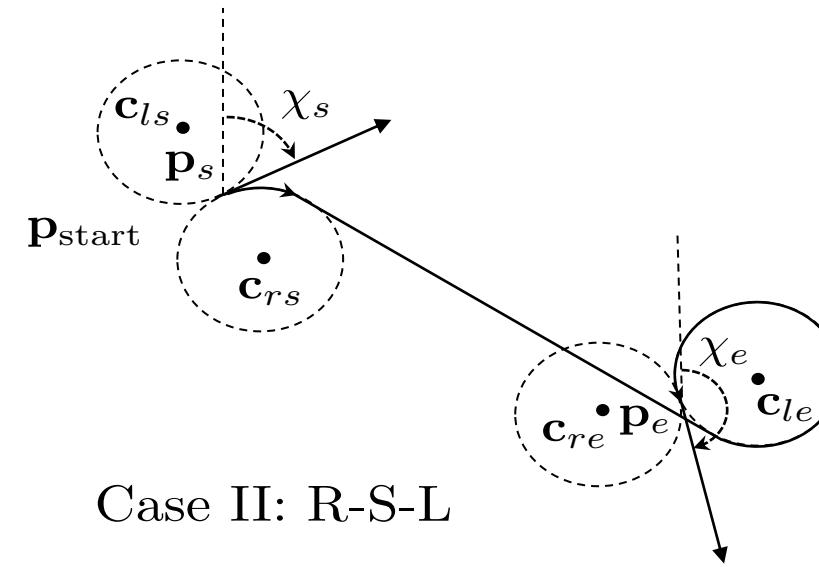
where V is constant and $u \in [-\bar{u}, \bar{u}]$, time-optimal path between two different configurations consists of circular arc, followed by straight line, and concluding with another circular arc to the final configuration, where the radius of the circular arcs is V/\bar{u} .



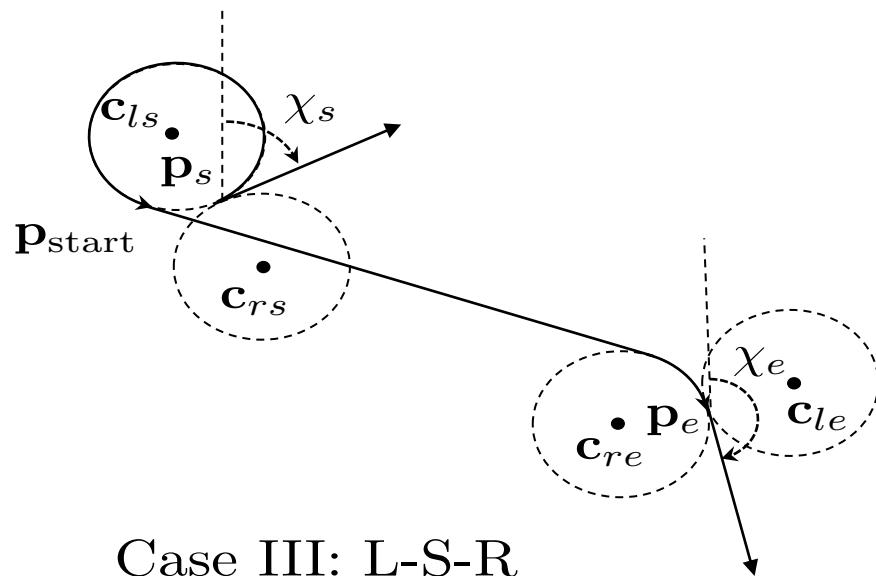
Dubins path is defined as shortest of four cases



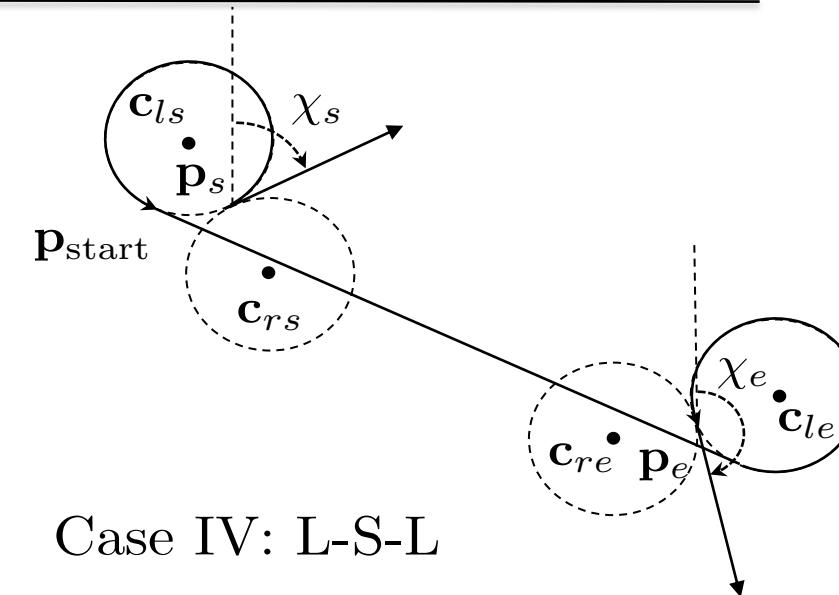
Case I: R-S-R



Case II: R-S-L



Case III: L-S-R



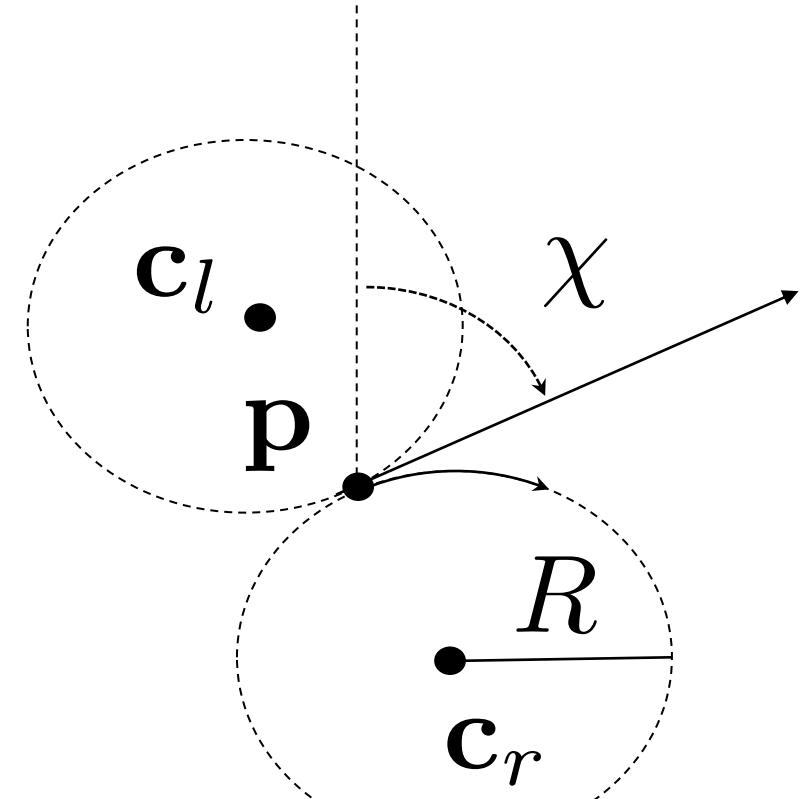
Case IV: L-S-L

Path Length Preliminaries

Given position \mathbf{p} , course χ , and radius R , centers of right and left turning circles are given by

$$\mathbf{c}_r = \mathbf{p} + R \left(\cos\left(\chi + \frac{\pi}{2}\right), \sin\left(\chi + \frac{\pi}{2}\right), 0 \right)^\top$$

$$\mathbf{c}_l = \mathbf{p} + R \left(\cos\left(\chi - \frac{\pi}{2}\right), \sin\left(\chi - \frac{\pi}{2}\right), 0 \right)^\top$$



Path Length Preliminaries

$$\langle 2\pi + \vartheta_2 - \vartheta_1 \rangle$$

For clockwise circles, angular distance between ϑ_1 and ϑ_2 given by

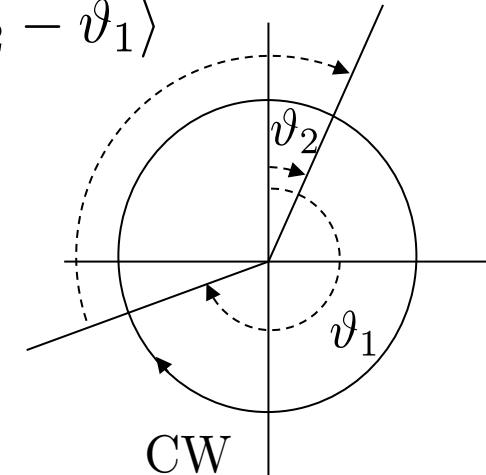
$$|\vartheta_2 - \vartheta_1|_{CW} \triangleq \langle 2\pi + \vartheta_2 - \vartheta_1 \rangle,$$

where

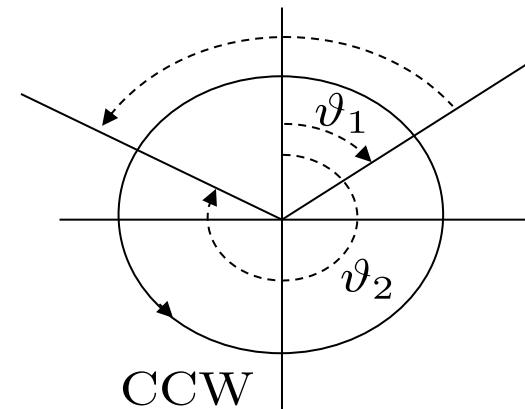
$$\langle \varphi \rangle \triangleq \varphi \mod 2\pi$$

For counter clockwise circles,

$$|\vartheta_2 - \vartheta_1|_{CCW} \triangleq \langle 2\pi - \vartheta_2 + \vartheta_1 \rangle$$



$$\langle 2\pi - \vartheta_2 + \vartheta_1 \rangle$$



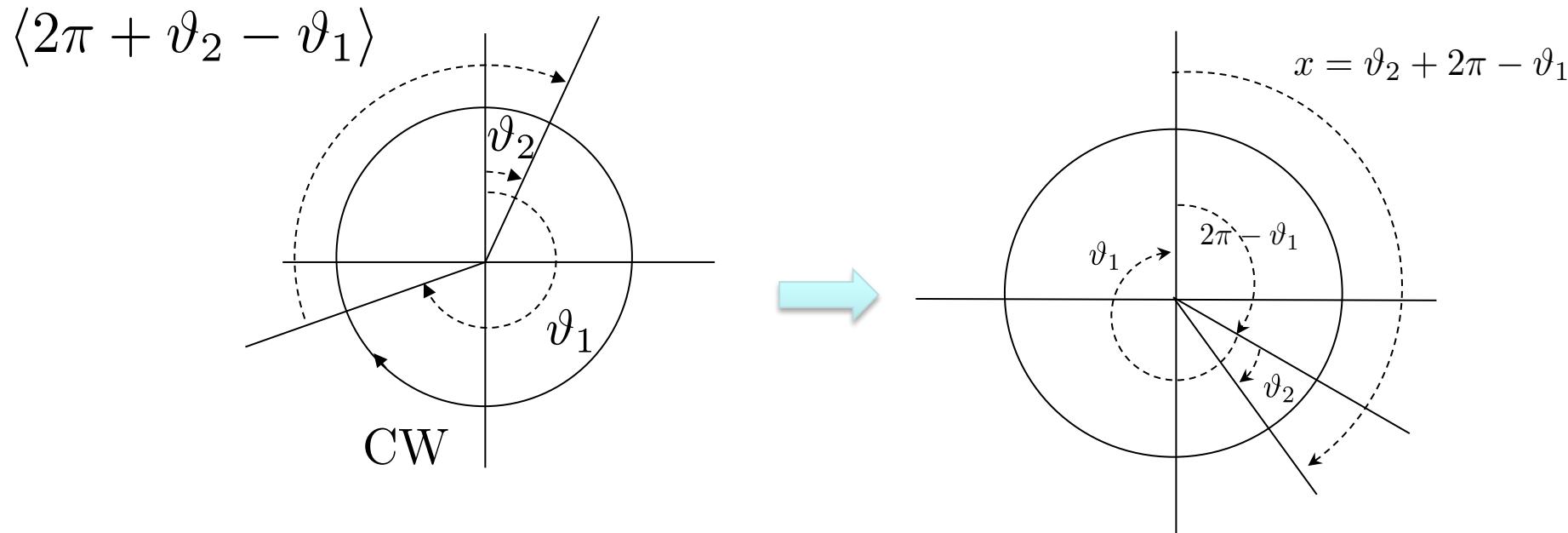
Alternative Viewpoint

For clockwise circles, angular distance between ϑ_1 and ϑ_2 given by

$$|\vartheta_2 - \vartheta_1|_{CW} \triangleq \langle 2\pi + \vartheta_2 - \vartheta_1 \rangle,$$

where

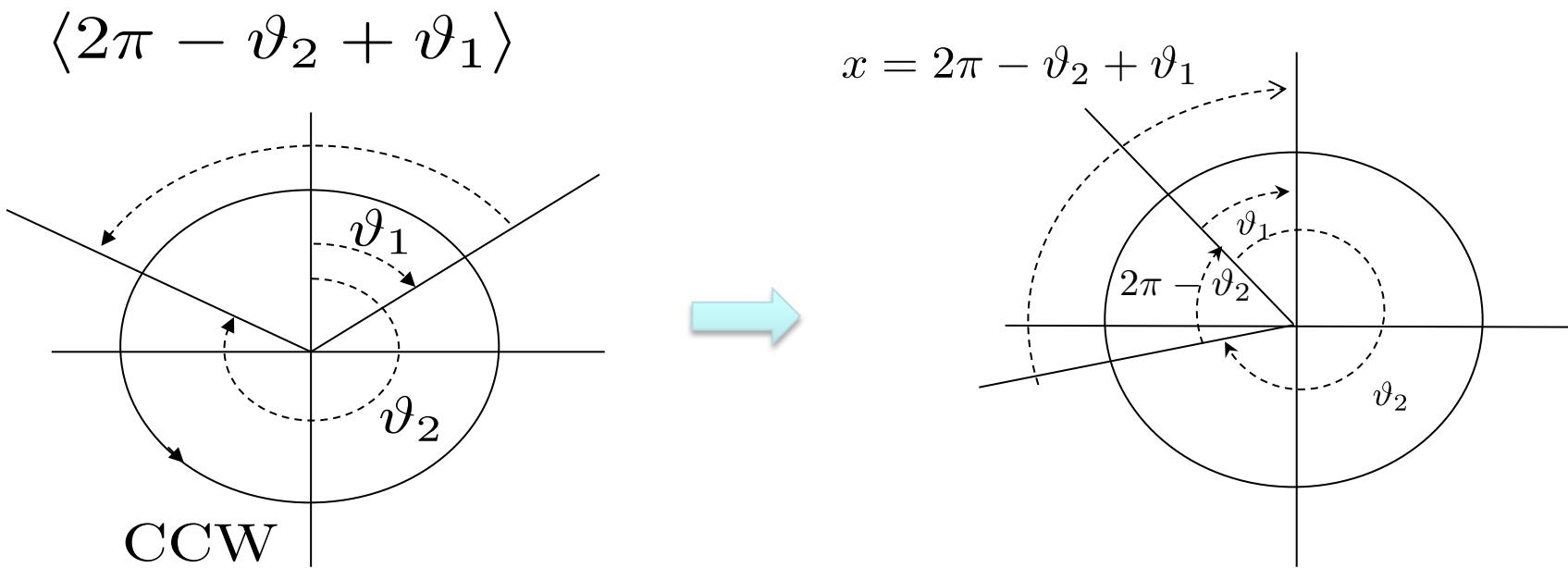
$$\langle \varphi \rangle \triangleq \varphi \mod 2\pi$$



Alternative Viewpoint

For counter clockwise circles,

$$|\vartheta_2 - \vartheta_1|_{CCW} \stackrel{\triangle}{=} \langle 2\pi - \vartheta_2 + \vartheta_1 \rangle$$



Dubins Case I: R-S-R

Distance traveled along \mathbf{c}_{rs}

$$R\left\langle 2\pi + \left\langle \vartheta - \frac{\pi}{2} \right\rangle - \left\langle \chi_s - \frac{\pi}{2} \right\rangle \right\rangle$$

Distance traveled along \mathbf{c}_{re}

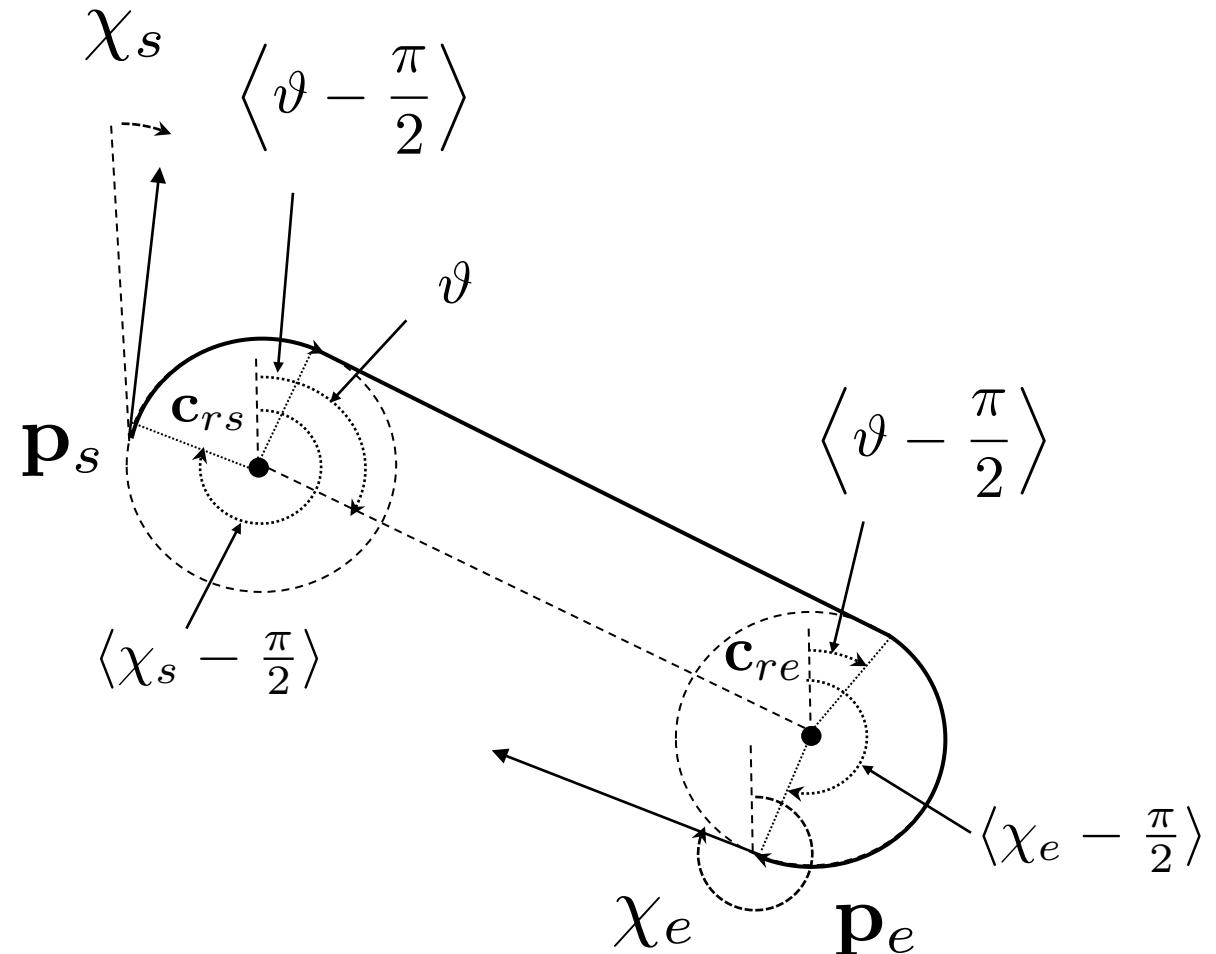
$$R\left\langle 2\pi + \left\langle \chi_e - \frac{\pi}{2} \right\rangle - \left\langle \vartheta - \frac{\pi}{2} \right\rangle \right\rangle$$

where

$$\vartheta = \text{atan2}(c_{ree} - c_{rse}, c_{ren} - c_{rsn})$$

Total path length:

$$L_1 = \|\mathbf{c}_{rs} - \mathbf{c}_{re}\| + R\left\langle 2\pi + \left\langle \vartheta - \frac{\pi}{2} \right\rangle - \left\langle \chi_s - \frac{\pi}{2} \right\rangle \right\rangle + R\left\langle 2\pi + \left\langle \chi_e - \frac{\pi}{2} \right\rangle - \left\langle \vartheta - \frac{\pi}{2} \right\rangle \right\rangle$$



Dubins Case II: R-S-L

Distance traveled along \mathbf{c}_{rs}

$$R\left\langle 2\pi + \left\langle \vartheta_2 \right\rangle - \left\langle \chi_s - \frac{\pi}{2} \right\rangle \right\rangle$$

Distance traveled along \mathbf{c}_{le}

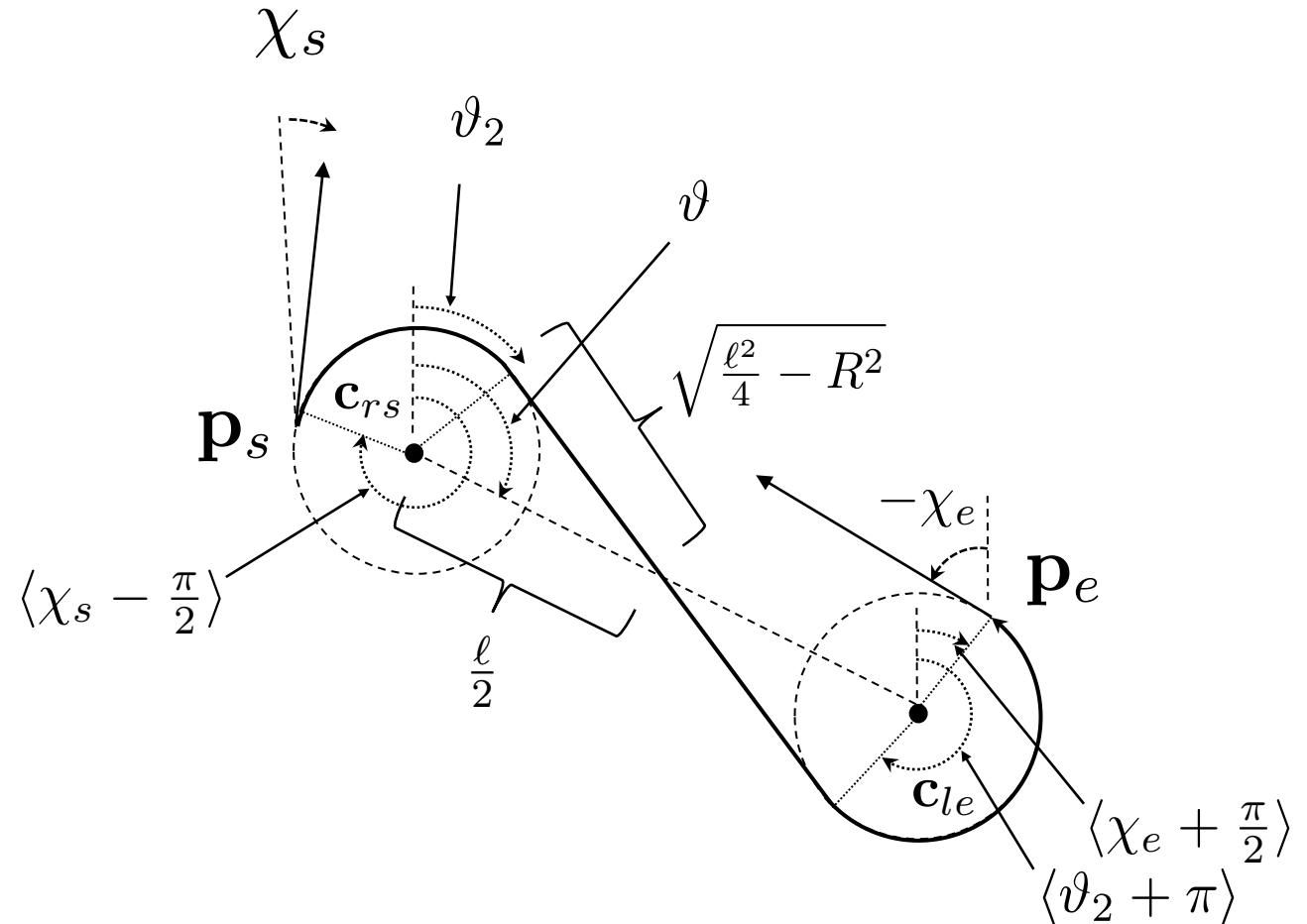
$$R\left\langle 2\pi + \left\langle \vartheta_2 + \pi \right\rangle - \left\langle \chi_e + \frac{\pi}{2} \right\rangle \right\rangle$$

where

$$\vartheta_2 = \vartheta - \frac{\pi}{2} + \sin^{-1} \left(\frac{2R}{\ell} \right)$$

Total path length:

$$L_2 = \sqrt{\ell^2 - 4R^2} + R\left\langle 2\pi + \left\langle \vartheta_2 \right\rangle - \left\langle \chi_s - \frac{\pi}{2} \right\rangle \right\rangle + R\left\langle 2\pi + \left\langle \vartheta_2 + \pi \right\rangle - \left\langle \chi_e + \frac{\pi}{2} \right\rangle \right\rangle$$



Dubins Case III: L-S-R

Distance traveled along \mathbf{c}_{ls}

$$R\left\langle 2\pi + \left\langle \chi_s + \frac{\pi}{2} \right\rangle - \langle \vartheta + \vartheta_2 \rangle \right\rangle$$

Distance traveled along \mathbf{c}_{re}

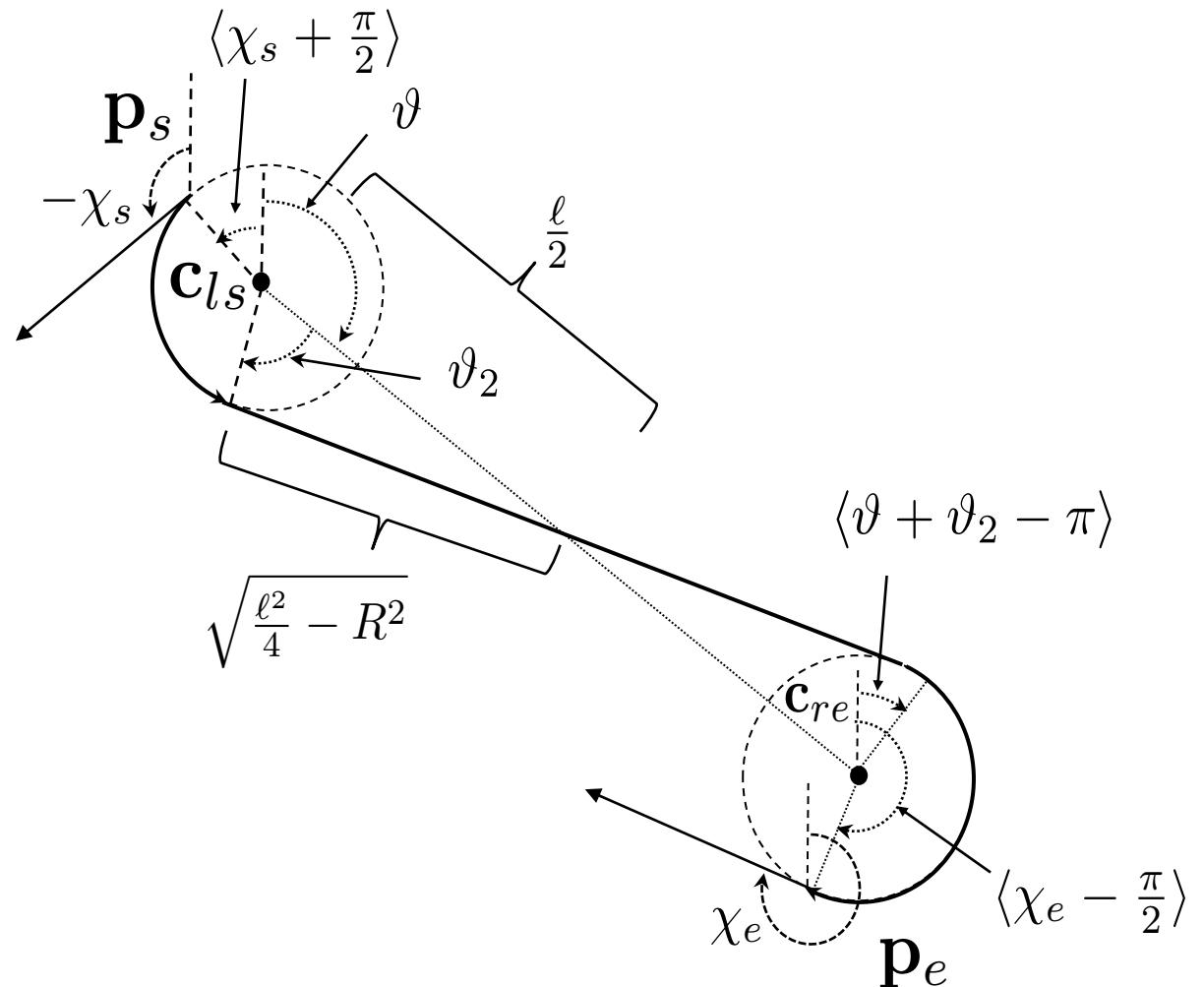
$$R\left\langle 2\pi + \left\langle \chi_e - \frac{\pi}{2} \right\rangle - \langle \vartheta + \vartheta_2 - \pi \rangle \right\rangle$$

where

$$\vartheta_2 = \cos^{-1} \left(\frac{2R}{\ell} \right)$$

Total path length:

$$L_3 = \sqrt{\ell^2 - 4R^2} + R\left\langle 2\pi + \left\langle \chi_s + \frac{\pi}{2} \right\rangle - \langle \vartheta + \vartheta_2 \rangle \right\rangle + R\left\langle 2\pi + \left\langle \chi_e - \frac{\pi}{2} \right\rangle - \langle \vartheta + \vartheta_2 - \pi \rangle \right\rangle$$



Dubins Case IV: L-S-L

Distance traveled along \mathbf{c}_{ls}

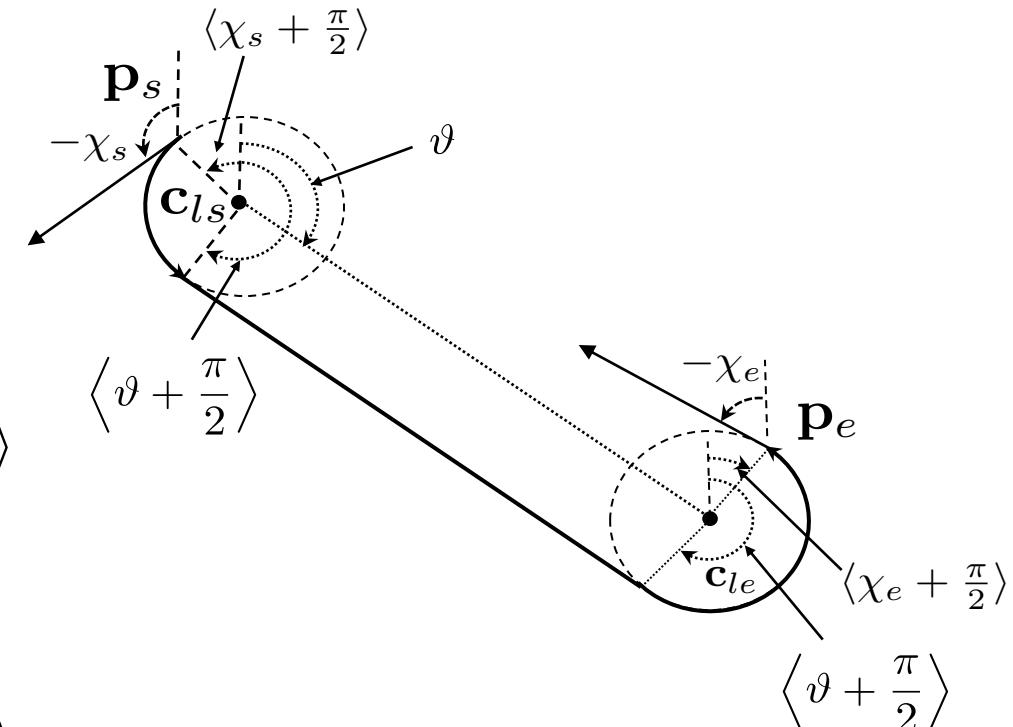
$$R\left\langle 2\pi + \left\langle \chi_s + \frac{\pi}{2} \right\rangle - \left\langle \vartheta + \frac{\pi}{2} \right\rangle \right\rangle$$

Distance traveled along \mathbf{c}_{le}

$$R\left\langle 2\pi + \left\langle \vartheta + \frac{\pi}{2} \right\rangle - \left\langle \chi_e + \frac{\pi}{2} \right\rangle \right\rangle$$

Total path length:

$$L_4 = \|\mathbf{c}_{ls} - \mathbf{c}_{le}\| + R\left\langle 2\pi + \left\langle \chi_s + \frac{\pi}{2} \right\rangle - \left\langle \vartheta + \frac{\pi}{2} \right\rangle \right\rangle + R\left\langle 2\pi + \left\langle \vartheta + \frac{\pi}{2} \right\rangle - \left\langle \chi_e + \frac{\pi}{2} \right\rangle \right\rangle$$



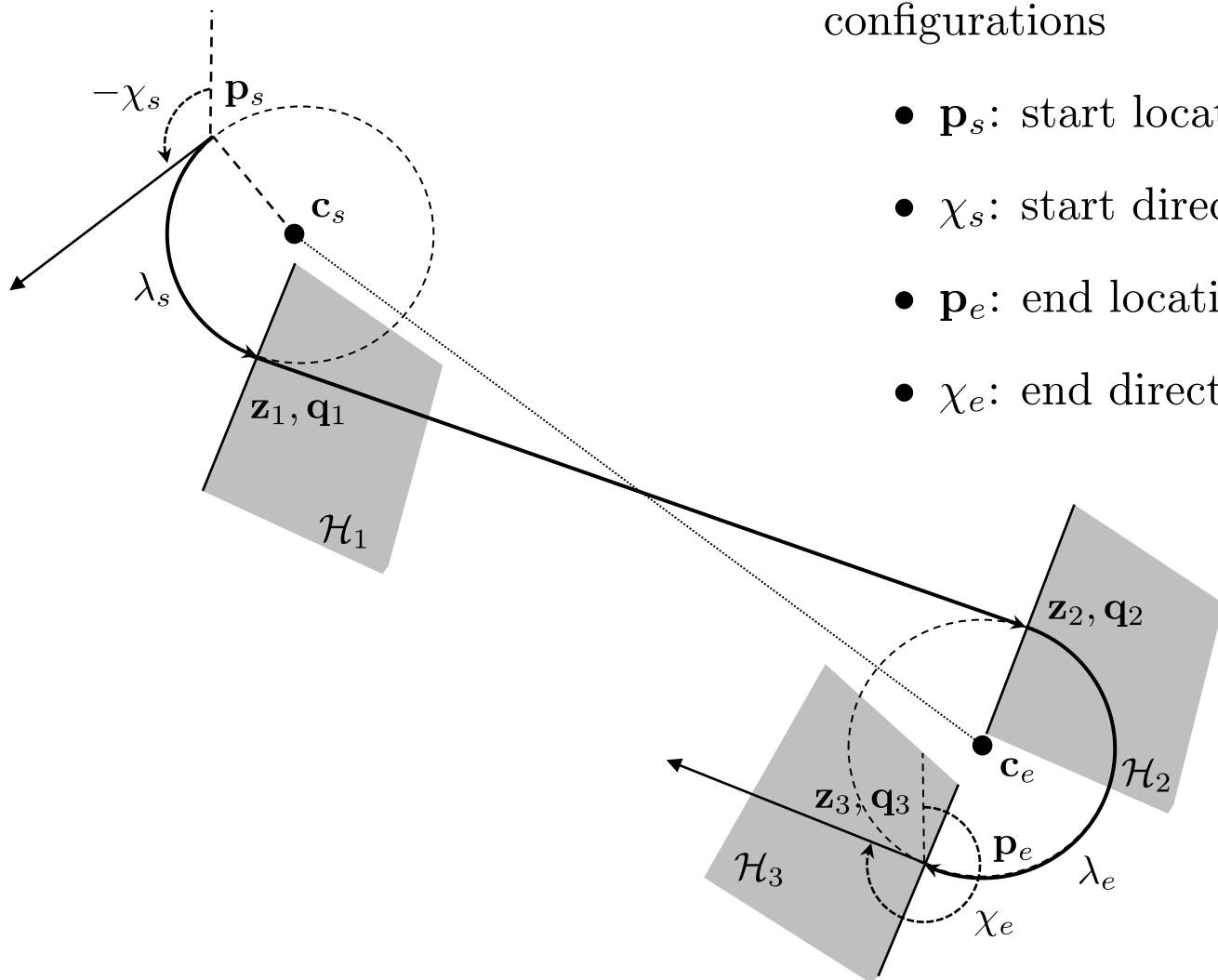
Dubins Path Half-plane Switching

Given desired start and end configurations

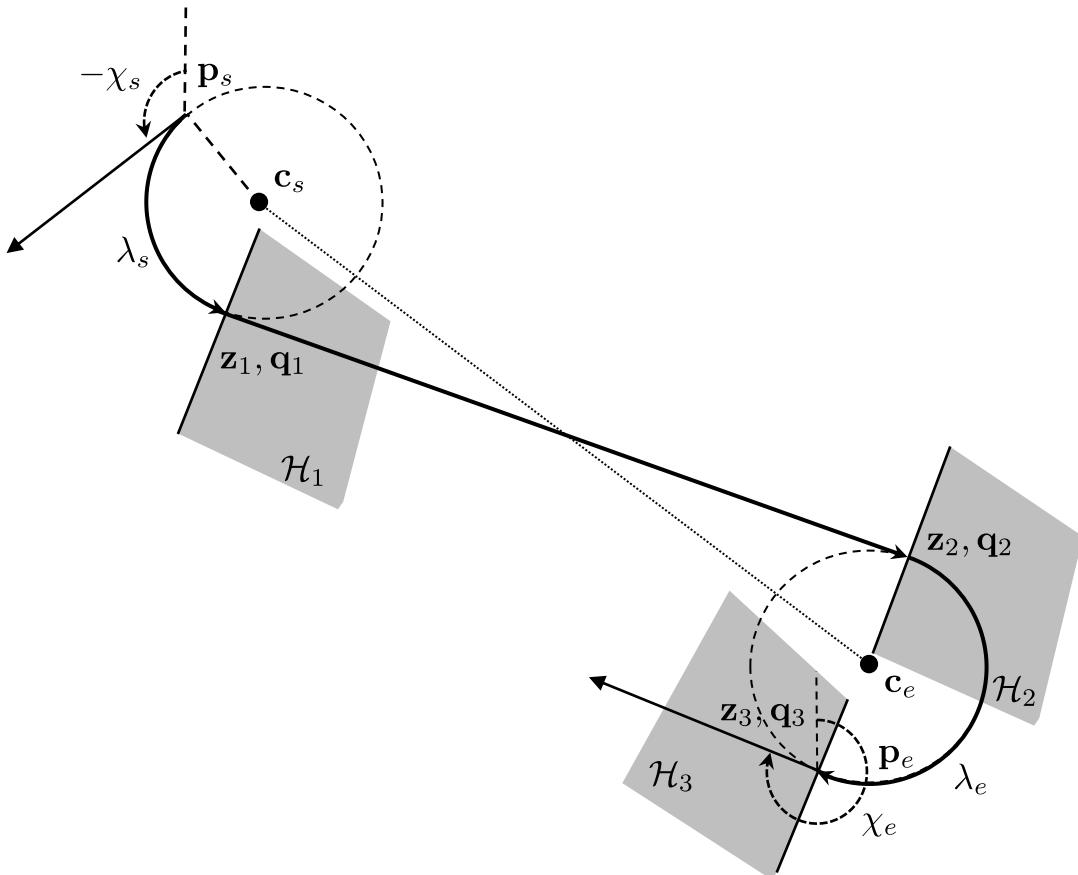
- \mathbf{p}_s : start location
- χ_s : start direction
- \mathbf{p}_e : end location
- χ_e : end direction

Calculate Dubins path parameters

- \mathbf{c}_s : start circle location
- λ_s : start circle direction
- \mathbf{c}_e : end circle location
- λ_e : end circle direction
- $\mathbf{z}_1, \mathbf{q}_1$: half-plane \mathcal{H}_1 parameters
- $\mathbf{z}_2, \mathbf{q}_2$: half-plane \mathcal{H}_2 parameters
- $\mathbf{z}_3, \mathbf{q}_3$: half-plane \mathcal{H}_3 parameters



Dubins Path Parameters Algorithm



Algorithm 7 Find Dubins Parameters:

$(L, \mathbf{c}_s, \lambda_s, \mathbf{c}_e, \lambda_e, \mathbf{z}_1, \mathbf{q}_1, \mathbf{z}_2, \mathbf{z}_3, \mathbf{q}_3) =$
findDubinsParameters($\mathbf{p}_s, \chi_s, \mathbf{p}_e, \chi_e, R$)

Input: Start configuration (\mathbf{p}_s, χ_s) , End configuration (\mathbf{p}_e, χ_e) , Radius R .

Require: $\|\mathbf{p}_s - \mathbf{p}_e\| \geq 3R$

Require: R is larger than minimum turn radius of MAV

- 1: $\mathbf{c}_{rs} \leftarrow \mathbf{p}_s + R\mathcal{R}_z\left(\frac{\pi}{2}\right)(\cos \chi_s, \sin \chi_s, 0)^T$
- 2: $\mathbf{c}_{ls} \leftarrow \mathbf{p}_s + R\mathcal{R}_z\left(-\frac{\pi}{2}\right)(\cos \chi_s, \sin \chi_s, 0)^T$
- 3: $\mathbf{c}_{re} \leftarrow \mathbf{p}_e + R\mathcal{R}_z\left(\frac{\pi}{2}\right)(\cos \chi_e, \sin \chi_e, 0)^T$
- 4: $\mathbf{c}_{le} \leftarrow \mathbf{p}_e + R\mathcal{R}_z\left(-\frac{\pi}{2}\right)(\cos \chi_e, \sin \chi_e, 0)^T$
- 5: Compute L_1, L_2, L_3 , and L_4 using equations (11.9) through (11.12).

