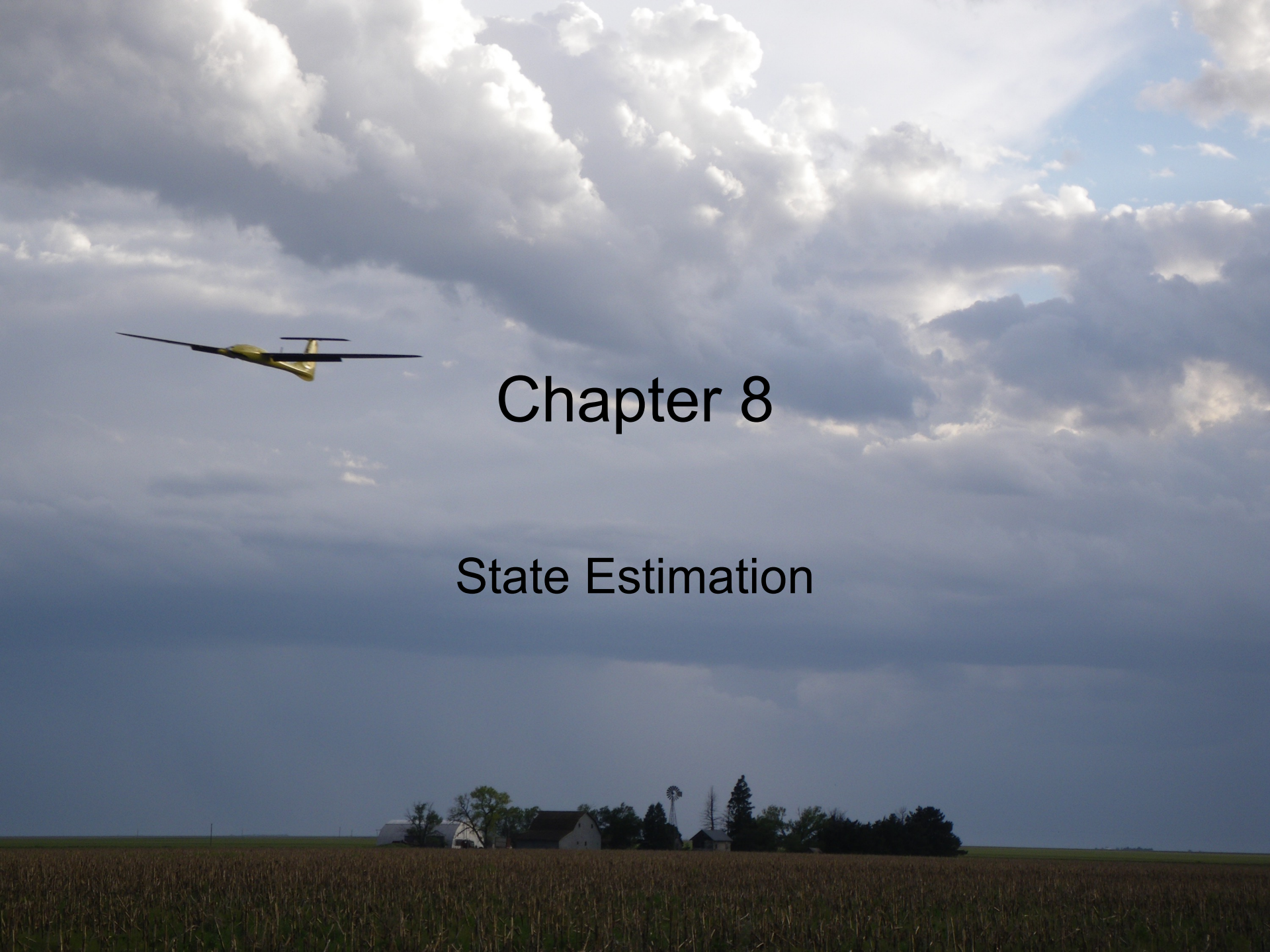
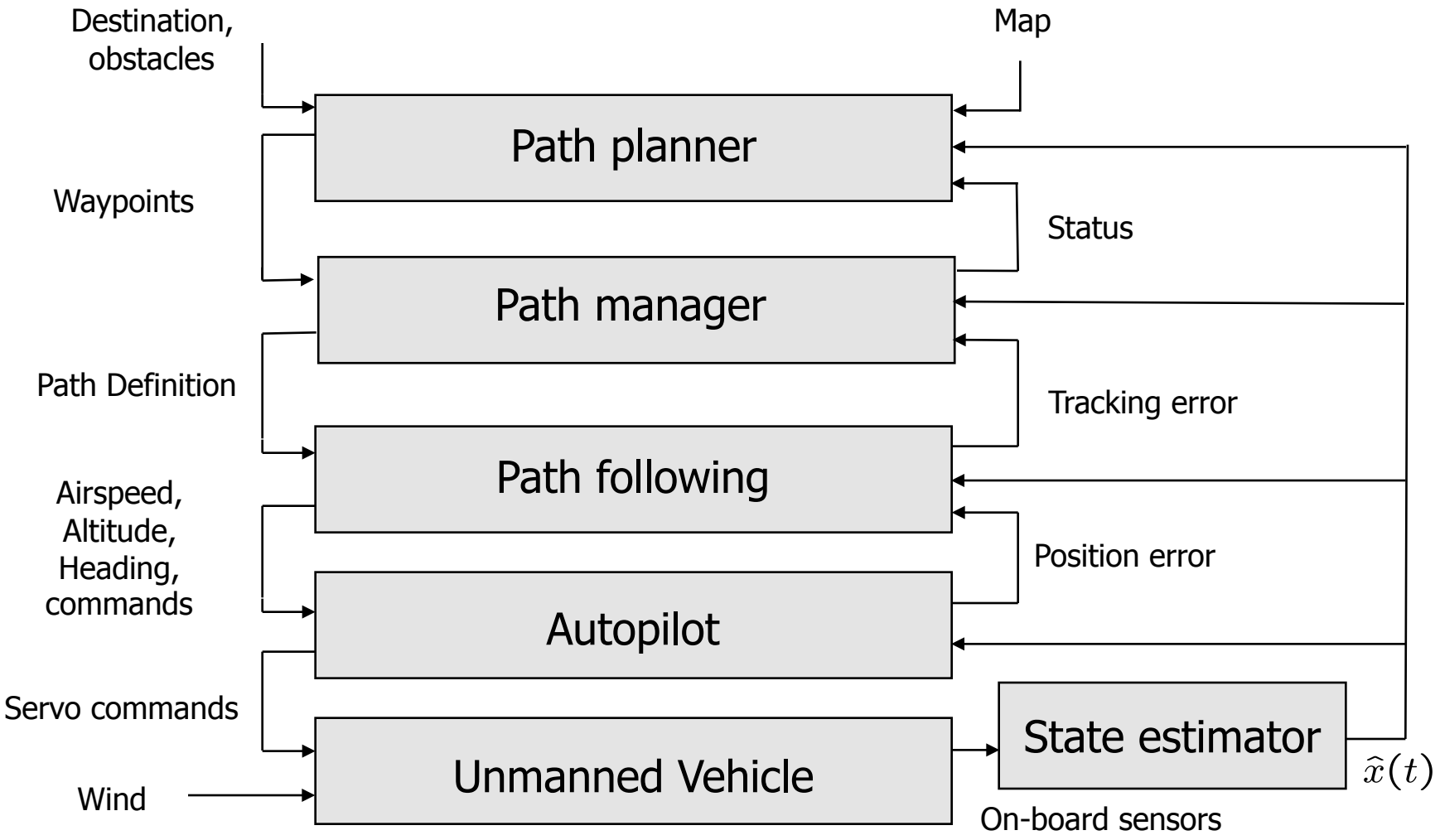


Chapter 8

State Estimation

Architecture

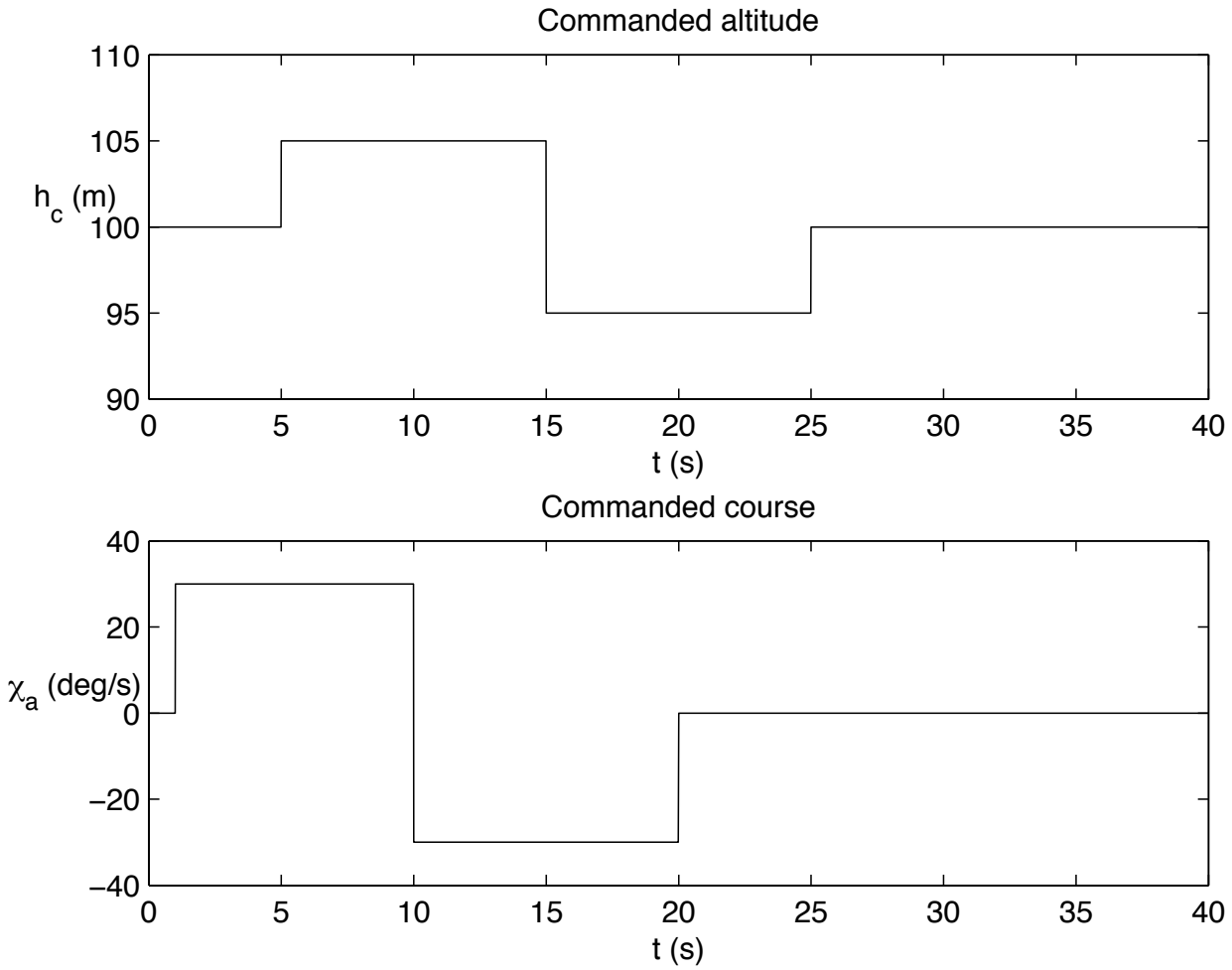


Why estimate states?