- 6: $L \leftarrow \min\{L_1, L_2, L_3, L_4\}$

- 7: if $\arg \min\{L_1, L_2, L_3, L_4\} = 1$ then

- 8: $\mathbf{c}_s \leftarrow \mathbf{c}_{rs}, \quad \lambda_s \leftarrow +1, \quad \mathbf{c}_e \leftarrow \mathbf{c}_{re}, \quad \lambda_e \leftarrow +1$
- 9: $\mathbf{q}_1 \leftarrow \frac{\mathbf{c}_e - \mathbf{c}_s}{\|\mathbf{c}_e - \mathbf{c}_s\|}$
- 10: $\mathbf{z}_1 \leftarrow \mathbf{c}_s + R\mathcal{R}_z\left(-\frac{\pi}{2}\right)\mathbf{q}_1$
- 11: $\mathbf{z}_2 \leftarrow \mathbf{c}_e + R\mathcal{R}_z\left(-\frac{\pi}{2}\right)\mathbf{q}_1$

- 12: else if $\arg \min\{L_1, L_2, L_3, L_4\} = 2$ then

- 13: $\mathbf{c}_s \leftarrow \mathbf{c}_{rs}, \quad \lambda_s \leftarrow +1, \quad \mathbf{c}_e \leftarrow \mathbf{c}_{le}, \quad \lambda_e \leftarrow -1$
- 14: $\ell \leftarrow \|\mathbf{c}_e - \mathbf{c}_s\|$
- 15: $\vartheta \leftarrow \text{angle}(\mathbf{c}_e - \mathbf{c}_s)$
- 16: $\vartheta_2 \leftarrow \vartheta - \frac{\pi}{2} + \sin^{-1} \frac{2R}{\ell}$
- 17: $\mathbf{q}_1 \leftarrow \mathcal{R}_z\left(\vartheta_2 + \frac{\pi}{2}\right)\mathbf{e}_1$
- 18: $\mathbf{z}_1 \leftarrow \mathbf{c}_s + R\mathcal{R}_z(\vartheta_2)\mathbf{e}_1$
- 19: $\mathbf{z}_2 \leftarrow \mathbf{c}_e + R\mathcal{R}_z(\vartheta_2 + \pi)\mathbf{e}_1$

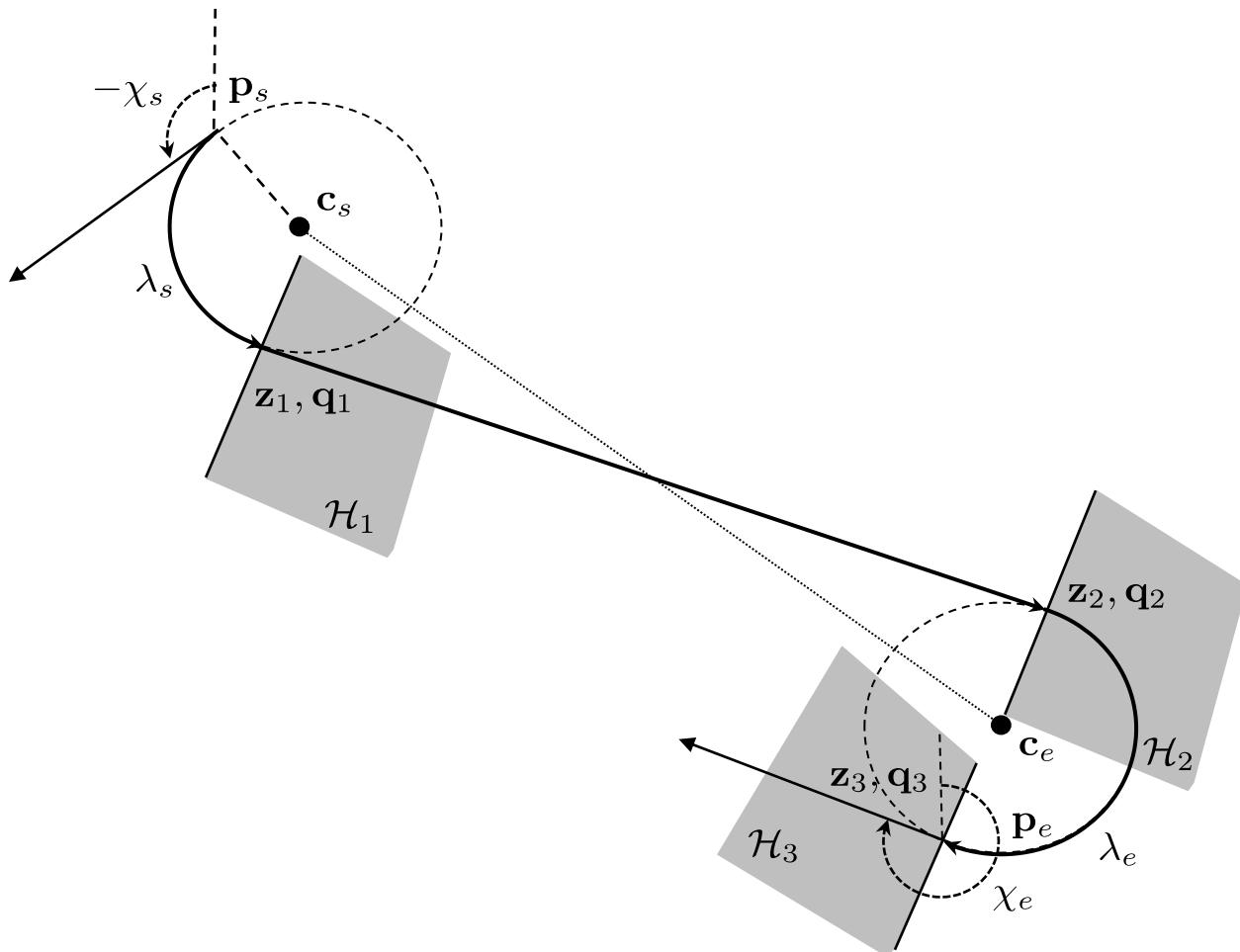
- 20: else if $\arg \min\{L_1, L_2, L_3, L_4\} = 3$ then

- 21: $\mathbf{c}_s \leftarrow \mathbf{c}_{ls}, \quad \lambda_s \leftarrow -1, \quad \mathbf{c}_e \leftarrow \mathbf{c}_{re}, \quad \lambda_e \leftarrow +1$
- 22: $\ell \leftarrow \|\mathbf{c}_e - \mathbf{c}_s\|$
- 23: $\vartheta \leftarrow \text{angle}(\mathbf{c}_e - \mathbf{c}_s)$,
- 24: $\vartheta_2 \leftarrow \cos^{-1} \frac{2R}{\ell}$
- 25: $\mathbf{q}_1 \leftarrow \mathcal{R}_z\left(\vartheta + \vartheta_2 - \frac{\pi}{2}\right)\mathbf{e}_1$,
- 26: $\mathbf{z}_1 \leftarrow \mathbf{c}_s + R\mathcal{R}_z(\vartheta + \vartheta_2)\mathbf{e}_1$,
- 27: $\mathbf{z}_2 \leftarrow \mathbf{c}_e + R\mathcal{R}_z(\vartheta + \vartheta_2 - \pi)\mathbf{e}_1$

- 28: else if $\arg \min\{L_1, L_2, L_3, L_4\} = 4$ then

- 29: $\mathbf{c}_s \leftarrow \mathbf{c}_{ls}, \quad \lambda_s \leftarrow -1, \quad \mathbf{c}_e \leftarrow \mathbf{c}_{le}, \quad \lambda_e \leftarrow -1$
- 30: $\mathbf{q}_1 \leftarrow \frac{\mathbf{c}_e - \mathbf{c}_s}{\|\mathbf{c}_e - \mathbf{c}_s\|}$,
- 31: $\mathbf{z}_1 \leftarrow \mathbf{c}_s + R\mathcal{R}_z\left(\frac{\pi}{2}\right)\mathbf{q}_1$,
- 32: $\mathbf{z}_2 \leftarrow \mathbf{c}_e + R\mathcal{R}_z\left(\frac{\pi}{2}\right)\mathbf{q}_2$
- 33: end if
- 34: $\mathbf{z}_3 \leftarrow \mathbf{p}_e$
- 35: $\mathbf{q}_3 \leftarrow \mathcal{R}_z(\chi_e)\mathbf{e}_1$

Dubins Path Following Algorithm



Algorithm 8 Follow Waypoints with Dubins: $(\text{flag}, \mathbf{r}, \mathbf{q}, \mathbf{c}, \rho, \lambda) = \text{followWppDubins}(\mathcal{P}, \mathbf{p}, R)$

Input: Configuration path $\mathcal{P} = \{(\mathbf{w}_1, \chi_1), \dots, (\mathbf{w}_N, \chi_N)\}$, MAV position $\mathbf{p} = (p_n, p_e, p_d)^\top$, fillet radius R .

Require: $N \geq 3$

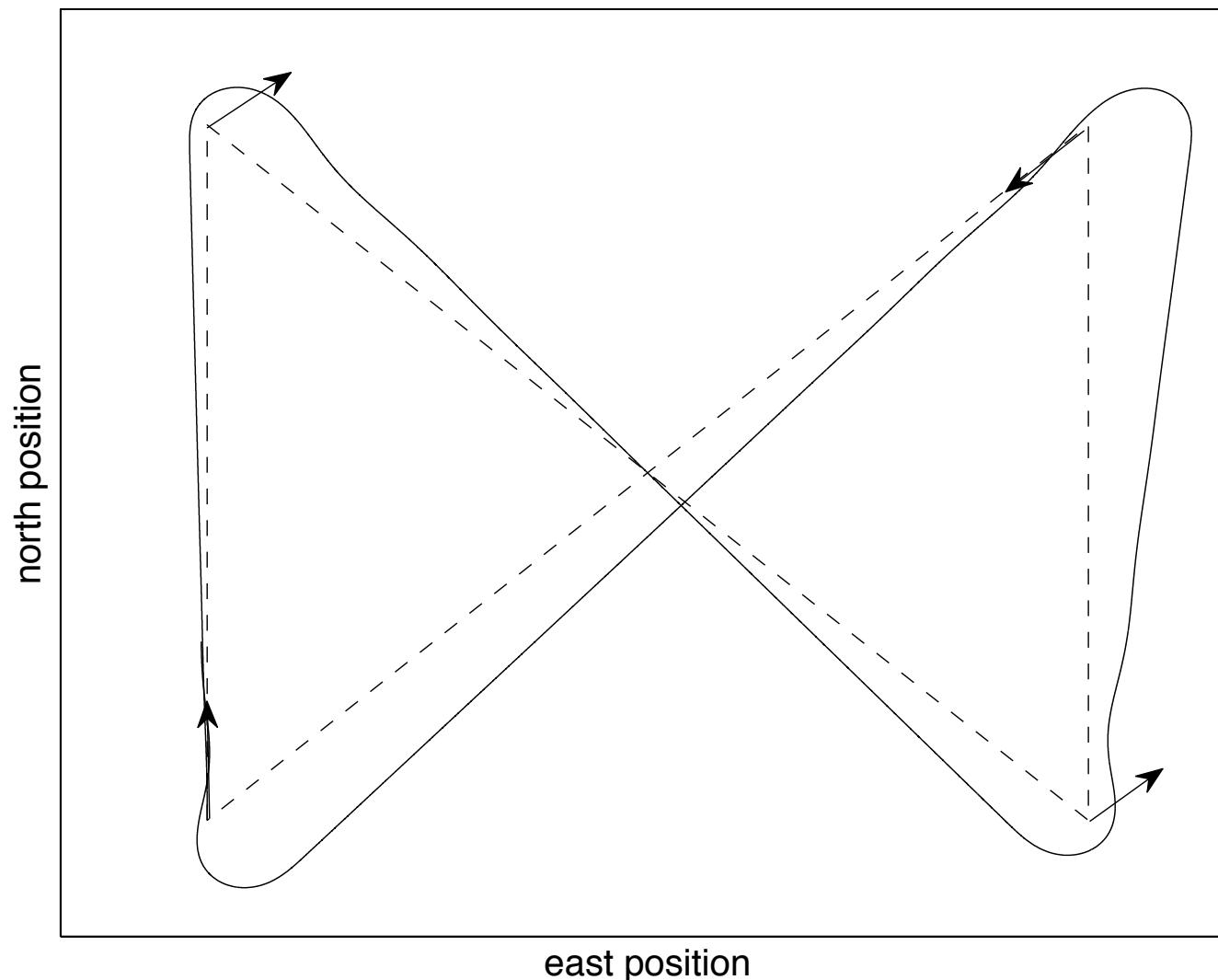
```

1: if New configuration path  $\mathcal{P}$  is received then
2:   Initialize waypoint pointer:  $i \leftarrow 2$ , and state machine: state  $\leftarrow 1$ .
3: end if
4:  $(L, \mathbf{c}_s, \lambda_s, \mathbf{c}_e, \lambda_e, \mathbf{z}_1, \mathbf{q}_1, \mathbf{z}_2, \mathbf{z}_3, \mathbf{q}_3) \leftarrow$ 
   findDubinsParameters( $\mathbf{w}_{i-1}, \chi_{i-1}, \mathbf{w}_i, \chi_i, R$ )
5: if state = 1 then
6:   flag  $\leftarrow 2$ ,  $\mathbf{c} \leftarrow \mathbf{c}_s$ ,  $\rho \leftarrow R$ ,  $\lambda \leftarrow \lambda_s$ 
7:   if  $\mathbf{p} \in \mathcal{H}(\mathbf{z}_1, -\mathbf{q}_1)$  then
8:     state  $\leftarrow 2$ 
9:   end if
10: else if state = 2 then
11:   if  $\mathbf{p} \in \mathcal{H}(\mathbf{z}_1, \mathbf{q}_1)$  then
12:     state  $\leftarrow 3$ 
13:   end if
14: else if state = 3 then
15:   flag  $\leftarrow 1$ ,  $\mathbf{r} \leftarrow \mathbf{z}_1$ ,  $\mathbf{q} \leftarrow \mathbf{q}_1$ 
16:   if  $\mathbf{p} \in \mathcal{H}(\mathbf{z}_2, \mathbf{q}_1)$  then
17:     state  $\leftarrow 4$ 
18:   end if
19: else if state = 4 then
20:   flag  $\leftarrow 2$ ,  $\mathbf{c} \leftarrow \mathbf{c}_e$ ,  $\rho \leftarrow R$ ,  $\lambda \leftarrow \lambda_e$ 
21:   if  $\mathbf{p} \in \mathcal{H}(\mathbf{z}_3, -\mathbf{q}_3)$  then
22:     state  $\leftarrow 5$ 
23:   end if
24: else if state = 5 then
25:   if  $\mathbf{p} \in \mathcal{H}(\mathbf{z}_3, \mathbf{q}_3)$  then
26:     state  $\leftarrow 1$ 
27:      $i \leftarrow (i + 1)$  until  $i = N$ .
28:      $(L, \mathbf{c}_s, \lambda_s, \mathbf{c}_e, \lambda_e, \mathbf{z}_1, \mathbf{q}_1, \mathbf{z}_2, \mathbf{z}_3, \mathbf{q}_3) \leftarrow$ 
       findDubinsParameters( $\mathbf{w}_{i-1}, \chi_{i-1}, \mathbf{w}_i, \chi_i, R$ )
29:   end if
30: end if
31: return flag,  $\mathbf{r}$ ,  $\mathbf{q}$ ,  $\mathbf{c}$ ,  $\rho$ ,  $\lambda$ .

```

Dubins Path Following Results

Path Manager – Dubins



Need to incorporate in this next set of slides into book, and also slide deck.

C-C-C DUBINS PATHS

- Dubins RRT in dense obstacle fields works better with short segment lengths
- While methods exist to determine optimality without calculating paths [6], an exhaustive search was implemented as in the textbook with help from [7]
- An additional state is needed in the path manager (state 3.5)

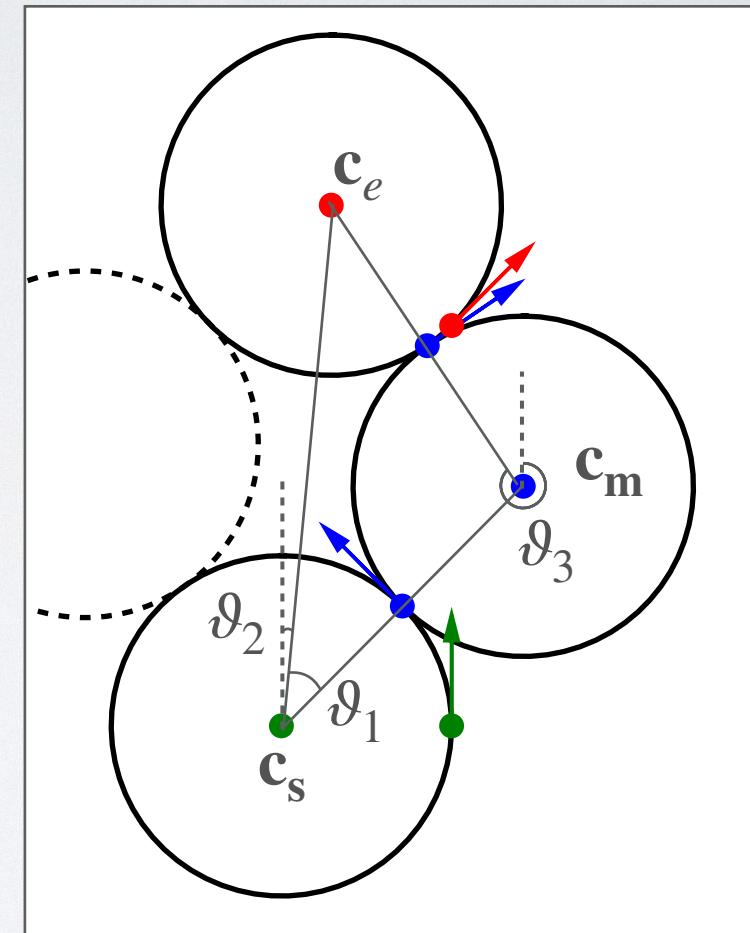
Case	Conditions for Validity
Case I: R-S-R	Always
Case II: R-S-L	$\ \mathbf{p}_e - \mathbf{p}_s\ \geq 4R$ and $\ \mathbf{c}_{le} - \mathbf{c}_{rs}\ \geq 4R$
Case III: L-S-R	$\ \mathbf{p}_e - \mathbf{p}_s\ \geq 4R$ and $\ \mathbf{c}_{re} - \mathbf{c}_{ls}\ \geq 4R$
Case IV: L-S-L	Always
Case V: R-L-R	$\ \mathbf{p}_e - \mathbf{p}_s\ \leq 4R$ and $\ \mathbf{c}_{re} - \mathbf{c}_{rs}\ \geq 4R$
Case VI: L-R-L	$\ \mathbf{p}_e - \mathbf{p}_s\ \leq 4R$ and $\ \mathbf{c}_{le} - \mathbf{c}_{ls}\ \geq 4R$

C-C-C DUBINS PATHS

- The angle between $\mathbf{c}_e - \mathbf{c}_s$ and the middle circle center \mathbf{c}_m is given by

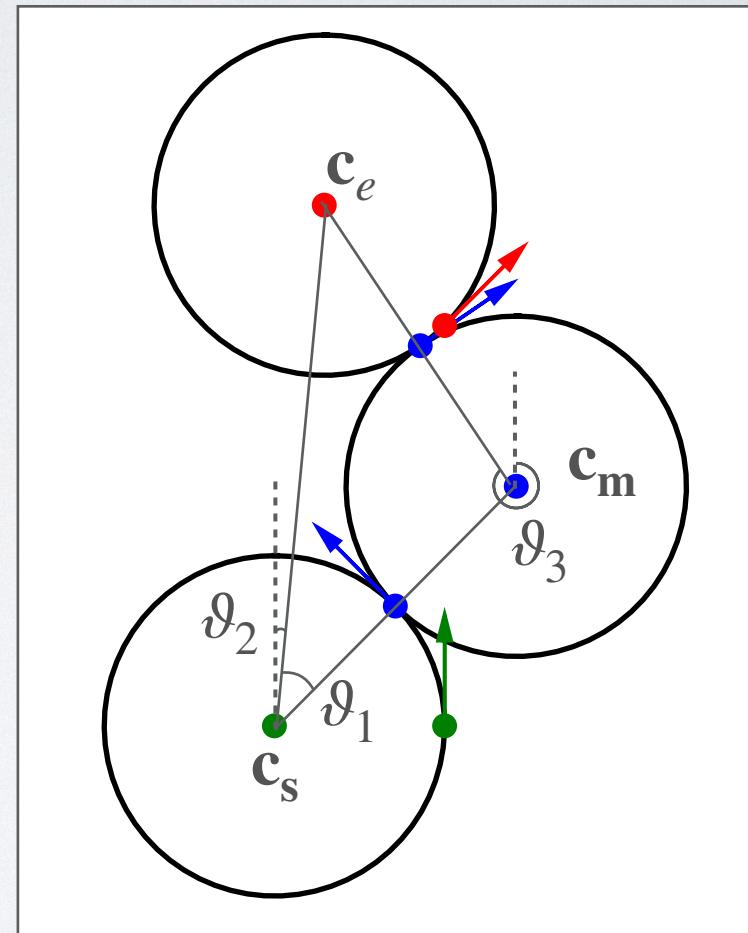
$$\vartheta_1 = \left\langle \pm \cos^{-1} \left(\frac{\ell}{4R} \right) \right\rangle$$

- The \pm in the equation for ϑ_1 indicates that there are two cases for both R-L-R and L-R-L: one where the circle is to the right of $\mathbf{c}_e - \mathbf{c}_s$ (+) and one where it is to the left (-)



C-C-C DUBINS PATHS

- The center of the middle circle is given by $\mathbf{c}_m = \mathbf{c}_{rs} + 2R\mathcal{R}_z(\vartheta_1)\frac{\mathbf{c}_{re} - \mathbf{c}_{rs}}{\ell}$, where $\ell = \|\mathbf{c}_{re} - \mathbf{c}_{rs}\|$ for R-L-R and $\ell = \|\mathbf{c}_{le} - \mathbf{c}_{ls}\|$ for L-R-L
- ϑ_2 and ϑ_1 are defined as the angles of the vectors $\mathbf{c}_m - \mathbf{c}_s$ and $\mathbf{c}_e - \mathbf{c}_m$ respectively



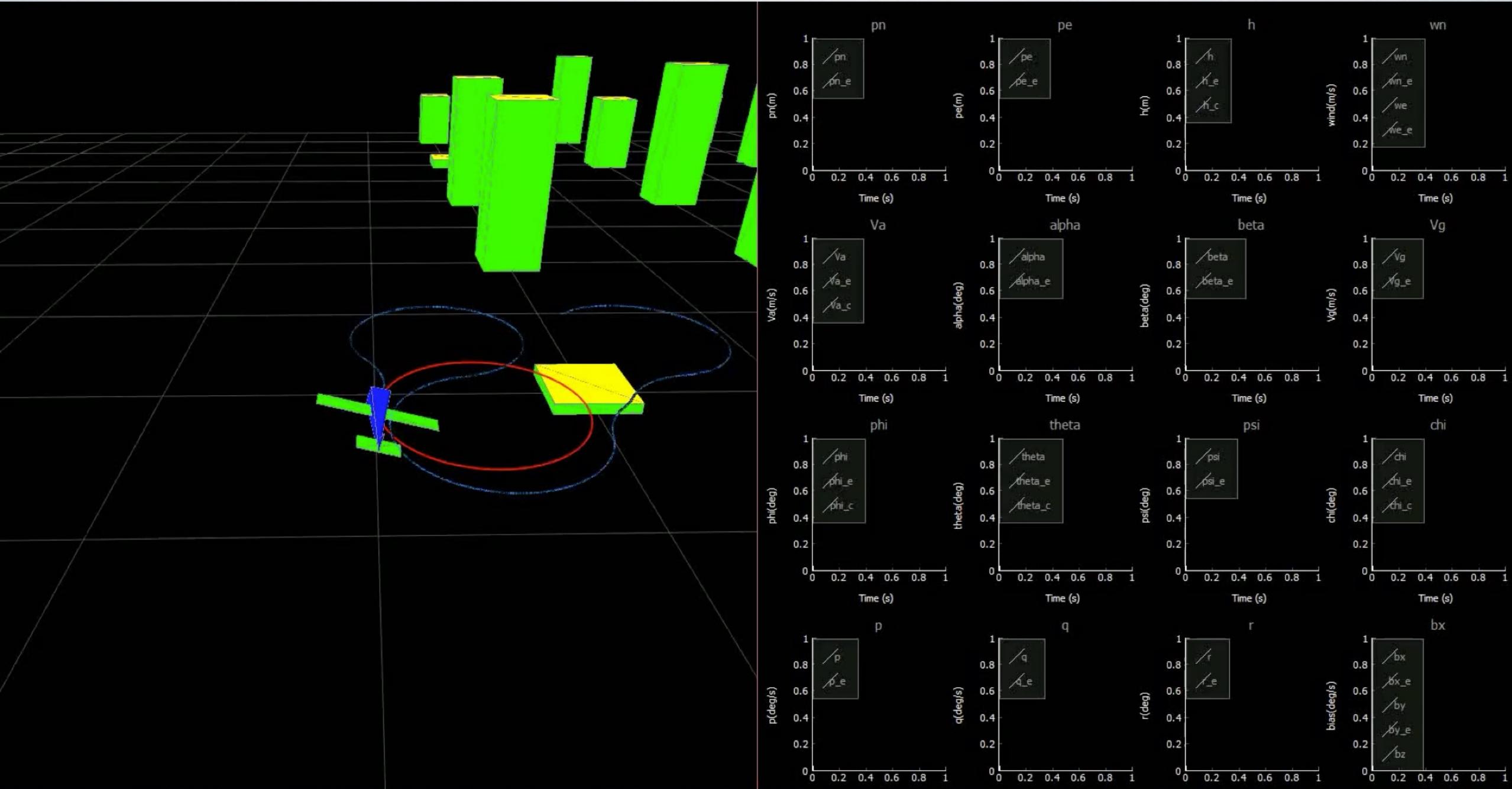
C-C-C DUBINS PATHS

- R-L-R Length:

$$L_5 = R \left\langle 2\pi + \left\langle \frac{\pi}{2} - \vartheta_2 \right\rangle - \left\langle \chi_s - \frac{\pi}{2} \right\rangle \right\rangle \\ + R \left\langle 2\pi - \left\langle \frac{\pi}{2} - \vartheta_3 \right\rangle + \left\langle \frac{3\pi}{2} - \vartheta_2 \right\rangle \right\rangle \\ + R \left\langle 2\pi + \left\langle \chi_e - \frac{\pi}{2} \right\rangle - \left\langle \frac{3\pi}{2} - \vartheta_3 \right\rangle \right\rangle$$

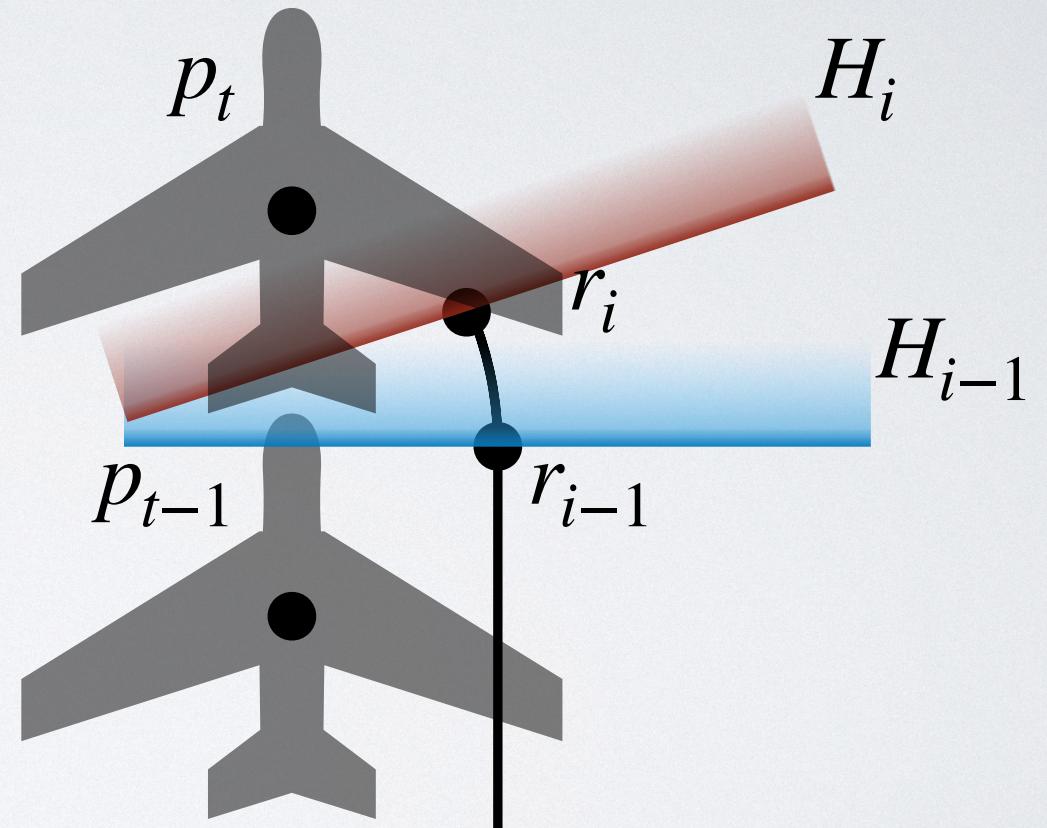
- L-R-L Length:

$$L_6 = R \left\langle 2\pi - \left\langle \frac{\pi}{2} - \vartheta_2 \right\rangle + \left\langle \chi_s - \frac{\pi}{2} \right\rangle \right\rangle \\ + R \left\langle 2\pi + \left\langle \frac{\pi}{2} - \vartheta_3 \right\rangle - \left\langle \frac{3\pi}{2} - \vartheta_2 \right\rangle \right\rangle \\ + R \left\langle 2\pi - \left\langle \chi_e - \frac{\pi}{2} \right\rangle + \left\langle \frac{3\pi}{2} - \vartheta_3 \right\rangle \right\rangle$$



ROBUST PATH MANAGEMENT

- The dual half-plane path manager occasionally flew a full orbit when paths included short arcs
- The negative half plane check should be skipped when points are close
 - $r_{i-1} \in -H_i$
 - $s < R\pi$
 - $\theta < \pi$



The best way to do this is check distance along circles and straight lines. If it is small, then just transition to the next segment.

REFERENCES

- [1] T.Wallace, D.Watkins, and J. Schwartz, “A Map of Every Building in America,” *The New York Times*, Oct 2018. Data available: <https://github.com/microsoft/USBuildingFootprints>.
- [2] J. O'Rourke *et al.*, *Computational Geometry in C*. Cambridge University Press, 2 ed., 1998.
- [3] S. Gillies *et al.*, “Shapely: Manipulation and Analysis of Geometric Objects,” 2007.
- [4] P. Virtanen *et al.*, “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python,” *Nature Methods*, vol. 17, pp. 261–272, 2020.
- [5] A. Hagberg, P. Swart, and D. S Chult, “Exploring Network Structure, Dynamics, and Function using NetworkX,” Tech. Rep. LA-UR-08-05495, Los Alamos National Laboratory, I 2008.
- [6] A. M. Shkel and V. Lumelsky, “Classification of the Dubins set,” *Robotics and Autonomous Systems*, vol. 34, no. 4, pp. 179–202, 2001.
- [7] A. Giese, “A Comprehensive, Step-by-Step Tutorial on Computing Dubins Curves,” 2014.

Dubins Airplane Model

Adapted from: Mark Owen, Randal W. Beard, Timothy W. McLain, "Implementing Dubins Airplane Paths on Fixed-wing UAVs," *Handbook of Unmanned Aerial Vehicles*, ed. Kimon P. Valavanis, George J. Vachtsevanos, Springer Verlag, Section XII, Chapter 68, p. 1677-1702, 2014.

Dubins Airplane model:

$$\dot{r}_n = V \cos \psi \cos \gamma^c$$

$$\dot{r}_e = V \sin \psi \cos \gamma^c$$

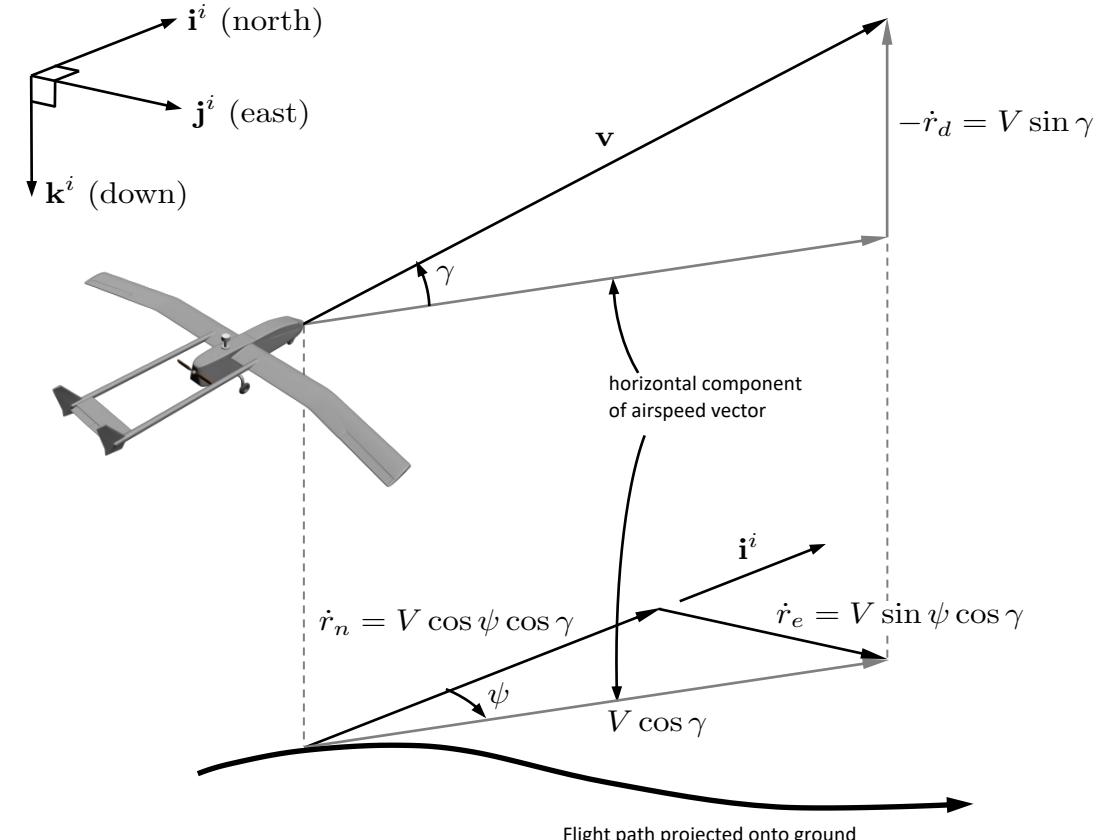
$$\dot{r}_d = -V \sin \gamma^c$$

$$\dot{\psi} = \frac{g}{V} \tan \phi^c$$

Where the commanded flight path angle γ^c and the commanded roll angle ϕ^c are constrained by

$$|\phi^c| \leq \bar{\phi}$$

$$|\gamma^c| \leq \bar{\gamma}.$$



3D Vector Field Path Following

Adapted from: V. M. Goncalves, L. C. A. Pimenta, C. A. Maia, B. C. O. Durtra, G. A. S. Pereira, B. C. O. Dutra, and G. A. S. Pereira, "Vector Fields for Robot Navigation Along Time-Varying Curves in n-Dimensions," IEEE Transactions on Robotics, vol. 26, pp. 647–659, Aug 2010.

The path is specified as the intersection of two 2D manifolds given by

$$\alpha_1(\mathbf{r}) = 0$$

$$\alpha_2(\mathbf{r}) = 0$$

$\mathbf{r} \in \mathbb{R}^3$. Define the composite function

$$W(\mathbf{r}) = \frac{1}{2}\alpha_1^2(\mathbf{r}) + \frac{1}{2}\alpha_2^2(\mathbf{r}),$$

Note that the gradient

$$\frac{\partial W}{\partial \mathbf{r}} = \alpha_1(\mathbf{r}) \frac{\partial \alpha_1}{\partial \mathbf{r}}(\mathbf{r}) + \alpha_2(\mathbf{r}) \frac{\partial \alpha_2}{\partial \mathbf{r}}(\mathbf{r}).$$

points away from the path.

3D Vector Field Path Following

The desired velocity vector can be chosen as

$$\mathbf{u}' = \underbrace{-K_1 \frac{\partial W}{\partial \mathbf{r}}}_{\text{velocity directed toward the path}} + \underbrace{K_2 \frac{\partial \alpha_1}{\partial \mathbf{r}} \times \frac{\partial \alpha_2}{\partial \mathbf{r}}}_{\text{velocity directed along the path}}$$

where $K_1 > 0$ and K_2 are symmetric tuning matrices, and the definiteness of K_2 determines the direction of travel along the path.

Since \mathbf{u}' may not equal V_a , normalize to get

$$\mathbf{u} = V_a \frac{\mathbf{u}'}{\|\mathbf{u}'\|}.$$

3D Vector Field Path Following

Setting the NED components of the velocity of the Dubins airplane model to $\mathbf{u} = (u_1, u_2, u_3)^\top$ gives

$$\begin{aligned} V \cos \psi^d \cos \gamma^c &= u_1 \\ V \sin \psi^d \cos \gamma^c &= u_2 \\ -V \sin \gamma^c &= u_3. \end{aligned}$$

Solving for γ^c , and ψ^d results in

$$\begin{aligned} \gamma^c &= -\text{sat}_{\bar{\gamma}} \left[\sin^{-1} \left(\frac{u_3}{V} \right) \right] \\ \psi^d &= \text{atan2}(u_2, u_1). \end{aligned}$$

Assuming the inner-loop lateral-directional dynamics are accurately modeled by the coordinated-turn equation, the commanded roll angle is

$$\phi^c = \text{sat}_{\bar{\phi}} \left[k_\phi (\psi^d - \psi) \right],$$

where k_ϕ is a positive constant.

3D Vector Field – Straight Line path

The straight line path is given by

$$\mathcal{P}_{\text{line}}(\mathbf{c}_\ell, \psi_\ell, \gamma_\ell) = \left\{ \mathbf{r} \in \mathbb{R}^3 : \mathbf{r} = \mathbf{c}_\ell + \sigma \mathbf{q}_\ell, \sigma \in \mathbb{R} \right\},$$

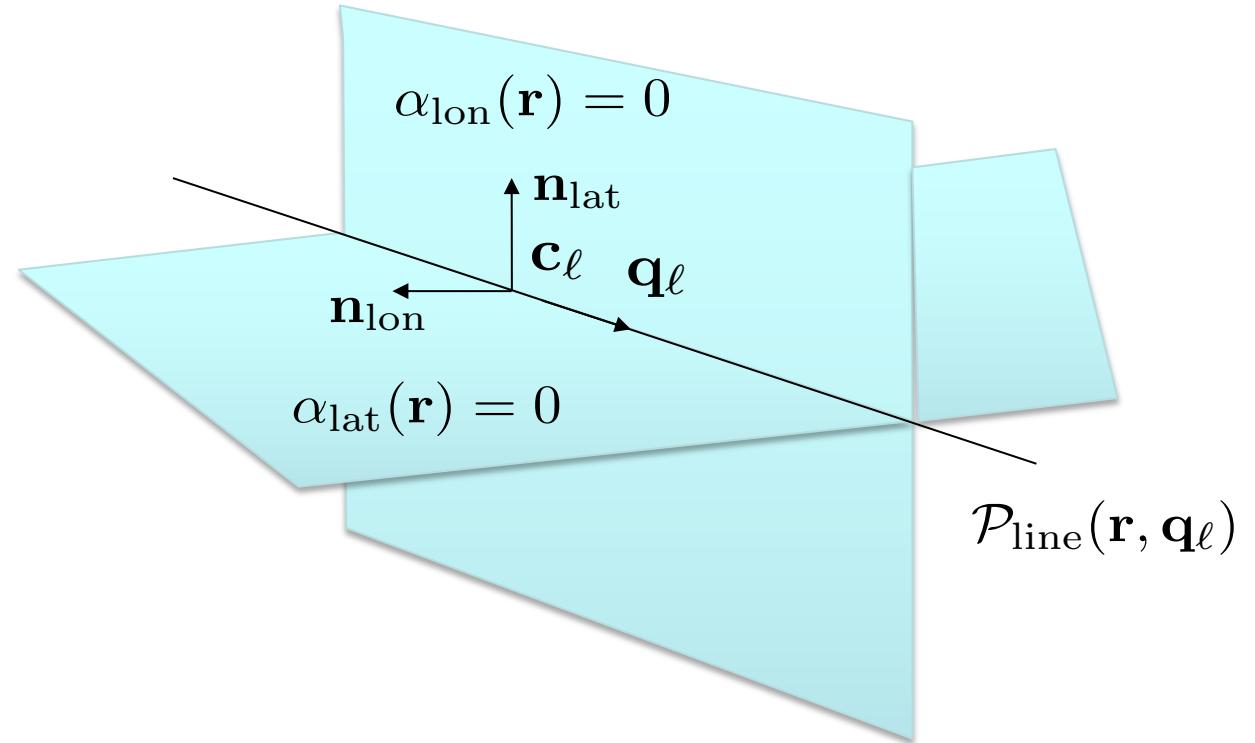
where

$$\mathbf{q}_\ell = \begin{pmatrix} q_n \\ q_e \\ q_d \end{pmatrix} = \begin{pmatrix} \cos \psi_\ell \cos \gamma_\ell \\ \sin \psi_\ell \cos \gamma_\ell \\ -\sin \gamma_\ell \end{pmatrix}.$$

Define

$$\mathbf{n}_{\text{lon}} = \begin{pmatrix} -\sin \psi_\ell \\ \cos \psi_\ell \\ 0 \end{pmatrix}$$

$$\mathbf{n}_{\text{lat}} = \mathbf{n}_{\text{lon}} \times \mathbf{q}_\ell = \begin{pmatrix} -\cos \psi_\ell \sin \gamma_\ell \\ -\sin \psi_\ell \sin \gamma_\ell \\ -\cos \gamma_\ell \end{pmatrix},$$



to get

$$\alpha_{\text{lon}}(\mathbf{r}) = \mathbf{n}_{\text{lon}}^\top (\mathbf{r} - \mathbf{c}_\ell) = 0$$

$$\alpha_{\text{lat}}(\mathbf{r}) = \mathbf{n}_{\text{lat}}^\top (\mathbf{r} - \mathbf{c}_\ell) = 0.$$

3D Vector Field – Helical Path

A helical path is then defined as

$$\mathcal{P}_{\text{helix}}(\mathbf{c}_h, \psi_h, \lambda_h, R_h, \gamma_h) = \{\mathbf{r} \in \mathbb{R}^3 : \alpha_{\text{cyl}}(\mathbf{r}) = 0 \text{ and } \alpha_{\text{pl}}(\mathbf{r}) = 0\}.$$

where

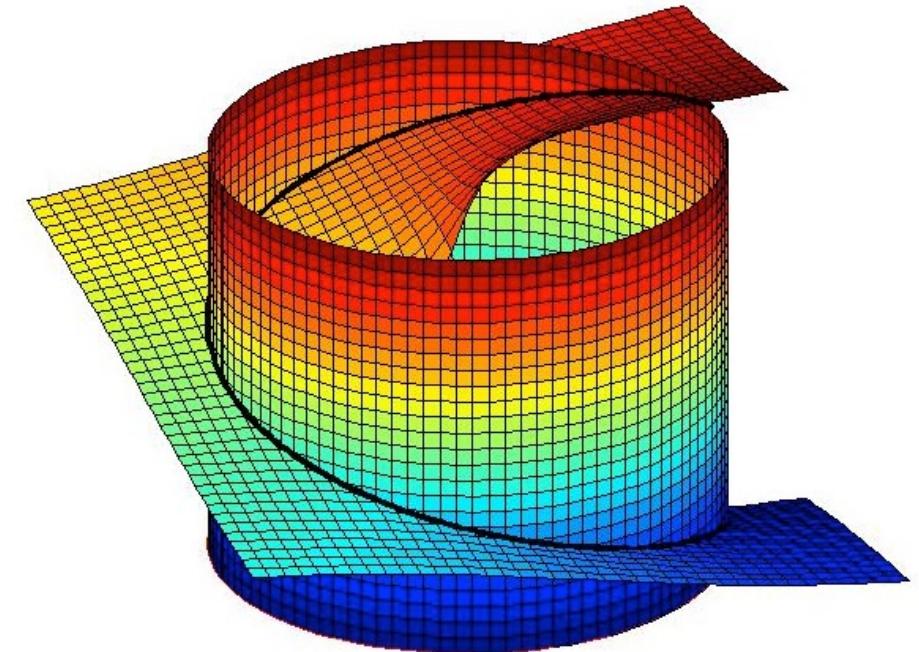
$$\alpha_{\text{cyl}}(\mathbf{r}) = \left(\frac{r_n - c_n}{R_h} \right)^2 + \left(\frac{r_e - c_e}{R_h} \right)^2 - 1$$

$$\alpha_{\text{pl}}(\mathbf{r}) = \left(\frac{r_d - c_d}{R_h} \right) + \frac{\tan \gamma_h}{\lambda_h} \left(\tan^{-1} \left(\frac{r_e - c_e}{r_n - c_n} \right) - \psi_h \right)$$

where the initial position along the helix is

$$\mathbf{r}(0) = \mathbf{c}_h + \begin{pmatrix} R_h \cos \psi_h \\ R_h \sin \psi_h \\ 0 \end{pmatrix}.$$

\mathbf{c}_h is the center of the helix, R_h is the radius, γ_h is the climb angle.