- 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 measurement 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

$$y_{\text{gyro,x}} = p + \beta_p + \eta_p$$

$$y_{\text{gyro,y}} = q + \beta_q + \eta_q$$

$$y_{\text{gyro,z}} = r + \beta_r + \eta_r$$

$$y_{\text{abs pres}} = \rho g h + \beta_{\text{abs}} + \eta_{\text{abs}}$$

$$y_{\text{diff pres}} = \frac{1}{2}\rho V_a^2 + \beta_{\text{diff}} + \eta_{\text{diff}}$$

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:

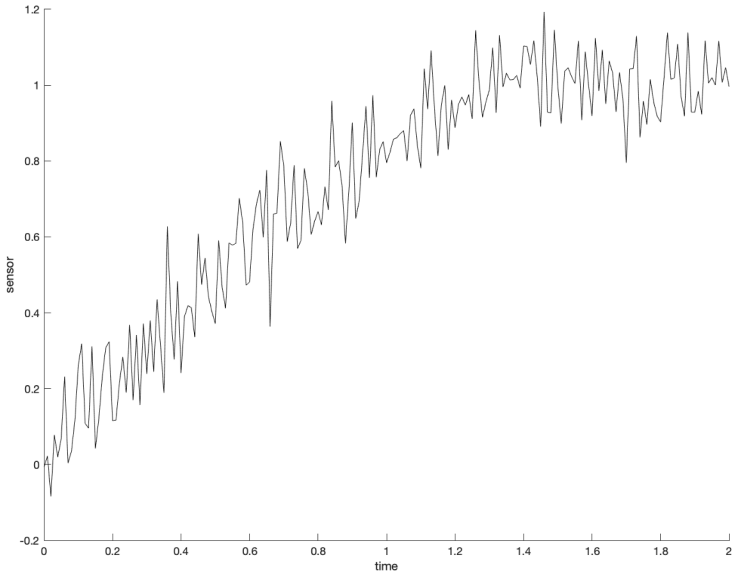
$$z_k = \alpha z_{k-1} + (1-\alpha) y_k$$

where

$0 \leq \alpha \leq 1$

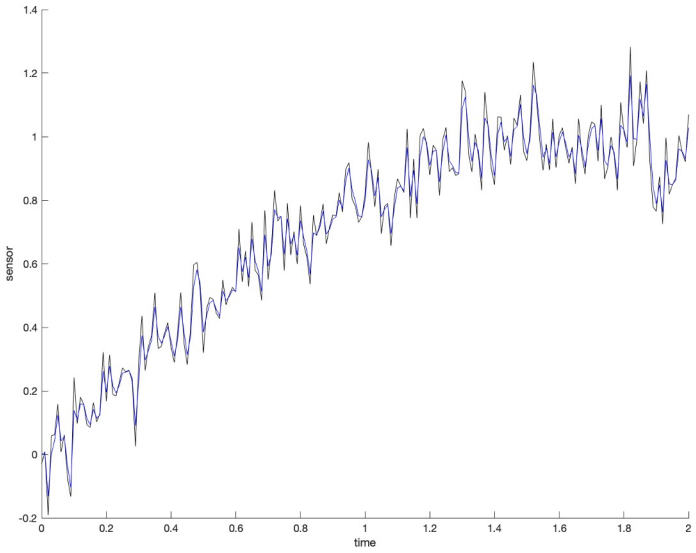
$y_k \equiv$ sensor output at time k

$z_k \equiv$ smoothed sensor output at time k

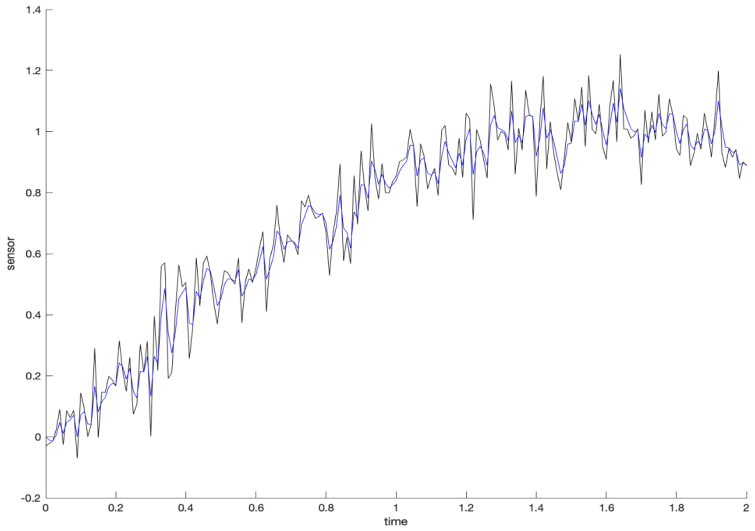


The alpha-filter

$$z_k = \alpha z_{k-1} + (1-\alpha) y_k$$



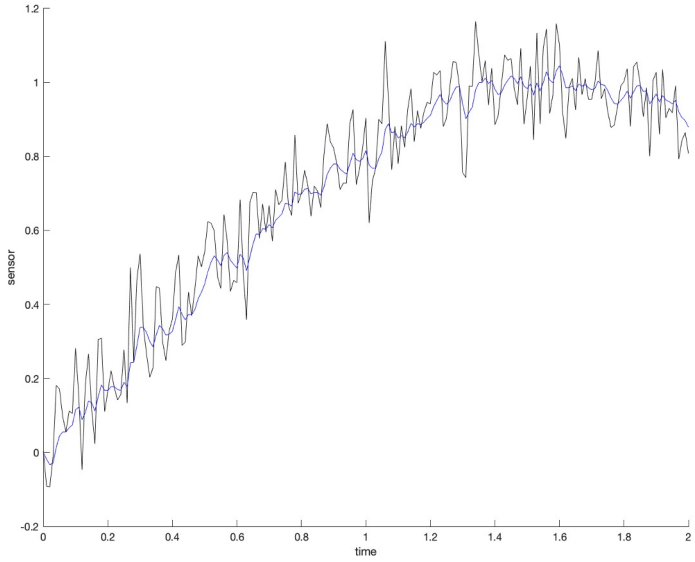
$\alpha = 0.3$



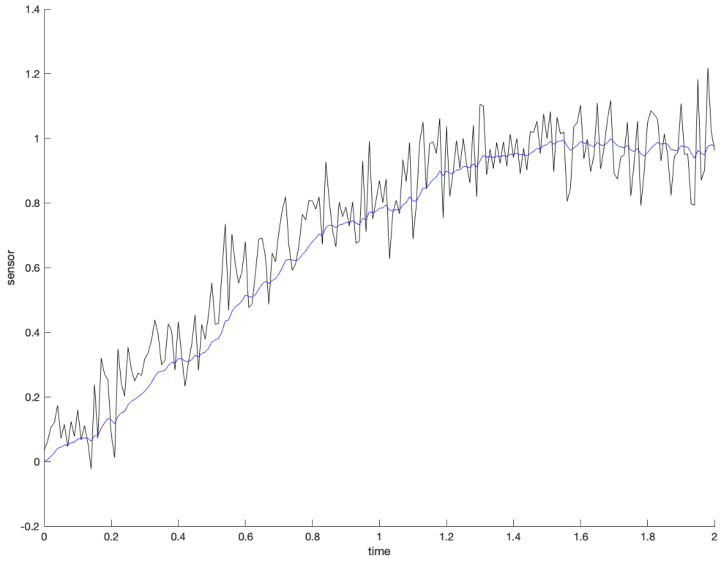
$\alpha = 0.5$

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

$$z(t) = e^{-a(t-t_0)} z(t_0) + \int_{t_0}^{t} e^{-a(t-\tau)} a y(\tau)\, d\tau$$

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

$$y_{\text{gyro,x}} = p + \beta_p + \eta_p$$

$$y_{\text{gyro,y}} = q + \beta_q + \eta_q$$

$$y_{\text{gyro,z}} = r + \beta_r + \eta_r$$

$$y_{\text{abs pres}} = \rho g h + \beta_{\text{abs}} + \eta_{\text{abs}}$$

$$y_{\text{diff pres}} = \frac{1}{2}\rho V_a^2 + \beta_{\text{diff}} + \eta_{\text{diff}}$$

State Estimate

$$\hat{p} = LPF(y_{\text{gyro,x}})$$

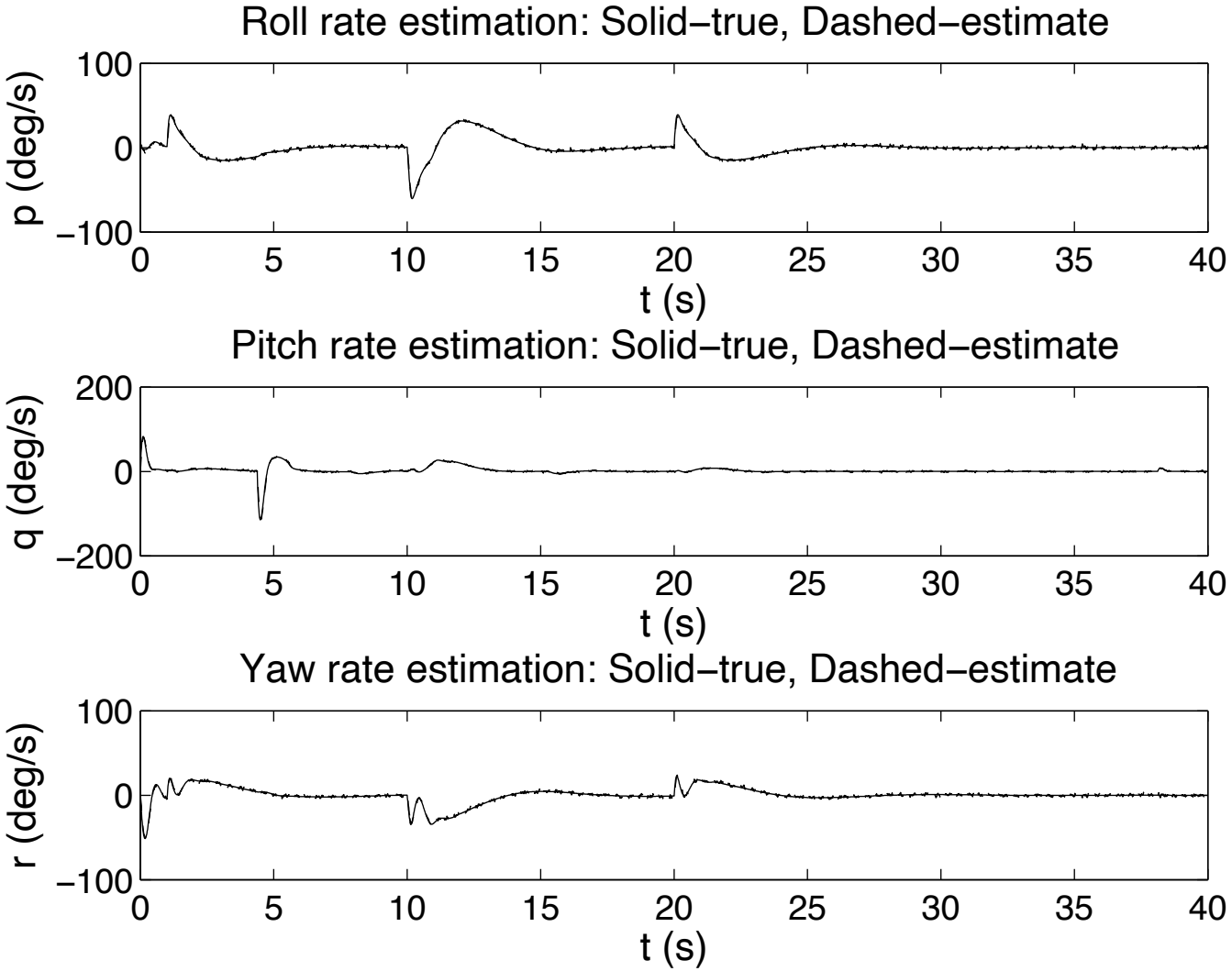
$$\hat{q} = LPF(y_{\text{gyro,y}})$$

$$\hat{r} = LPF(y_{\text{gyro,z}})$$

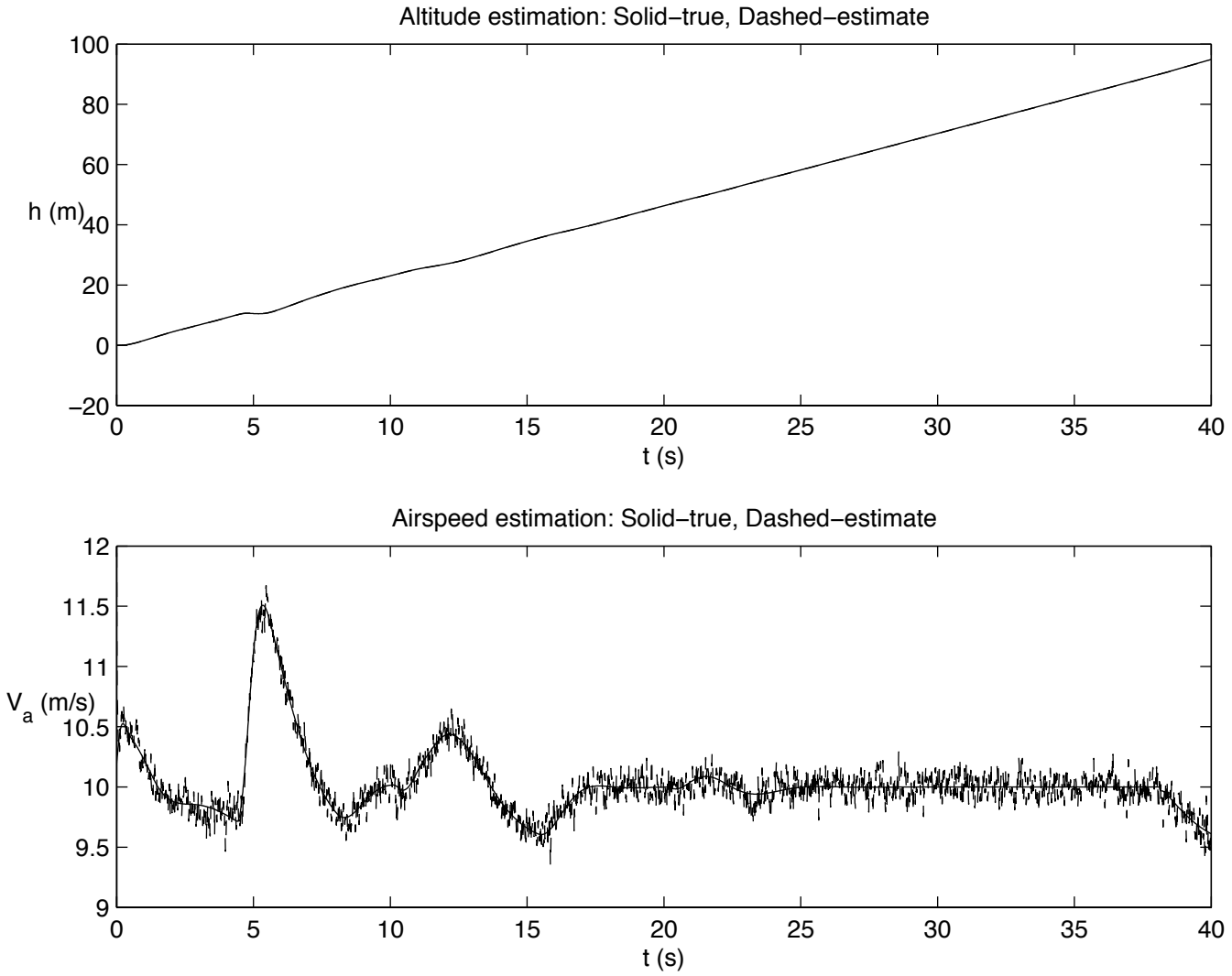
$$\hat{h} = \frac{LPF(y_{\text{static pres}})}{\rho g}$$

$$\hat{V}_a = \sqrt{\frac{2}{\rho} LPF(y_{\text{diff pres}})}$$

Model Inversion - p, q, r



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

$$
\begin{aligned}
y_{\text{accel,x}} &= \dot{u} + qw - rv + g\sin\theta + \eta_{\text{accel,x}} \\
y_{\text{accel,y}} &= \dot{v} + ru - pw - g\cos\theta\sin\phi + \eta_{\text{accel,y}} \\
y_{\text{accel,z}} &= \dot{w} + pv - qu - g\cos\theta\cos\phi + \eta_{\text{accel,z}}
\end{aligned}
$$

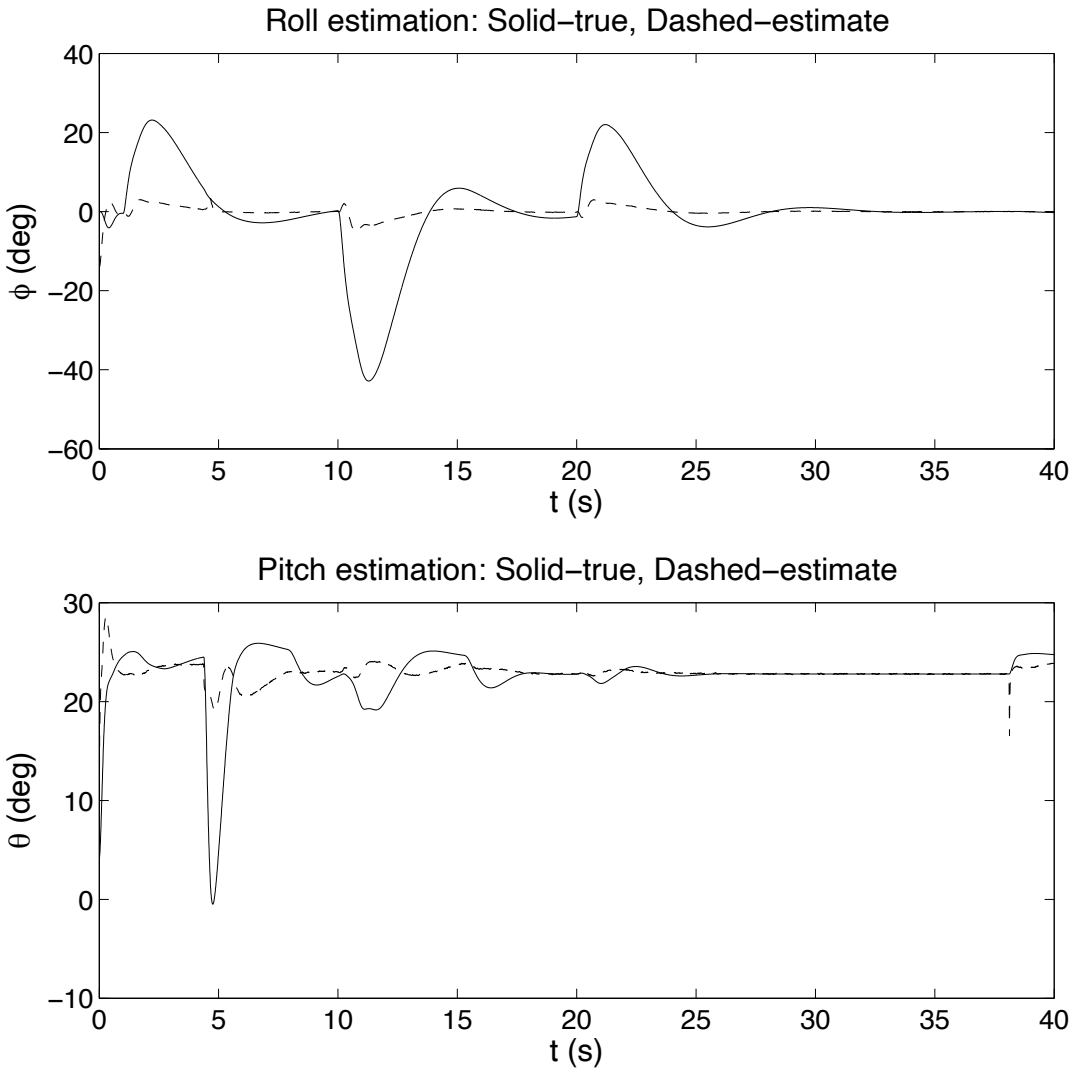
In unaccelerated flight, $\dot{u} = \dot{v} = \dot{w} = p = q = r = 0$, and

$$
\begin{aligned}
LPF(y_{\text{accel,x}}) &= g\sin\theta \\
LPF(y_{\text{accel,y}}) &= -g\cos\theta\sin\phi \\
LPF(y_{\text{accel,z}}) &= -g\cos\theta\cos\phi
\end{aligned}
$$