Dubins Airplane Paths

Given the start configuration $\mathbf{z}_s = (z_{ns}, z_{es}, z_{ds}, \psi_s)^\top$ and the end configuration $\mathbf{z}_e = (z_{ne}, z_{ee}, z_{de}, \psi_e)^\top$ and the turn radius R , let $L_{\text{car}}(R, \mathbf{z}_s, \mathbf{z}_e)$ be the length of the Dubins car path.

Recall that $\bar{\gamma}$ is the limit of the flight path angle. There are three possible cases for the commanded altitude gain:

Low Altitude:

$$|z_{de} - z_{ds}| \leq L_{\text{car}}(R_{\min}) \tan \bar{\gamma},$$

i.e., the altitude gain can be achieved by following the Dubins car path with a flight path angle $|\gamma^c| \leq \bar{\gamma}$.

High Altitude:

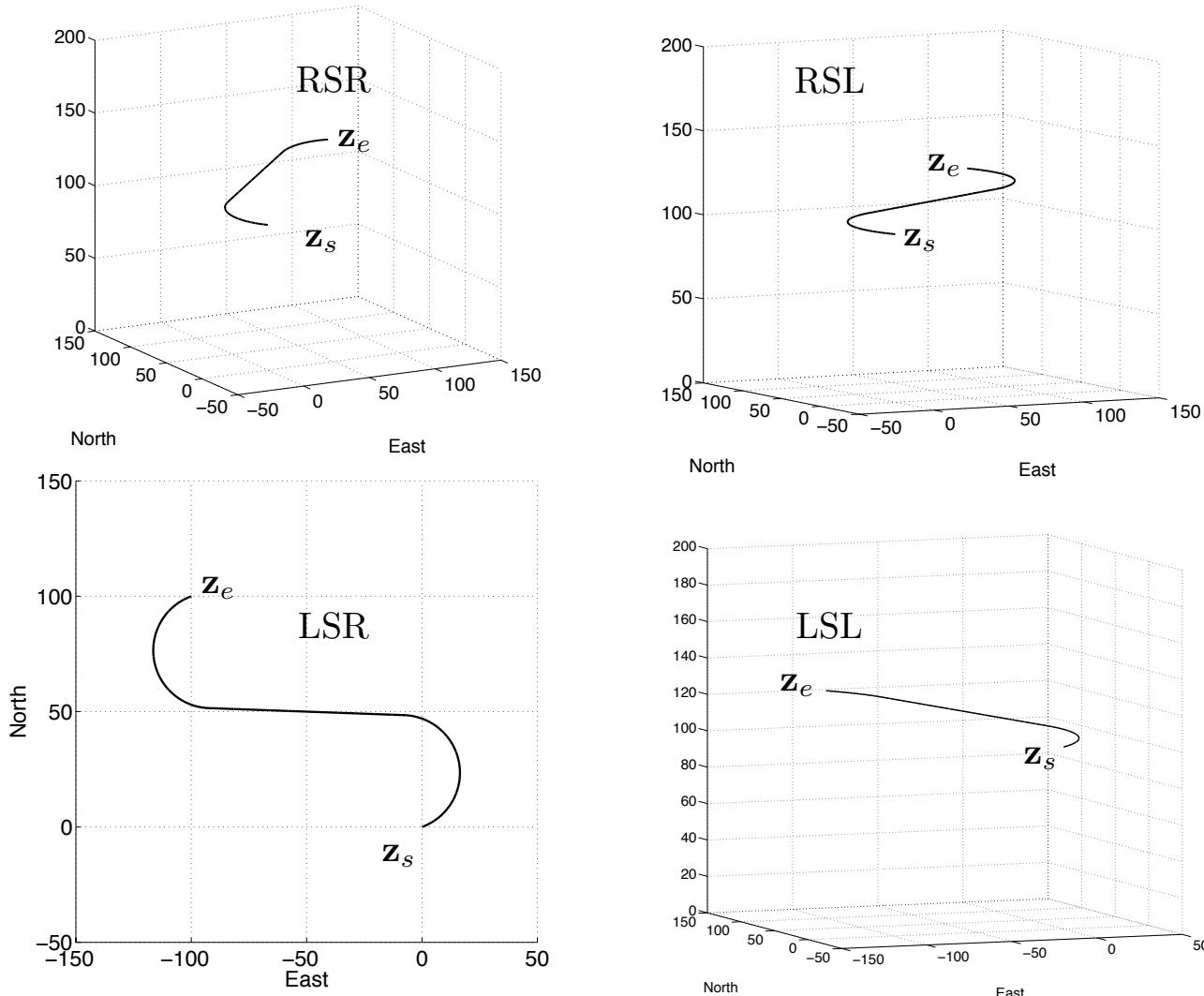
$$|z_{de} - z_{ds}| > [L_{\text{car}}(R_{\min}) + 2\pi R_{\min}] \tan \bar{\gamma}.$$

i.e., the altitude gain is larger than following the Dubins car path plus one orbit, at flight path angle $\bar{\gamma}$.

Medium Altitude:

$$L_{\text{car}}(R_{\min}) \tan \bar{\gamma} < |z_{de} - z_{ds}| \leq [L_{\text{car}}(R_{\min}) + 2\pi R_{\min}] \tan \bar{\gamma}.$$

Low Altitude Dubins Airplane Paths

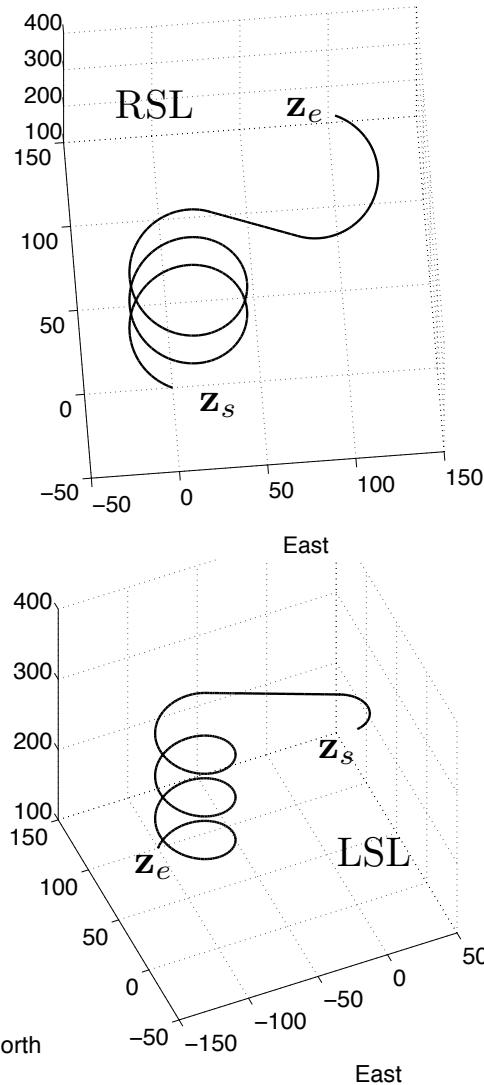
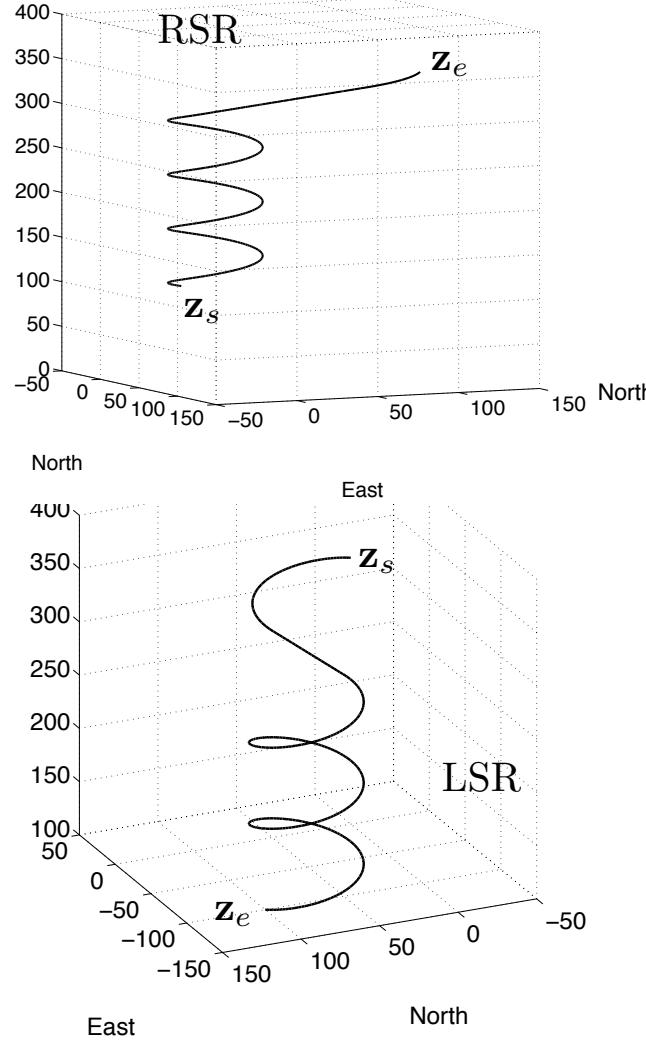


$$\gamma^* = \tan^{-1} \left(\frac{|z_{de} - z_{ds}|}{L_{\text{car}}(R_{\min})} \right)$$

$$R^* = R_{\min}$$

$$L_{\text{air}}(R_{\min}, \gamma^*) = \frac{L_{\text{car}}(R_{\min})}{\cos \gamma^*}.$$

High Altitude Dubins Airplane Paths



Find smallest integer k such that

$$(L_{\text{car}}(R_{\min}) + 2\pi k R_{\min}) \tan \bar{\gamma} \leq |z_{de} - z_{ds}| < (L_{\text{car}}(R_{\min}) + 2\pi(k+1)R_{\min}) \tan \bar{\gamma}.$$

Increase the radius R^* so that

$$(L_{\text{car}}(R^*) + 2\pi k R^*) \tan \bar{\gamma} = |z_{de} - z_{ds}|.$$

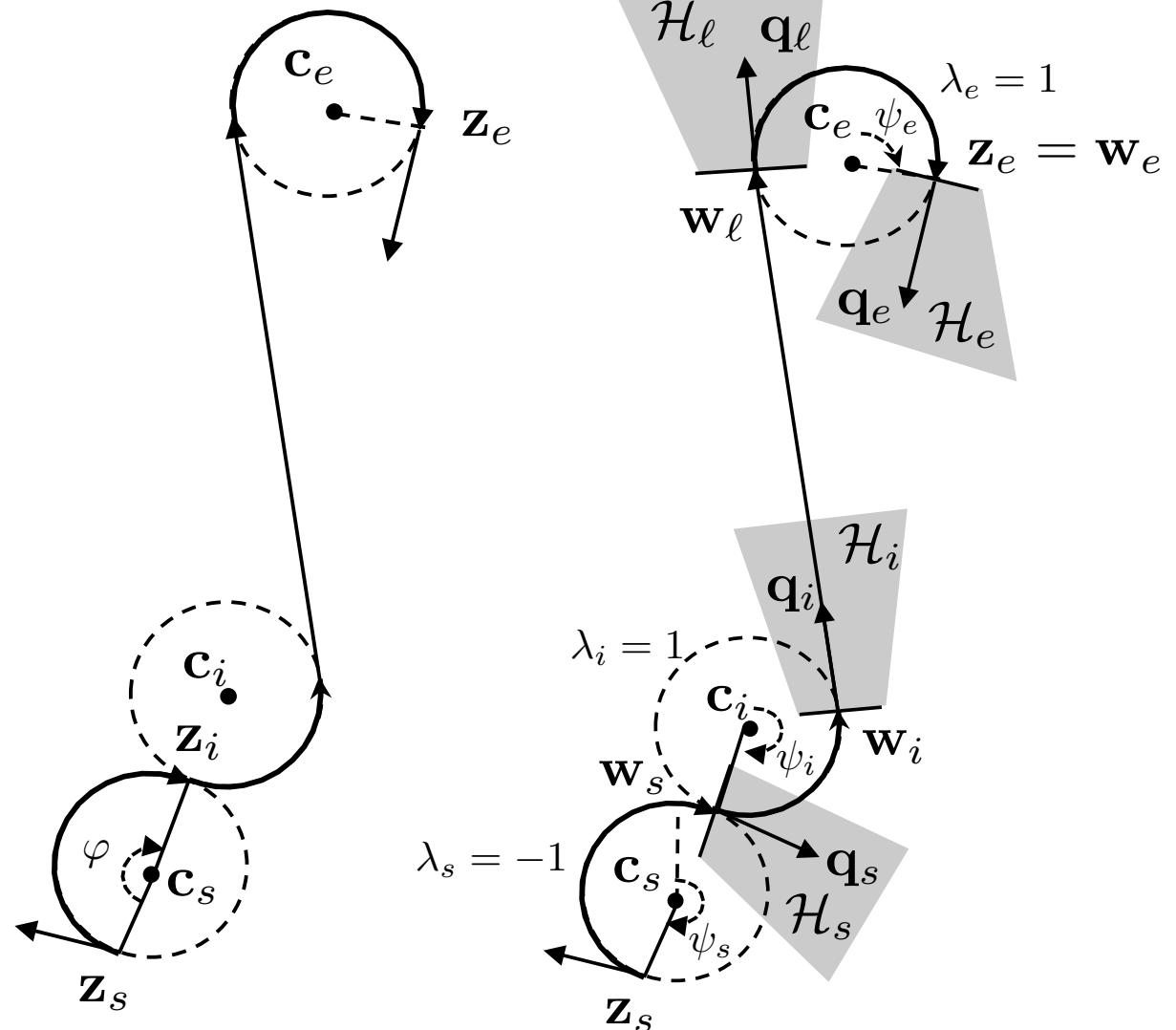
The resulting path is

$$L_{\text{air}}(R^*, \bar{\gamma}) = \frac{L_{\text{car}}(R^*)}{\cos \bar{\gamma}}.$$

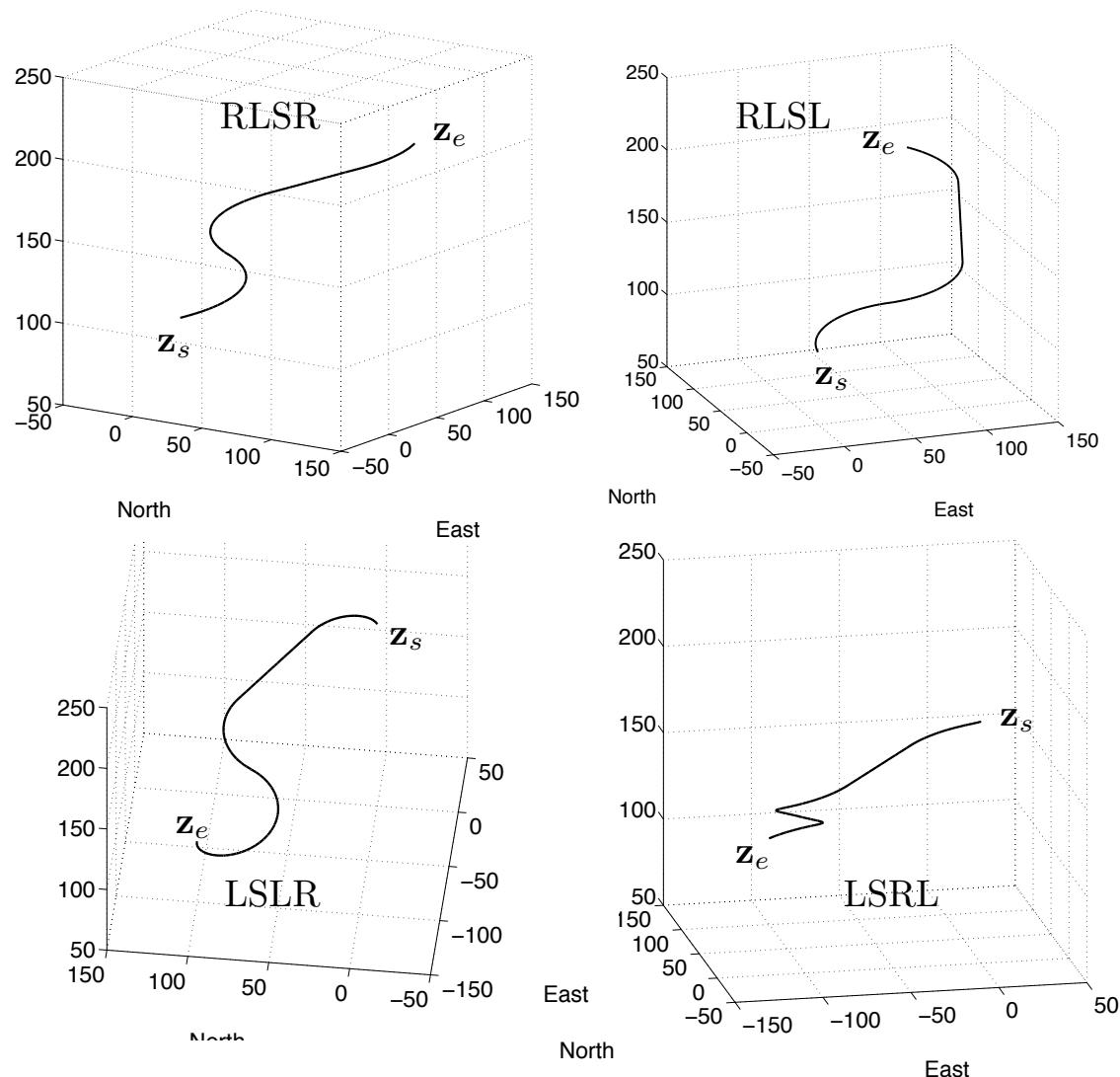
Medium Altitude Dubins Airplane Paths

Key idea: Add an intermediate helix along the path.

Could add intermediate helix at start, end, or in the middle of path.



Medium Altitude Dubins Airplane Paths

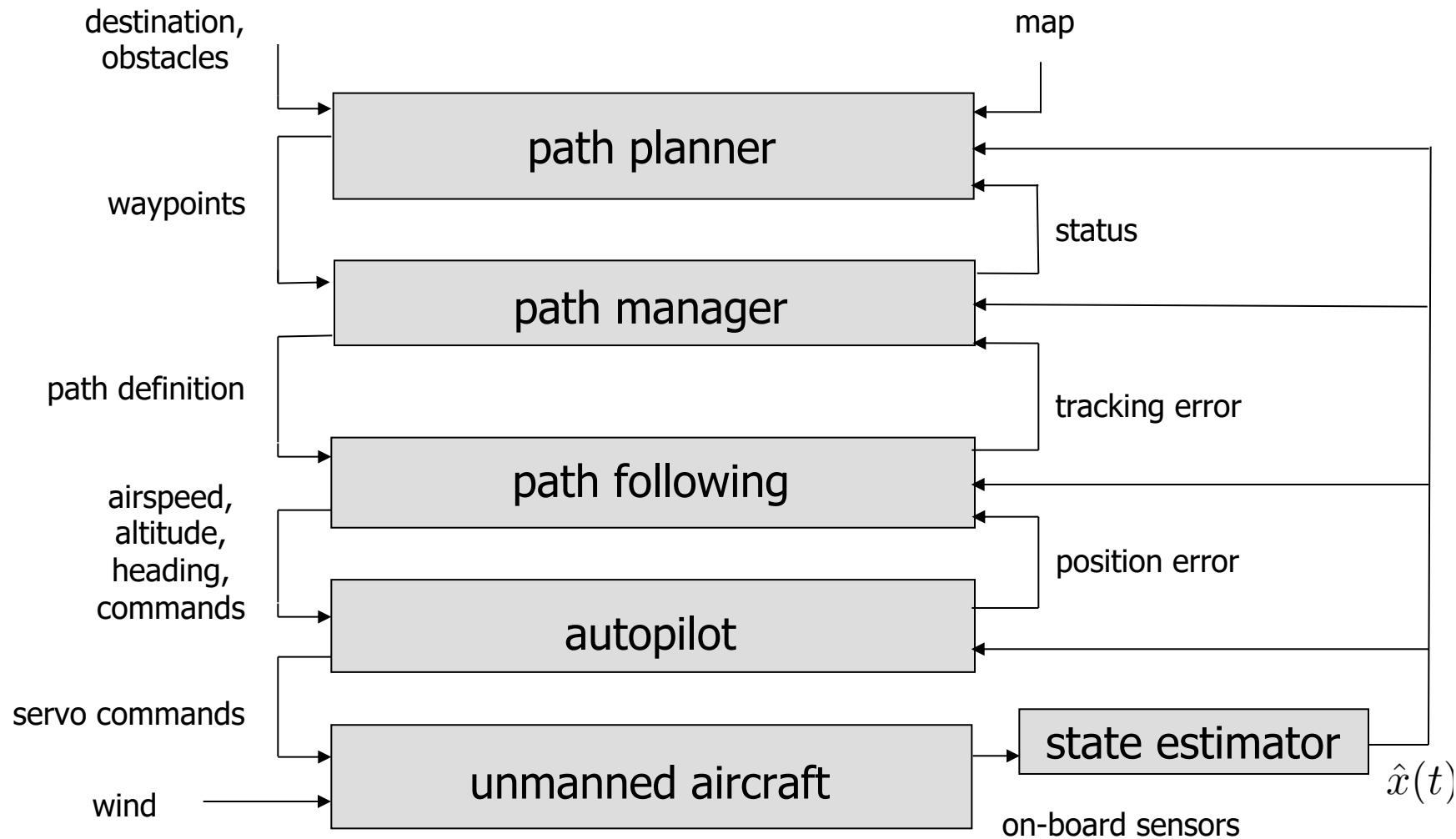




Chapter 12

Path Planning

Control Architecture

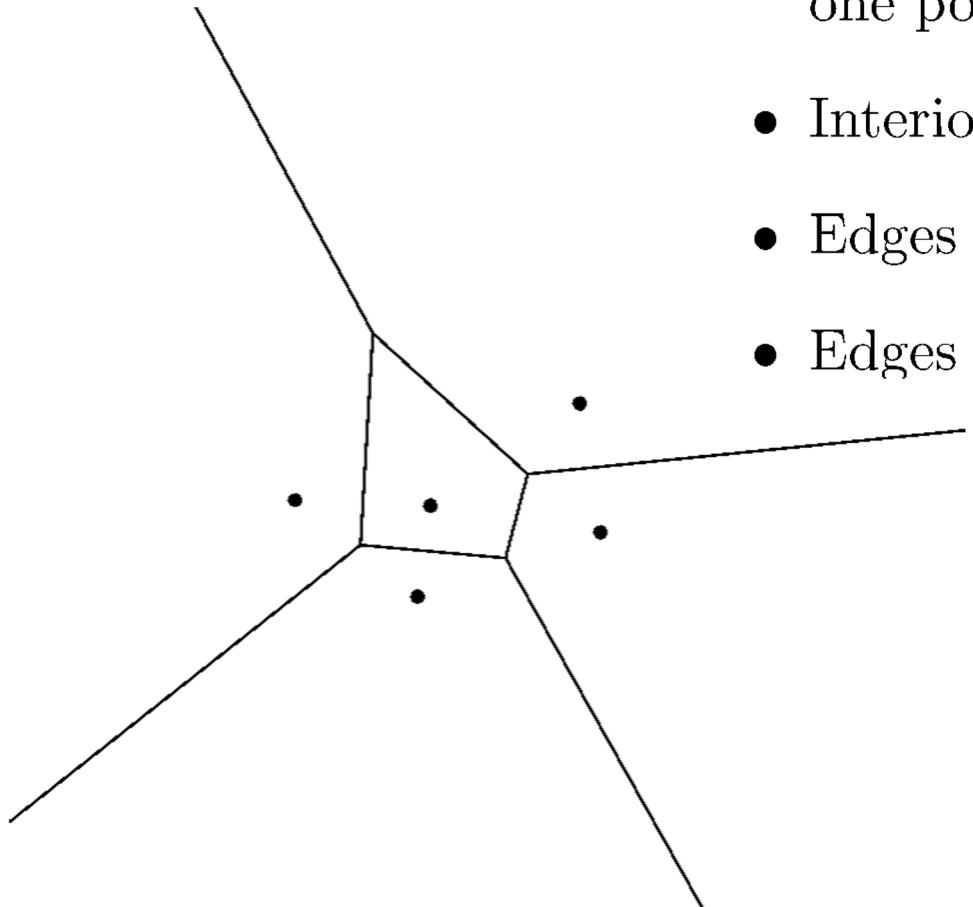


Path Planning Approaches

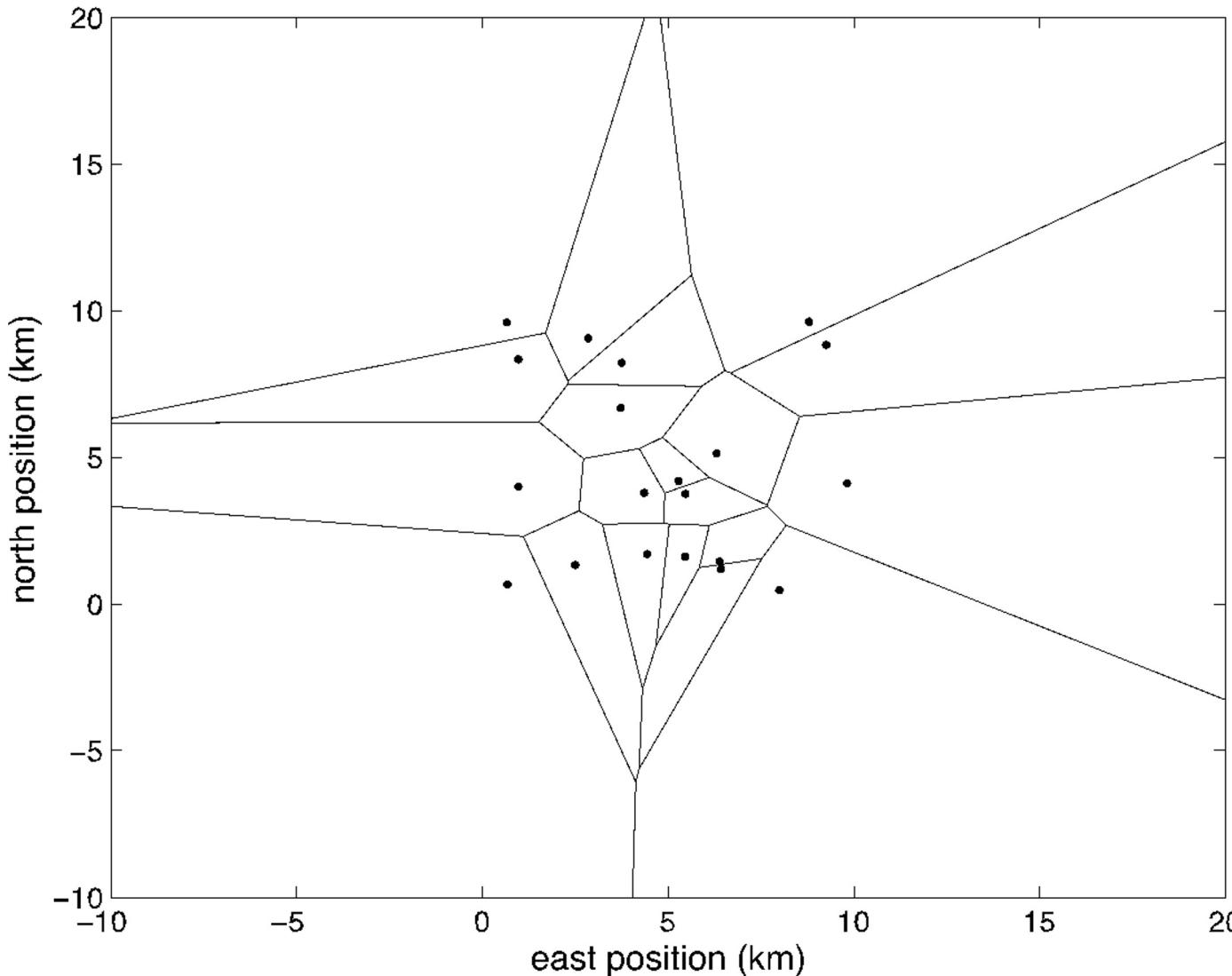
- Deliberative
 - Based on global world knowledge
 - Requires a good map of terrain, obstacles, etc.
 - Can be too computationally intense for dynamic environments
 - Usually executed before the mission
- Reactive
 - Based on what sensors detect on immediate horizon
 - Can respond to dynamic environments
 - Not usually used for entire mission

Voronoi Graphs

- For finite number Q point obstacles, Voronoi graph divides search plane into Q convex cells, each containing one point obstacle
- Interior of cell is closer to its point than any other point in \mathcal{Q}
- Edges of graph are perpendicular bisectors between points in \mathcal{Q}
- Edges of path maximize distance from point obstacles

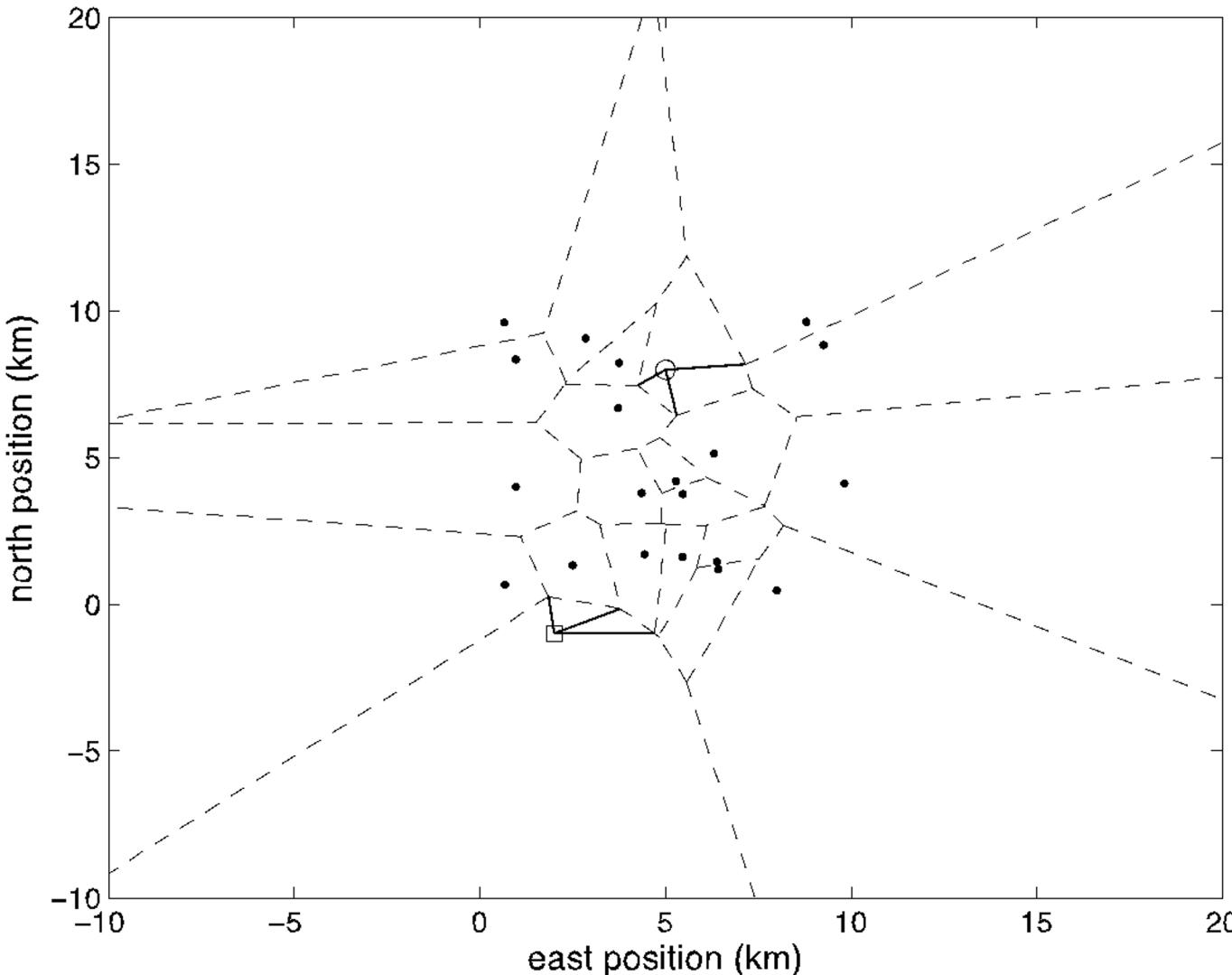


Voronoi Graph Example



- Graph generated using `voronoi` command in Matlab
- 20 point obstacles
- Start and end points of path not shown

Voronoi Graph Example



- Add *start* and *end* points to graph
- Find 3 closest graph nodes to *start* and *end* points
- Add graph edges to *start* and *end* points
- Search graph to find “best” path
- Must define “best”
 - Shortest?
 - Furthest from obstacles?

Path Cost Calculation

Nodes of graph edge: \mathbf{v}_1 and \mathbf{v}_2

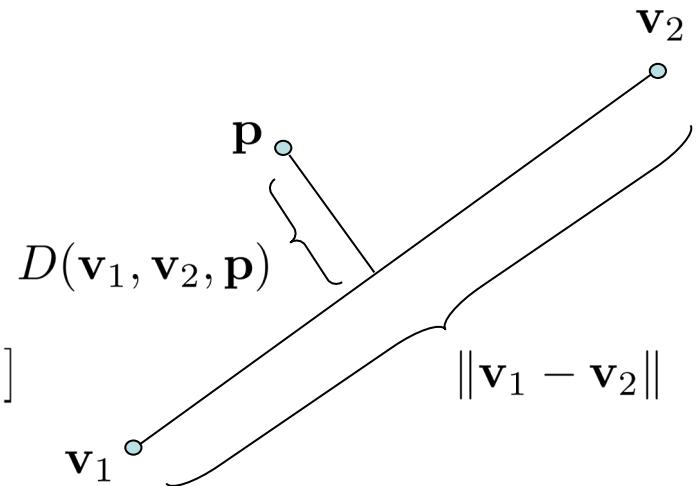
Point obstacle: \mathbf{p}

Length of edge: $\|\mathbf{v}_1 - \mathbf{v}_2\|$

Any point on line segment: $\mathbf{w}(\sigma) = (1 - \sigma)\mathbf{v}_1 + \sigma\mathbf{v}_2$ where $\sigma \in [0, 1]$

Minimum distance between \mathbf{p} and graph edge

$$\begin{aligned} D(\mathbf{v}_1, \mathbf{v}_2, \mathbf{p}) &\stackrel{\triangle}{=} \min_{\sigma \in [0,1]} \|\mathbf{p} - \mathbf{w}(\sigma)\| \\ &= \min_{\sigma \in [0,1]} \sqrt{(\mathbf{p} - \mathbf{w}(\sigma))^{\top} (\mathbf{p} - \mathbf{w}(\sigma))} \\ &= \min_{\sigma \in [0,1]} \sqrt{\|\mathbf{p} - \mathbf{v}_1\|^2 + 2\sigma(\mathbf{p} - \mathbf{v}_1)^{\top}(\mathbf{v}_1 - \mathbf{v}_2) + \sigma^2 \|\mathbf{v}_1 - \mathbf{v}_2\|^2} \end{aligned}$$



Path Cost Calculation

Value for σ that minimizes D is

$$\sigma^* = \frac{(\mathbf{v}_1 - \mathbf{p})^\top (\mathbf{v}_1 - \mathbf{v}_2)}{\|\mathbf{v}_1 - \mathbf{v}_2\|^2}$$

Location along edge for which D is minimum:

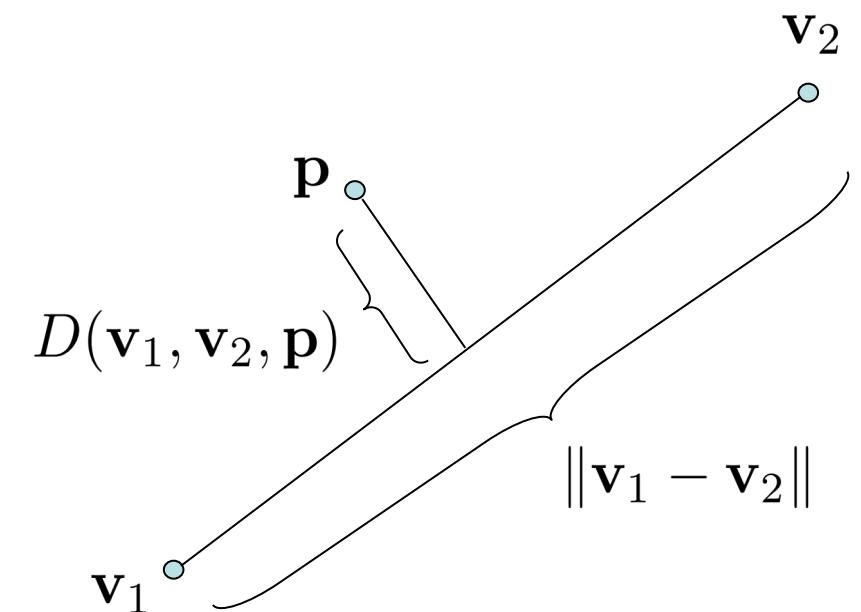
$$\mathbf{w}(\sigma^*) = \sqrt{\|\mathbf{p} - \mathbf{v}_1\|^2 - \frac{((\mathbf{v}_1 - \mathbf{p})^\top (\mathbf{v}_1 - \mathbf{v}_2))^2}{\|\mathbf{v}_1 - \mathbf{v}_2\|^2}}$$

Define distance to edge for $\sigma^* < 0$, $\sigma^* > 1$:

$$D'(\mathbf{v}_1, \mathbf{v}_2, \mathbf{p}) \triangleq \begin{cases} \mathbf{w}(\sigma^*) & \text{if } \sigma^* \in [0, 1] \\ \|\mathbf{p} - \mathbf{v}_1\| & \text{if } \sigma^* < 0 \\ \|\mathbf{p} - \mathbf{v}_2\| & \text{if } \sigma^* > 1 \end{cases}$$

Distance between point set \mathcal{Q} and line segment $\overline{\mathbf{v}_1 \mathbf{v}_2}$:

$$D(\mathbf{v}_1, \mathbf{v}_2, \mathcal{Q}) = \min_{\mathbf{p} \in \mathcal{Q}} D'(\mathbf{v}_1, \mathbf{v}_2, \mathbf{p})$$



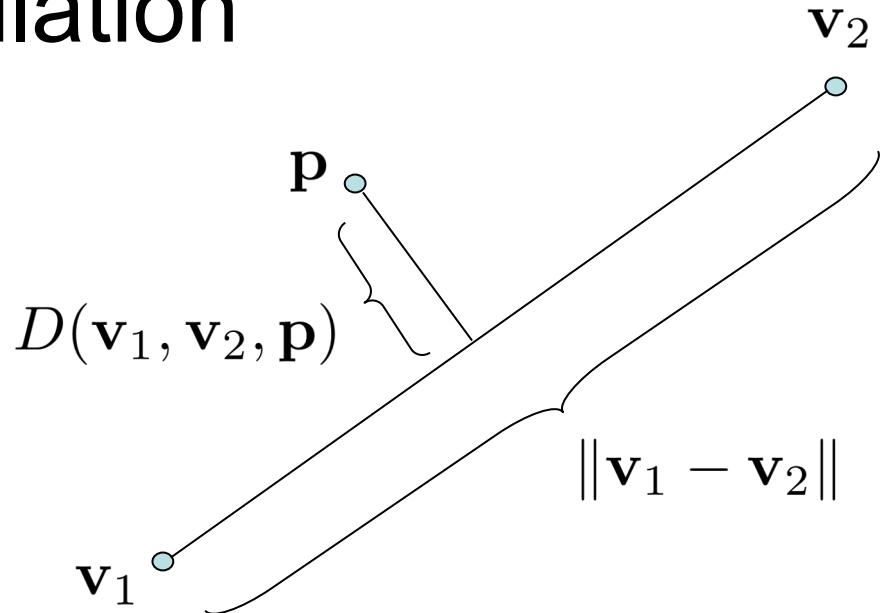
Path Cost Calculation

Cost for traveling along edge $(\mathbf{v}_1, \mathbf{v}_2)$ is assigned as

$$J(\mathbf{v}_1, \mathbf{v}_2) = \underbrace{k_1 \|\mathbf{v}_1 - \mathbf{v}_2\|}_{\text{length of edge}} + \underbrace{\frac{k_2}{D(\mathbf{v}_1, \mathbf{v}_2, \mathcal{Q})}}_{\text{reciprocal of distance to closest point in } \mathcal{Q}}$$

k_1 and k_2 are positive weights

Choice of k_1 and k_2 allow tradeoff between path length and proximity to threats.



Voronoi Path Planning Algorithm

Algorithm 9 Plan Voronoi Path: $\mathcal{W} = \text{planVoronoi}(\mathcal{Q}, \mathbf{p}_s, \mathbf{p}_e)$

Input: Obstacle points \mathcal{Q} , start position \mathbf{p}_s , end position \mathbf{p}_e

Require: $|\mathcal{Q}| \geq 10$ Randomly add points if necessary.