Solving for ϕ and θ:

$$
\begin{aligned}
\hat{\phi}_{\text{accel}} &= \tan^{-1}\left(\frac{LPF(y_{\text{accel,y}})}{LPF(y_{\text{accel,z}})}\right) \\
\hat{\theta}_{\text{accel}} &= \sin^{-1}\left(\frac{LPF(y_{\text{accel,x}})}{g}\right)
\end{aligned}
$$

Model Inversion - ϕ, θ



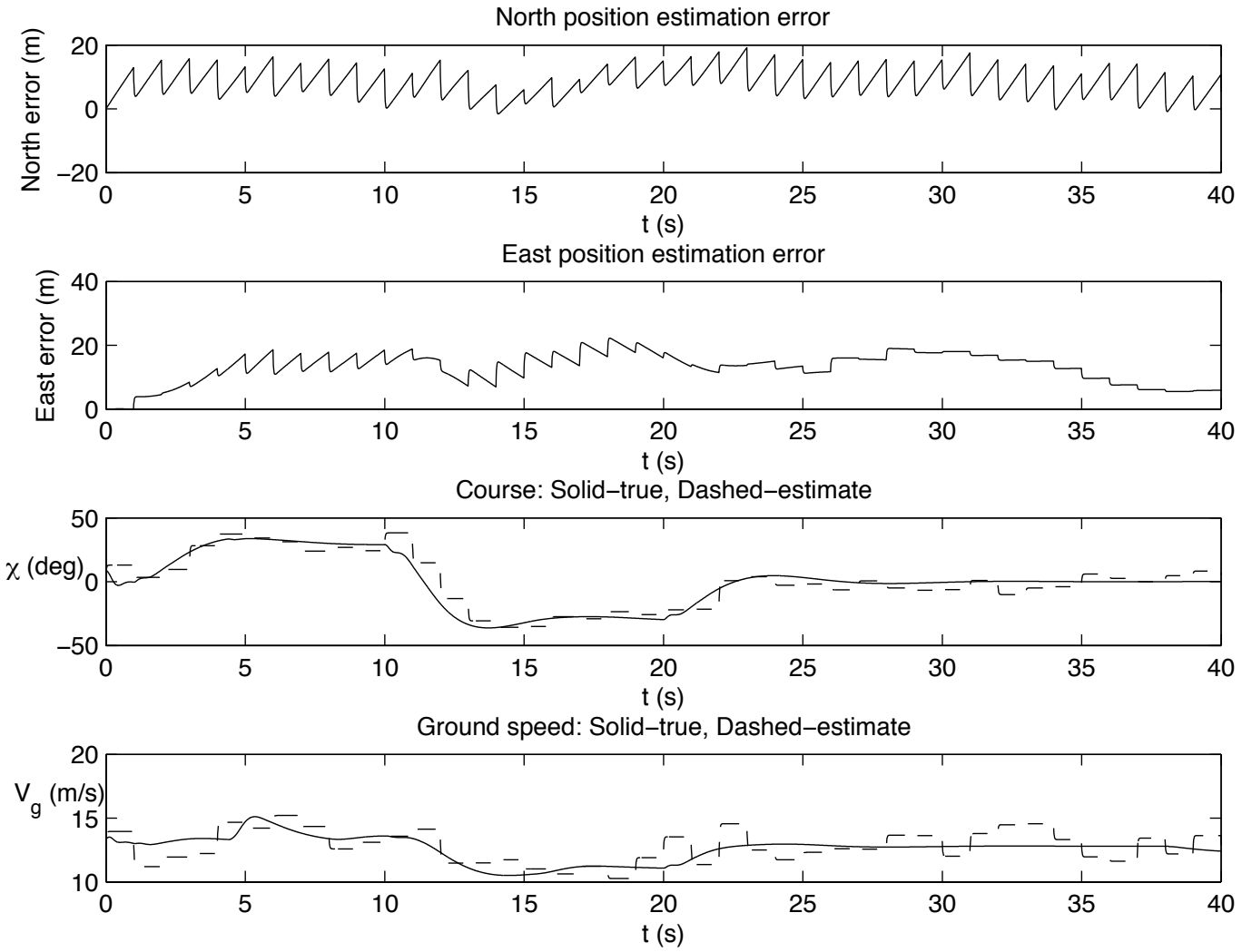
Inverting Sensor Measurement Model

- Can also invert sensor models for GPS signals

$$\hat{p}_n = LPF(y_{\text{GPS,n}})$$
$$\hat{p}_e = LPF(y_{\text{GPS,e}})$$
$$\hat{\chi} = LPF(y_{\text{GPS},\chi})$$
$$\hat{V}_g = LPF(y_{\text{GPS},V_g})$$

- 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

$$\dot{x} = Ax + Bu$$
$$y = Cx.$$

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 $\to 0$ if $\text{eig}(A - LC)$ in LHP.

Dynamic Observer Theory, cont.

For sampled data systems we use a predictor-corrector structure:
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):

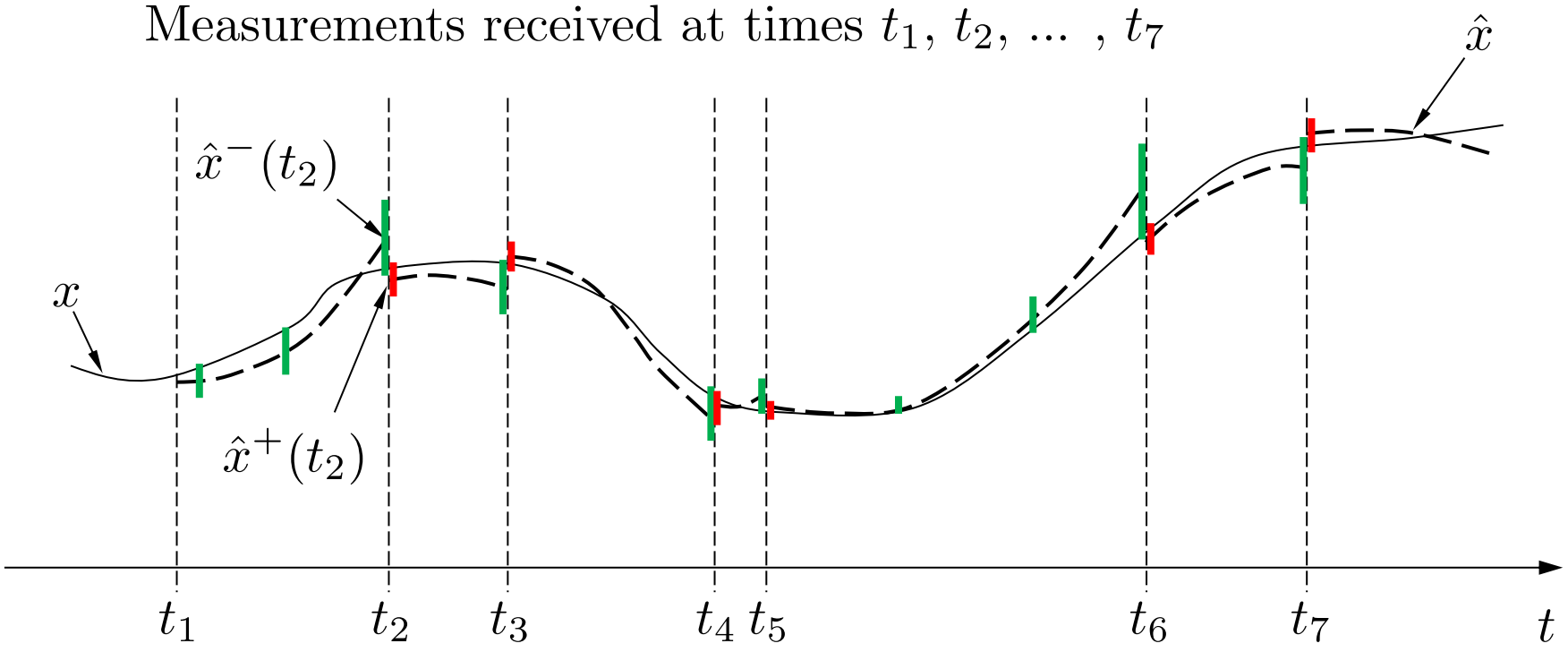
$$\dot{\hat{x}} = f(\hat{x}, u)$$

At a Measurements (corrector):

$$\hat{x}^+ = \hat{x}^- + L(y(t_n) - h(\hat{x}^-)).$$

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.

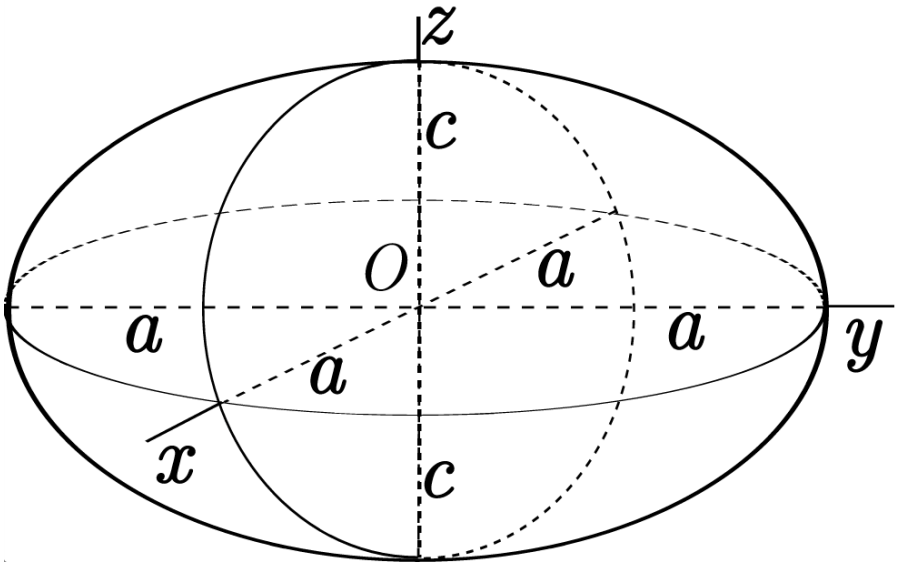


Quadratic Forms

Recall that for a positive definite matrix $P = P^\top > 0$, the set

$$\mathcal{E}(k) = \left\{ y \in \mathbb{R}^n : y^\top P^{-1} y = k^2 \right\}$$

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

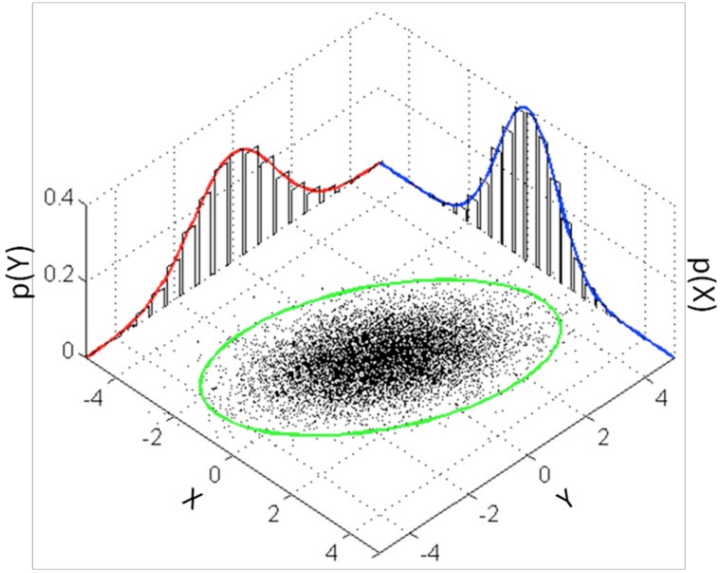
The multivariate Gaussian distribution is given by

$$\mathcal{N}(\mu, P) = \frac{1}{(2\pi)^{k/2}\|P\|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^\top P^{-1}(x-\mu)\right),$$

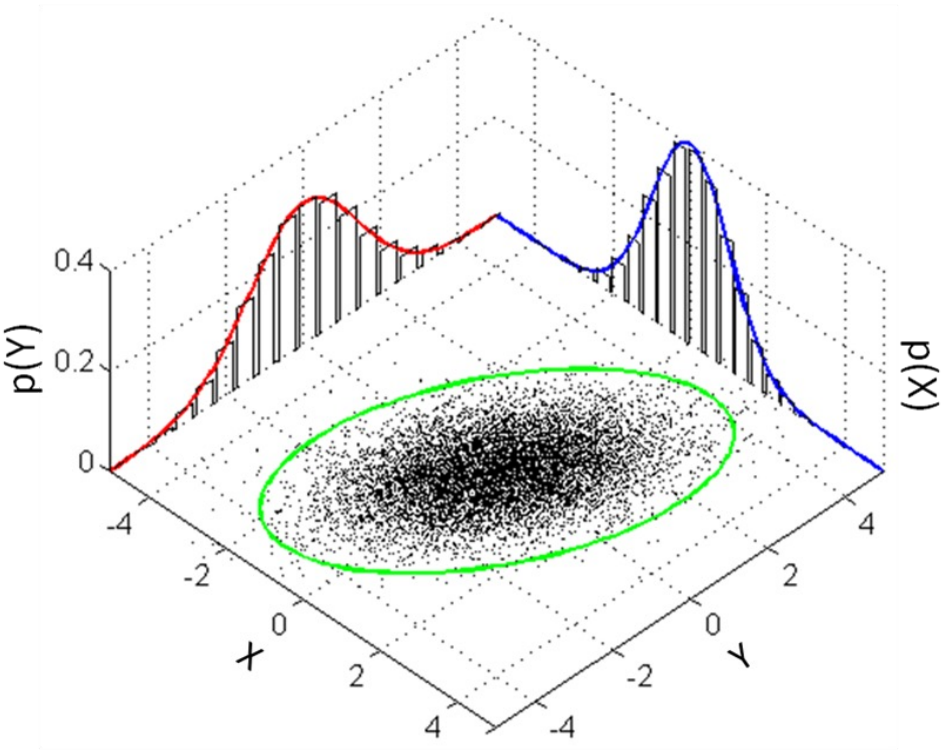
where μ is the mean and P is the covariance matrix. The k-σ ellipsoid is defined as

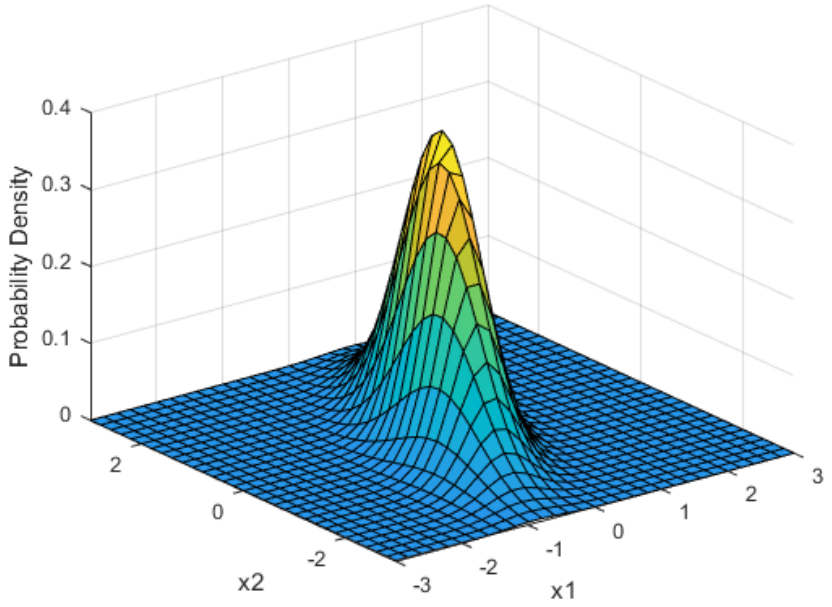
$$\mathcal{E}(k\sigma) = \left\{x \in \mathbb{R}^k : (x-\mu)^\top P^{-1}(x-\mu) = k\sigma\right\}$$

where 1-σ or $k = 1$ corresponds to the ellipsoid that represents one standard deviation from the mean μ.



Multivariate Gaussians



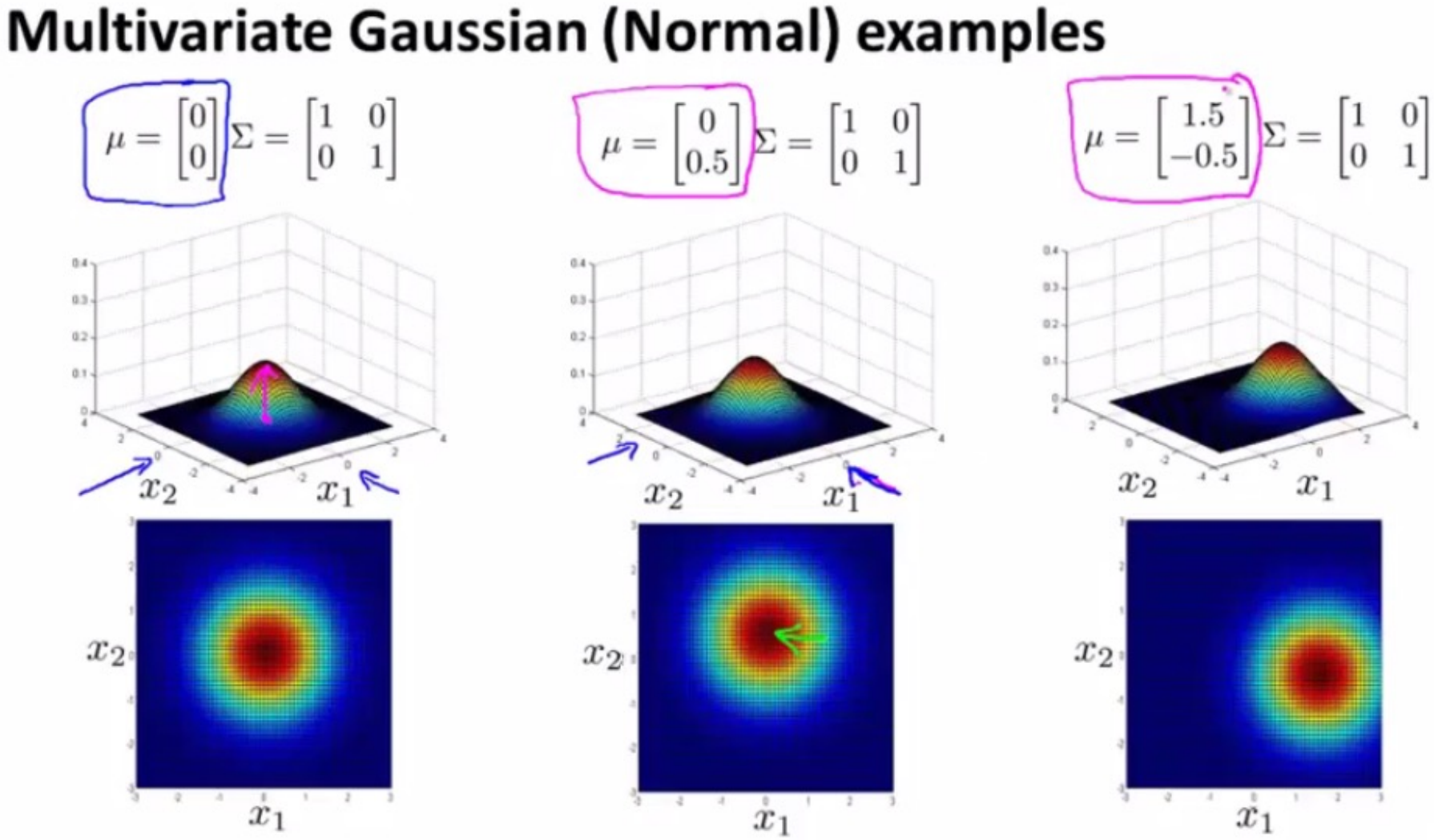


Gaussian filters

- Gaussian filters represent belief with a Gaussian (normal) probability distribution
- Examples of Gaussian filters include
  - Kalman filter
  - Extended Kalman filter
  - Unscented Kalman filter
  - Information filter