- 1: $(V, E) = \text{constructVoronoiGraph}(\mathcal{Q})$
 - 2: $V^+ = V \cup \{\mathbf{p}_s\} \cup \{\mathbf{p}_e\}$
 - 3: Find $\{\mathbf{v}_{1s}, \mathbf{v}_{2s}, \mathbf{v}_{3s}\}$, the three closest points in V to \mathbf{p}_s , and
 $\{\mathbf{v}_{1e}, \mathbf{v}_{2e}, \mathbf{v}_{3e}\}$, the three closest points in V to \mathbf{p}_e
 - 4: $E^+ = E \cup_{i=1,2,3} (\mathbf{v}_{is}, \mathbf{p}_s) \cup_{i=1,2,3} (\mathbf{v}_{ie}, \mathbf{p}_e)$
 - 5: **for** Each element $(\mathbf{v}_a, \mathbf{v}_b) \in E$ **do**
 - 6: Assign edge cost $J_{ab} = J(\mathbf{v}_a, \mathbf{v}_b)$ according to equation (12.1).
 - 7: **end for**
 - 8: $\mathcal{W} = \text{DijkstraSearch}(V^+, E^+, J)$
 - 9: **return** \mathcal{W}
-

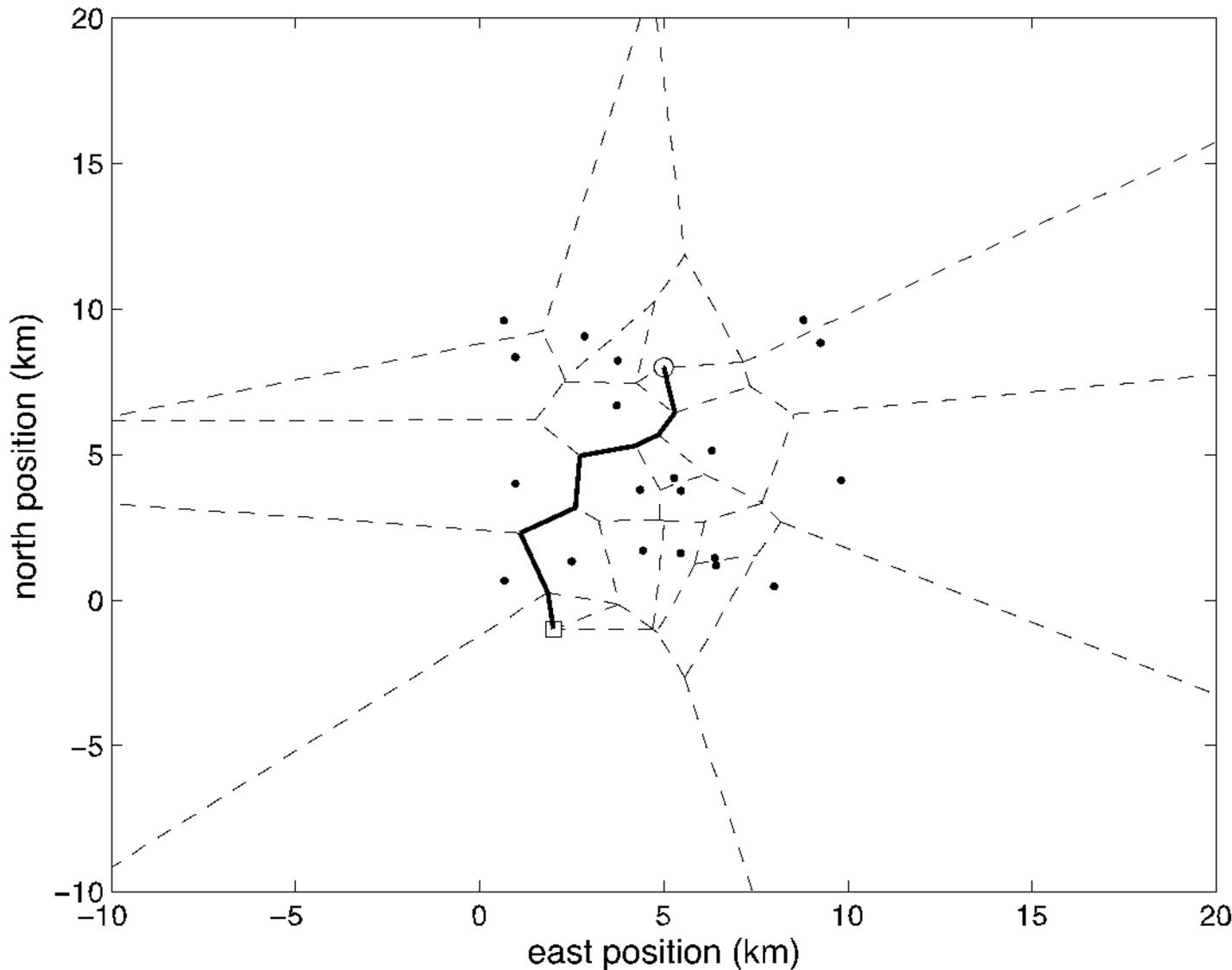
Dijkstra's algorithm is used to search the graph

Voronoi graph and Dijkstra's algorithm code are commonly available
Matlab:

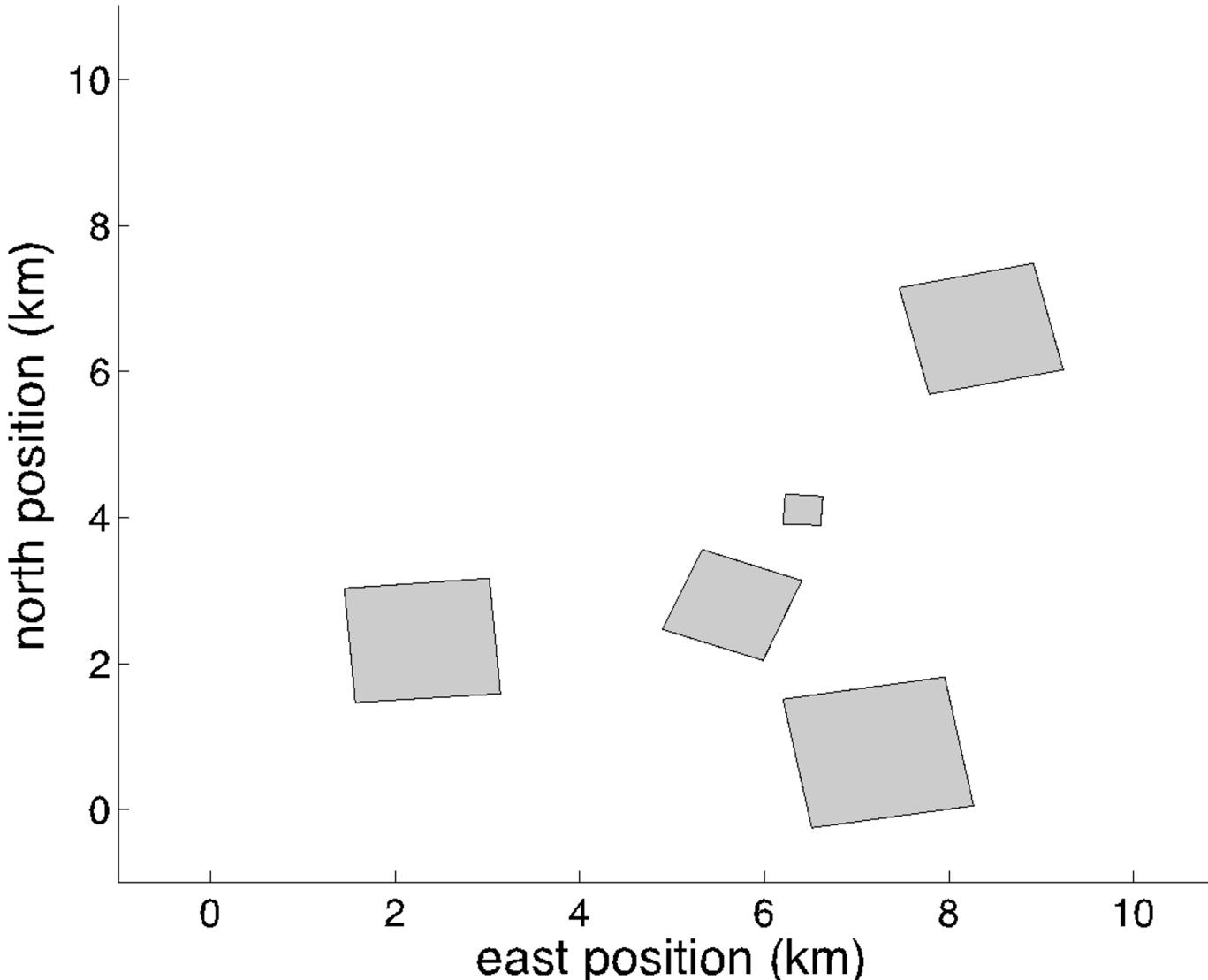
Voronoi -> voronoi

Dijkstra -> shortestpath

Voronoi Path Planning Result



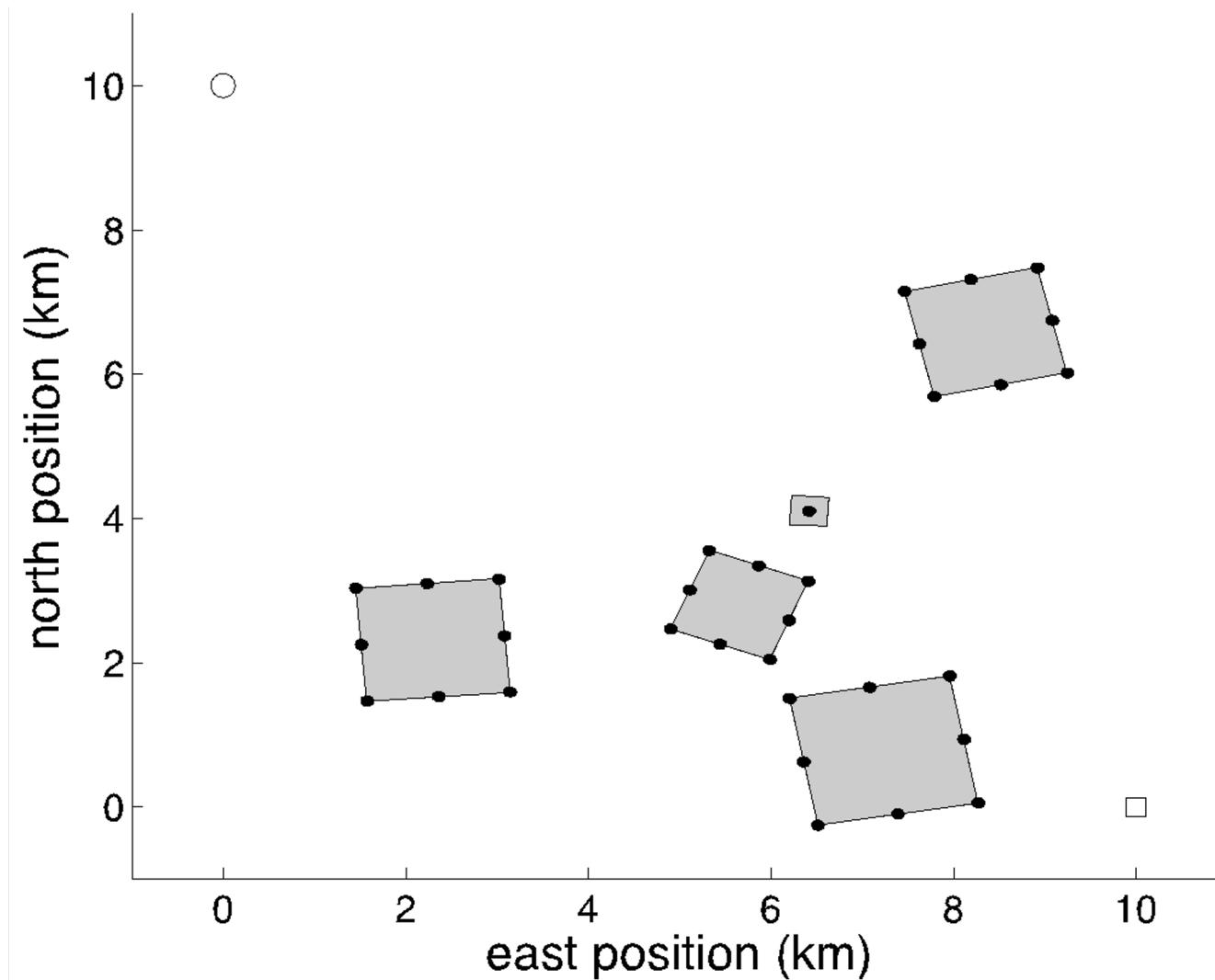
Voronoi Path Planning – Non-point Obstacles



- How do we handle solid obstacles?
- Point obstacles at center of spatial obstacles won't provide safe path options around obstacles

Non-point Obstacles – Step 1

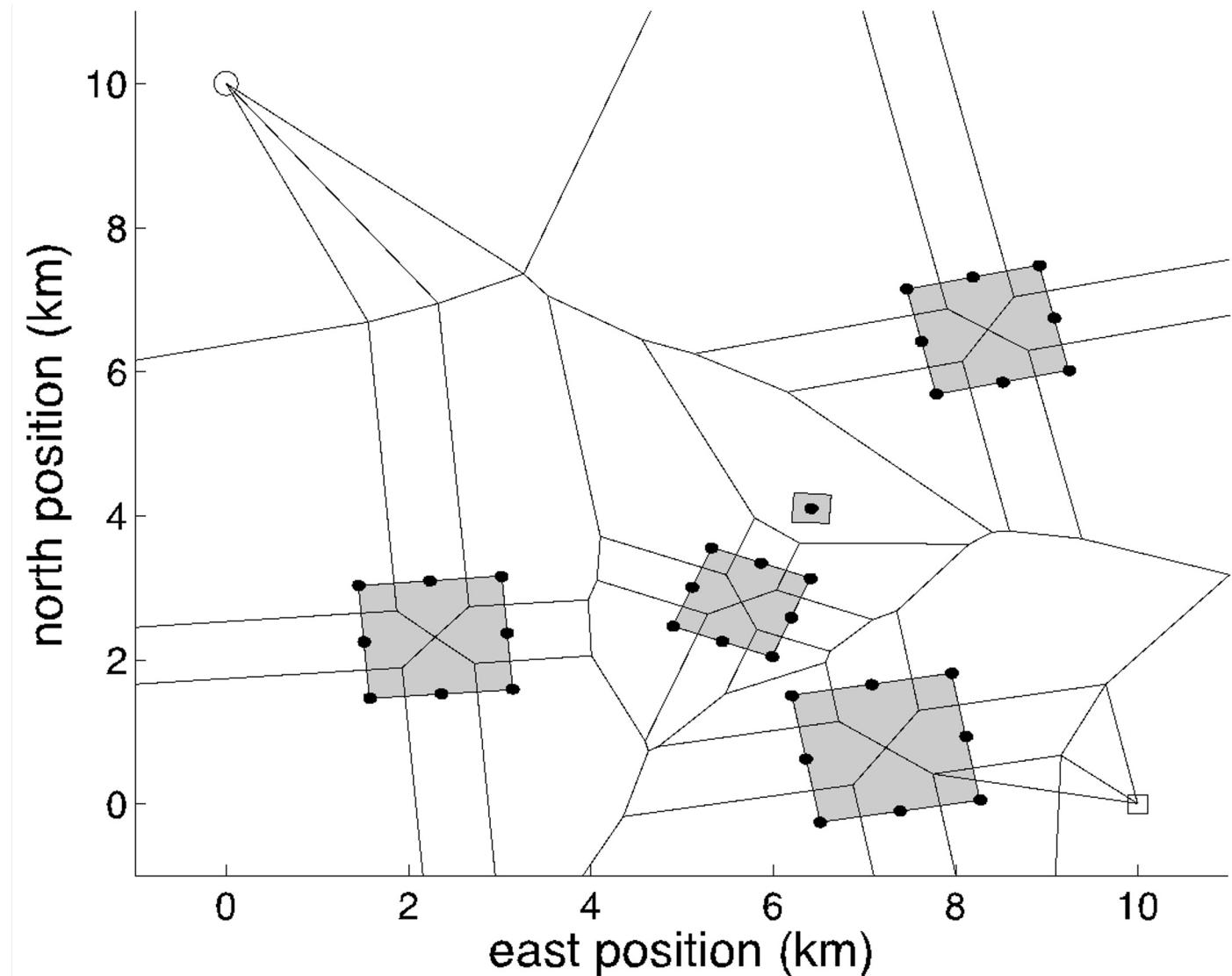
Insert points around perimeter of obstacles



Non-point Obstacles – Step 2

Construct Voronoi graph

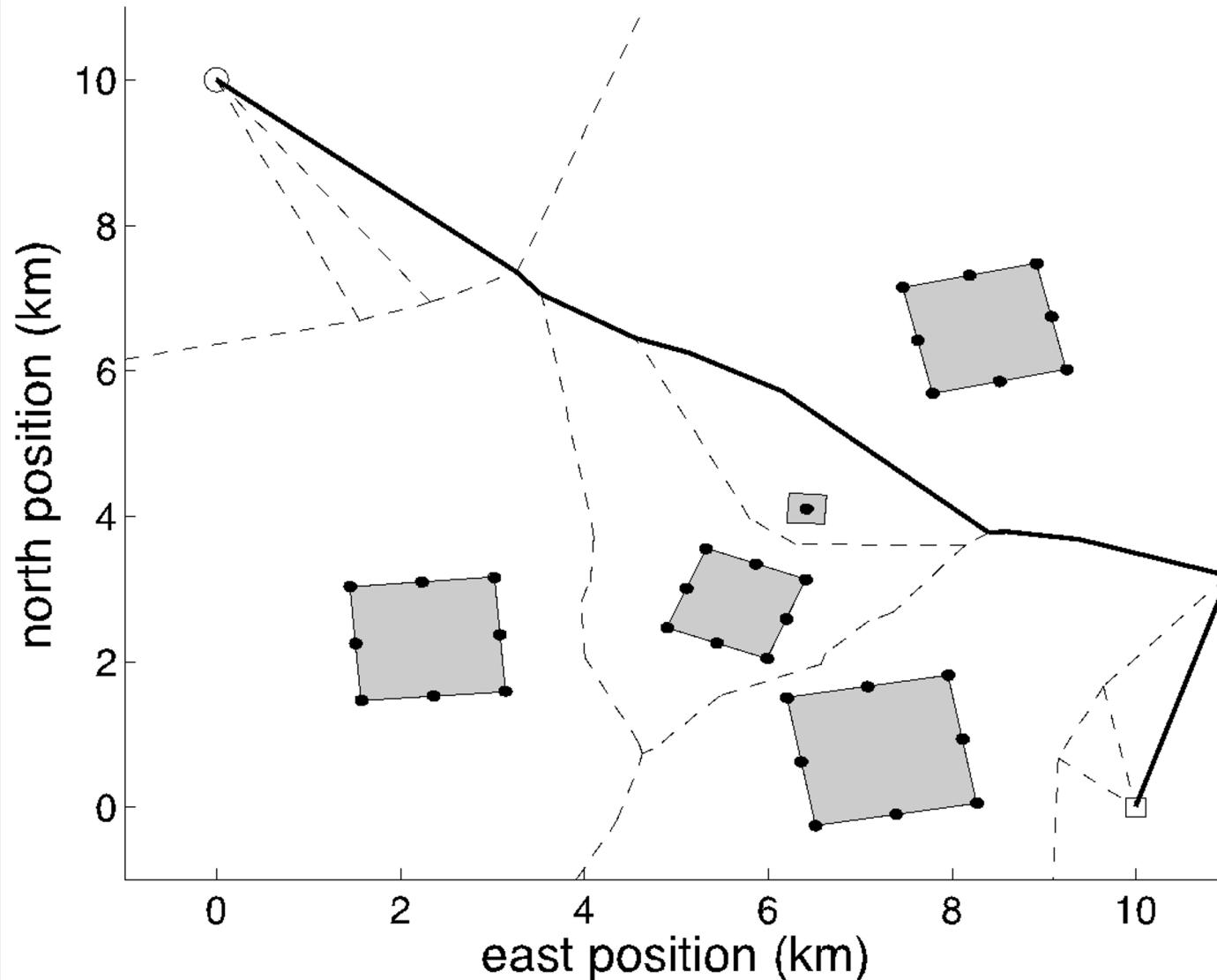
Note infeasible path
edges inside obstacles



Non-point Obstacles – Step 3

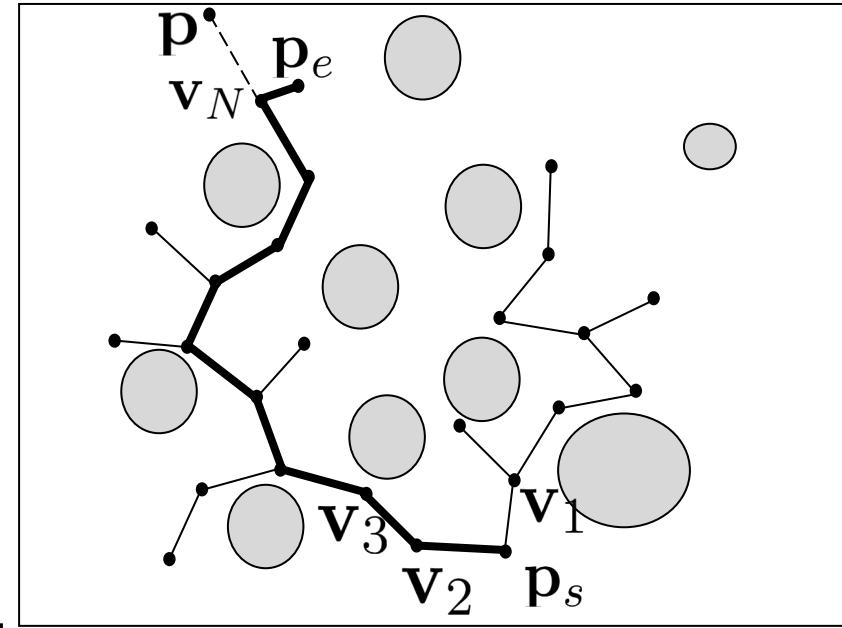
Remove infeasible path edges from graph

Search graph for best path



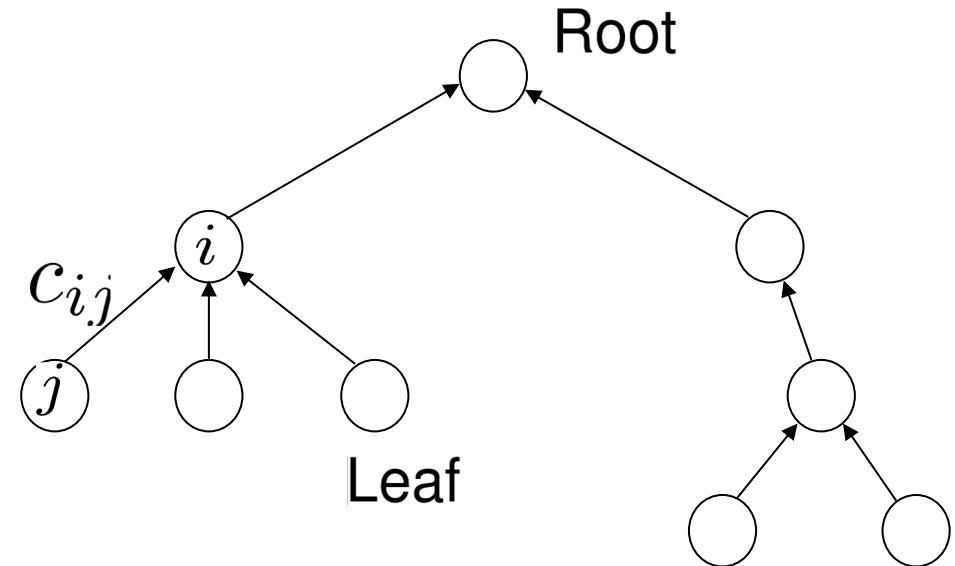
Rapidly Exploring Random Trees (RRT)

- Exploration algorithm that randomly, but uniformly explores search space
- Can accommodate vehicles with complicated, nonlinear dynamics
- Obstacles represented in a terrain map
- Map can be queried to detect possible collisions
- RRTs can be used to generate a single feasible path or a tree with many feasible paths that can be searched to determine the best one
- If algorithm runs long enough, the optimal path through the terrain will be found



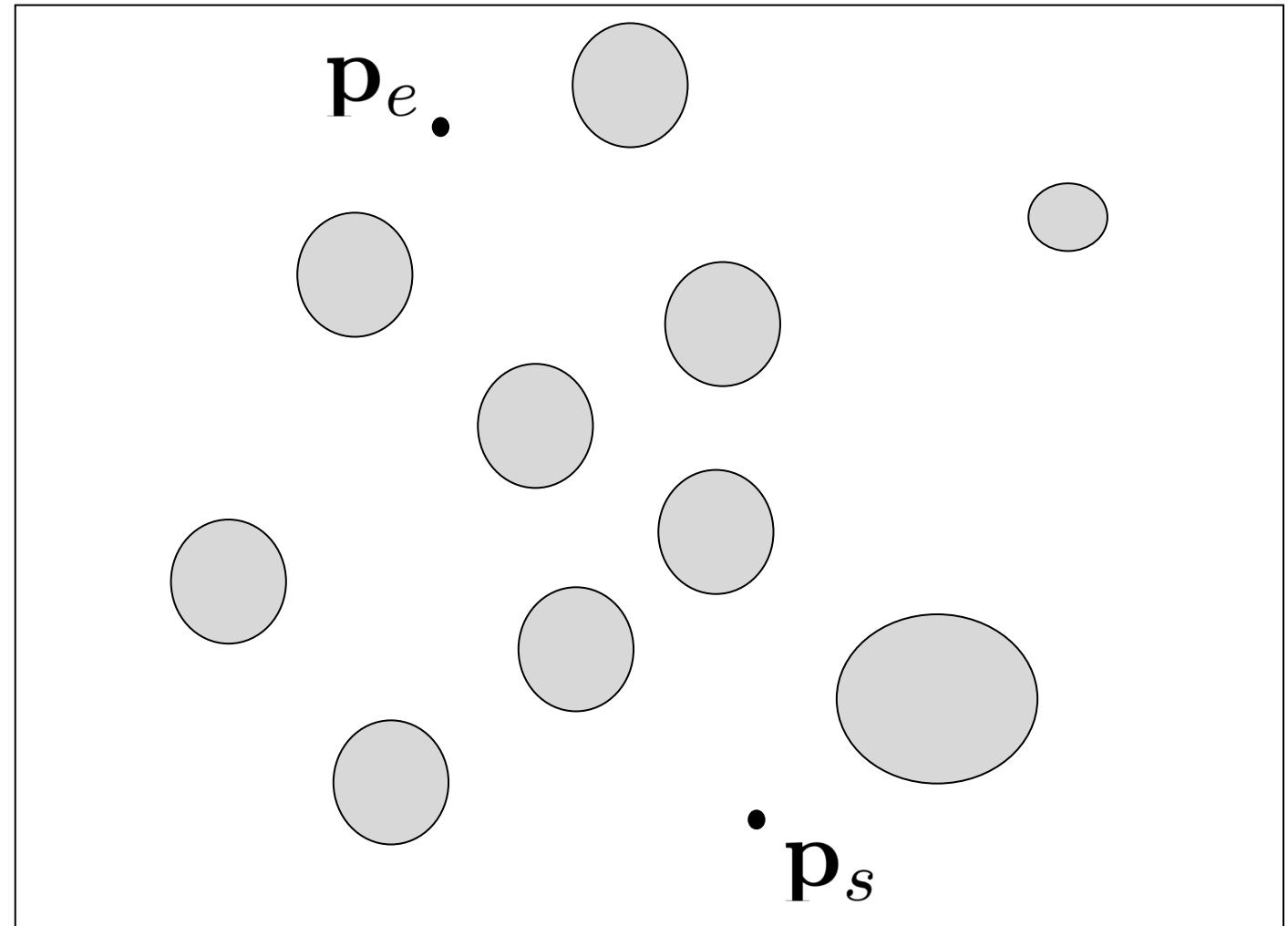
RRT Tree Structure

- RRT algorithm is implemented using tree data structure
- Tree is special case of directed graph
- Edges are directed from child node to parent
- Every node has one parent, except root
- Nodes represent physical states or configurations
- Edges represent feasible paths between states
- Each edge has cost associated traversing feasible path between states



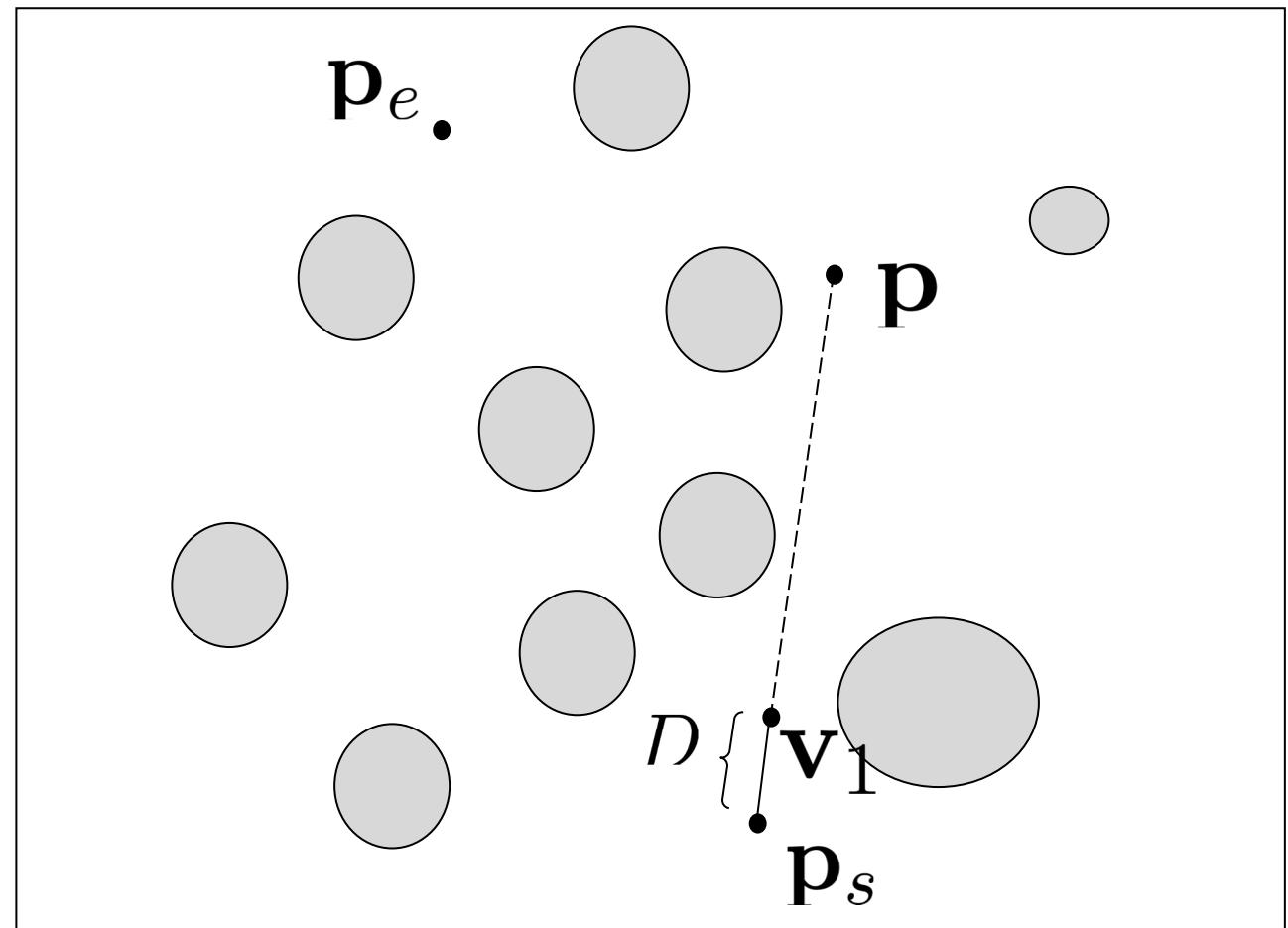
RRT Algorithm

- Algorithm initialized
 - start node
 - end node
 - terrain/obstacle map



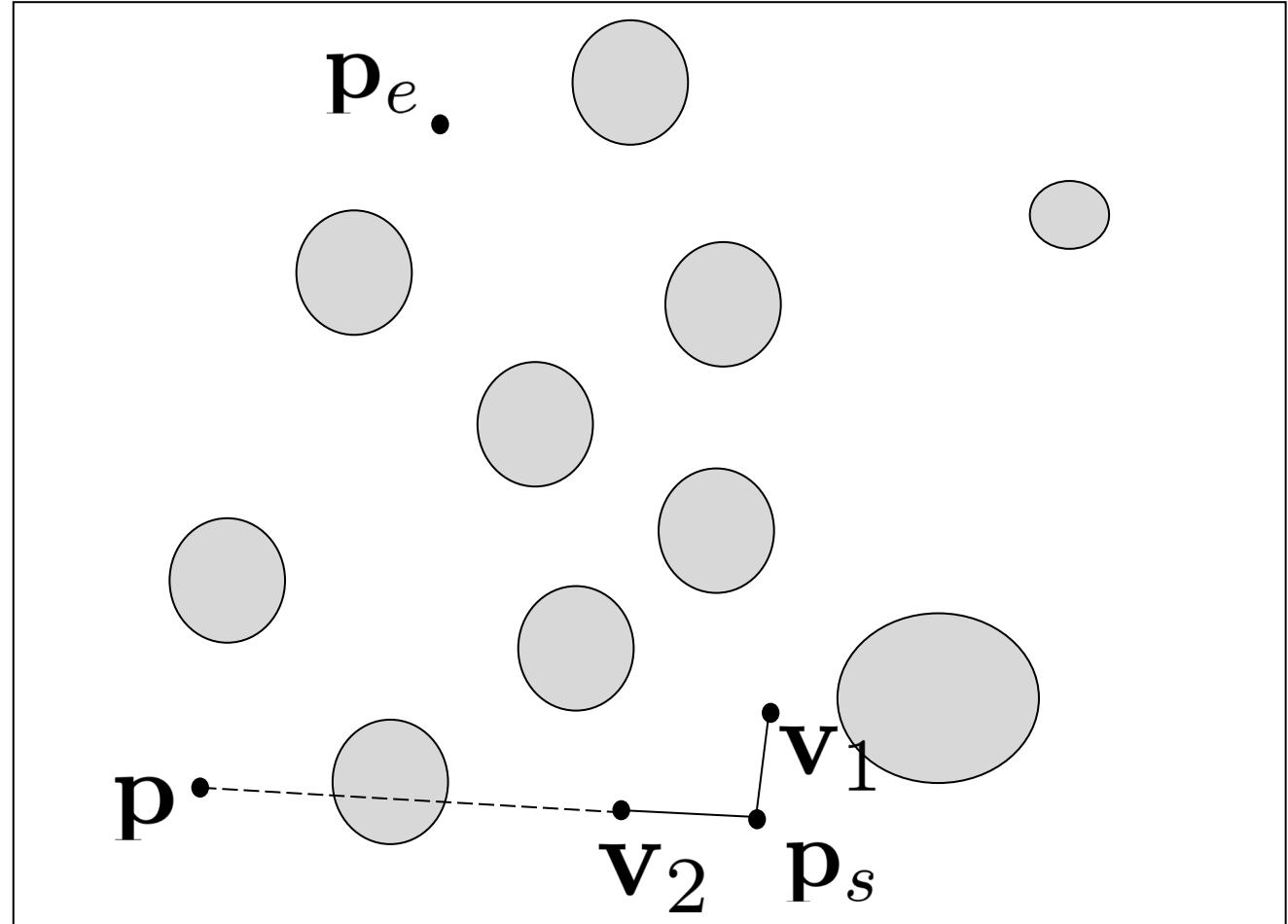
RRT Algorithm

- Randomly select configuration \mathbf{p} in workspace
- Select new configuration \mathbf{v}_1 fixed distance D from start point along line connecting \mathbf{p}_s and \mathbf{p}
- Check new segment for collisions
- If collision-free, insert \mathbf{v}_1 into tree.



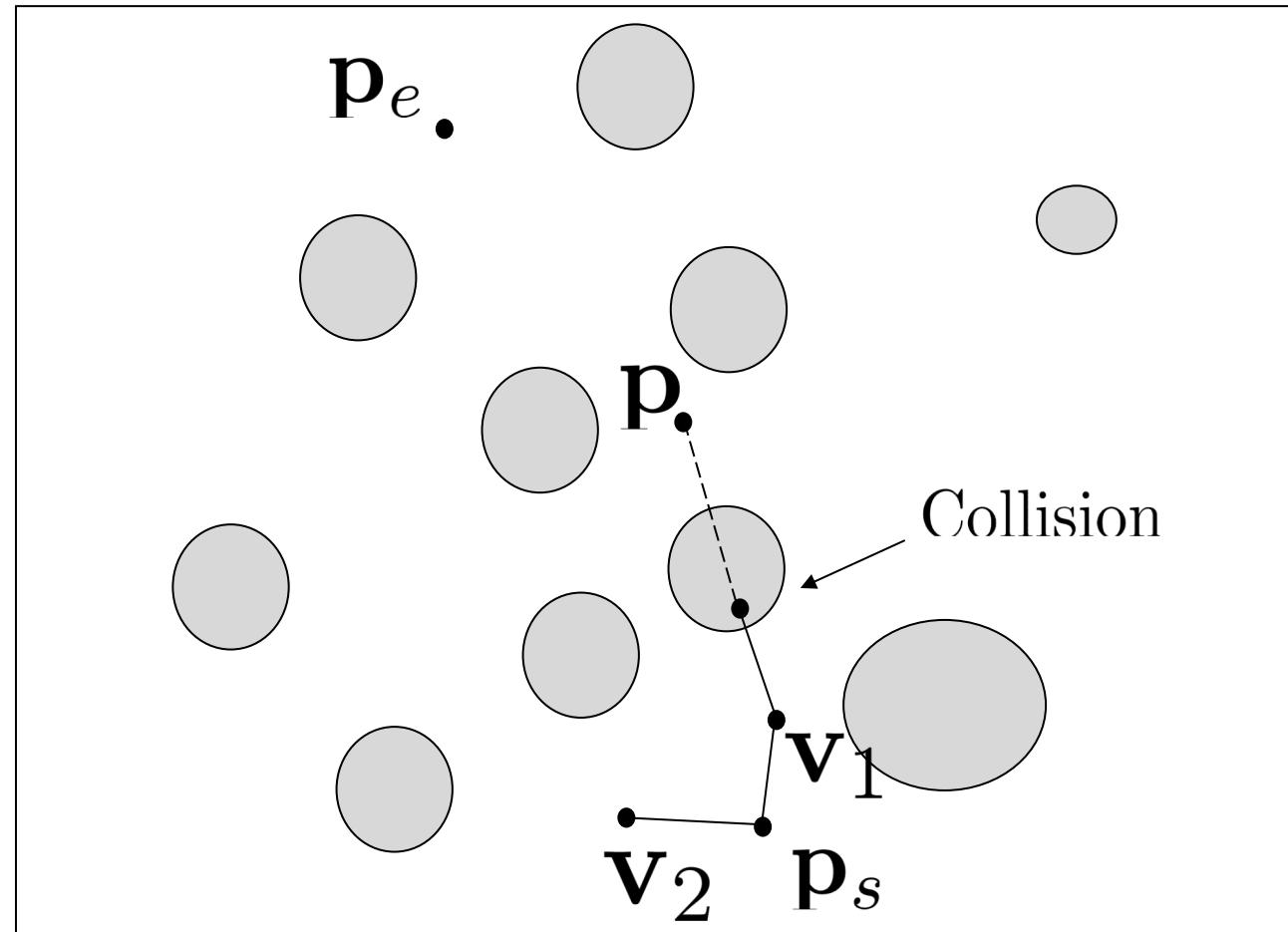
RRT Algorithm

- Generate random configuration \mathbf{p} in workspace
- Search tree to find node closest to \mathbf{p}
- From node closest to \mathbf{p} , move distance D along line connecting node and \mathbf{p} to establish configuration \mathbf{v}_2
- Check new segment for collisions
- If collision-free, insert \mathbf{v}_2 into tree



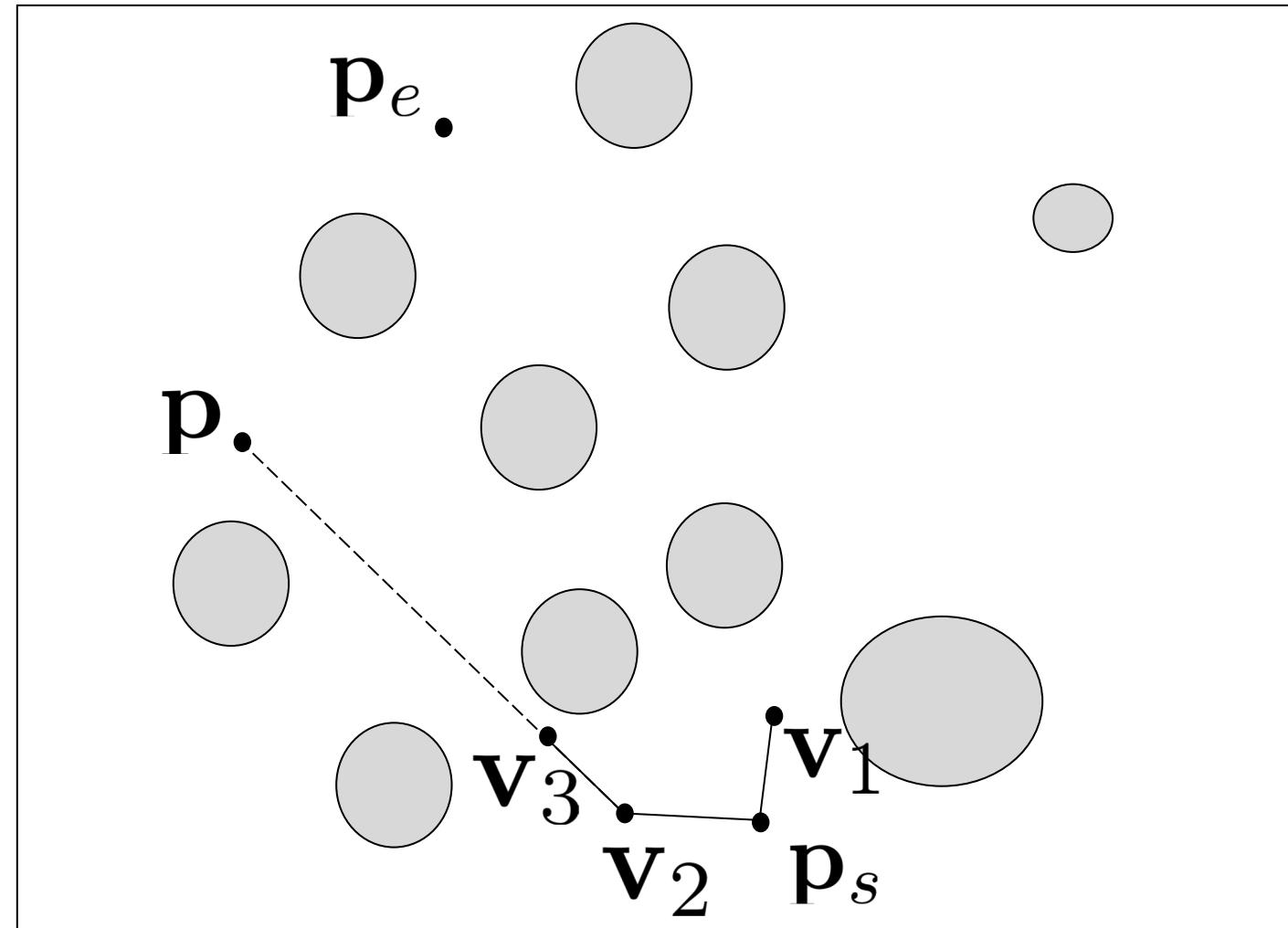
RRT Algorithm

- Generate random configuration \mathbf{p} in workspace
- Search tree to find node closest to \mathbf{p}
- From node closest to \mathbf{p} , move distance D along line connecting node and \mathbf{p} to establish new node
- Check new segment for collisions
- If collision-free, insert new node into tree



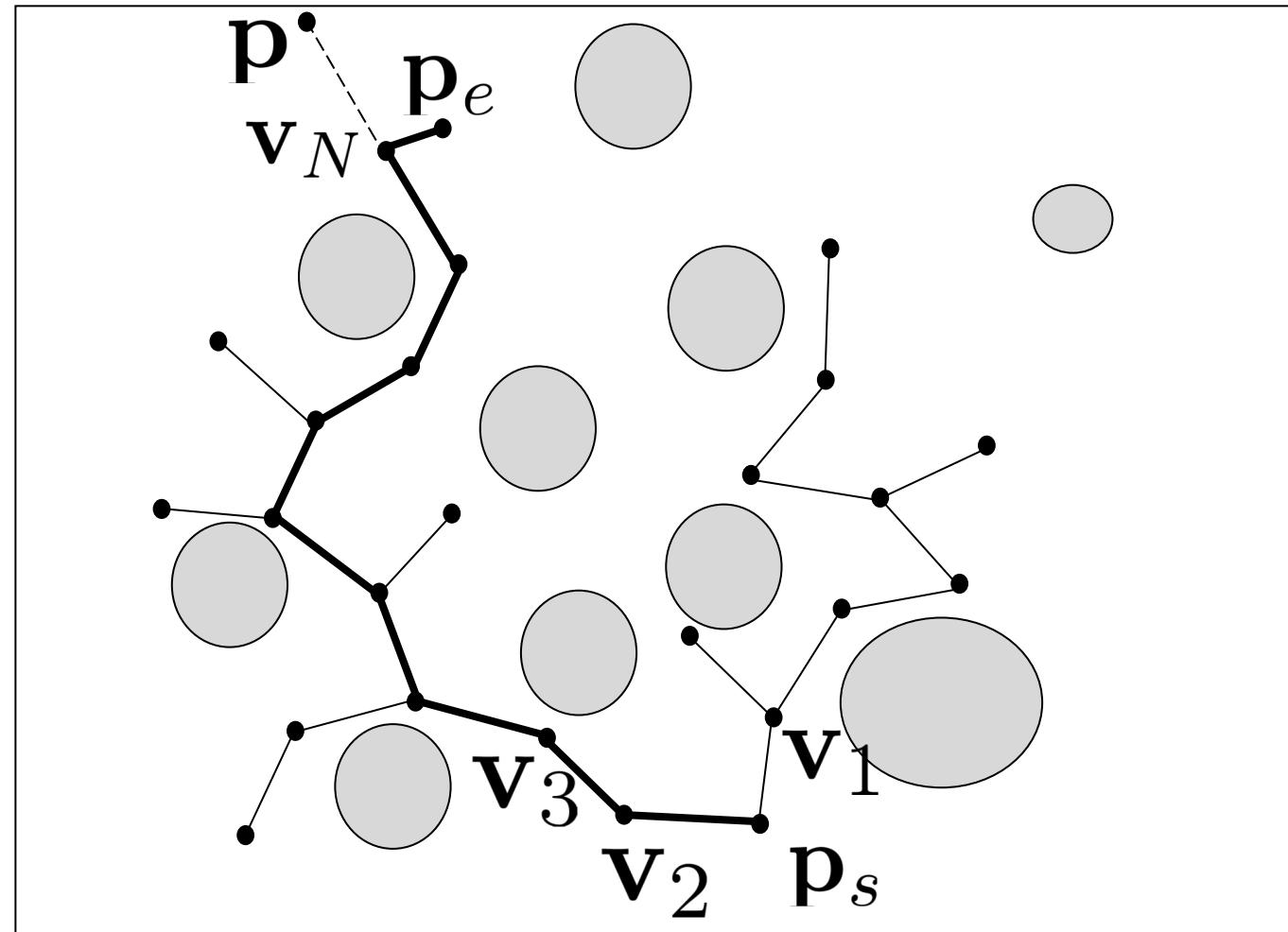
RRT Algorithm

- Continue adding nodes and checking for collisions



RRT Algorithm

- Continue until node is generated that is within distance D of end node
- At this point, terminate algorithm or search for additional feasible paths