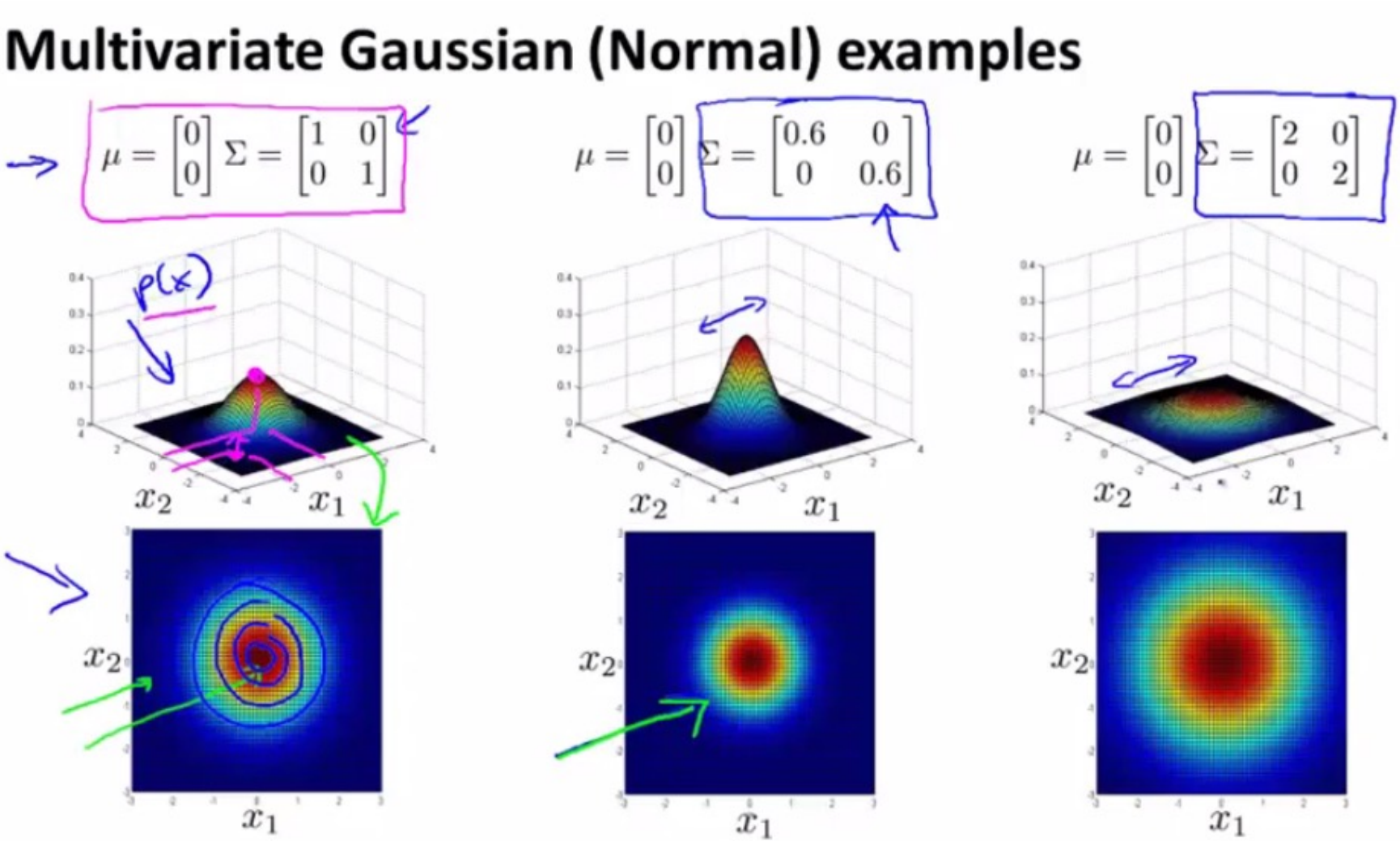
- Normal distribution characterized by mean and variance
  - Often abbreviated as $\mathcal{N}(x; \mu, \sigma^2)$ or $\mathcal{N}(x; \mu, \Sigma)$

Mean and covariance review



Andrew Ng, Machine Learning, Coursera

Mean and covariance review

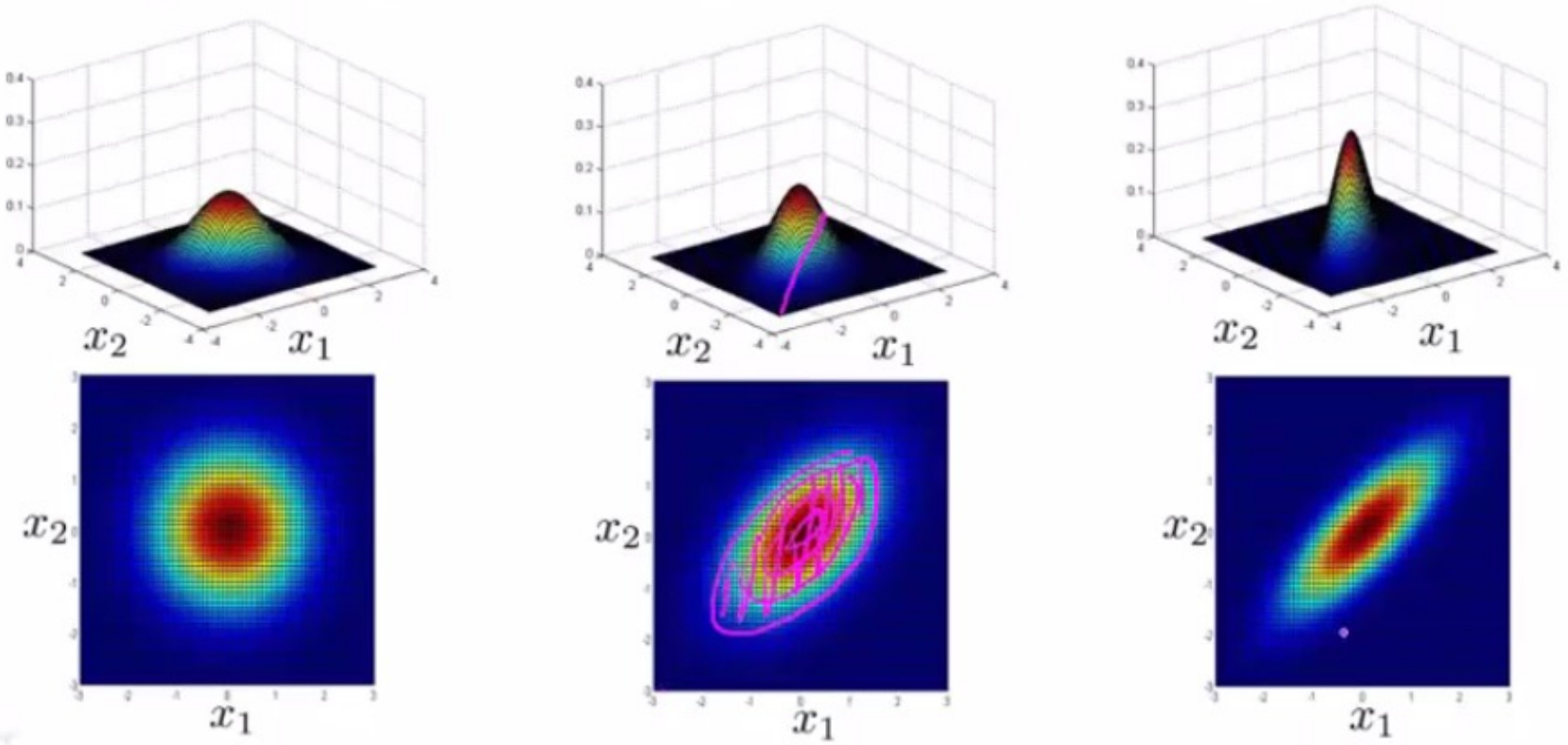


Andrew Ng, Machine Learning, Coursera

Mean and covariance review

Multivariate Gaussian (Normal) examples

$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ $\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$ $\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix}$

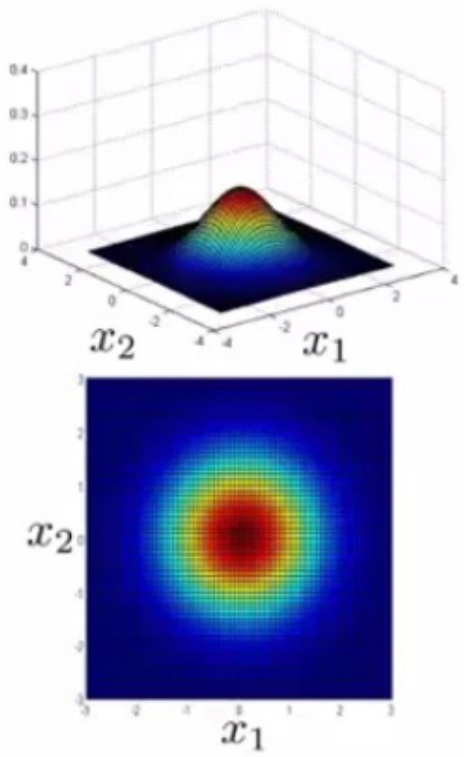


Andrew Ng, Machine Learning, Coursera

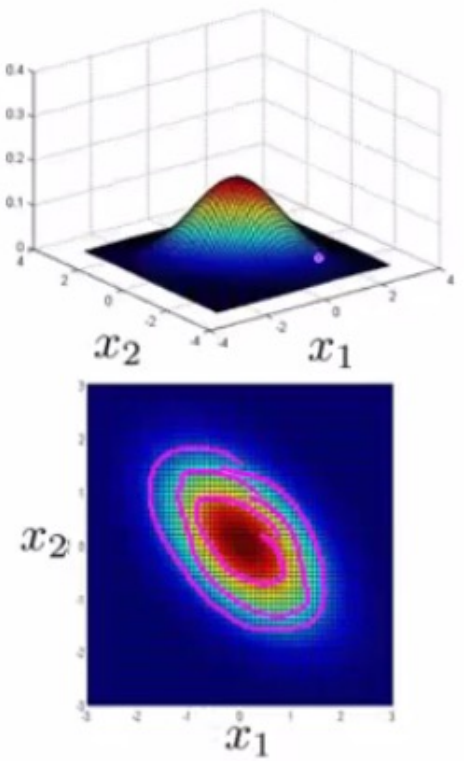
Mean and covariance review

Multivariate Gaussian (Normal) examples

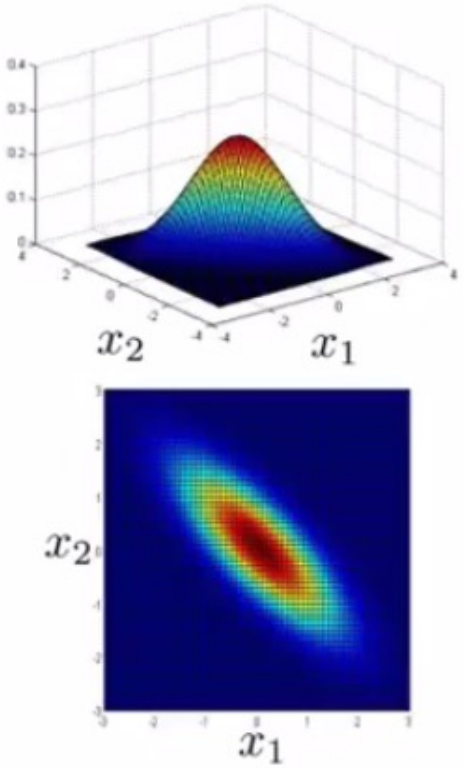
$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$



$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & -0.5 \\ -0.5 & 1 \end{bmatrix}$$



$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & -0.8 \\ -0.8 & 1 \end{bmatrix}$$



Andrew Ng, Machine Learning, Coursera

Kalman Filter

Assume continuous linear system dynamics, with discrete measurement update

$$\dot{x} = Ax + Bu + \xi$$
$$y[n] = Cx[n] + \eta[n]$$

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 tuning parameter for the Kalman filter.

A derivation of the linear, continuous-discrete Kalman filter is provided in the textbook.

Challenge: Our dynamics are nonlinear, so we must utilize the *extended* Kalman filter.

Extended Kalman Filter Overview

Basic idea of Kalman Filter (and extended Kalman Filter - EKF):

- 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:

$$\dot{x} = f(x, u) + \xi$$
$$y_i[n] = h_i(x[n], u[n]) + \eta_i[n],$$

where

$$\xi \sim \mathcal{N}(0, Q)$$
$$\eta_i \sim \mathcal{N}(0, R_i)$$

Extended Kalman Filter

Between Measurements (prediction):

$$\dot{\hat{x}} = f(\hat{x}, u)$$
$$A = \frac{\partial f}{\partial x}(\hat{x}, u)$$
$$\dot{P} = AP + PA^\top + Q$$

Numerical errors in integrating prediction equations forward in time using Euler integration can cause covariance matrix to not remain positive definite. Since it is linear, we can convert to a difference equation.

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

Revised EKF

Between Measurements (prediction):

$$\dot{\hat{x}} = f(\hat{x}, u)$$
$$A = \frac{\partial f}{\partial x}(\hat{x}, u)$$
$$A_d = I + AT_s + A^2\frac{T_s^2}{2}$$
$$P_{k+1} = A_d P_k A_d^\top + T_s^2 Q$$

P_{k+1} is sum of two positive-definite matrices and therefore remains positive definite

At the i^{th} Measurements (correction):

$$C_i = \frac{\partial h_i}{\partial x}(\hat{x}^-)$$
$$L_i = P^- C_i^\top (R_i + C_i P^- C_i^\top)^{-1}$$
$$\hat{x}^+ = \hat{x}^- + L_i(y_i(t_n) - h_i(\hat{x}^-)),$$
$$P^+ = (I - L_i C_i)P^-(I - L_i C_i)^\top + L_i R_i L_i^\top$$

This is Joseph's form of the covariance update (preserves symmetry)

where L_i is called the Kalman gain for sensor i

To do list:

- Improve estimation for roll and pitch angles, north and east positions, ground speed, and heading
- How?
- Extended Kalman filter

Nonlinear dynamic model:

$$\dot{x} = f(x, u) + \xi$$
$$y[n] = h(x[n], u[n]) + \eta[n]$$

ξ: zero-mean Gaussian random process with covariance Q
$\eta[n]$: zero-mean Gaussian random variable with covariance R

EKF Algorithm

Algorithm 1 Continuous-Discrete Extended Kalman Filter

Initialize: $\hat{x} = 0$.
Pick an output sample rate T_{out} which is much less than the sample rates of the sensors.
At each sample time T_{out}:
for $i = 1$ to N **do** {Prediction}
  $T_p = T_{out}/N$
  $\hat{x} \leftarrow \hat{x} + T_p f(\hat{x}, u)$
  $A = \frac{\partial f}{\partial x}(\hat{x}, u)$
  $A_d = I + AT_p + A^2 T_p^2$
  $P \leftarrow A_d P A_d^\top + T_p^2 Q$
end for
if Measurement has been received from sensor i **then** {Correction}
  $C_i = \frac{\partial h_i}{\partial x}(\hat{x}, u[n])$
  $L_i = PC_i^\top (R_i + C_i P C_i^\top)^{-1}$
  $P \leftarrow (I - L_i C_i) P (I - L_i C_i)^\top + L_i R_i L_i^\top$
  $\hat{x} \leftarrow \hat{x} + L_i \left(y_i[n] - h(\hat{x}, u[n])\right)$.
end if

If R is diagonal (uncorrelated sensors), then update one measurement at a time.

Attitude Estimation Using EKF

Use the nonlinear propagation model

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

where

$$\xi_\phi \sim \mathcal{N}(0, Q_\phi) \qquad \text{and} \qquad \xi_\theta \sim \mathcal{N}(0, Q_\theta)$$

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 $\eta = (\eta_\phi, \eta_\theta)^\top$, gives

$$\dot{x} = f(x,u) + \xi \qquad Q = E\{\xi\xi^\top\}$$
$$y = h(x,u) + \eta \qquad R = E\{\eta\eta^\top\}$$

where

$$f(x,u) = \begin{pmatrix} p + q\sin\phi\tan\theta + r\cos\phi\tan\theta \\ q\cos\phi - r\sin\phi \end{pmatrix}$$

$$h(x,u) = \begin{pmatrix} qV_a\sin\theta + g\sin\theta \\ rV_a\cos\theta - pV_a\sin\theta - g\cos\theta\sin\phi \\ -qV_a\cos\theta - g\cos\theta\cos\phi \end{pmatrix}$$

Attitude Estimation Using EKF - Improved

Defining $x = (\phi, \theta)^\top$, $u = (p, q, r, V_a)^\top$, $\xi = (\xi_\phi, \xi_\theta)^\top$, $\eta = (\eta_\phi, \eta_\theta)^\top$ and noting that there is noise on the gyros and pressure sensor, i.e., $u_{\text{actual}} = u + \xi_u$ where $\xi_u = (\xi_p, \xi_q, \xi_r, \xi_{V_a})^\top$, gives

$$\dot{x} = f(x,u) + G(x)\xi_u + \xi \qquad Q = E\{\xi\xi^\top\}$$
$$y = h(x,u) + \eta \qquad R = E\{\eta\eta^\top\}$$

where

$$f(x,u) = \begin{pmatrix} p + q\sin\phi\tan\theta + r\cos\phi\tan\theta \\ q\cos\phi - r\sin\phi \end{pmatrix}$$

$$h(x,u) = \begin{pmatrix} qV_a\sin\theta + g\sin\theta \\ rV_a\cos\theta - pV_a\sin\theta - g\cos\theta\sin\phi \\ -qV_a\cos\theta - g\cos\theta\cos\phi \end{pmatrix}$$

$$G(x) = \begin{pmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta & 0 \\ 0 & \cos\phi & -\sin\phi & 0 \end{pmatrix}$$

Note that

$$E\{(G\xi_u + \xi)(G\xi_u + \xi)^\top\} = GE\{\xi_u\xi_u^\top\}G^\top + E\{\xi\xi^\top\} = GQ_uG^\top + Q$$

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,

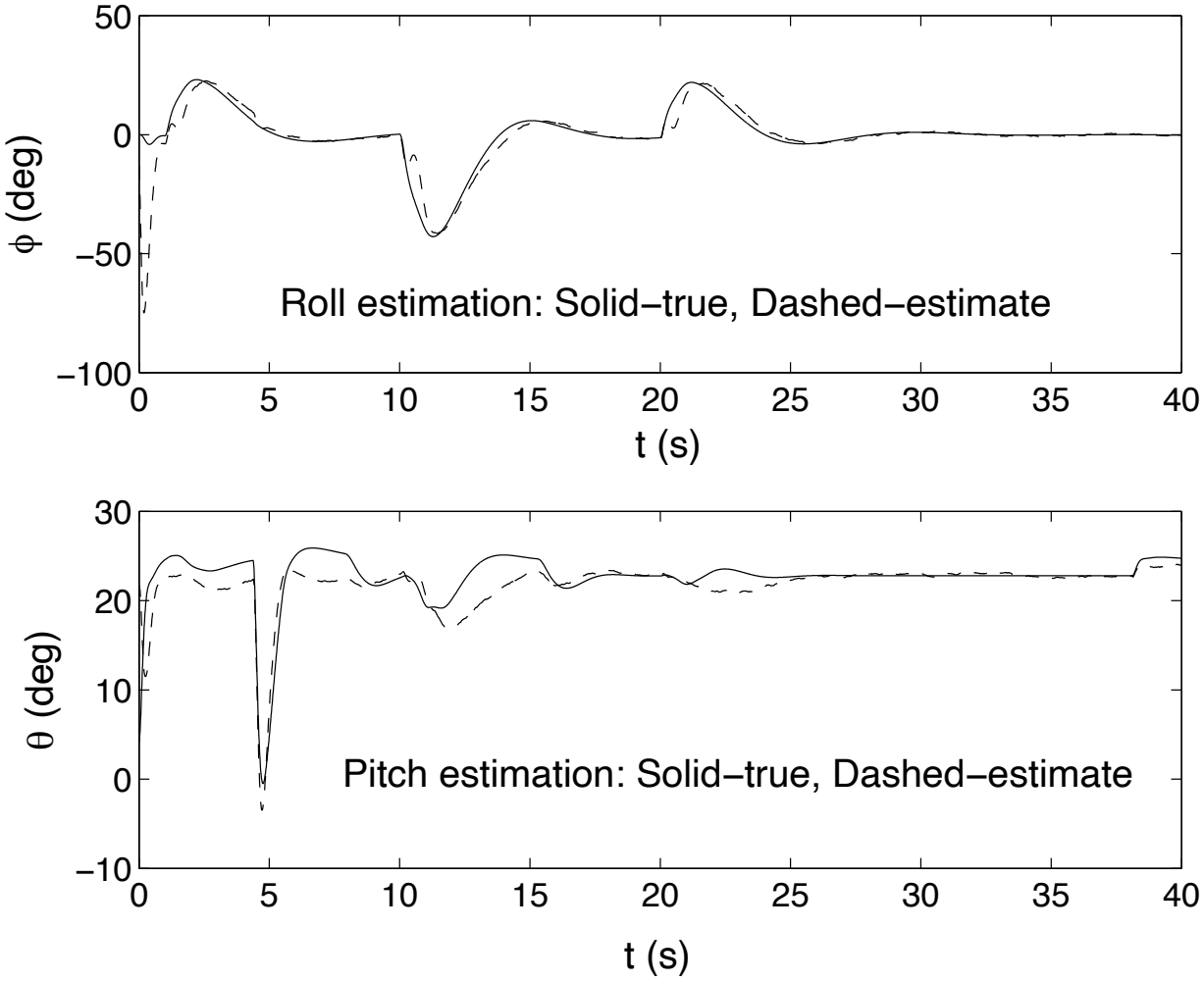
$$\frac{\partial f}{\partial x} = \begin{pmatrix} q\cos\phi\tan\theta - r\sin\phi\tan\theta & \frac{q\sin\phi + r\cos\phi}{\cos^2\theta} \\ -q\sin\phi - r\cos\phi & 0 \end{pmatrix}$$

$$\frac{\partial h}{\partial x} = \begin{pmatrix} 0 & qV_a\cos\theta + g\cos\theta \\ -g\cos\phi\cos\theta & -rV_a\sin\theta - pV_a\cos\theta + g\sin\phi\sin\theta \\ g\sin\phi\cos\theta & (qV_a + g\cos\phi)\sin\theta \end{pmatrix}$$