RRT Algorithm

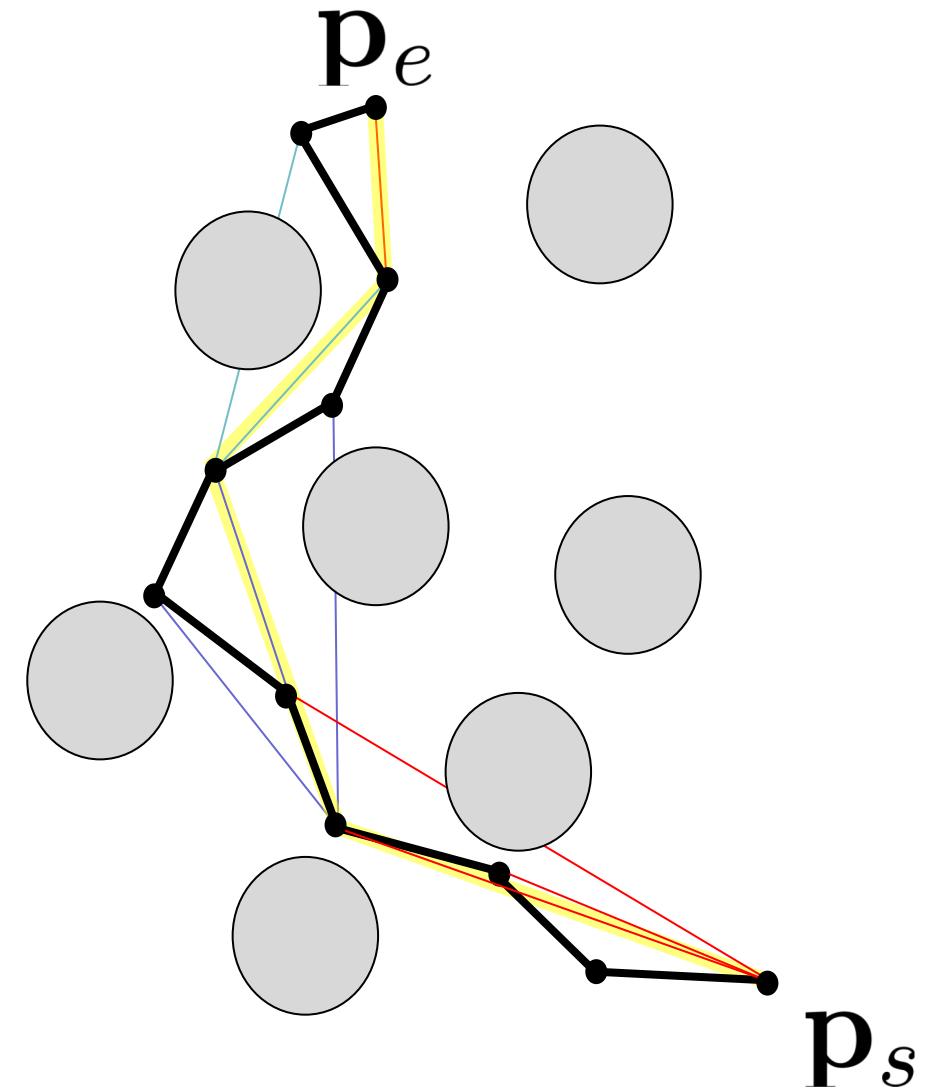
Algorithm 12 Plan RRT Path: $\mathcal{W} = \text{planRRT}(\mathcal{T}, \mathbf{p}_s, \mathbf{p}_e)$

Input: Terrain map \mathcal{T} , start configuration \mathbf{p}_s , end configuration \mathbf{p}_e

```
1: Initialize RRT graph  $G = (V, E)$  as  $V = \{\mathbf{p}_s\}$ ,  $E = \emptyset$ .
2: while The end node  $\mathbf{p}_e$  is not connected to  $G$ , i.e.,  $\mathbf{p}_e \notin V$  do
3:    $\mathbf{p} \leftarrow \text{generateRandomConfiguration}(\mathcal{T})$ 
4:    $\mathbf{v}^* \leftarrow \text{findClosestConfiguration}(\mathbf{p}, V)$ 
5:    $\mathbf{v}^+ \leftarrow \text{planPath}(\mathbf{v}^*, \mathbf{p}, D)$ 
6:   if existFeasiblePath( $\mathcal{T}, \mathbf{v}^*, \mathbf{v}^+$ ) then
7:     Update graph  $G = (V, E)$  as  $V \leftarrow V \cup \{\mathbf{v}^+\}$ ,
8:      $E \leftarrow E \cup \{(\mathbf{v}^*, \mathbf{v}^+)\}$ 
9:     Update edge costs as  $C[(\mathbf{v}^*, \mathbf{v}^+)] \leftarrow \text{pathLength}(\mathbf{v}^*, \mathbf{v}^+)$ 
10:    if existFeasiblePath( $\mathcal{T}, \mathbf{v}^+, \mathbf{p}_e$ ) then
11:      Update graph  $G = (V, E)$  as  $V \leftarrow V \cup \{\mathbf{p}_e\}$ ,
12:       $E \leftarrow E \cup \{(\mathbf{v}^+, \mathbf{p}_e)\}$ 
13:      Update edge costs as  $C[(\mathbf{v}^+, \mathbf{p}_e)] \leftarrow \text{pathLength}(\mathbf{v}^+, \mathbf{p}_e)$ 
14:    end if
15:  end if
16: end while
17:  $\mathcal{W} = \text{findShortestPath}(G, C)$  return  $\mathcal{W}$ 
```

Path Smoothing

- Start with initial point (1)
- Make connections to subsequent points in path (2), (3), (4) ...
- When connection collides with obstacle, add previous waypoint to smoothed path
- Continue smoothing from this point to end of path



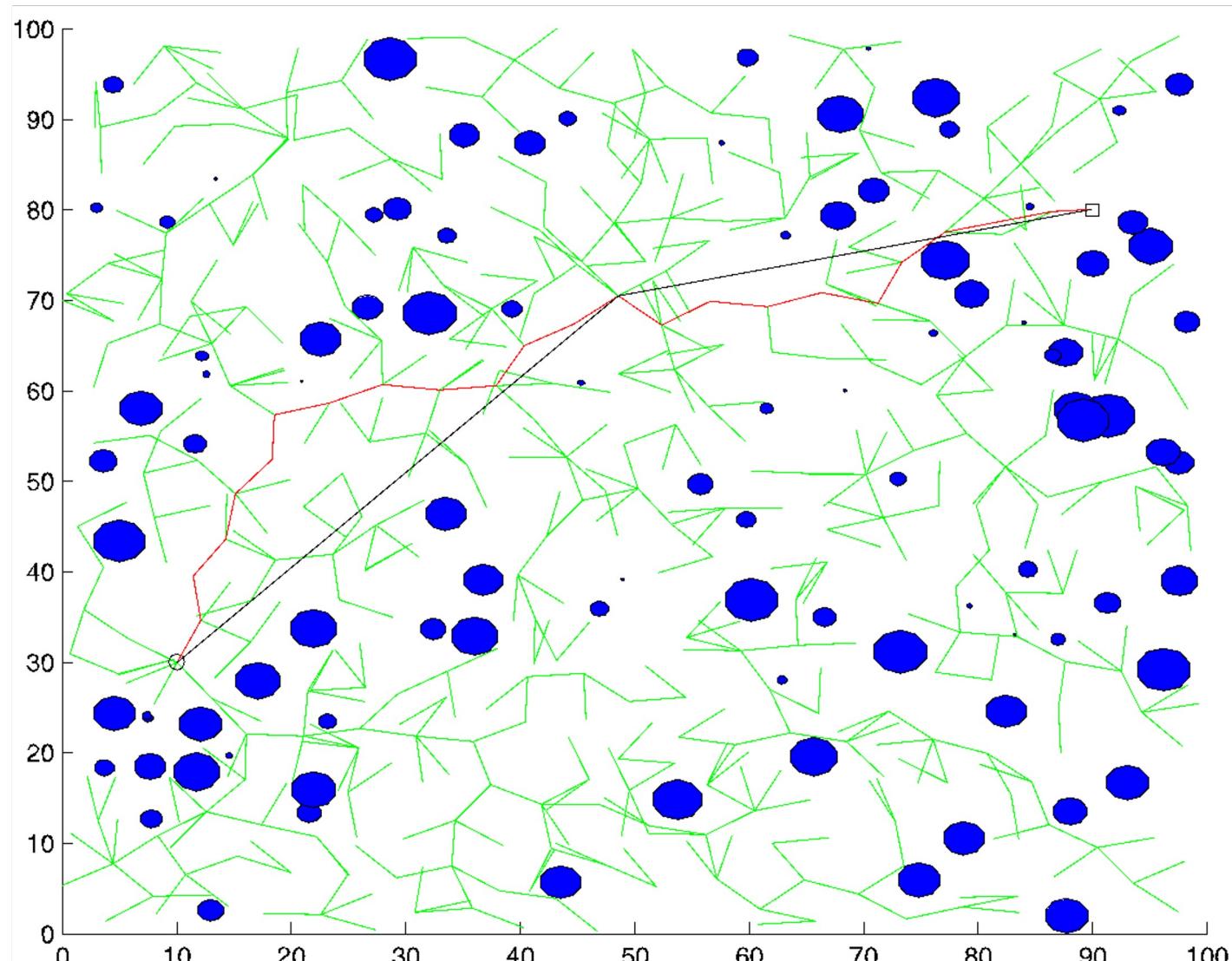
Path Smoothing Algorithm

Algorithm 13 Smooth RRT Path: $(\mathcal{W}_s, C_s) = \text{smoothRRT}(\mathcal{T}, \mathcal{W}, C)$

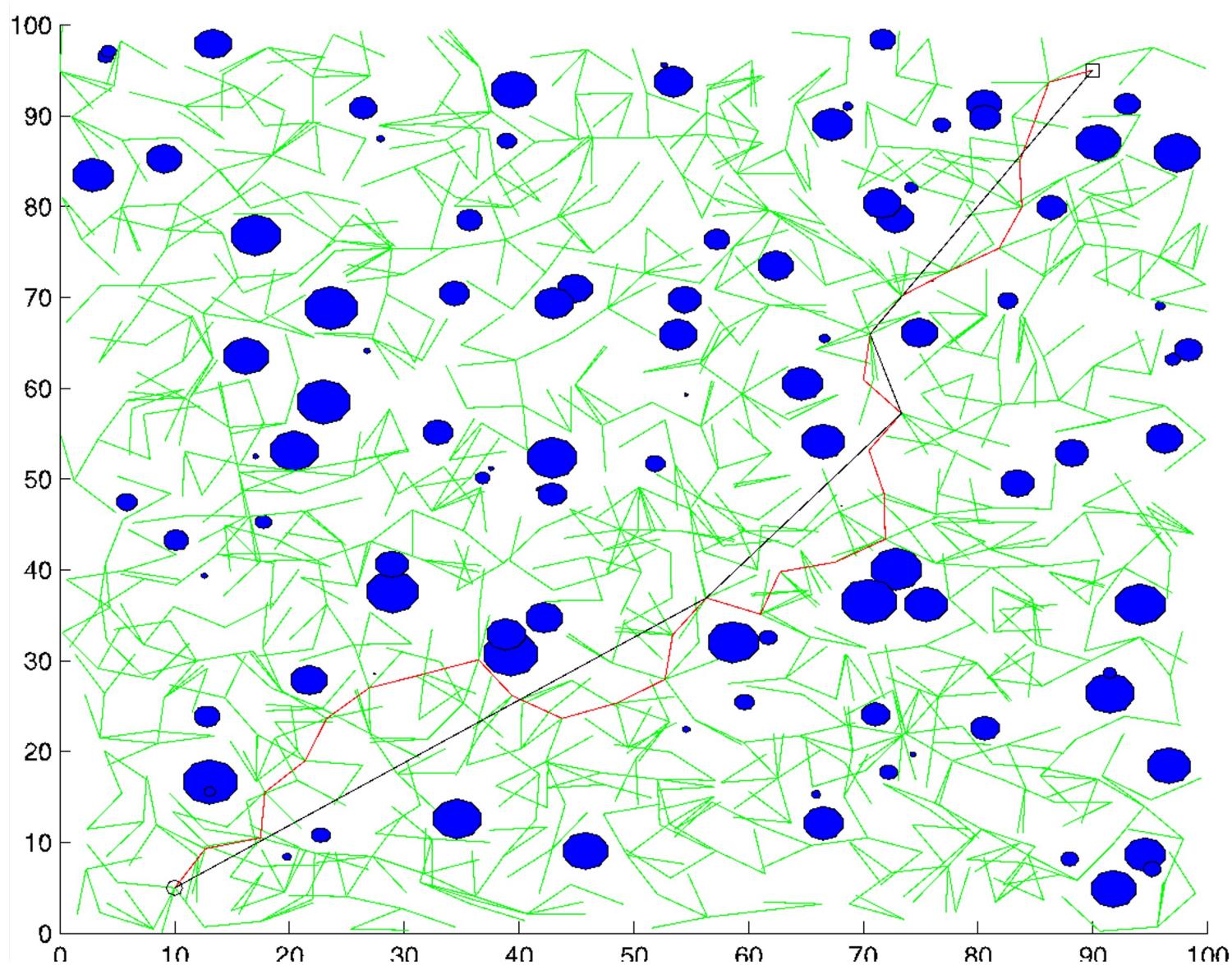
Input: Terrain map \mathcal{T} , waypoint path $\mathcal{W} = \{\mathbf{w}_1, \dots, \mathbf{w}_N\}$, cost matrix C

```
1: Initialized smoothed path  $\mathcal{W}_s \leftarrow \{\mathbf{w}_1\}$ 
2: Initialize pointer to current node in  $\mathcal{W}_s$ :  $i \leftarrow 1$ 
3: Initialize pointer to next node in  $\mathcal{W}$ :  $j \leftarrow 2$ 
4: while  $j < N$  do
5:    $\mathbf{w}_s \leftarrow \text{getNode}(\mathcal{W}_s, i)$ 
6:    $\mathbf{w}^+ \leftarrow \text{getNode}(\mathcal{W}, j + 1)$ 
7:   if  $\text{existFeasiblePath}(\mathcal{T}, \mathbf{w}_s, \mathbf{w}^+) = \text{FALSE}$  then
8:     Get last node:  $\mathbf{w} \leftarrow \text{getNode}(\mathcal{W}, j)$ 
9:     Add deconflicted node to smoothed path:  $\mathcal{W}_s \leftarrow \mathcal{W}_s \cup \{\mathbf{w}\}$ 
10:    Update smoothed cost:  $C_s[(\mathbf{w}_s, \mathbf{w})] \leftarrow \text{pathLength}(\mathbf{w}_s, \mathbf{w})$ 
11:     $i \leftarrow j$ 
12:   end if
13:    $j \leftarrow j + 1$ 
14: end while
15: Add last node from  $\mathcal{W}$ :  $\mathcal{W}_s \leftarrow \mathcal{W}_s \cup \{\mathbf{w}_N\}$ 
16: Update smoothed cost:  $C_s[(\mathbf{w}_i, \mathbf{w}_N)] \leftarrow \text{pathLength}(\mathbf{w}_i, \mathbf{w}_N)$  return
       $\mathcal{W}_s$ 
```

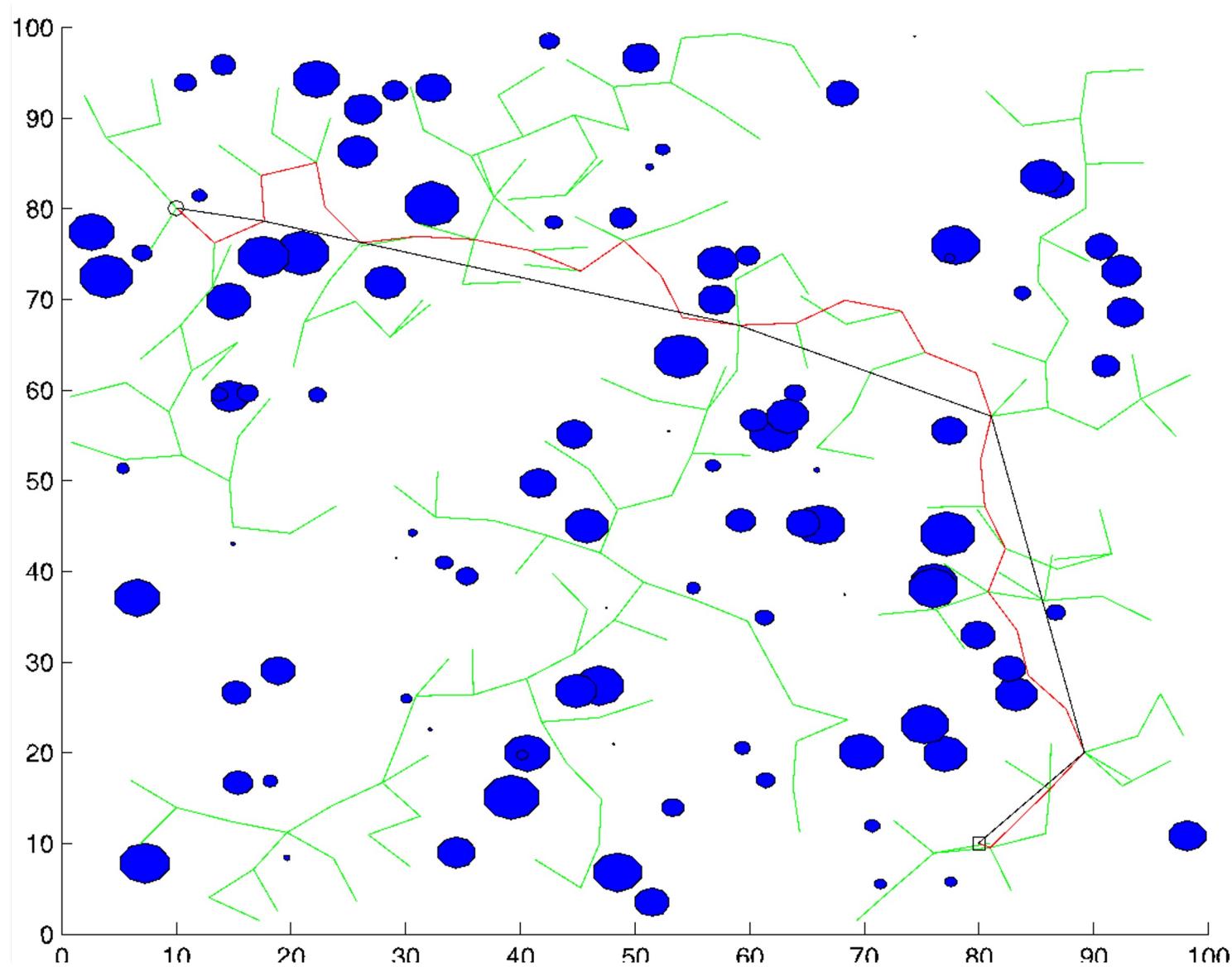
RRT Results



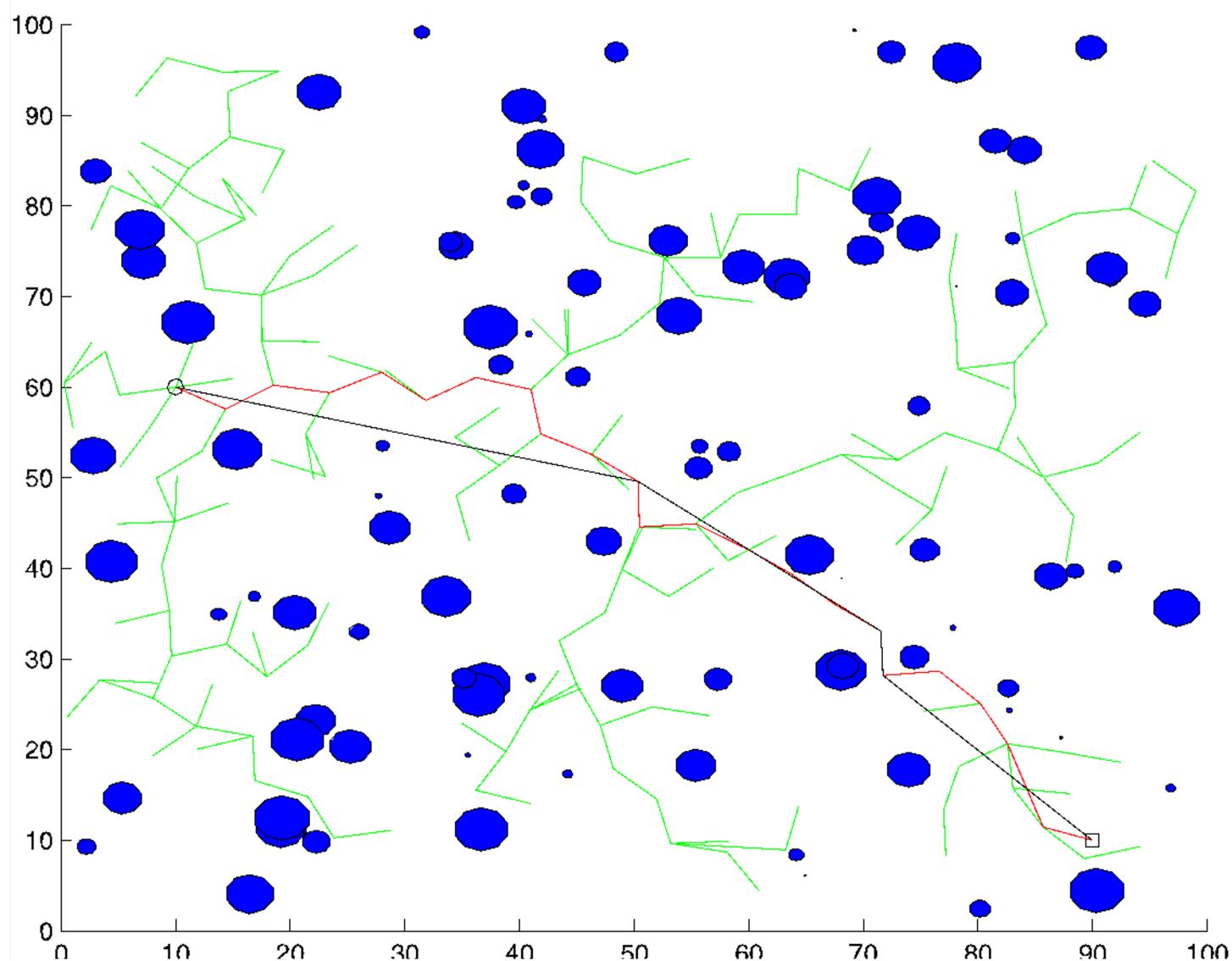
RRT Results



RRT Results

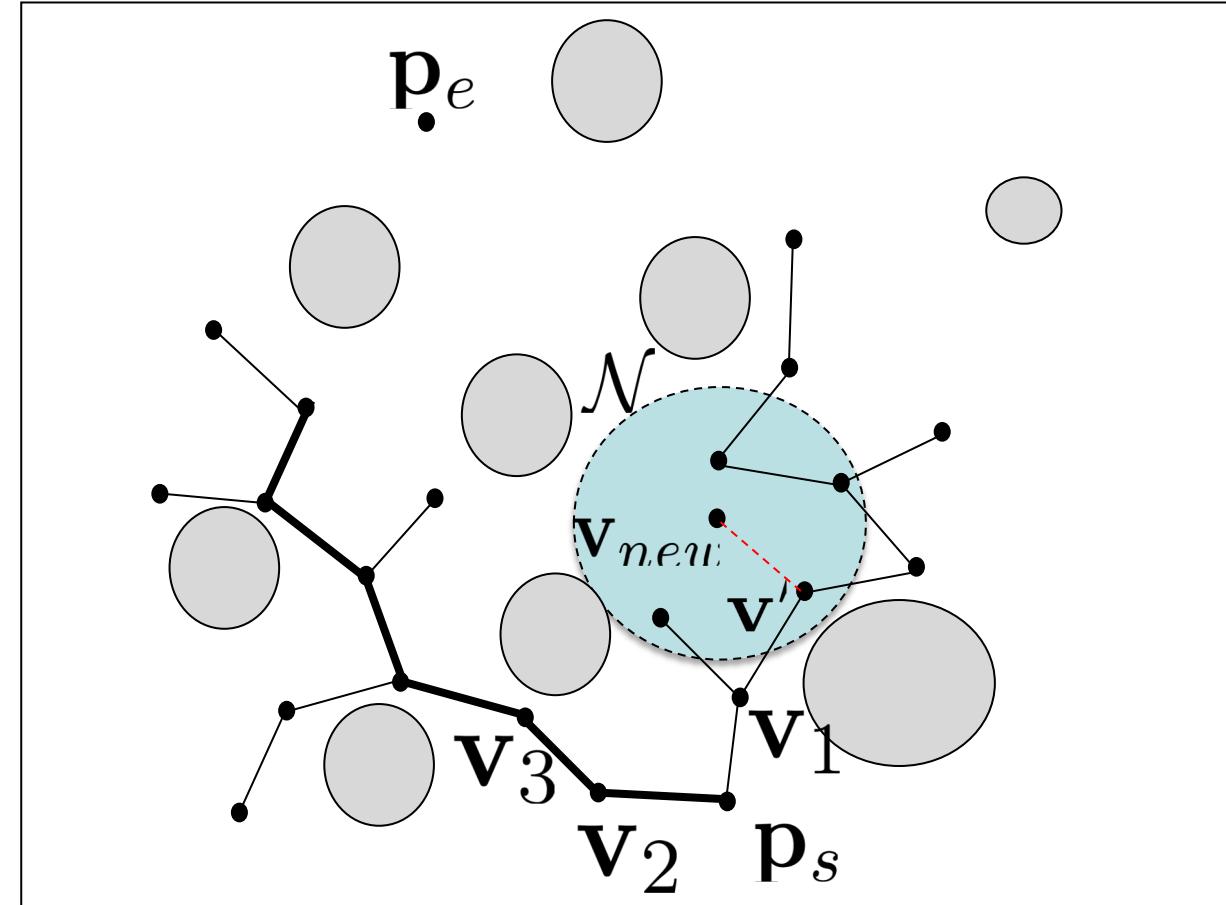


RRT Results



RRT* Algorithm – Extend Step

- Generate new potential node \mathbf{v}_{new} identical to RRT.
- Instead of finding the closest node in the tree, find all nodes within neighborhood \mathcal{N} .
- Let $C(\mathbf{v})$ be the path cost from \mathbf{p}_s to \mathbf{v} , and let $c(\mathbf{v}_1, \mathbf{v}_2)$ be the path cost from \mathbf{v}_1 to \mathbf{v}_2 .
- Let $\mathbf{v}' = \arg \min_{\mathbf{v} \in \mathcal{N}} (C(\mathbf{v}) + c(\mathbf{v}_{new}, \mathbf{v}))$ (i.e., closest node in \mathcal{N} to \mathbf{v}_{new}).
- Add node $V \leftarrow V \cup \{\mathbf{v}_{new}\}$.
- Add edge $E \leftarrow E \cup \{(\mathbf{v}', \mathbf{v}_{new})\}$.
- Set $C(\mathbf{v}_{new}) = C(\mathbf{v}') + c(\mathbf{v}_{new}, \mathbf{v})$.



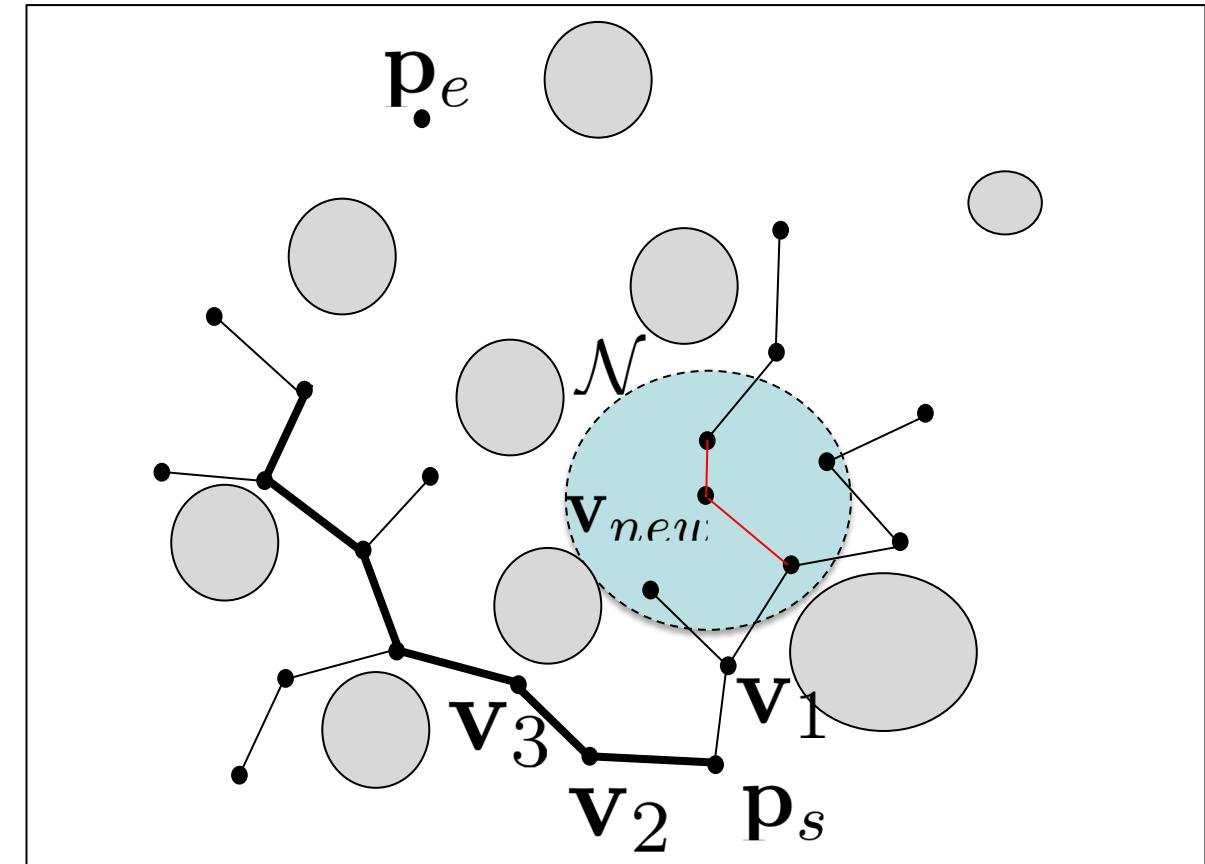
RRT* Algorithm – Re-wire Step

- Check all nodes in $\mathbf{v} \in \mathcal{N}$ to see if

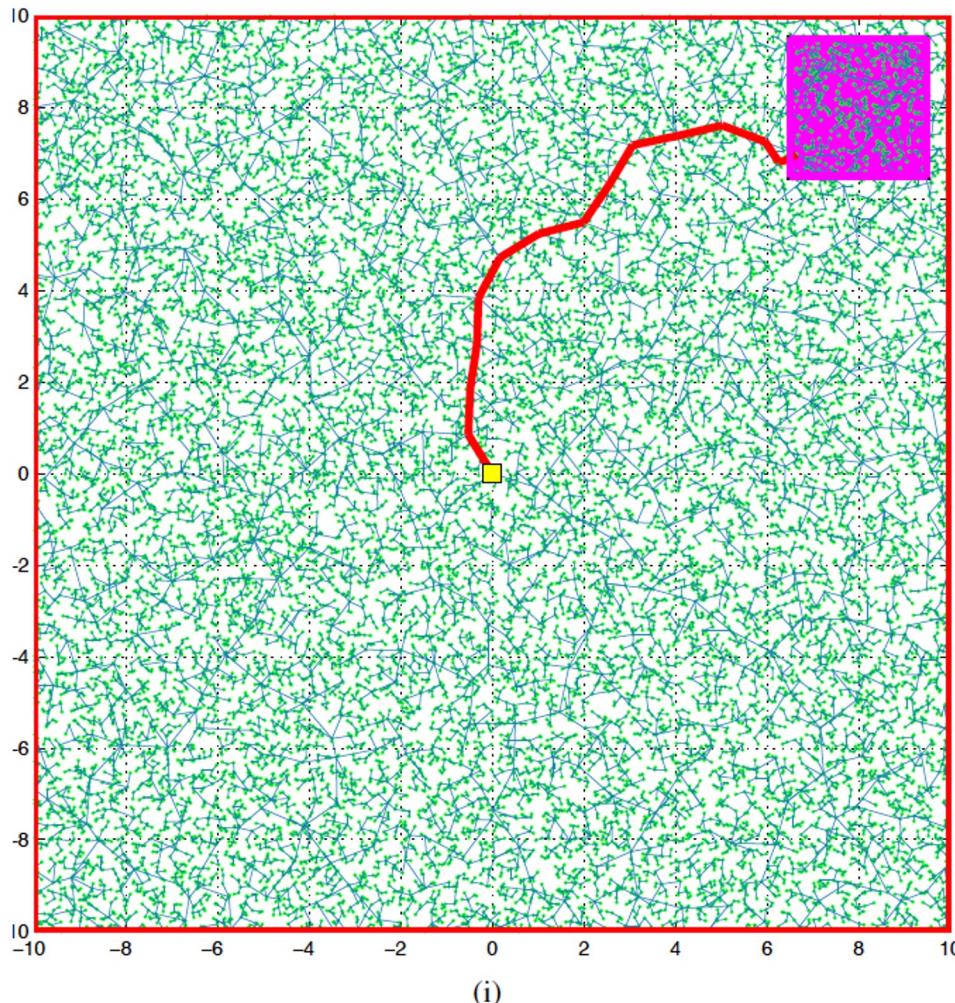
$$C(\mathbf{v}_{new}) + c(\mathbf{v}_{new}, \mathbf{v}) < C(\mathbf{v})$$

i.e., if re-routing through \mathbf{v}_{new} reduces the path length.

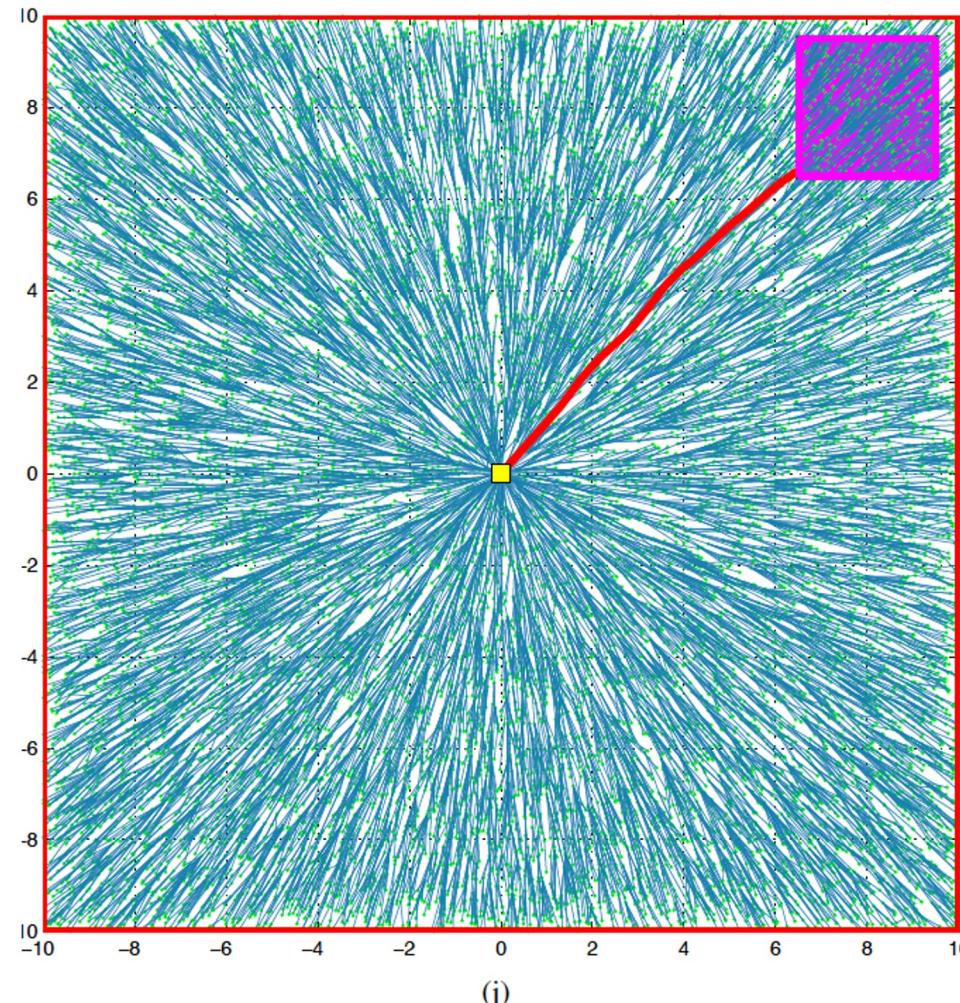
- If so, remove edge between \mathbf{v} and its parent, and add edge between \mathbf{v} and \mathbf{v}_{new} .



RRT vs. RRT*



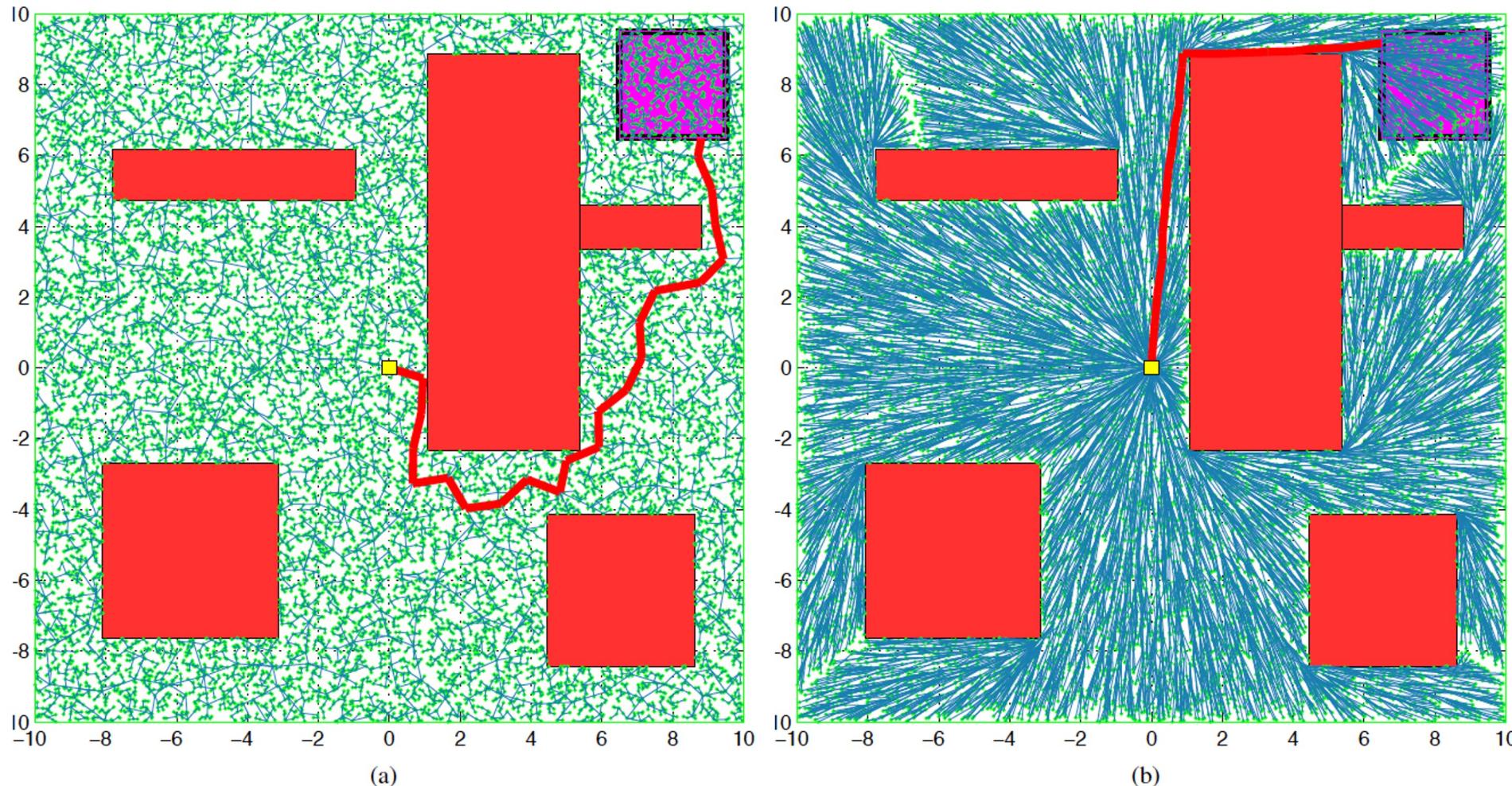
(i)



(j)

From S. Karaman and E. Frazzoli, “Incremental Sampling-based Algorithms for Optimal Motion Planning,” International Journal of Robotic Research, 2010.

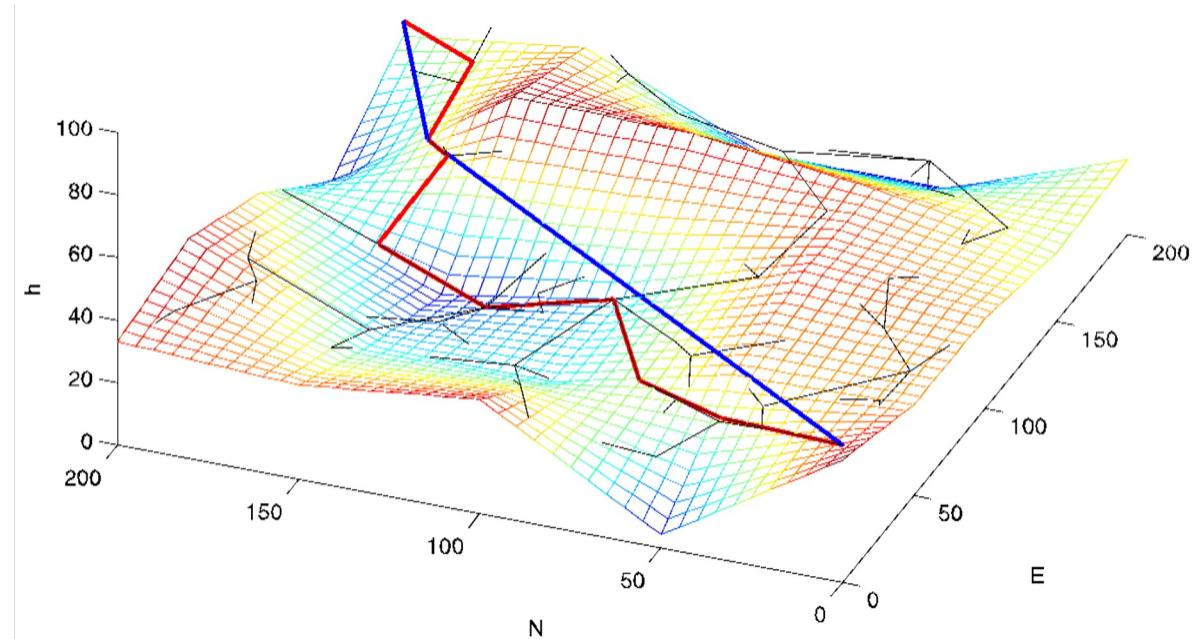
RRT vs. RRT*



From S. Karaman and E. Frazzoli, "Incremental Sampling-based Algorithms for Optimal Motion Planning," International Journal of Robotic Research, 2010.