We now 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:

$$\dot{p}_n = V_g \cos\chi$$
$$\dot{p}_e = V_g \sin\chi$$

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:

$$\begin{aligned}\dot{V}_g &= \frac{(V_a\cos\psi + w_n)(-V_a\dot{\psi}\sin\psi) + (V_a\sin\psi + w_e)(V_a\dot{\psi}\cos\psi)}{V_g} \\ &= \frac{V_a}{V_g}\dot{\psi}(-w_n\sin\psi + w_e\cos\psi)\end{aligned}$$

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:
$x = (p_n, p_e, V_g, \chi, w_n, w_e, \psi)^\top$
$u = (V_a, q, r, \phi, \theta)^\top$:

$$f(x,u) \triangleq \begin{pmatrix} V_g \cos\chi \\ V_g \sin\chi \\ \frac{V_a}{V_g}\dot{\psi}\left(-w_n \sin\psi + w_e \cos\psi\right) \\ \frac{g}{V_g}\tan\phi\cos(\chi-\psi) \\ 0 \\ 0 \\ q\frac{\sin\phi}{\cos\theta} + r\frac{\cos\phi}{\cos\theta} \end{pmatrix}$$

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:

$$\frac{\partial f}{\partial x} = \begin{pmatrix} 0 & 0 & \cos\chi & -V_g \sin\chi & 0 & 0 & 0 \\ 0 & 0 & \sin\chi & V_g \cos\chi & 0 & 0 & 0 \\ 0 & 0 & -\frac{\dot{V}_g}{V_g} & 0 & -\frac{\dot{\psi} V_a \sin\psi}{V_g} & \frac{\dot{\psi} V_a \cos\psi}{V_g} & \frac{\partial \dot{V}_g}{\partial \psi} \\ 0 & 0 & \frac{\partial \dot{\chi}}{\partial V_g} & \frac{\partial \dot{\chi}}{\partial \chi} & 0 & 0 & \frac{\partial \dot{\chi}}{\partial \psi} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

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

$$y_{GPS} = h(x,u) + \eta_{GPS}$$

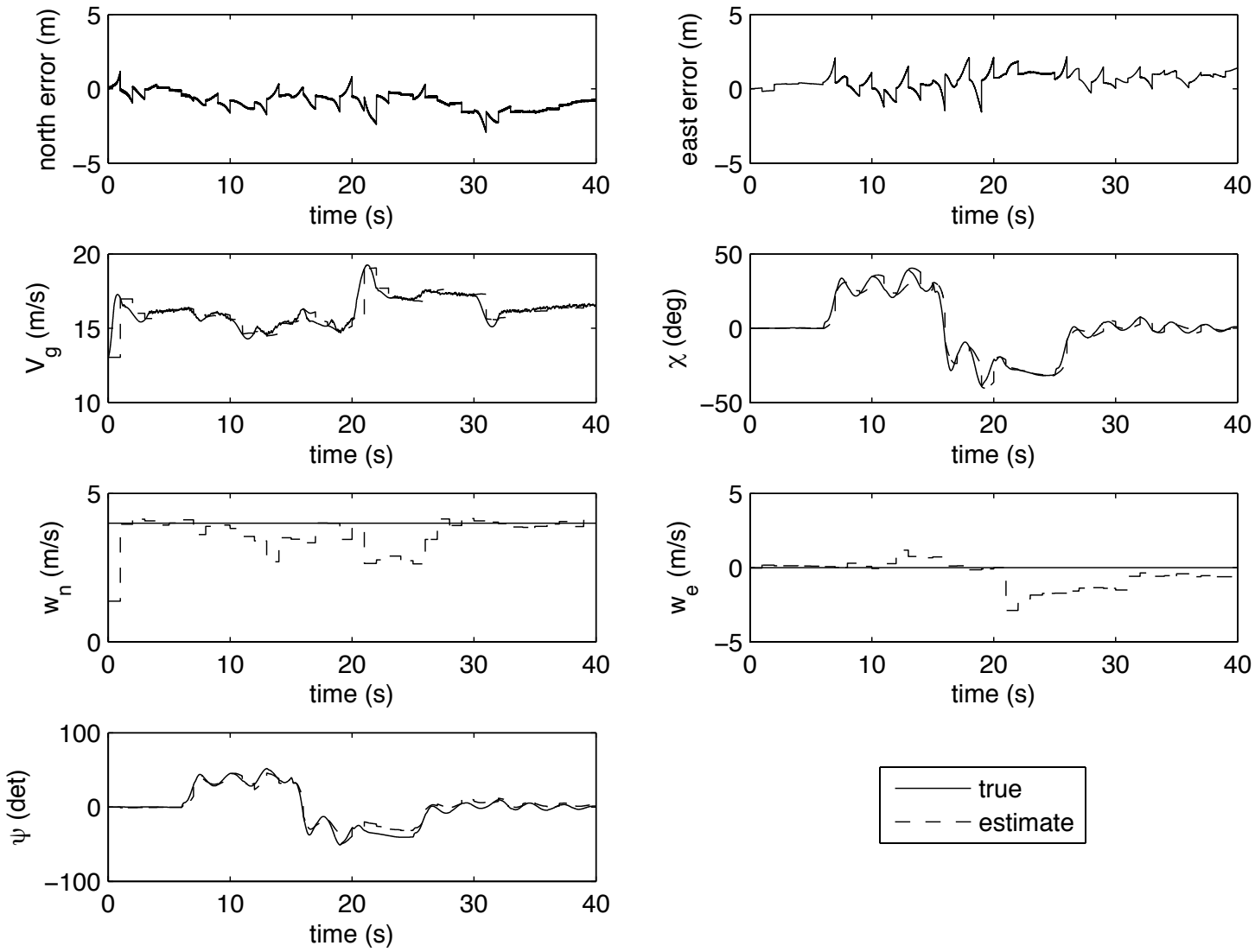
where $y_{GPS} = (y_{GPS,n}, y_{GPS,e}, y_{GPS,V_g}, y_{GPS,\chi}, y_{\text{wind,n}}, y_{\text{wind,e}})$, and

$$h(x,u) = \begin{pmatrix} p_n \\ p_e \\ V_g \\ \chi \\ V_a \cos\psi + w_n - V_g \cos\chi \\ V_a \sin\psi + w_e - V_g \sin\chi \end{pmatrix}$$

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



Python Simulation

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

$$S_k = E\{v_k v_k^\top\} = E\{(\eta_k + C\tilde{x}_k^-)(\eta_k + C\tilde{x}_k^-)^\top\} = R + CP_k^- C^\top$$

We can use this covariance as a check (or gate) on the validity of a measurement by comparing the norm of the innovation (or residual) to this covariance

One way to do this is by computing the Mahalanobis distance, which is a scaled measurement of the distance between the measurement y_k and the predicted measurement $C\hat{x}_k^-$ (or $h(\hat{x}_k^-)$ in the case of the EKF)

Measurement Gating

The Mahalanobis distance is computed as

$$d_{M,k} = \sqrt{(y_k - h(\hat{x_k}))^\top S_k^{-1} (y_k - h(\hat{x_k}))}$$

Notionally, $d_{M,k}$ tells us how many standard deviations separate the measurement from the predicted measurement. By setting a gate value at 2 (95% confidence) or 3 (99% confidence), we can identify a measurement as an outlier and reject it accordingly. This is a bit of a hack since $d_{M,k}$ is not normally distributed.

A better gating approach is to recognize that the Mahalonibis distance and its square

$$d_{M,k}^2 = (y_k - h(\hat{x_k}))^\top S_k^{-1} (y_k - h(\hat{x_k}))$$

are χ^2 (chi-squared) random variables with m degrees of freedom. We will gate our measurements by accepting all measurements that are less than a gate threshold value γ, such that $d_{M,k}^2 < \gamma$.

The gate threshold is obtained from the inverse χ^2 cumulative distribution at a significance level α and k degrees of freedom. The measurement gate is a region of acceptance such that $100(1-\alpha)\%$ of true measurements are accepted. Typical values are $\alpha = 0.01$ or $\alpha = 0.05$.

Measurement Gating

An example of chi-squared gating on the measurement update residual (3 degrees of freedom, 0.01% level of significance $\rightarrow \chi^2_{3,0.01} = 11.3$) is given below.

```
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!

GPS Smoothing – Simplified

Define:

$x = (p_n, p_e, V_g, \chi, \psi)^\top$

$u = (V_a, q, r, \phi, \theta)^\top$

$$f(x,u) = \begin{pmatrix} V_g \cos\chi \\ V_g \sin\chi \\ \dot{\psi} V_a \sin(\chi - \psi) \\ \frac{g}{V_g} \tan\phi \cos(\chi - \psi) \\ q \frac{\sin\phi}{\cos\theta} + r \frac{\cos\phi}{\cos\theta} \end{pmatrix}$$

GPS Smoothing – Simplified

Define:
$x = (p_n, p_e, V_g, \chi, \psi)^\top$
$u = (V_a, q, r, \phi, \theta)^\top$

Jacobian of f:

$$\frac{\partial f}{\partial x} = \begin{pmatrix} 0 & 0 & \cos\chi & -V_g \sin\chi & 0 \\ 0 & 0 & \sin\chi & V_g \cos\chi & 0 \\ 0 & 0 & -\frac{\dot{V}_g}{V_g} & \frac{\partial \dot{V}_g}{\partial \chi} & \frac{\partial \dot{V}_g}{\partial \psi} \\ 0 & 0 & \frac{\partial \dot{\chi}}{\partial V_g} & \frac{\partial \dot{\chi}}{\partial \chi} & \frac{\partial \dot{\chi}}{\partial \psi} \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

where

$$\frac{\partial \dot{V}_g}{\partial \chi} = \dot{\psi} V_a \cos(\chi - \psi)$$

$$\frac{\partial \dot{V}_g}{\partial \psi} = -\dot{\psi} V_a \cos(\chi - \psi)$$

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

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

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

GPS Smoothing – Simplified

The resulting measurement model is given by

$$y_{\text{GPS}} = h(x, u) + \eta_{\text{GPS}}$$

where $y_{\text{GPS}} = (y_{\text{GPS},n}, y_{\text{GPS},e}, y_{\text{GPS},V_g}, y_{\text{GPS},\chi})^\top$, and

$$h(x, u) = \begin{pmatrix} p_n \\ p_e \\ V_g \\ \chi \end{pmatrix}$$

and where the Jacobian is given by

$$\frac{\partial h}{\partial x}(\hat{x}, u) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

GPS Smoothing – Simplified

Wind calculations:

$$w_n = V_g \cos\chi - V_a \cos\psi$$
$$w_e = V_g \sin\chi - V_a \sin\psi$$

Simplified GPS Smoothing Results