RRT Path Planning Over 3D Terrain

- Assume terrain map can be queried to determine altitude of terrain at any north-east location
- Must be able to determine altitude for random configuration p in RRT algorithm
- Must be able to detect collisions with terrain – reject random candidate paths leading to collision
- Options:
 - random altitude within predetermined range
 - random selection of discrete altitudes in desired range
 - set altitude above ground level
- Test candidate paths to ensure flight path angles are feasible – reject if infeasible



RRT Algorithm – 3D Terrain

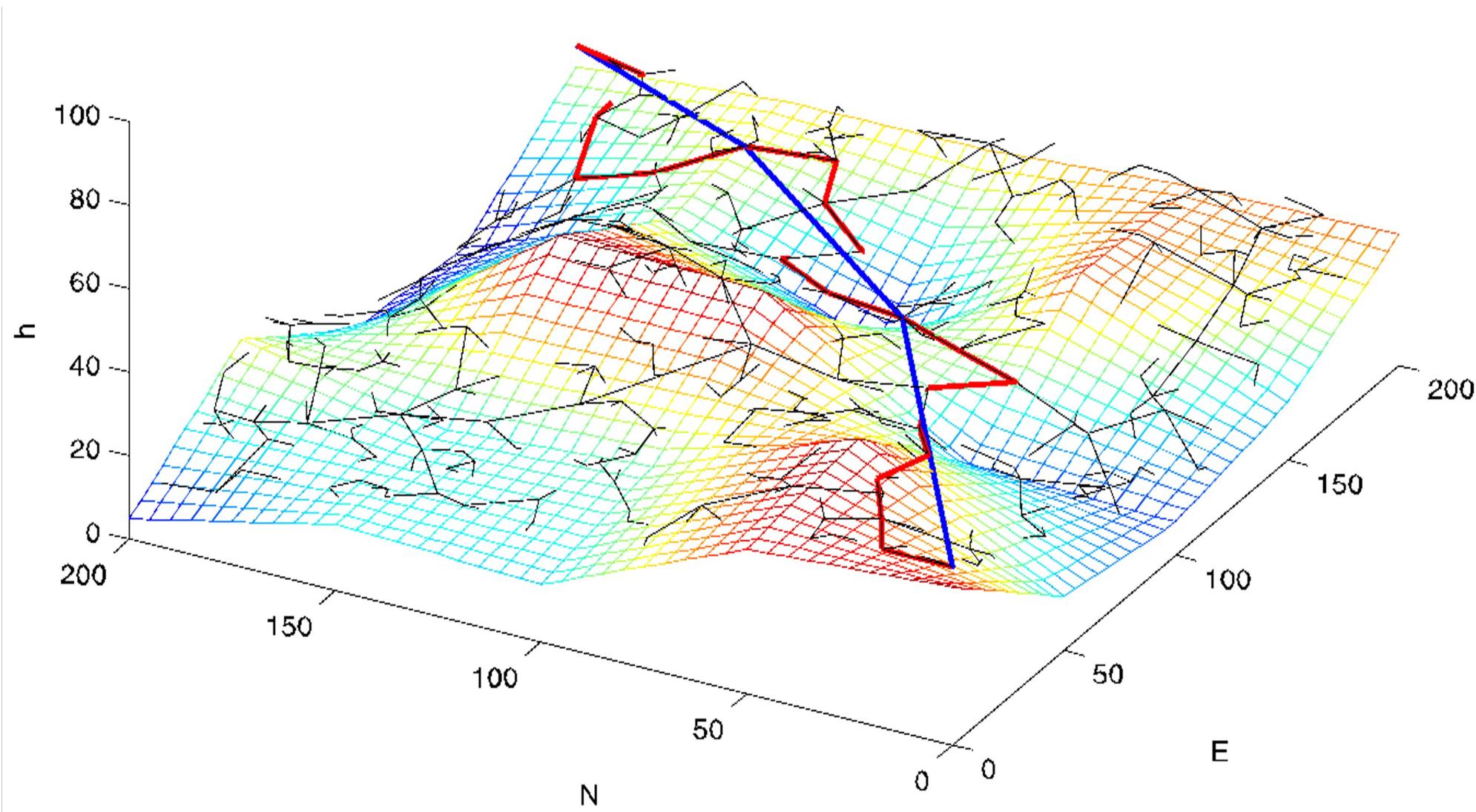
Algorithm 10 Plan RRT Path: $\mathcal{W} = \text{planRRT}(\mathcal{T}, \mathbf{p}_s, \mathbf{p}_e)$

Input: Terrain map \mathcal{T} , start configuration \mathbf{p}_s , end configuration \mathbf{p}_e

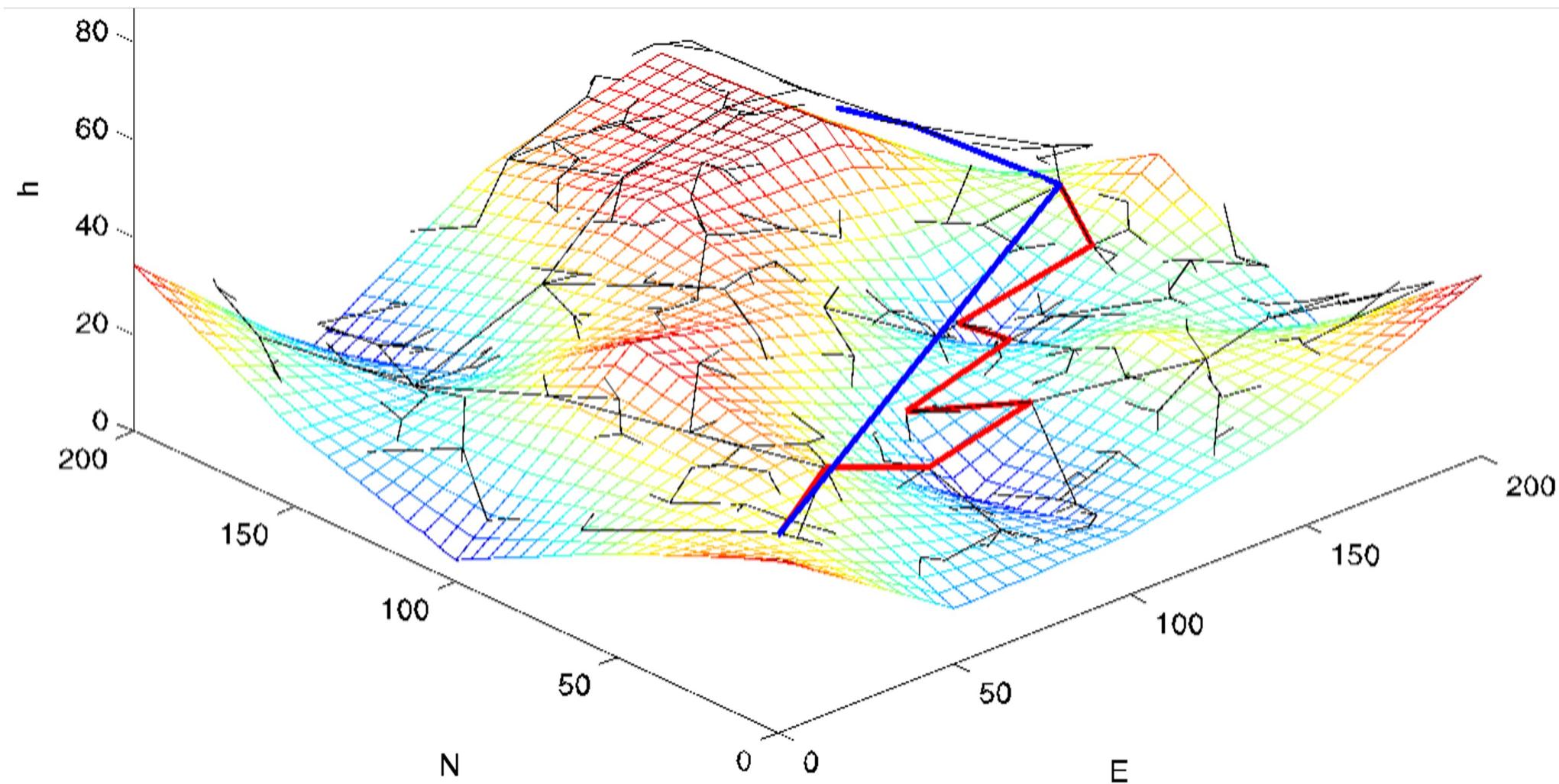
```
1: Initialize RRT graph  $G = (V, E)$  as  $V = \{\mathbf{p}_s\}$ ,  $E = \emptyset$ 
2: while The end node  $\mathbf{p}_e$  is not connected to  $G$ , i.e.,  $\mathbf{p}_e \notin V$  do
3:    $\mathbf{p} \leftarrow \text{generateRandomConfiguration}(\mathcal{T})$ 
4:    $\mathbf{v}^* \leftarrow \text{findClosestConfiguration}(\mathbf{p}, V)$ 
5:    $\mathbf{v}^+ \leftarrow \text{planPath}(\mathbf{v}^*, \mathbf{p}, D)$ 
6:   if existFeasiblePath( $\mathcal{T}$ ,  $\mathbf{v}^*$ ,  $\mathbf{v}^+$ ) then
7:     Update graph  $G = (V, E)$  as  $V \leftarrow V \cup \{\mathbf{v}^+\}$ ,  $E \leftarrow E \cup \{(\mathbf{v}^*, \mathbf{v}^+)\}$ 
8:     Update edge costs as  $C[(\mathbf{v}^*, \mathbf{v}^+)] \leftarrow \text{pathLength}(\mathbf{v}^*, \mathbf{v}^+)$ 
9:   end if
10:  if existFeasiblePath( $\mathcal{T}$ ,  $\mathbf{v}^+$ ,  $\mathbf{p}_e$ ) then
11:    Update graph  $G = (V, E)$  as  $V \leftarrow V \cup \{\mathbf{p}_e\}$ ,  $E \leftarrow E \cup \{(\mathbf{v}^*, \mathbf{p}_e)\}$ 
12:    Update edge costs as  $C[(\mathbf{v}^*, \mathbf{p}_e)] \leftarrow \text{pathLength}(\mathbf{v}^*, \mathbf{p}_e)$ 
13:  end if
14: end while
15:  $\mathcal{W} = \text{findShortestPath}(G, C)$ .
16: return  $\mathcal{W}$ 
```

modify to test for collisions
with terrain and flight path
angle feasibility

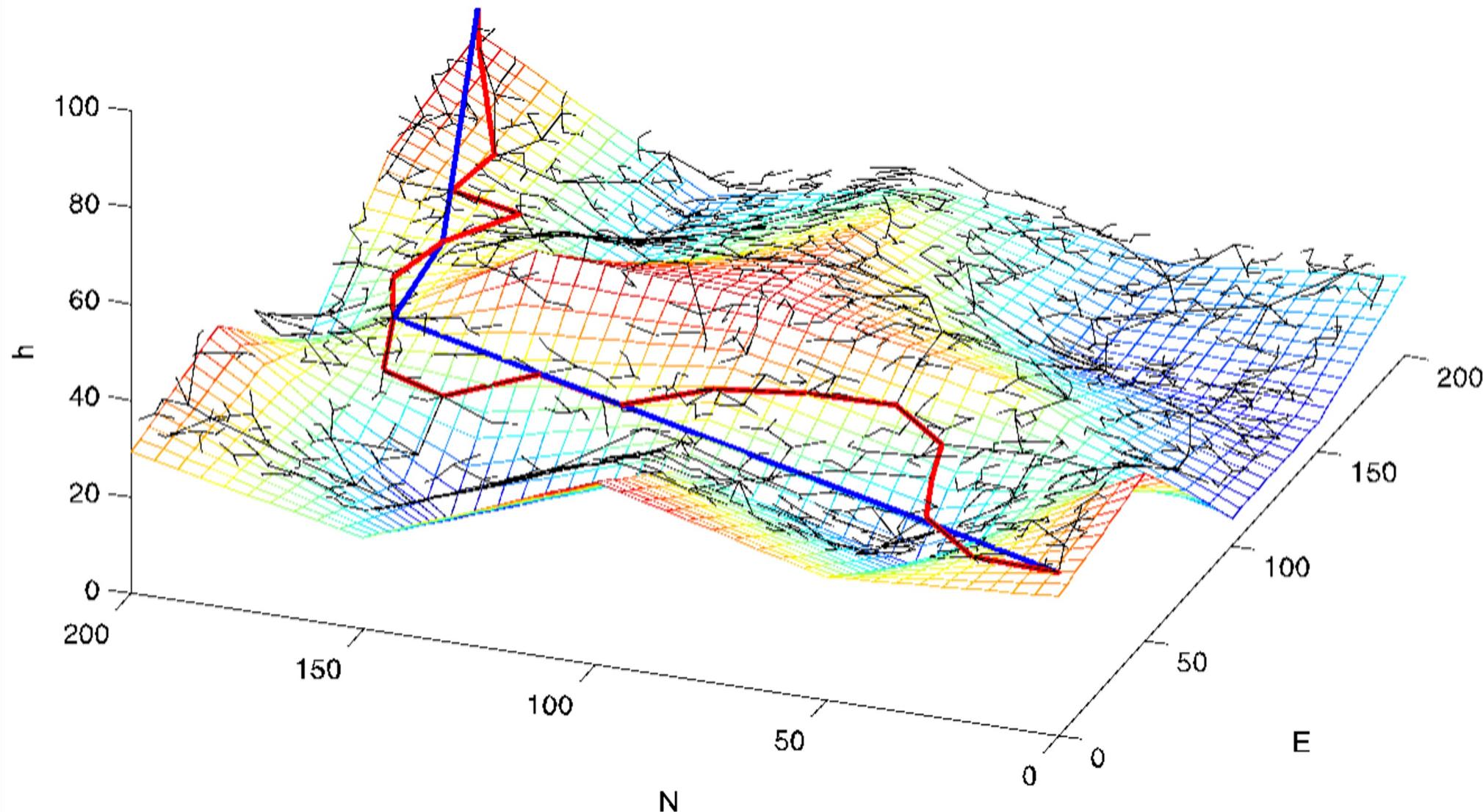
RRT 3D Terrain Results



RRT 3D Terrain Results



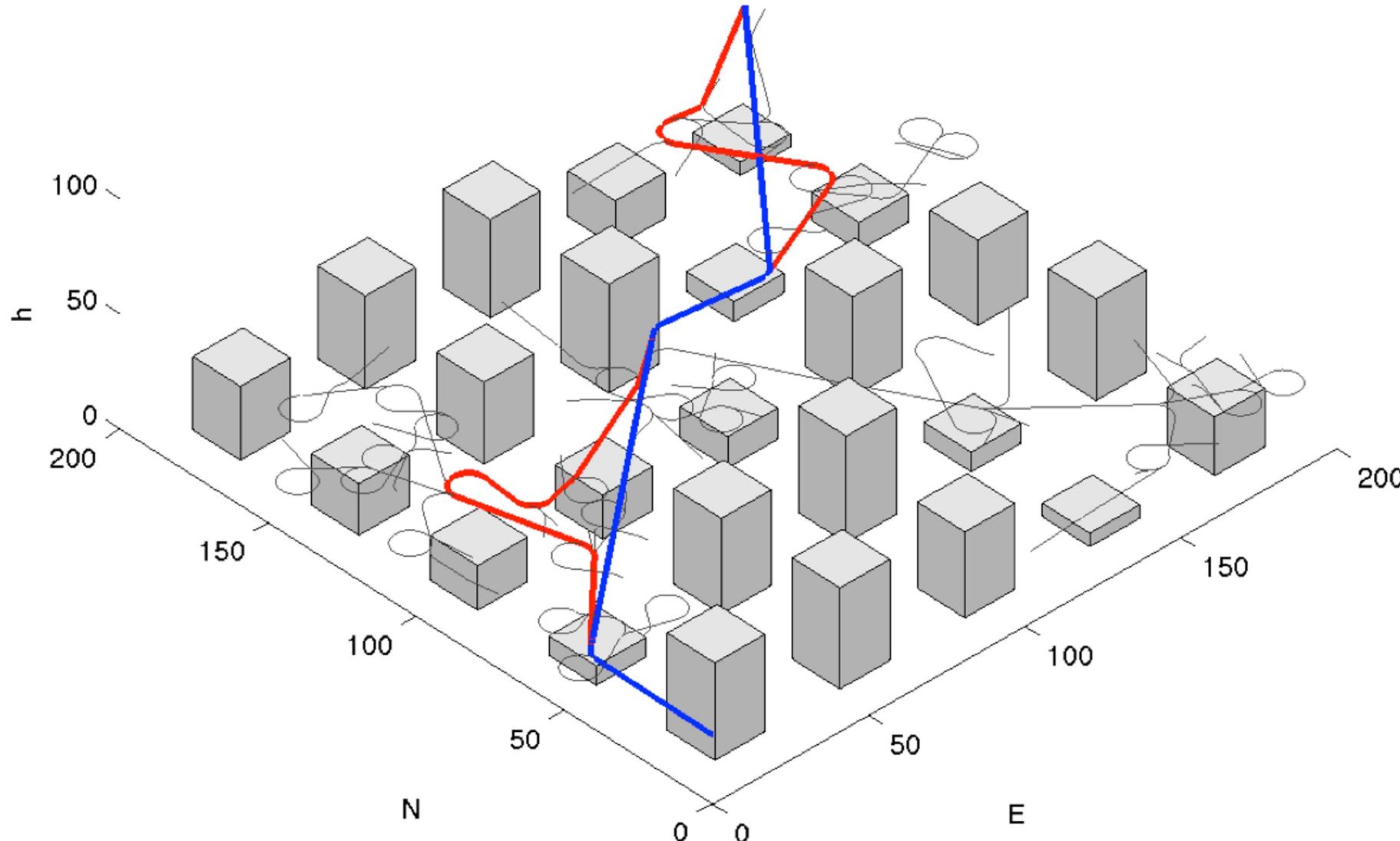
RRT 3D Terrain Results



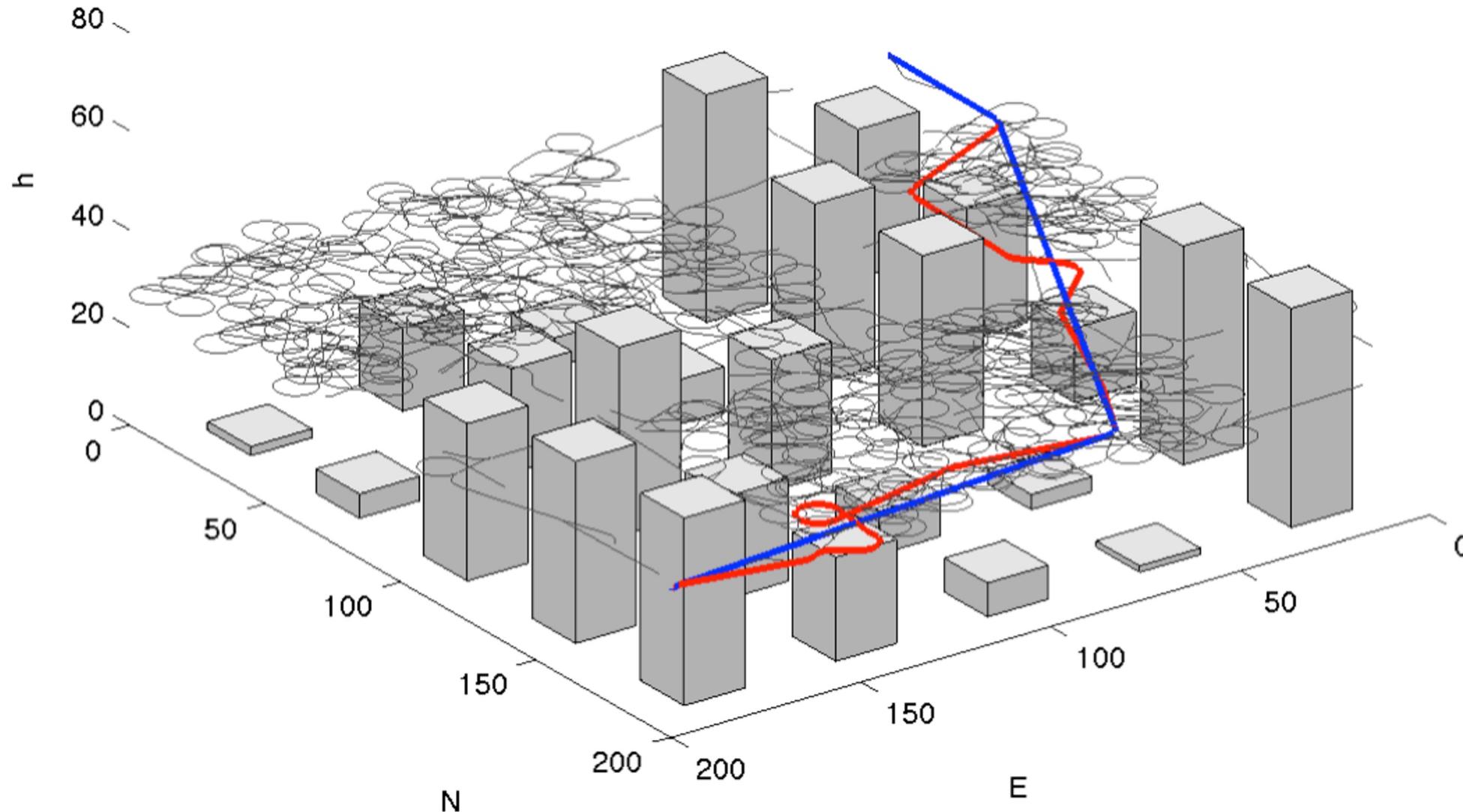
RRT Dubins Approach

- To generate a new random configuration for tree:
 - Generate random N-E position in environment
 - Find closest node in RRT graph to new random point
 - Select a position of distance L from the closest RRT node in direction of new point – use this position as N-E coordinates of new configuration
 - Define course angle for new configuration as the angle of the line connecting the RRT graph to the new configuration

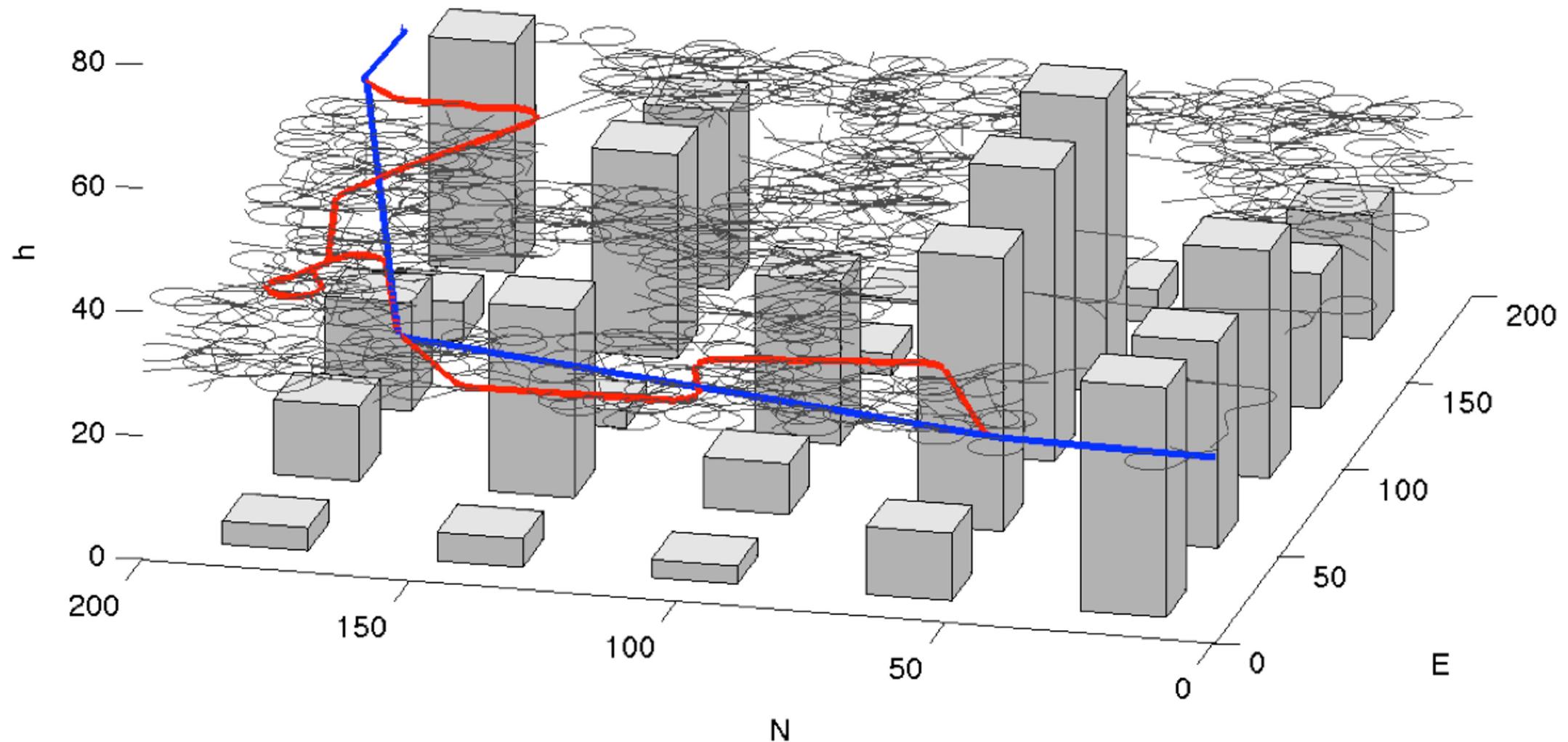
RRT Dubins Results



RRT Dubins Results

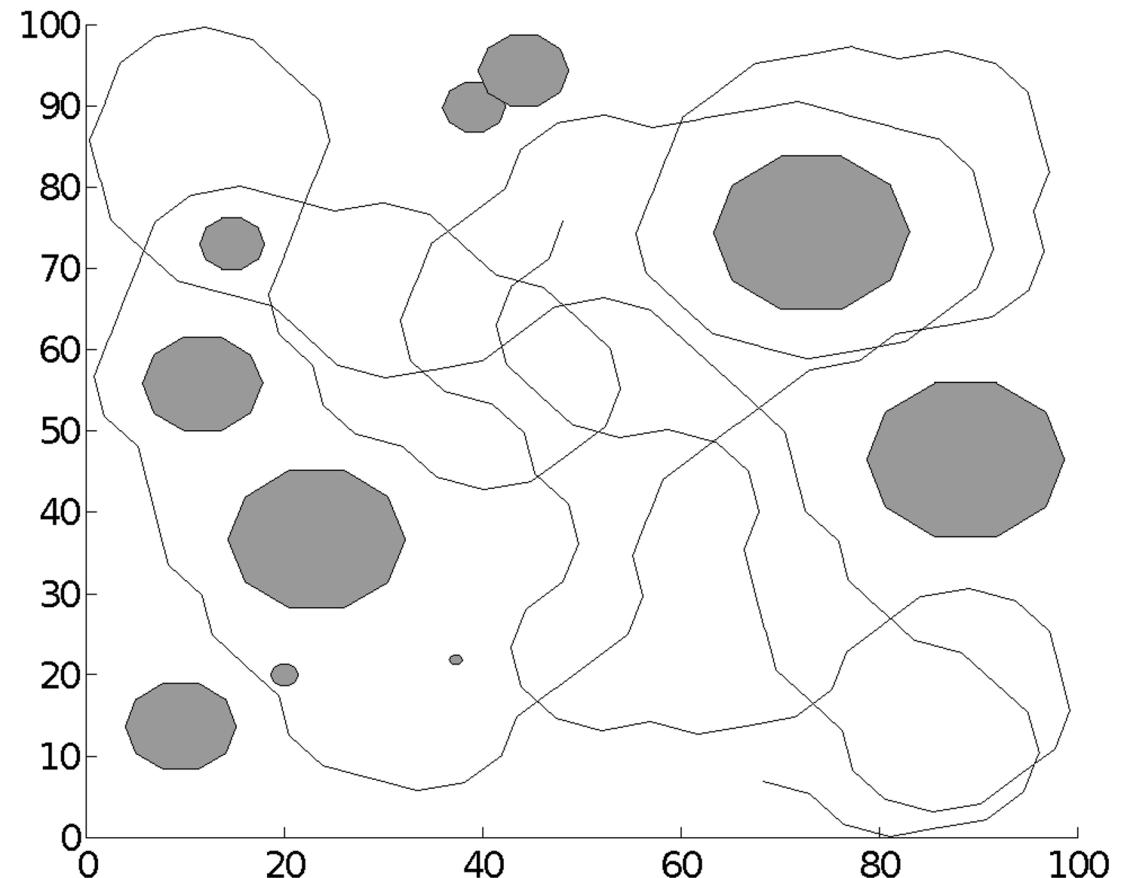


RRT Dubins Results



Coverage Algorithms

- Goal: Survey an area
 - Pass sensor footprint over entire area
- Algorithms often cell based
 - Goal: visit every cell



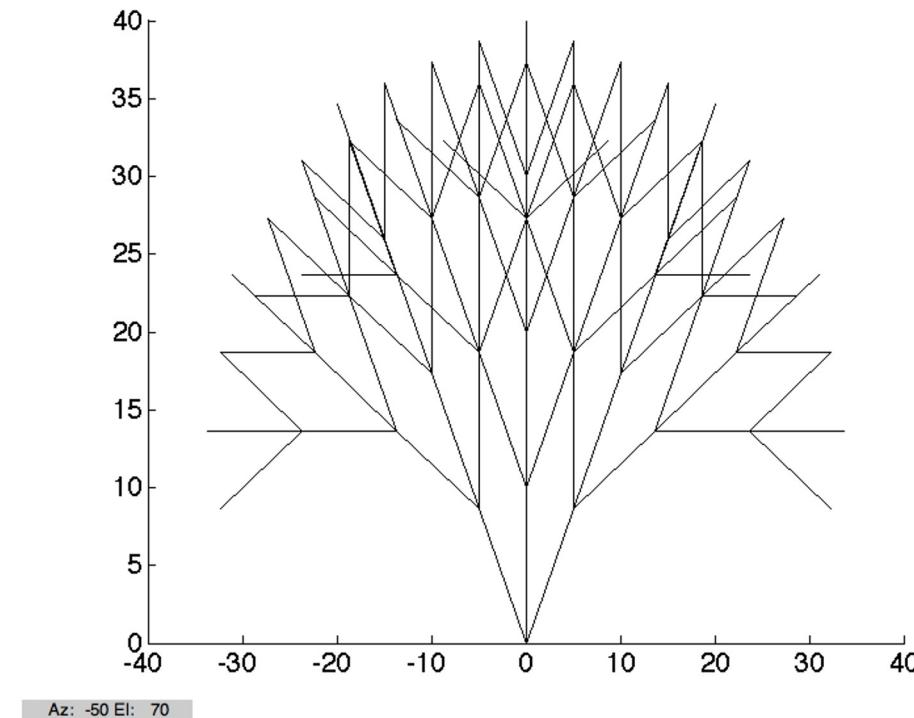
Coverage Algorithm

- Two maps in memory
 - terrain map
 - used to detect collisions with environment
 - coverage or return map
 - used to track coverage of terrain
- Return map stores value of returning to particular location
 - Return map initialized so that all locations have same return value
 - As locations are visited, return value of that location is decremented by fixed amount:

$$\Upsilon_i[k] = \Upsilon_i[k - 1] - c$$

Coverage Algorithm

- Finite look ahead tree search used to determine where to go
- Tree generated from current MAV configuration
- Tree searched to determine path that maximizes return value
- Two methods for look ahead tree
 - Uniform branching
 - Predetermined depth
 - Uniform branch length
 - Uniform branch separation
 - RRT



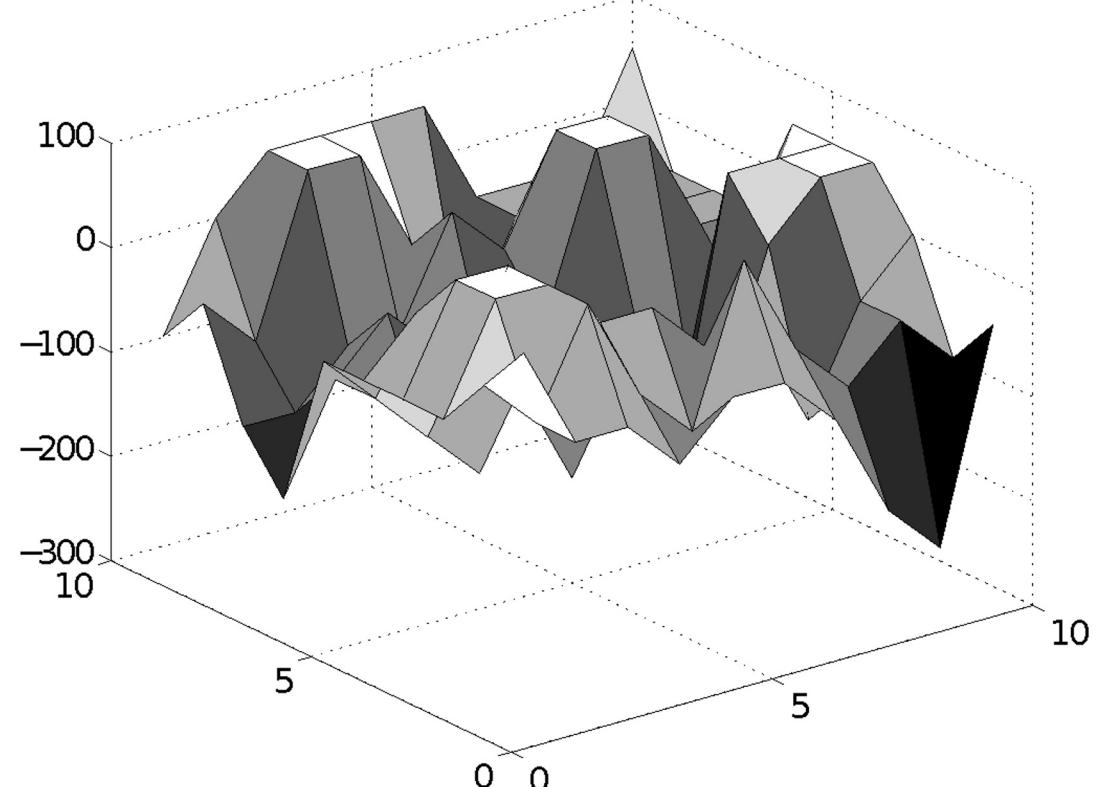
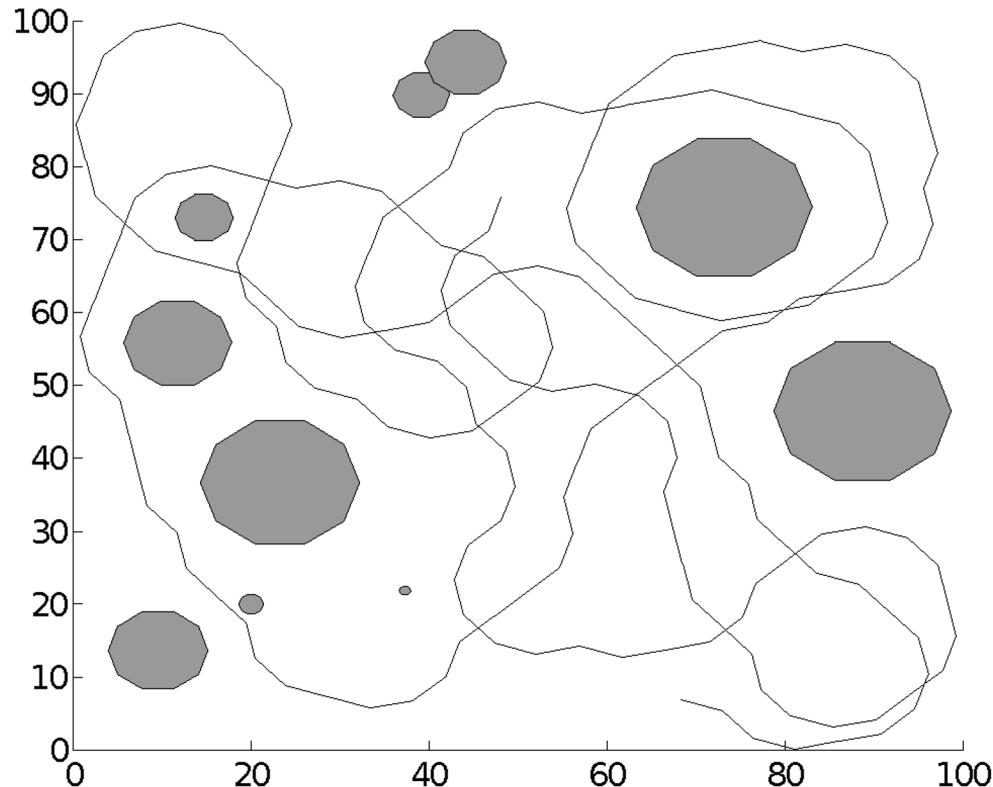
Coverage Algorithm

Algorithm 12 Plan Cover Path: $\text{planCover}(\mathcal{T}, \Upsilon, \mathbf{p})$

Input: Terrain map \mathcal{T} , return map Υ , initial configuration \mathbf{p}_s

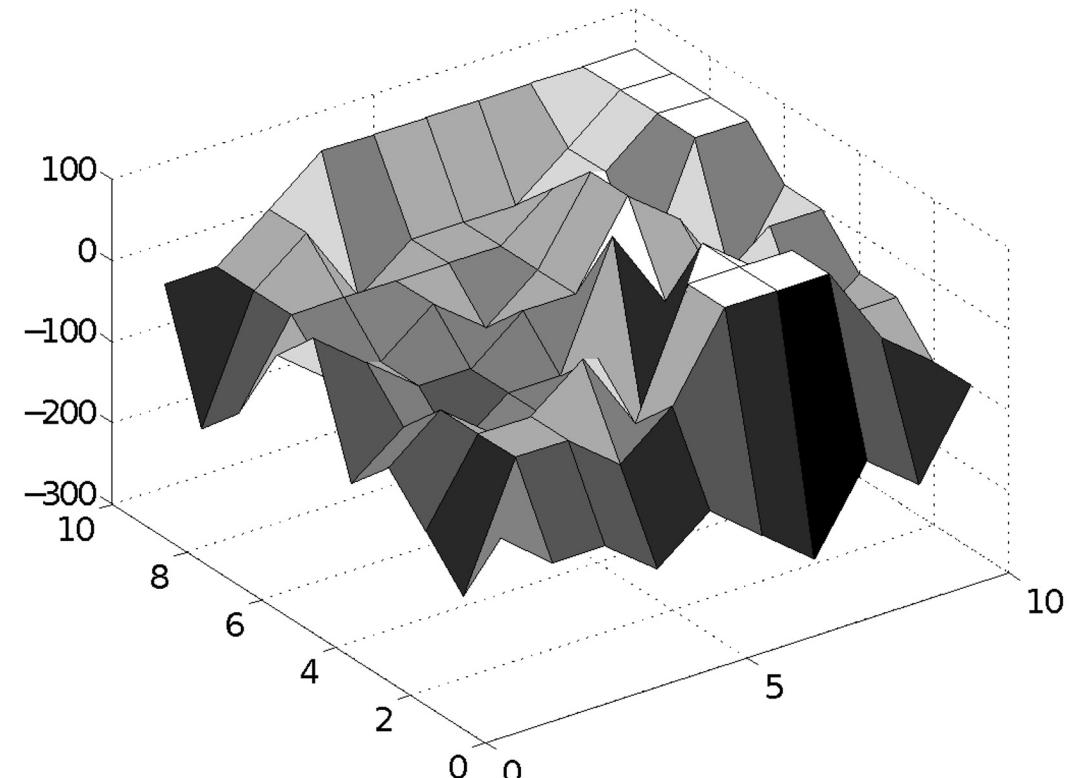
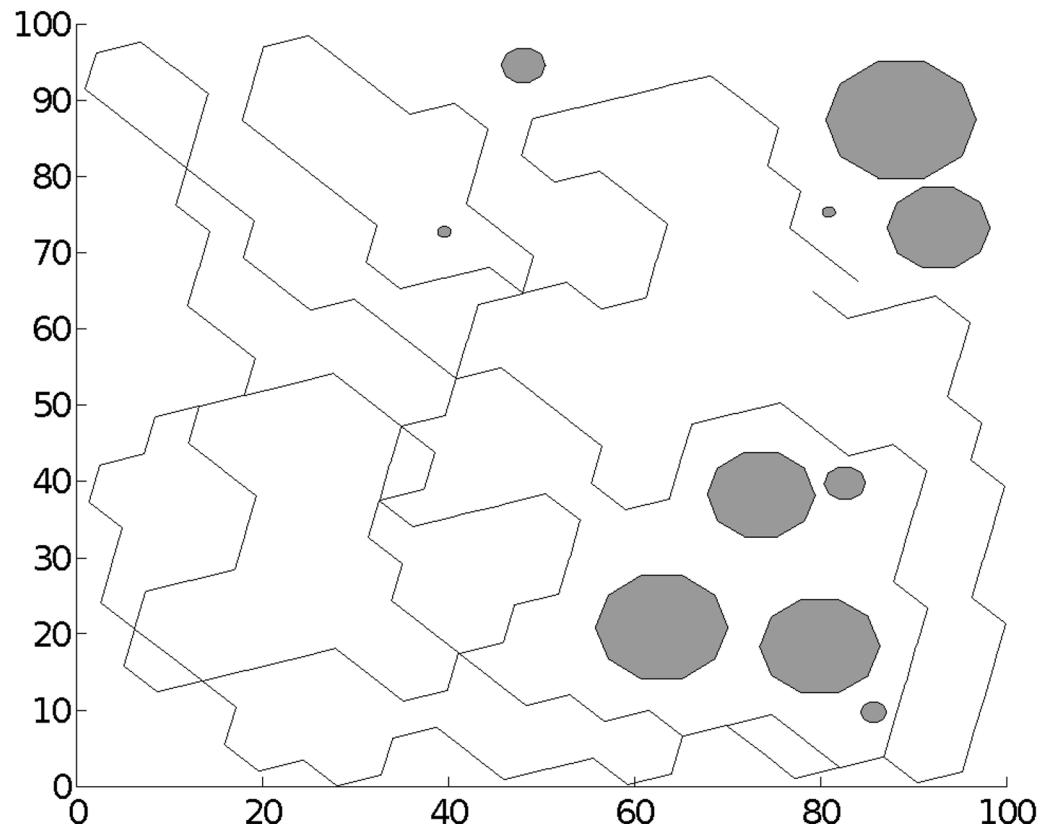
- 1: Initialize look-ahead tree $G = (V, E)$ as $V = \{\mathbf{p}_s\}$, $E = \emptyset$
 - 2: Initialize return map $\Upsilon = \{\Upsilon_i : i \text{ indexes the terrain}\}$
 - 3: $\mathbf{p} = \mathbf{p}_s$
 - 4: **for** Each planning cycle **do**
 - 5: $G = \text{generateTree}(\mathbf{p}, \mathcal{T}, \Upsilon)$
 - 6: $\mathcal{W} = \text{highestReturnPath}(G)$
 - 7: Update \mathbf{p} by moving along the first segment of \mathcal{W}
 - 8: Reset $G = (V, E)$ as $V = \{\mathbf{p}\}$, $E = \emptyset$
 - 9: $\Upsilon = \text{updateReturnMap}(\Upsilon, \mathbf{p})$
 - 10: **end for**
-

Coverage Planning Results – Uniform Tree



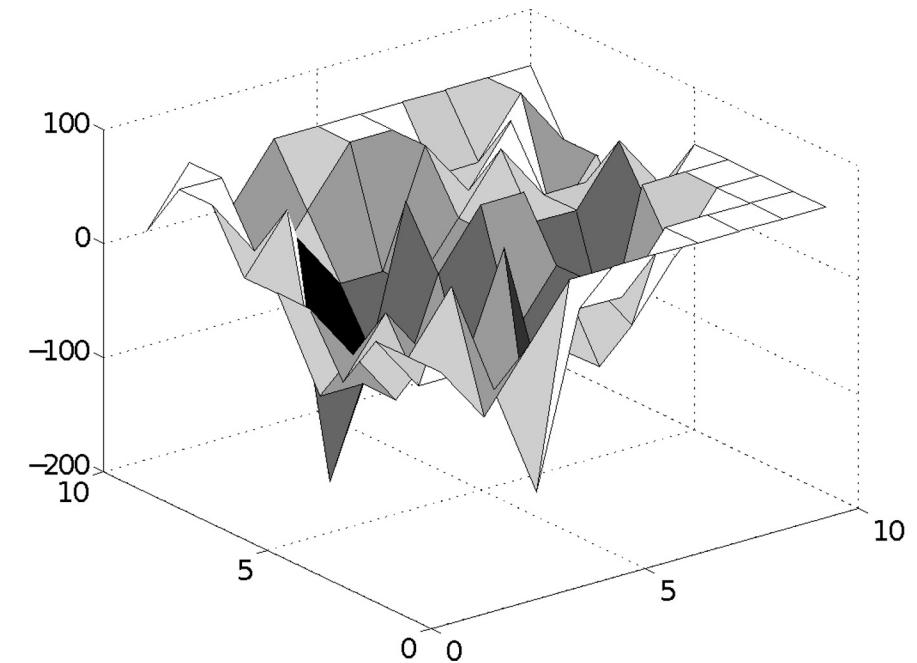
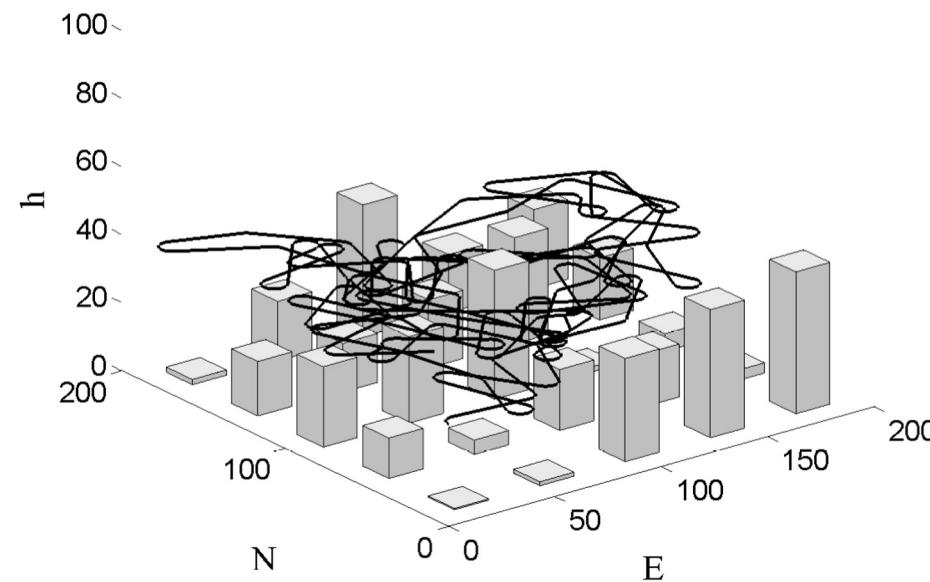
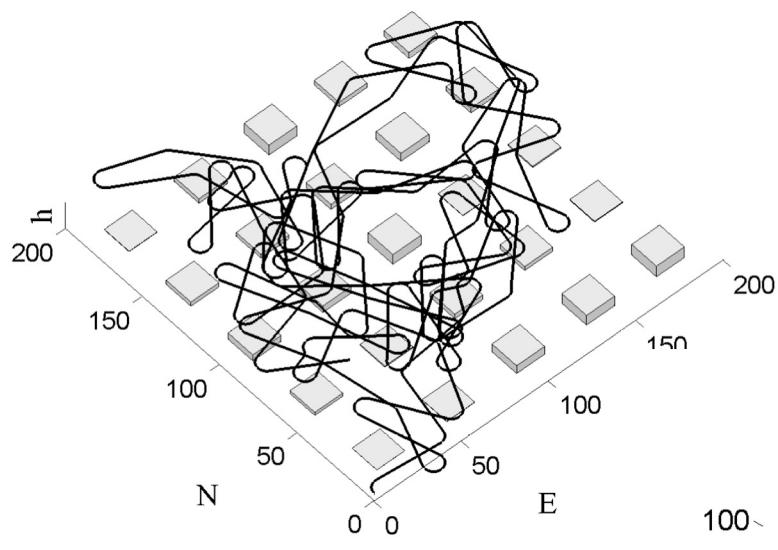
Look ahead length = 5
Heading change = 30 deg
Tree depth = 3
Iterations = 200

Coverage Planning Results – Uniform Tree

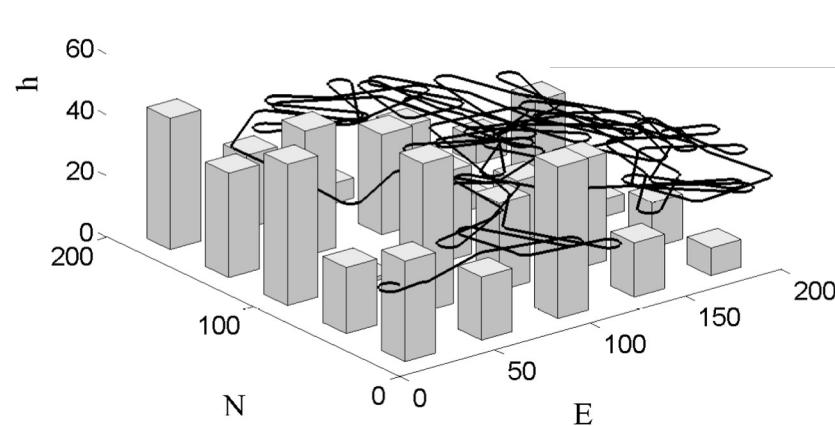
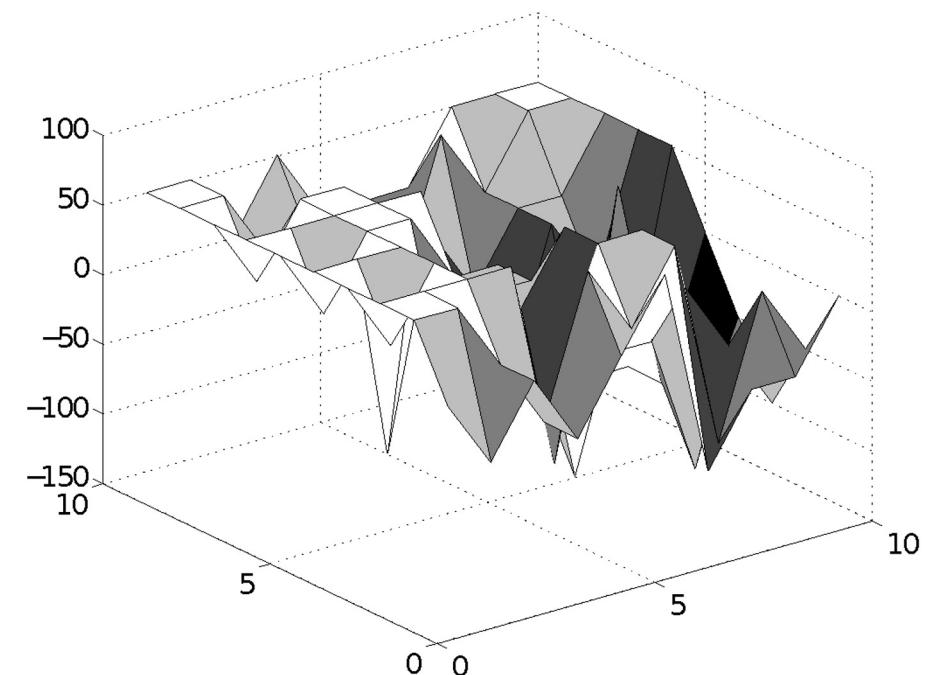
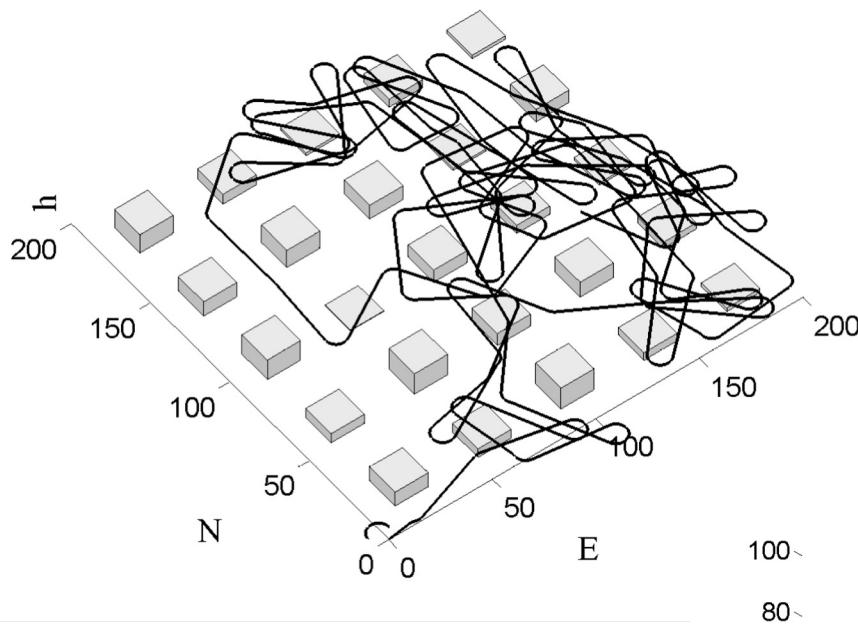


Look ahead length = 5
Heading change = 60 deg
Tree depth = 3
Iterations = 200

Coverage Planning Results – RRT Dubins



Coverage Planning Results – RRT Dubins

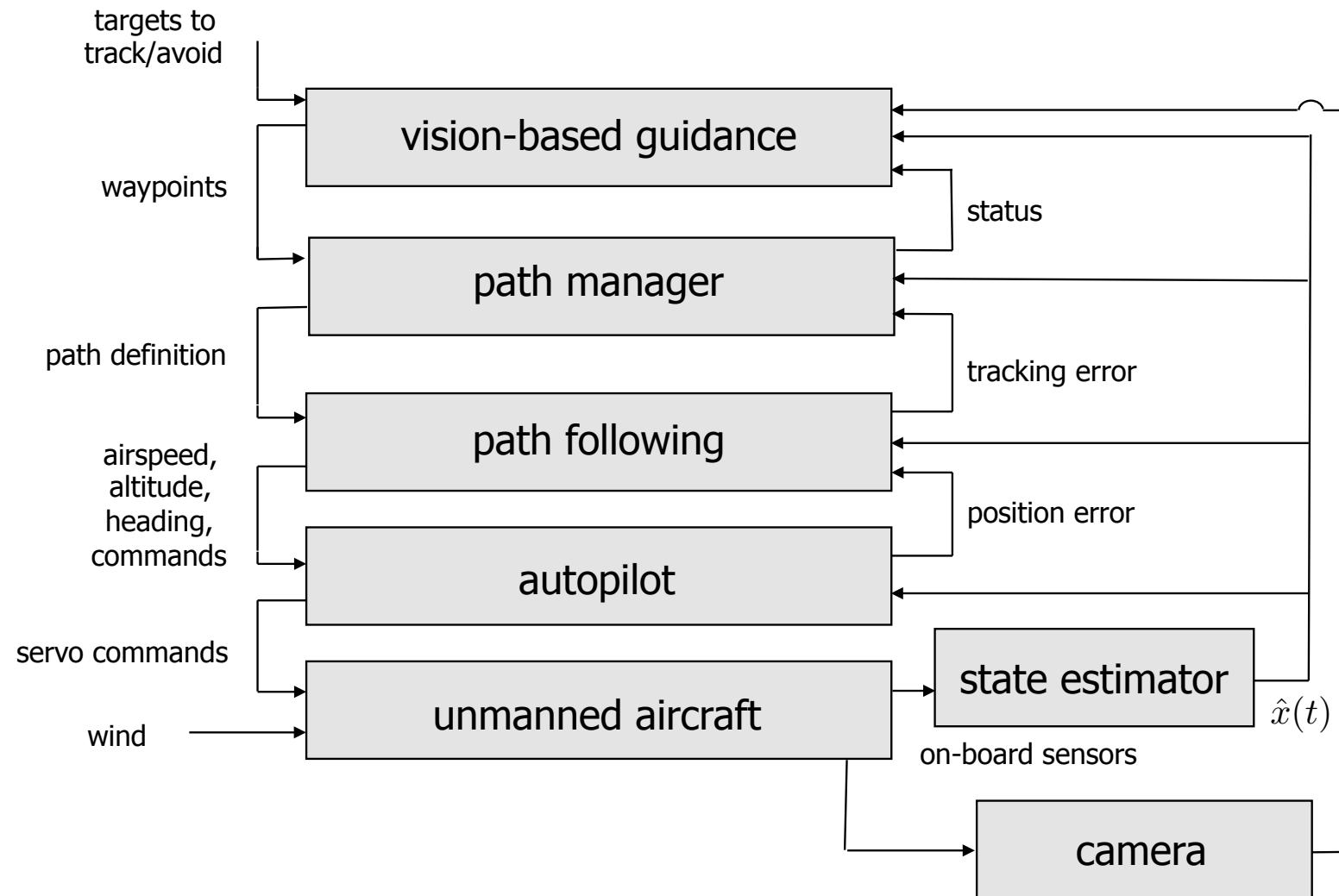




Chapter 13

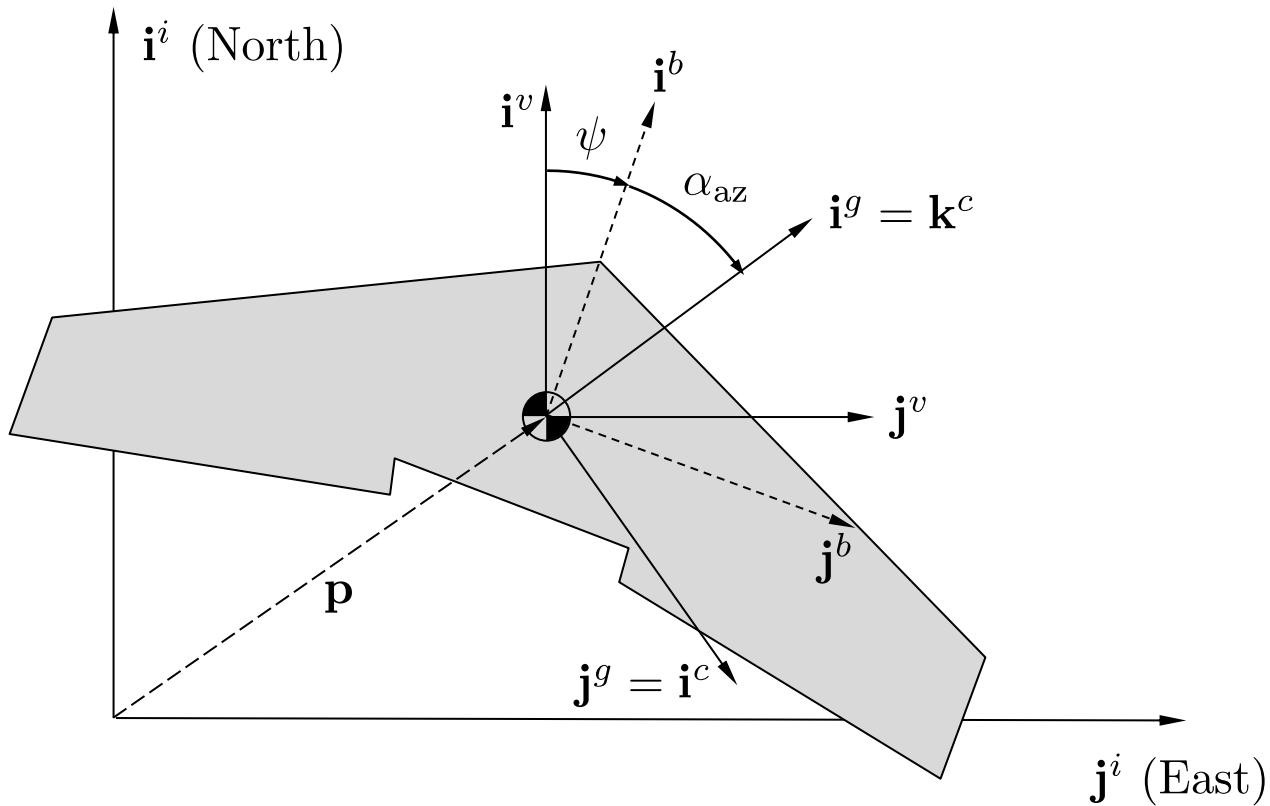
Vision Based Guidance

Architecture w/ Camera



Gimbal and Camera Reference Frames

Gimbal-1 frame: $\mathcal{F}^{g1} = (\mathbf{i}^{g1}, \mathbf{j}^{g1}, \mathbf{k}^{g1})$
Gimbal frame: $\mathcal{F}^g = (\mathbf{i}^g, \mathbf{j}^g, \mathbf{k}^g)$
Camera frame: $\mathcal{F}^c = (\mathbf{i}^c, \mathbf{j}^c, \mathbf{k}^c)$



Gimbal-1 frame:
Rotate body frame about \mathbf{k}^b axis
by gimbal azimuth angle, α_{az}

$$\mathcal{R}_b^{g1}(\alpha_{az}) \triangleq \begin{pmatrix} \cos \alpha_{az} & \sin \alpha_{az} & 0 \\ -\sin \alpha_{az} & \cos \alpha_{az} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

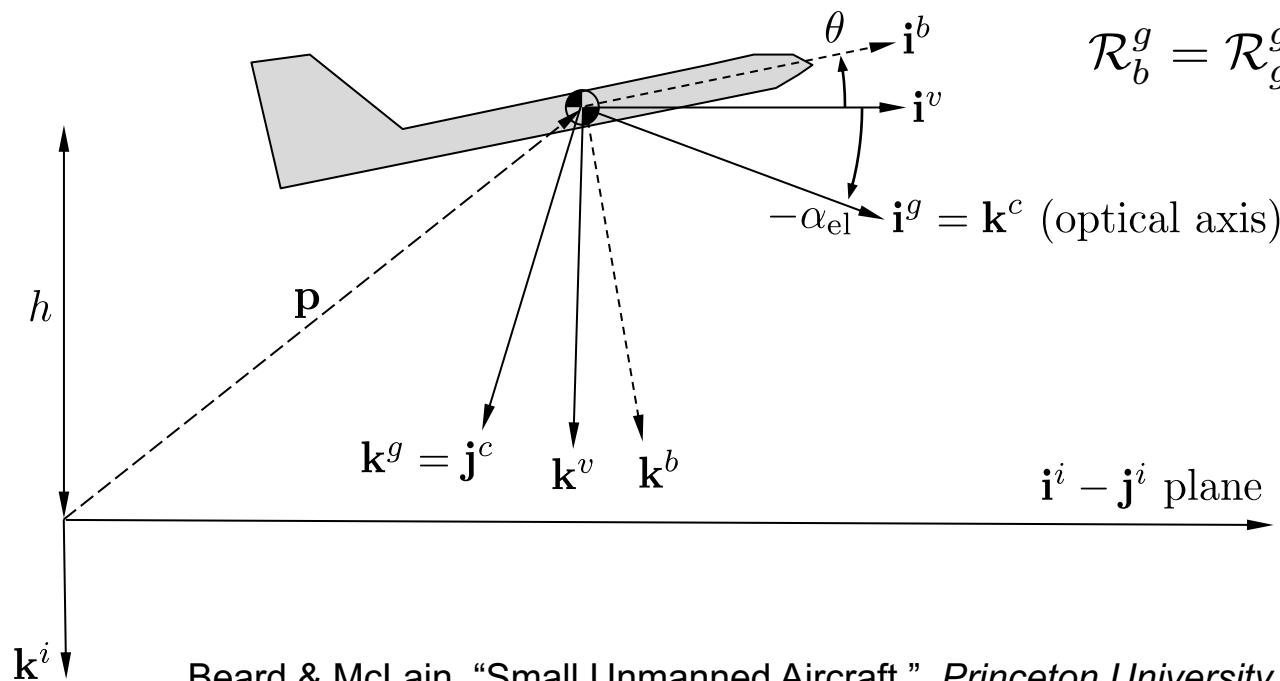
Gimbal Reference Frames

Gimbal frame:

Rotate gimbal-1 frame about \mathbf{j}^{g1} axis
by gimbal elevation angle, α_{el}

$$\mathcal{R}_{g1}^g(\alpha_{el}) \triangleq \begin{pmatrix} \cos \alpha_{el} & 0 & -\sin \alpha_{el} \\ 0 & 1 & 0 \\ \sin \alpha_{el} & 0 & \cos \alpha_{el} \end{pmatrix}$$

$$\mathcal{R}_b^g = \mathcal{R}_{g1}^g \mathcal{R}_b^{g1} = \begin{pmatrix} \cos \alpha_{el} \cos \alpha_{az} & \cos \alpha_{el} \sin \alpha_{az} & -\sin \alpha_{el} \\ -\sin \alpha_{az} & \cos \alpha_{az} & 0 \\ \sin \alpha_{el} \cos \alpha_{az} & \sin \alpha_{el} \sin \alpha_{az} & \cos \alpha_{el} \end{pmatrix}$$



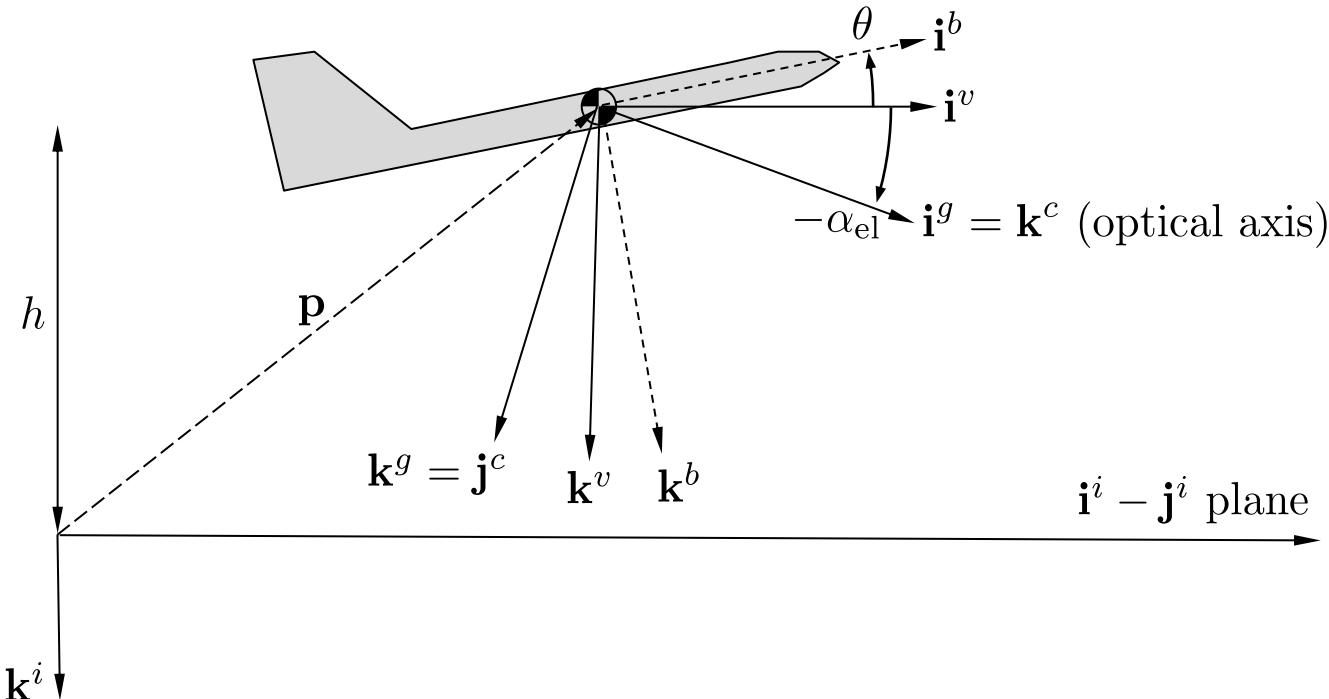
Camera Reference Frame

Camera frame:

\mathbf{i}^c points to the right in the image

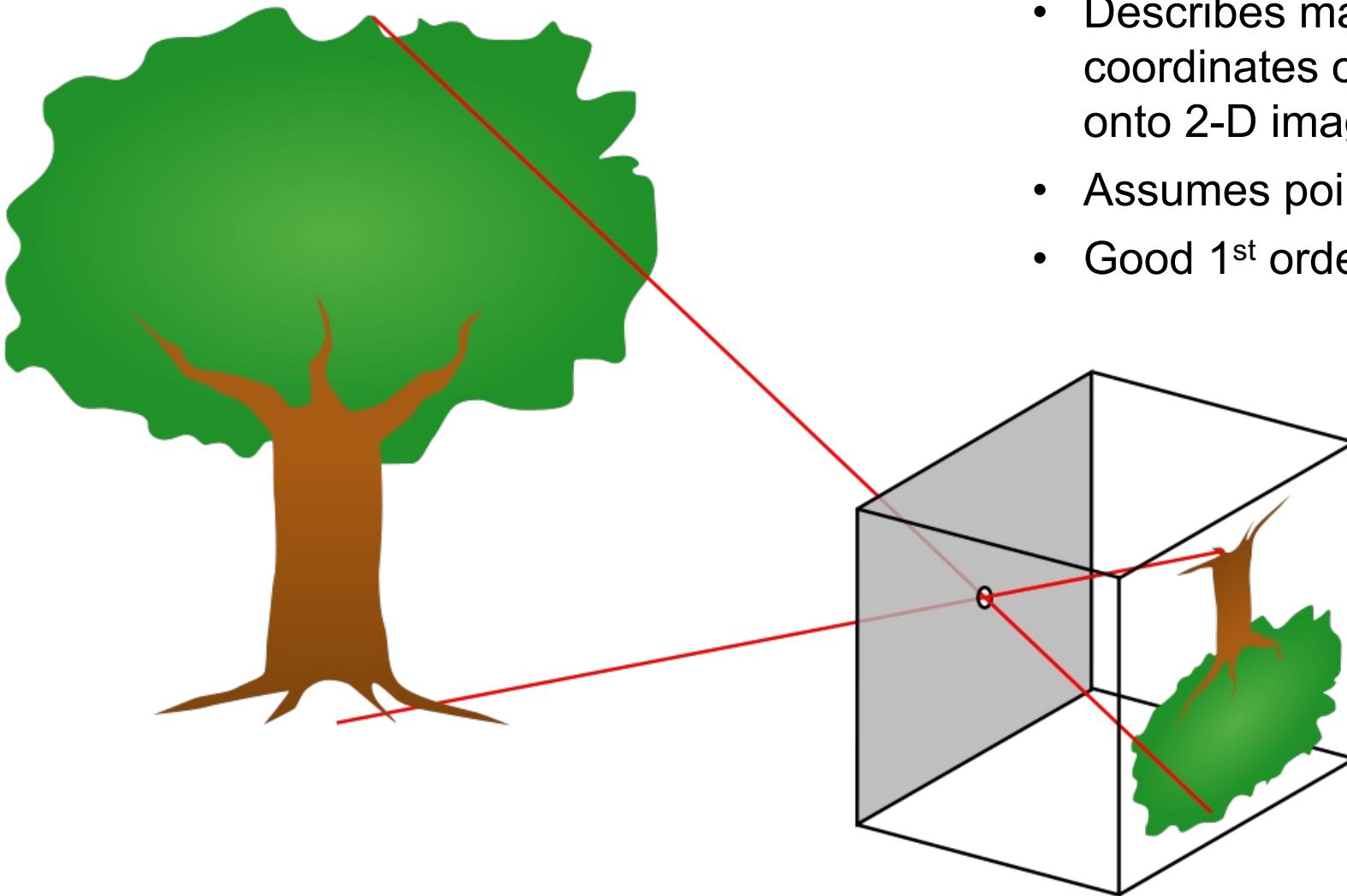
\mathbf{j}^c points down in the image

\mathbf{k}^c points along the optical axis



$$\mathcal{R}_g^c = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

Pinhole Camera Model



- Describes mathematical relationship between coordinates of 3-D point and its projection onto 2-D image plane
- Assumes point aperture, no lenses
- Good 1st order approximation

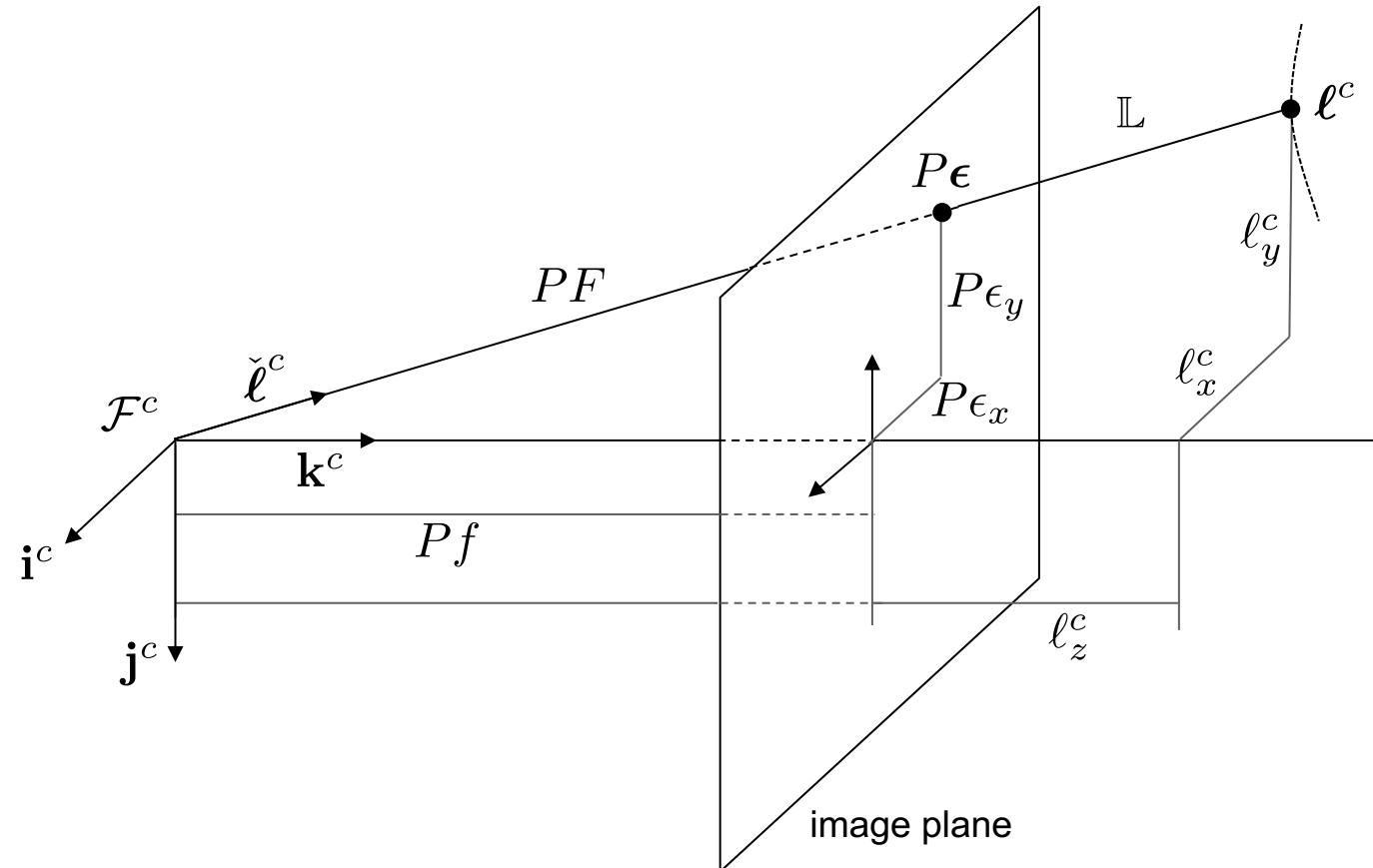
Pinhole Camera Model

Goal:

From pixel location (ϵ_x, ϵ_y) in image plane and camera parameters, determine location of object in world ℓ^c

Approach:

Use similar triangles geometry



Camera Model

f : focal length in pixels

P : converts pixels to meters

M : width of square pixel array

v : field of view

ℓ^c : 3-D location of point

$$f = \frac{M}{2 \tan\left(\frac{v}{2}\right)}$$

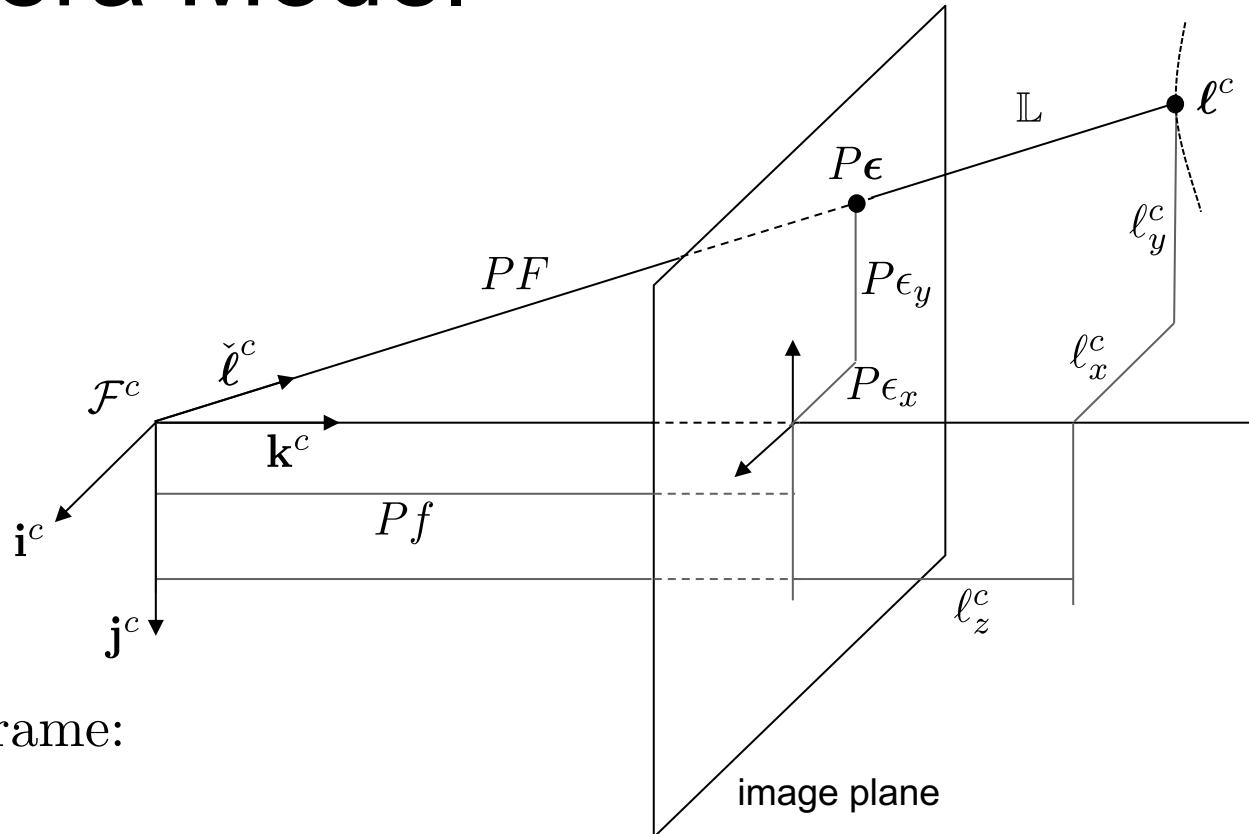
Location of projection of object in camera frame:

$(P\epsilon_x, P\epsilon_y, Pf)$

Pixel location (in pixels): ϵ_x, ϵ_y

Distance from origin of camera frame to object image:

$$F = \sqrt{f^2 + \epsilon_x^2 + \epsilon_y^2}$$



Camera Model

By similar triangles:

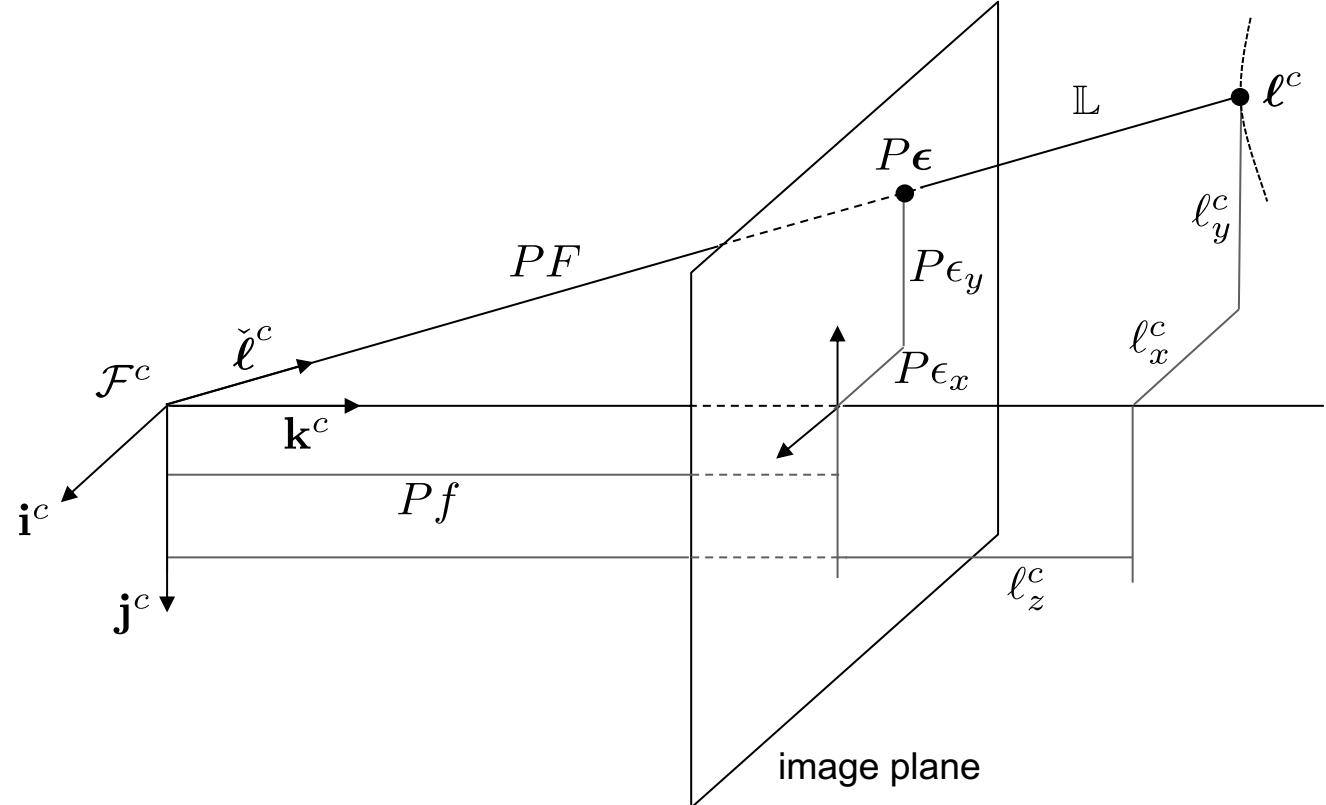
$$\frac{\ell_x^c}{\mathbb{L}} = \frac{P\epsilon_x}{PF} = \frac{\epsilon_x}{F}$$

Similarly:

$$\ell_y^c/\mathbb{L} = \epsilon_y/F \quad \text{and} \quad \ell_z^c/\mathbb{L} = f/F$$

Combining:

$$\boldsymbol{\ell}^c = \begin{pmatrix} \ell_x^c \\ \ell_y^c \\ \ell_z^c \end{pmatrix} = \frac{\mathbb{L}}{F} \begin{pmatrix} \epsilon_x \\ \epsilon_y \\ f \end{pmatrix}$$



Problem: \mathbb{L} is unknown...

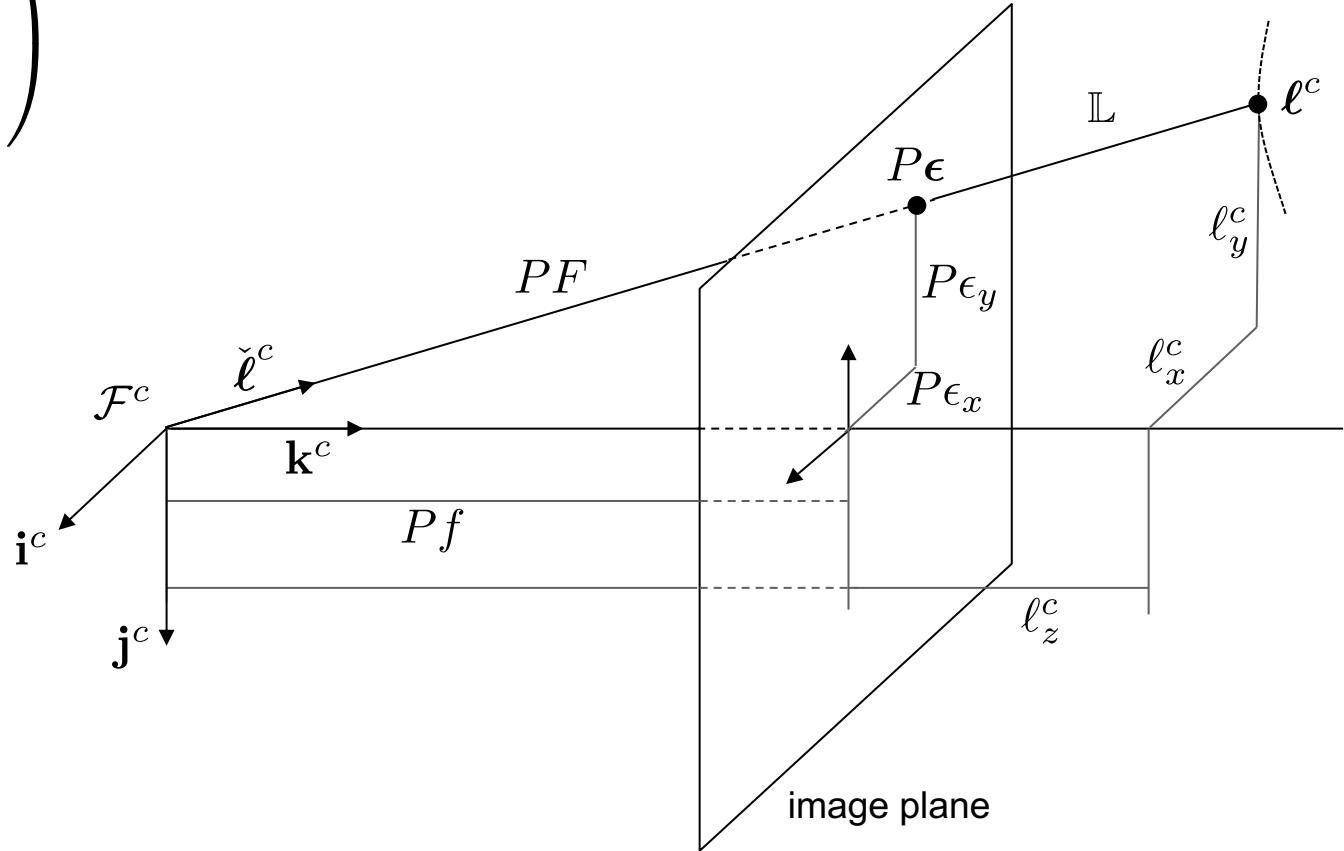
Camera Model

Unit direction vector to target:

$$\frac{\ell^c}{\mathbb{L}} = \frac{1}{F} \begin{pmatrix} \epsilon_x \\ \epsilon_y \\ f \end{pmatrix} = \frac{1}{\sqrt{\epsilon_x^2 + \epsilon_y^2 + f^2}} \begin{pmatrix} \epsilon_x \\ \epsilon_y \\ f \end{pmatrix}$$

Define notation:

$$\check{\ell} \triangleq \begin{pmatrix} \check{\ell}_x \\ \check{\ell}_y \\ \check{\ell}_z \end{pmatrix} \triangleq \frac{\ell}{\mathbb{L}}$$



Gimbal Pointing

- Two scenarios
 - Point gimbal at given world coordinate
 - “Point to this GPS location”
 - Point gimbal so that optical axis aligns with certain point in image plane
 - “Point at this object”
- Gimbal dynamics
 - Assume rate control inputs

$$\dot{\alpha}_{az} = u_{az}$$

$$\dot{\alpha}_{el} = u_{el}$$

Scenario 1: Point Gimbal at World Coordinate

$\mathbf{p}_{\text{obj}}^i$: known location of object in inertial frame (world coordinate)

$\mathbf{p}_{\text{MAV}}^i = (p_n, p_e, p_d)^\top$: location of MAV in inertial frame

Objective:

Align optical axis of camera with desired relative position vector

$$\boldsymbol{\ell}_d^i \triangleq \mathbf{p}_{\text{obj}}^i - \mathbf{p}_{\text{MAV}}^i$$

Body-frame unit vector that points in desired direction of specified world coordinates

$$\check{\boldsymbol{\ell}}_d^b = \frac{1}{\|\boldsymbol{\ell}_d^i\|} \mathcal{R}_i^b \boldsymbol{\ell}_d^i$$

Scenario 2: Point Gimbal at Object in Image

Objective:

Maneuver gimbal so that object at pixel location ϵ is pushed to center of image

Desired direction of optical axis:

$$\check{\ell}_d^c = \frac{1}{\sqrt{f^2 + \epsilon_x^2 + \epsilon_y^2}} \begin{pmatrix} \epsilon_x \\ \epsilon_y \\ f \end{pmatrix}$$

Desired direction of optical axis expressed in body frame:

$$\check{\ell}_d^b = \mathcal{R}_g^b \mathcal{R}_c^g \check{\ell}_d^c$$

Gimbal Pointing Angles

We know direction to point gimbal

What are corresponding azimuth and elevation angles?

Optical axis in camera frame = $(0, 0, 1)^\top$

Objective: Select commanded gimbal angles α_{az}^c and α_{el}^c so that

$$\begin{aligned}\check{\ell}_d^b &\triangleq \begin{pmatrix} \check{\ell}_{xd}^b \\ \check{\ell}_{yd}^b \\ \check{\ell}_{zd}^b \end{pmatrix} = \mathcal{R}_g^b(\alpha_{\text{az}}^c, \alpha_{\text{el}}^c) \mathcal{R}_c^g \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \\ &= \begin{pmatrix} \cos \alpha_{\text{el}}^c \cos \alpha_{\text{az}}^c & -\sin \alpha_{\text{az}}^c & \sin \alpha_{\text{el}}^c \cos \alpha_{\text{az}}^c \\ \cos \alpha_{\text{el}}^c \sin \alpha_{\text{az}}^c & \cos \alpha_{\text{az}}^c & \sin \alpha_{\text{el}}^c \sin \alpha_{\text{az}}^c \\ -\sin \alpha_{\text{el}}^c & 0 & \cos \alpha_{\text{el}}^c \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} \cos \alpha_{\text{el}}^c \cos \alpha_{\text{az}}^c \\ \cos \alpha_{\text{el}}^c \sin \alpha_{\text{az}}^c \\ -\sin \alpha_{\text{el}}^c \end{pmatrix}\end{aligned}$$

Gimbal Pointing Angles

Objective: Select commanded gimbal angles α_{az}^c and α_{el}^c so that

$$\begin{pmatrix} \check{\ell}_{xd}^b \\ \check{\ell}_{yd}^b \\ \check{\ell}_{zd}^b \end{pmatrix} = \begin{pmatrix} \cos \alpha_{\text{el}}^c \cos \alpha_{\text{az}}^c \\ \cos \alpha_{\text{el}}^c \sin \alpha_{\text{az}}^c \\ -\sin \alpha_{\text{el}}^c \end{pmatrix}$$

Solving for α_{el}^c and α_{az}^c gives desired azimuth and elevation angles:

$$\alpha_{\text{az}}^c = \tan^{-1} \left(\frac{\check{\ell}_{yd}^b}{\check{\ell}_{xd}^b} \right)$$
$$\alpha_{\text{el}}^c = -\sin^{-1} (\check{\ell}_{zd}^b)$$

Gimbal servo commands can be selected as:

$$u_{\text{az}} = k_{\text{az}} (\alpha_{\text{az}}^c - \alpha_{\text{az}})$$
$$u_{\text{el}} = k_{\text{el}} (\alpha_{\text{el}}^c - \alpha_{\text{el}})$$

k_{az} and k_{el} are positive control gains

Geolocation

Relative position vector between target and MAV: $\ell = \mathbf{p}_{\text{obj}} - \mathbf{p}_{\text{MAV}}$

Define:

$$\mathbb{L} = \|\ell\| \text{ (range to target)}$$

$$\check{\ell} = \ell / \mathbb{L} \text{ (unit vector pointing from aircraft to target)}$$

From geometry:

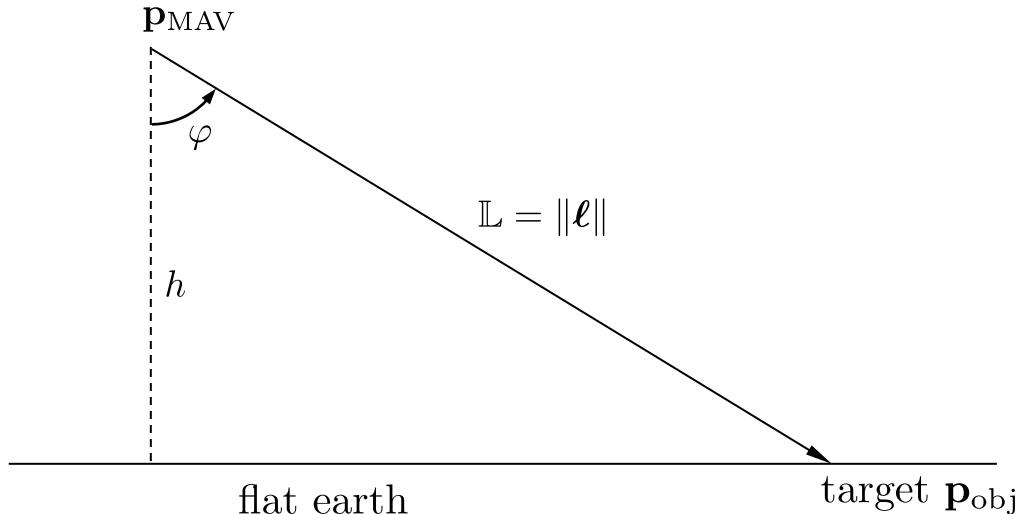
$$\begin{aligned}\mathbf{p}_{\text{obj}}^i &= \mathbf{p}_{\text{MAV}}^i + \mathcal{R}_b^i \mathcal{R}_g^b \mathcal{R}_c^g \check{\ell}^c \\ &= \mathbf{p}_{\text{MAV}}^i + \mathbb{L} (\mathcal{R}_b^i \mathcal{R}_g^b \mathcal{R}_c^g \check{\ell}^c)\end{aligned}$$

where

$$\mathbf{p}_{\text{MAV}}^i = (p_n, p_e, p_d)^\top$$

$$\mathcal{R}_b^i = \mathcal{R}_b^i(\phi, \theta, \psi)$$

$$\mathcal{R}_g^b = \mathcal{R}_g^b(\alpha_{\text{az}}, \alpha_{\text{el}})$$



\mathbb{L} is unknown \rightarrow solving geolocation problem reduces to estimating range to target \mathbb{L}

Range to Target – Flat Earth Model

From geometry: $\cos \varphi = \frac{h}{\mathbb{L}}$

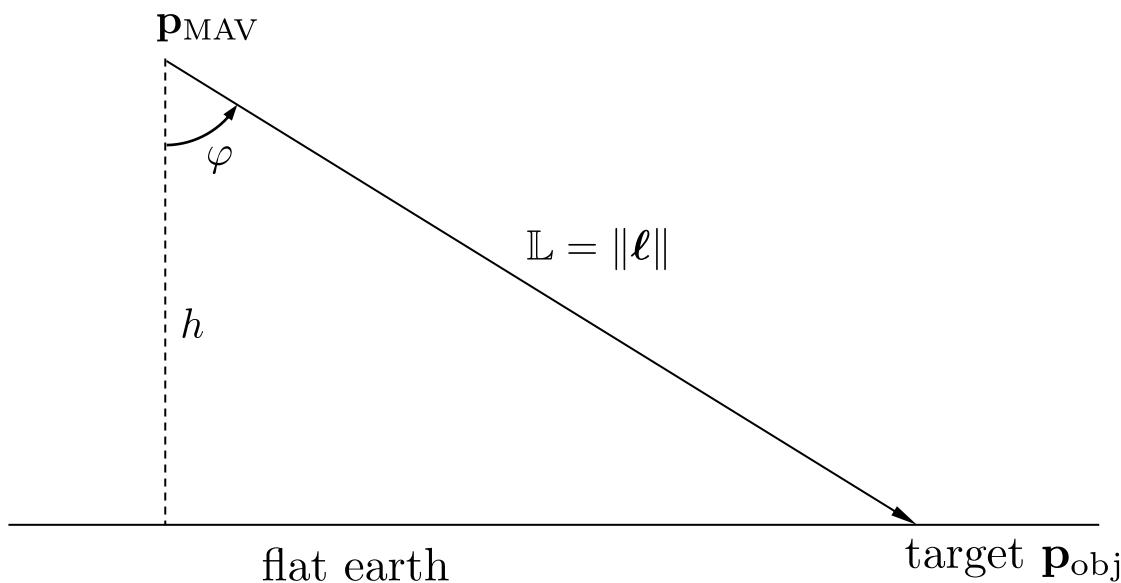
From Cauchy-Schwartz equality: $\cos \varphi = \mathbf{k}^i \cdot \check{\ell}^i = \mathbf{k}^i \cdot \mathcal{R}_b^i \mathcal{R}_g^b \mathcal{R}_c^g \check{\ell}^c$

Equating: $\mathbb{L} = \frac{h}{\mathbf{k}^i \cdot \mathcal{R}_b^i \mathcal{R}_g^b \mathcal{R}_c^g \check{\ell}^c}$

From before: $\mathbf{p}_{\text{obj}}^i = \mathbf{p}_{\text{MAV}}^i + \mathbb{L} (\mathcal{R}_b^i \mathcal{R}_g^b \mathcal{R}_c^g \check{\ell}^c)$

Therefore:

$$\mathbf{p}_{\text{obj}}^i = \begin{pmatrix} p_n \\ p_e \\ p_d \end{pmatrix} + h \frac{\mathcal{R}_b^i \mathcal{R}_g^b \mathcal{R}_c^g \check{\ell}^c}{\mathbf{k}^i \cdot \mathcal{R}_b^i \mathcal{R}_g^b \mathcal{R}_c^g \check{\ell}^c}$$



Geolocation Errors

The position of the object is

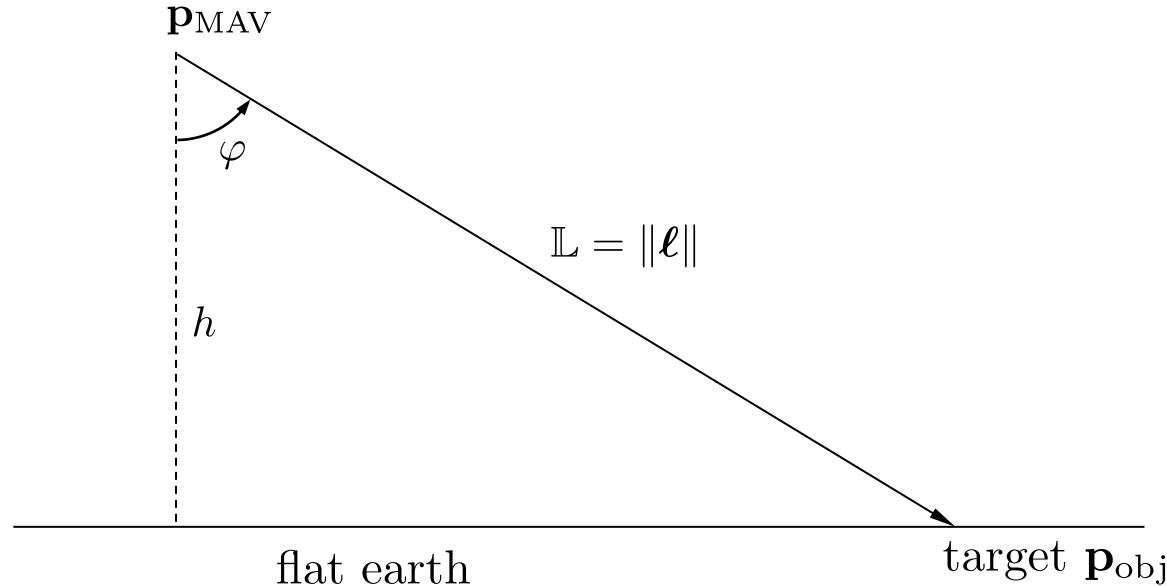
$$\mathbf{p}_{\text{obj}}^i = \begin{pmatrix} p_n \\ p_e \\ p_d \end{pmatrix} + h \frac{\mathcal{R}_b^i \mathcal{R}_g^b \mathcal{R}_c^g \check{\ell}^c}{\mathbf{k}^i \cdot \mathcal{R}_b^i \mathcal{R}_g^b \mathcal{R}_c^g \check{\ell}}$$

Equation is highly sensitive to measurement errors

- attitude estimation errors
- gimbal pointing angle errors
- gimbal/autopilot alignment errors

Two types of errors

- bias errors (address with calibration)
- random errors (address with EKF)



Geolocation Using EKF

The states are $x = (\mathbf{p}_{\text{obj}}^{i\top}, \mathbb{L})^\top$.

Need propagation equations $\dot{x} = f(x, u)$.

Assuming the object is stationary: $\dot{\mathbf{p}}_{\text{obj}}^i = 0$

Also:

$$\dot{\mathbb{L}} = \frac{d}{dt} \sqrt{(\mathbf{p}_{\text{obj}}^i - \mathbf{p}_{\text{MAV}}^i)^\top (\mathbf{p}_{\text{obj}}^i - \mathbf{p}_{\text{MAV}}^i)} = \frac{(\mathbf{p}_{\text{obj}}^i - \mathbf{p}_{\text{MAV}}^i)^\top (\dot{\mathbf{p}}_{\text{obj}}^i - \dot{\mathbf{p}}_{\text{MAV}}^i)}{\mathbb{L}} = -\frac{(\mathbf{p}_{\text{obj}}^i - \mathbf{p}_{\text{MAV}}^i)^\top \dot{\mathbf{p}}_{\text{MAV}}^i}{\mathbb{L}}$$

where: $\mathbf{p}_{\text{MAV}}^i = \mathbf{p}_{\text{obj}}^i - \mathbb{L} \left(\mathcal{R}_b^i \mathcal{R}_g^b \mathcal{R}_c^g \check{\ell}^c \right)$

and for constant-altitude flight: $\dot{\mathbf{p}}_{\text{MAV}}^i = \begin{pmatrix} \hat{V}_g \cos \hat{\chi} \\ \hat{V}_g \sin \hat{\chi} \\ 0 \end{pmatrix}$

Geolocation Using EKF

Prediction equations:

$$\begin{pmatrix} \dot{\hat{\mathbf{p}}}_{\text{obj}}^i \\ \dot{\hat{\mathbb{L}}} \end{pmatrix} = \begin{pmatrix} 0 \\ -\frac{(\hat{\mathbf{p}}_{\text{obj}}^i - \hat{\mathbf{p}}_{\text{MAV}}^i)^\top \dot{\hat{\mathbf{p}}}_{\text{MAV}}^i}{\hat{\mathbb{L}}} \end{pmatrix}$$

Jacobian of prediction equation:

$$\frac{\partial f}{\partial x} = \begin{pmatrix} 0 & 0 \\ -\frac{\dot{\hat{\mathbf{p}}}_{\text{MAV}}^{iT}}{\hat{\mathbb{L}}} & \frac{(\hat{\mathbf{p}}_{\text{obj}}^i - \hat{\mathbf{p}}_{\text{MAV}}^i)^\top \dot{\hat{\mathbf{p}}}_{\text{MAV}}^i}{\hat{\mathbb{L}}^2} \end{pmatrix}$$

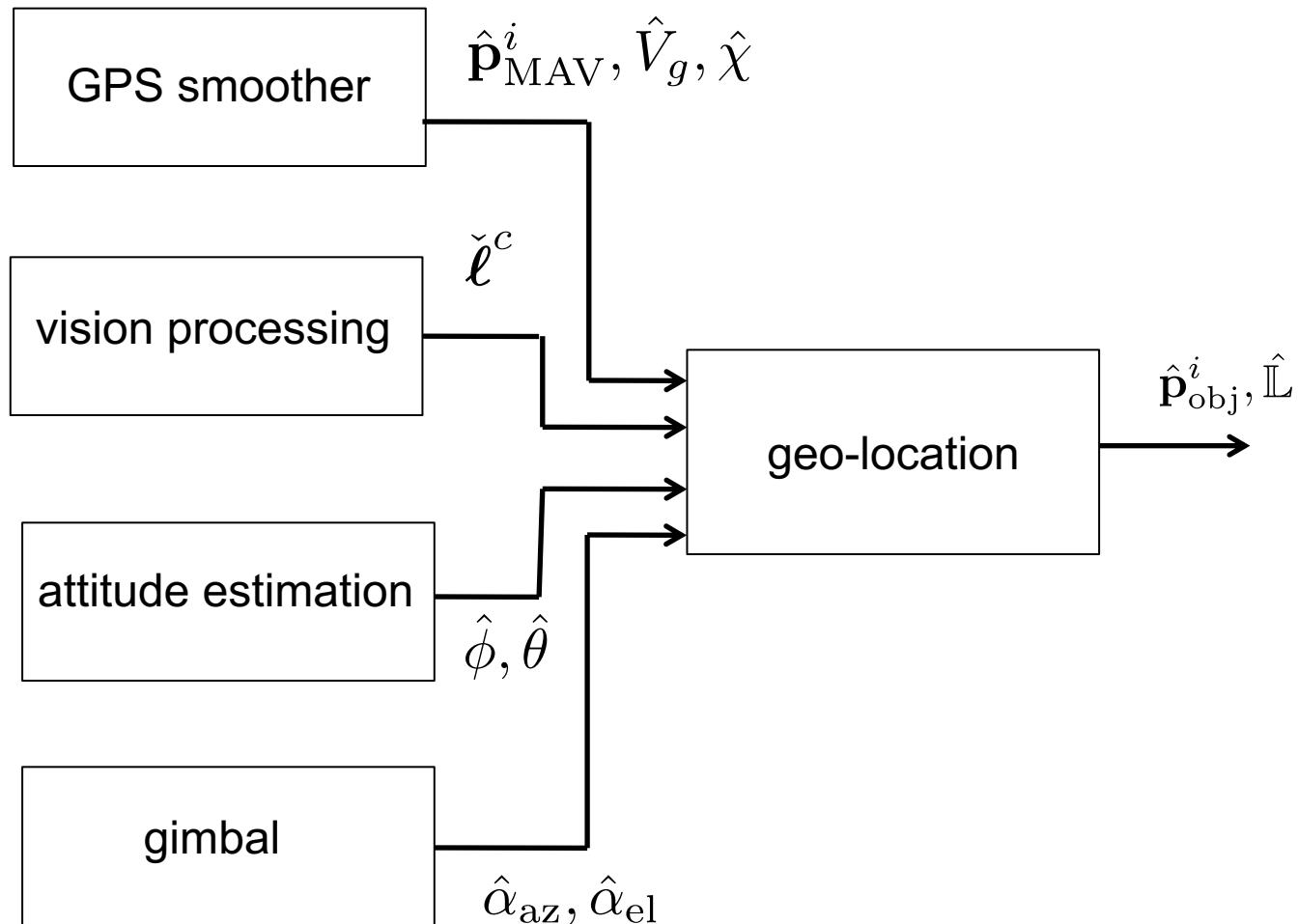
Measurement equation:

$$\mathbf{p}_{\text{MAV}}^i = \mathbf{p}_{\text{obj}}^i - \mathbb{L} \left(\mathcal{R}_b^i \mathcal{R}_g^b \mathcal{R}_c^g \check{\ell}^c \right)$$

Jacobian of measurement equation:

$$\frac{\partial h}{\partial x} = (I \quad \mathcal{R}_b^i \mathcal{R}_g^b \mathcal{R}_c^g \check{\ell}^c)$$

Geolocation Architecture



Target Motion in Image Plane

- The objective is to track targets seen in the image plane.
- Feature tracking, or other computer vision algorithms, are used to track pixels in image plane.
- Feature tracking is noisy, and so pixel location must be low pass filtered.
- Motion of the target in the image plane is due to two sources:
 - Self motion, or ego motion. Primarily due to rotation of the MAV.
 - Relative target motion.
- Since we are primarily interested in the relative motion, we must subtract the pixel movement due to ego motion.

Pixel LPF and Differentiation

Define the following

$$\bar{\epsilon} = (\bar{\epsilon}_x, \bar{\epsilon}_y)^\top, \quad \text{Raw pixel measurements,}$$

$$\epsilon = (\epsilon_x, \epsilon_y)^\top, \quad \text{Filtered pixel location,}$$

$$\dot{\epsilon} = (\dot{\epsilon}_x, \dot{\epsilon}_y)^\top, \quad \text{Filtered pixel velocity}$$

Basic idea: LPF $\bar{\epsilon}$ to obtain ϵ , and use dirty derivative to obtain $\dot{\epsilon}$:

$$\epsilon(s) = \frac{1}{\tau s + 1} \bar{\epsilon}, \quad \dot{\epsilon}(s) = \frac{s}{\tau s + 1} \bar{\epsilon}.$$

Digital Approximation

Tustin approximation

$$s \mapsto \frac{2}{T_s} \frac{z-1}{z+1},$$

Converting to the z -domain gives

$$\begin{aligned}\epsilon[z] &= \frac{1}{\frac{2\tau}{T_s} \frac{z-1}{z+1} + 1} \bar{\epsilon} = \frac{\frac{T_s}{2\tau+T_s}(z+1)}{z - \frac{2\tau-T_s}{2\tau+T_s}} \bar{\epsilon} \\ \dot{\epsilon}[z] &= \frac{\frac{2}{T_s} \frac{z-1}{z+1}}{\frac{2\tau}{T_s} \frac{z-1}{z+1} + 1} \bar{\epsilon} = \frac{\frac{2}{2\tau+T_s}(z-1)}{z - \frac{2\tau-T_s}{2\tau+T_s}} \bar{\epsilon}.\end{aligned}$$

Taking the inverse z -transform gives the difference equations

$$\begin{aligned}\epsilon[n] &= \left(\frac{2\tau - T_s}{2\tau + T_s} \right) \epsilon[n-1] + \left(\frac{T_s}{2\tau + T_s} \right) (\bar{\epsilon}[n] + \bar{\epsilon}[n-1]) \\ \dot{\epsilon}[n] &= \left(\frac{2\tau - T_s}{2\tau + T_s} \right) \dot{\epsilon}[n-1] + \left(\frac{2}{2\tau + T_s} \right) (\bar{\epsilon}[n] - \bar{\epsilon}[n-1]),\end{aligned}$$

where $\epsilon[0] = \bar{\epsilon}[0]$ and $\dot{\epsilon}[0] = 0$.

Digital Approximation

Define $\beta = (2\tau - T_s)/(2\tau + T_s)$, and note that $1 - \beta = 2T_s/(2\tau + T_s)$. Then

$$\epsilon[n] = \beta \epsilon[n-1] + (1 - \beta) \underbrace{\left(\frac{\bar{\epsilon}[n] + \bar{\epsilon}[n-1]}{2} \right)}_{\text{Average of last two samples}}$$
$$\dot{\epsilon}[n] = \beta \dot{\epsilon}[n-1] + (1 - \beta) \underbrace{\left(\frac{\bar{\epsilon}[n] - \bar{\epsilon}[n-1]}{T_s} \right)}_{\text{Euler approximation of derivative}},$$

where $\epsilon[0] = \bar{\epsilon}[0]$ and $\dot{\epsilon}[0] = 0$.

Apparent Motion

Let $\check{\ell} \triangleq \ell/\mathbb{L} = (\mathbf{p}_{\text{obj}} - \mathbf{p}_{\text{MAV}})/\|\mathbf{p}_{\text{obj}} - \mathbf{p}_{\text{MAV}}\|$ be the normalized relative position vector.

The Coriolis formula (expressed in camera frame) gives

$$\frac{d\check{\ell}^c}{dt_i} = \frac{d\check{\ell}^c}{dt_c} + \boldsymbol{\omega}_{c/i}^c \times \check{\ell}^c.$$

True relative
translational motion
between target and MAV

Motion of target
in image plane

Apparent motion
due to rotation of
MAV and gimbal

Apparent Motion, cont.

Motion of target in image plane:

$$\begin{aligned}\frac{d\check{\ell}^c}{dt_c} &= \frac{d}{dt_c} \frac{\begin{pmatrix} \epsilon_x \\ \epsilon_y \\ f \end{pmatrix}}{F} = \frac{F \begin{pmatrix} \dot{\epsilon}_x \\ \dot{\epsilon}_y \\ 0 \end{pmatrix} - \dot{F} \begin{pmatrix} \epsilon_x \\ \epsilon_y \\ f \end{pmatrix}}{F^2} = \frac{F \begin{pmatrix} \dot{\epsilon}_x \\ \dot{\epsilon}_y \\ 0 \end{pmatrix} - \frac{\epsilon_x \dot{\epsilon}_x + \epsilon_y \dot{\epsilon}_y}{F} \begin{pmatrix} \epsilon_x \\ \epsilon_y \\ f \end{pmatrix}}{F^2} \\ &= \frac{1}{F^3} \begin{pmatrix} F^2 - \epsilon_x^2 & -\epsilon_x \epsilon_y & \\ -\epsilon_x \epsilon_y & F^2 - \epsilon_y^2 & \\ -\epsilon_x f & -\epsilon_y f & \end{pmatrix} \begin{pmatrix} \dot{\epsilon}_x \\ \dot{\epsilon}_y \\ \end{pmatrix} = \frac{1}{F^3} \begin{pmatrix} \epsilon_y^2 + f^2 & -\epsilon_x \epsilon_y & \\ -\epsilon_x \epsilon_y & \epsilon_x^2 + f^2 & \\ -\epsilon_x f & -\epsilon_y f & \end{pmatrix} \dot{\epsilon} \\ &= Z(\epsilon) \dot{\epsilon},\end{aligned}$$

where

$$Z(\epsilon) \triangleq \frac{1}{F^3} \begin{pmatrix} \epsilon_y^2 + f^2 & -\epsilon_x \epsilon_y & \\ -\epsilon_x \epsilon_y & \epsilon_x^2 + f^2 & \\ -\epsilon_x f & -\epsilon_y f & \end{pmatrix} \dot{\epsilon}.$$

Apparent Motion, cont.

Ego Motion, $\omega_{c/i}^c \times \dot{\ell}^c$:

$$\omega_{c/i}^c = \underbrace{\omega_{c/g}^c}_0 + \underbrace{\omega_{g/b}^c}_{\mathcal{R}_g^c \mathcal{R}_b^g \begin{pmatrix} -\sin(\alpha_{az})\dot{\alpha}_{el} \\ \cos(\alpha_{az})\dot{\alpha}_{el} \\ \dot{\alpha}_{az} \end{pmatrix}} + \underbrace{\omega_{b/i}^c}_{\begin{pmatrix} p \\ q \\ r \end{pmatrix}}.$$

Therefore

$$\dot{\tilde{\ell}}_{app}^c \triangleq \omega_{c/i}^c \times \dot{\ell}^c = \frac{1}{F} \left[\mathcal{R}_g^c \mathcal{R}_b^g \begin{pmatrix} p - \sin(\alpha_{az})\dot{\alpha}_{el} \\ q + \cos(\alpha_{az})\dot{\alpha}_{el} \\ r + \dot{\alpha}_{az} \end{pmatrix} \right] \times \begin{pmatrix} \epsilon_x \\ \epsilon_y \\ f \end{pmatrix}$$

Total Motion in Image Plane

$$\frac{d\check{\ell}^c}{dt_i} = \underbrace{Z(\epsilon)\dot{\epsilon}}_{\text{Target Motion}} + \underbrace{\frac{1}{F} \left[\mathcal{R}_g^c \mathcal{R}_b^g \begin{pmatrix} p - \sin(\alpha_{az})\dot{\alpha}_{el} \\ q + \cos(\alpha_{az})\dot{\alpha}_{el} \\ r + \dot{\alpha}_{az} \end{pmatrix} \right]}_{\text{Ego Motion}} \times \begin{pmatrix} \epsilon_x \\ \epsilon_y \\ f \end{pmatrix}$$

Time to Collision

For all objects in the camera field of view, it is possible to compute the time to collision to that obstacle, defined as:

The time to collision if the vehicle were to proceed along the line-of-sight vector to the obstacle at the current closing velocity.

Therefore, for each obstacle, the time to collision is given by

$$t_c \triangleq -\frac{\mathbb{L}}{\dot{\mathbb{L}}}$$

Impossible to compute using only a monocular camera because of scale ambiguity.

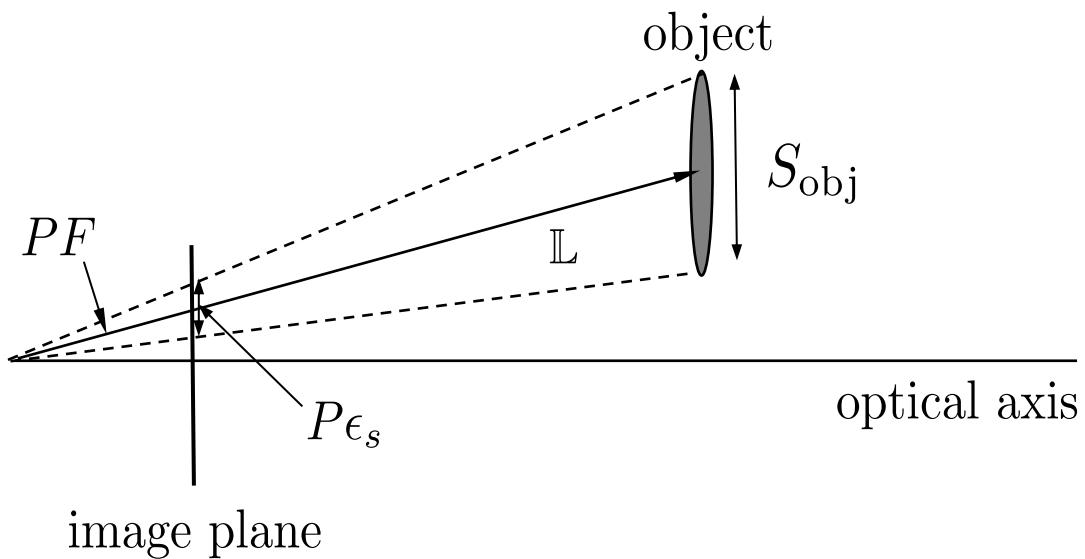
Two techniques:

- Looming ← Requires target size.
- Flat earth approximation ← Requires height above ground.

Time to Collision - Looming

From similar triangles:

$$\frac{S_{\text{obj}}}{\mathbb{L}} = \frac{P\epsilon_s}{PF} = \frac{\epsilon_s}{F}$$



If S_{obj} is not changing, then

$$\begin{aligned}\dot{\mathbb{L}} &= \frac{\mathbb{L}}{S_{\text{obj}}} \left[\frac{\epsilon_s}{F} \frac{\dot{F}}{F} - \frac{\dot{\epsilon}_s}{F} \right] \\ &= \frac{F}{\epsilon_s} \left[\frac{\epsilon_s}{F} \frac{\dot{F}}{F} - \frac{\dot{\epsilon}_s}{F} \right] \\ &= \frac{\dot{F}}{F} - \frac{\dot{\epsilon}_s}{\epsilon_s} \\ &= \frac{\epsilon_x \dot{\epsilon}_x + \epsilon_y \dot{\epsilon}_y}{F} - \frac{\dot{\epsilon}_s}{\epsilon_s},\end{aligned}$$

the inverse of which is the time to collision t_c .

Time to Collision – Flat Earth

$$\mathbb{L} = \frac{h}{\cos \varphi}$$

Differentiation gives

$$\frac{\dot{\mathbb{L}}}{\mathbb{L}} = \frac{\dot{h}}{h} + \dot{\varphi} \tan \varphi.$$

From geometry we have

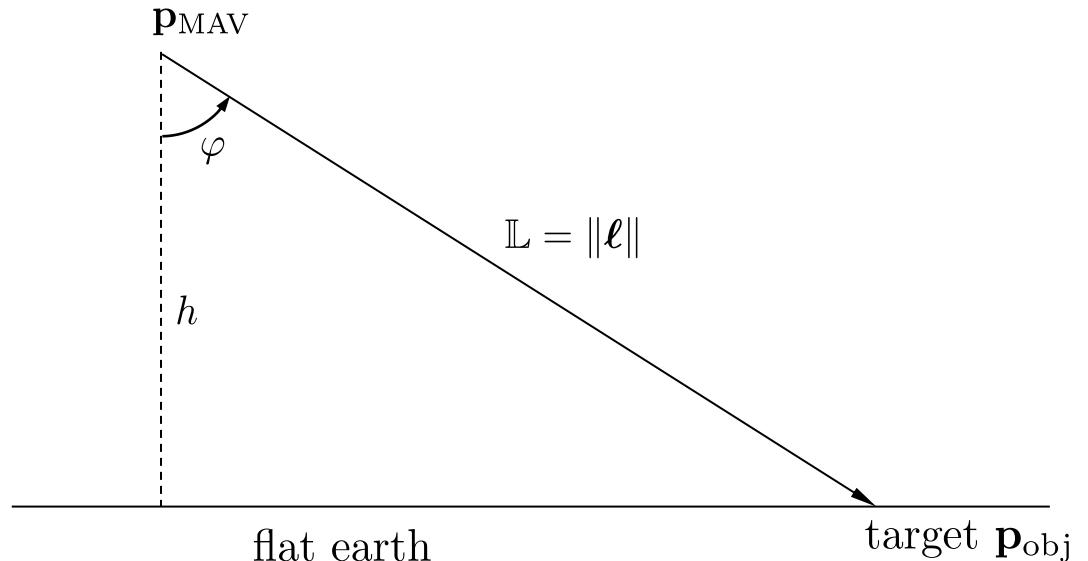
$$\cos \varphi = \check{\ell}_z^i.$$

Differentiate and solve for $\dot{\varphi}$:

$$\dot{\varphi} = -\frac{1}{\sin \varphi} \frac{d}{dt_i} \check{\ell}_z^i$$

Therefore,

$$\dot{\varphi} \tan \varphi = -\frac{1}{\cos \varphi} \frac{d}{dt_i} \check{\ell}_z^i = -\frac{1}{\check{\ell}_z^i} \frac{d}{dt_i} \check{\ell}_z^i$$



Precision Landing

For precision landing, we use the guidance model

$$\dot{p}_n = V_g \cos \chi \cos \gamma$$

$$\dot{p}_e = V_g \sin \chi \cos \gamma$$

$$\dot{p}_d = -V_g \sin \gamma$$

$$\dot{\chi} = \frac{g}{V_g} \tan \phi$$

$$\dot{\phi} = u_1$$

$$\dot{\gamma} = u_2.$$

Define:

$$\mathbf{p}_{\text{MAV}}^i = (p_n, \quad p_e, \quad p_d)^\top$$

$$\mathbf{v}_{\text{MAV}}^i = (V_g \cos \chi \cos \gamma, \quad V_g \sin \chi \cos \gamma, \quad -V_g \sin \gamma)^\top$$

$$\mathbf{v}_{\text{MAV}}^{v_2} = (V_g, \quad 0, \quad 0)^\top$$

Proportional Navigation (PN)

Proportional Navigation: align the line-of-sight rate $\dot{\ell}$ with the negative line-of-sight vector $-\ell$.

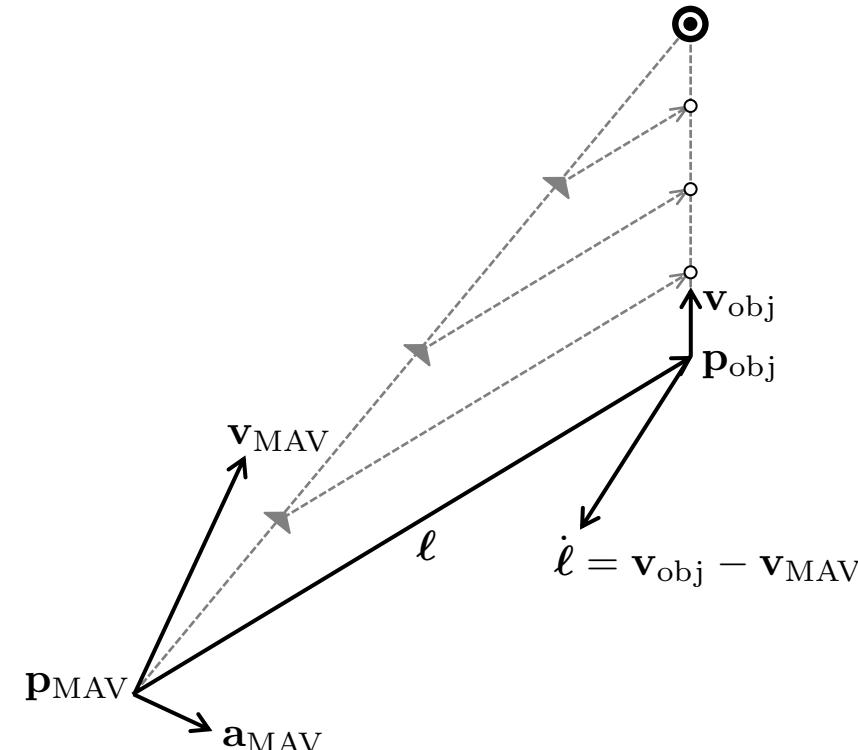
Define

$$\Omega_{\perp} = \check{\ell} \times \frac{\dot{\ell}}{L}.$$

Assuming constant velocity, we can only accelerate in the plane perpendicular to the velocity vector. Therefore, to zero Ω_{\perp} , the desired acceleration is

$$\mathbf{a}_{\text{MAV}} = N \Omega_{\perp} \times \mathbf{v}_{\text{MAV}},$$

where $N > 0$ is the (tunable) navigation constant.



Acceleration Command

To implement the acceleration command, must convert to vehicle-2 frame as

$$\begin{aligned}\mathbf{a}_{\text{MAV}}^{v2} &= \mu N \boldsymbol{\Omega}_{\perp}^{v2} \times \mathbf{v}_{\text{MAV}}^{v2} \\ &= \mu N \begin{pmatrix} \Omega_{\perp,x}^{v2} \\ \Omega_{\perp,y}^{v2} \\ \Omega_{\perp,z}^{v2} \end{pmatrix} \times \begin{pmatrix} V_g \\ 0 \\ 0 \end{pmatrix} \\ &= \begin{pmatrix} 0 \\ \mu NV \Omega_{\perp,z}^{v2} \\ -\mu NV \Omega_{\perp,y}^{v2} \end{pmatrix},\end{aligned}$$

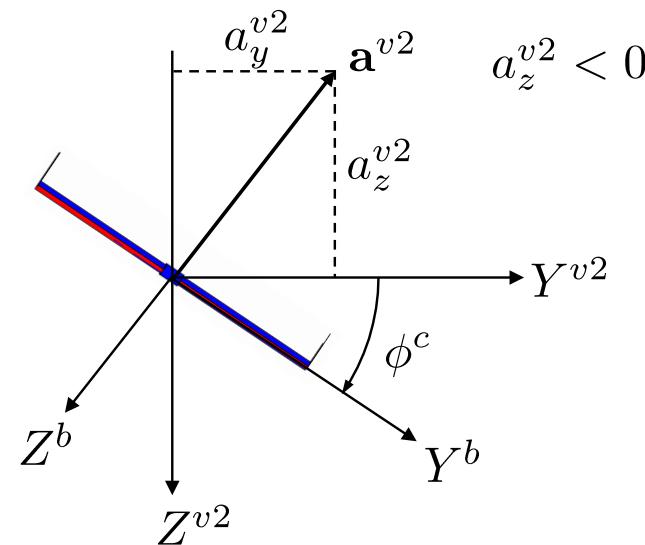
where

$$\begin{aligned}\boldsymbol{\Omega}_{\perp}^{v2} &= \mathcal{R}_b^{v2} \mathcal{R}_g^b \mathcal{R}_c^g \boldsymbol{\Omega}_{\perp}^c \\ \boldsymbol{\Omega}_{\perp}^c &= \check{\ell}^c \times \frac{\dot{\ell}^c}{\mathbb{L}} \\ \frac{\dot{\ell}^c}{\mathbb{L}} &= \frac{\dot{\mathbb{L}}}{\mathbb{L}} \check{\ell}^c + Z(\epsilon) \dot{\epsilon} + \dot{\check{\ell}}_{\text{app}}^c\end{aligned}$$

where the inverse of time to collision $\dot{\mathbb{L}}/\mathbb{L}$ can be estimated using looming or flat earth model.

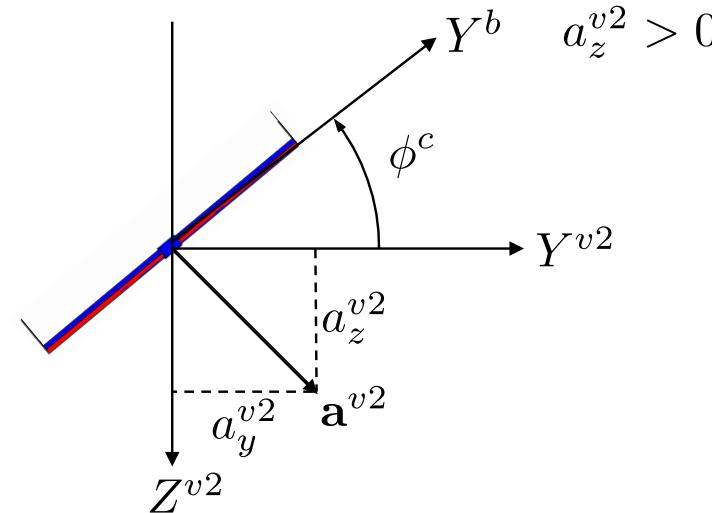
Polar Converting Logic

Use polar converting logic to convert acceleration command \mathbf{a}^{v^2} to roll angle and flight path angle commands:



$$\phi^c = \tan^{-1} \left(\frac{a_y^{v^2}}{-a_z^{v^2}} \right)$$

$$V_g \dot{\gamma}^c = \sqrt{(a_y^{v^2})^2 + (a_z^{v^2})^2}.$$



$$\phi^c = \tan^{-1} \left(\frac{a_y^{v^2}}{a_z^{v^2}} \right)$$

$$V_g \dot{\gamma}^c = -\sqrt{(a_y^{v^2})^2 + (a_z^{v^2})^2}.$$

Polar Converting Logic, cont.

General rule:

$$\phi^c = \tan^{-1} \left(\frac{a_y^{v2}}{|a_z^{v2}|} \right)$$
$$\dot{\gamma}^c = -\text{sign}(a_z^{v2}) \frac{1}{V_g} \sqrt{(a_y^{v2})^2 + (a_z^{v2})^2}$$

Note discontinuity in ϕ^c at $(a_y^{v2}, a_z^{v2}) = (0, 0)$. When $a_z^{v2} = 0$, then

- When $a_y^{v2} > 0 \rightarrow \phi^c = \pi/2$
- When $a_y^{v2} < 0 \rightarrow \phi^c = -\pi/2$

Remove discontinuity as

$$\phi^c = \sigma(a_y^{v2}) \tan^{-1} \left(\frac{a_y^{v2}}{|a_z^{v2}|} \right)$$

where

$$\sigma(a_y^{v2}) = \text{sign}(a_y^{v2}) \frac{1 - e^{-ka_y^{v2}}}{1 + e^{-ka_y^{v2}}}$$