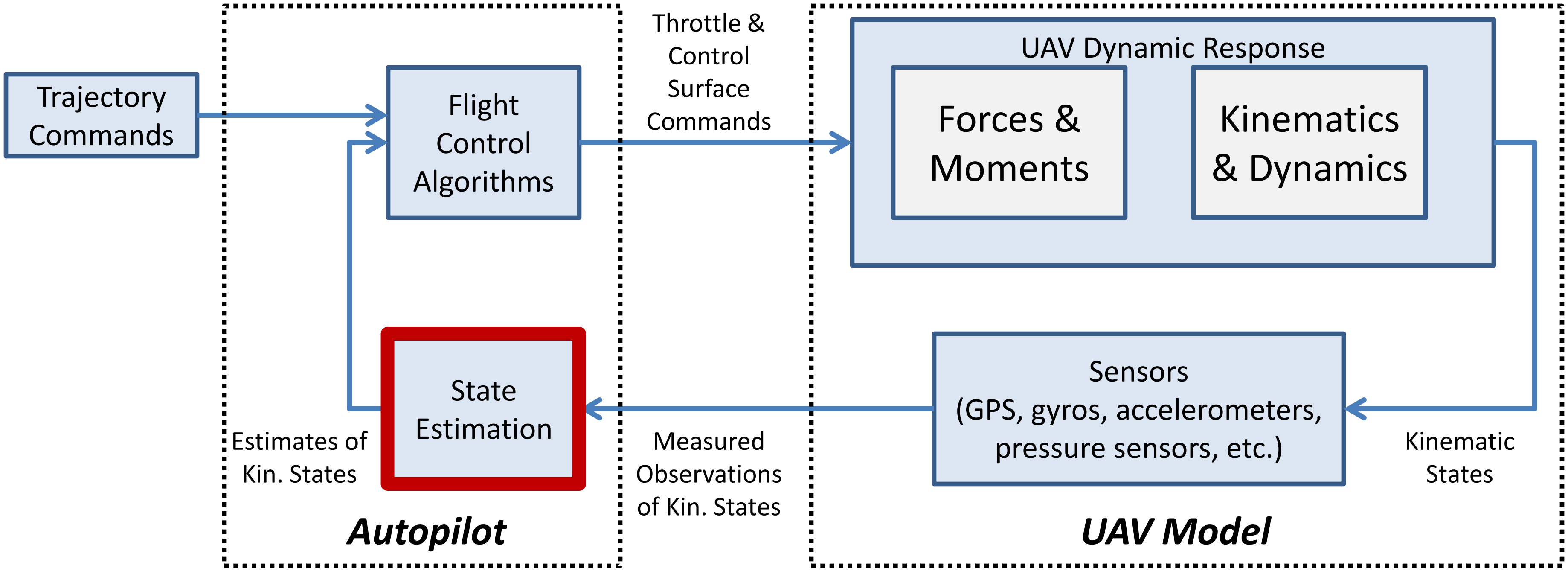
UAV Systems & Control
Lecture 10

State Estimates

Kalman Filtering Introduction

UAV System


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

- In the previous lectures we:
  - Developed models of common UAV sensors
- In this section we will:
  - Form state estimates from sensor measurements, where applicable
  - Introduce Kalman Filtering, which will be used to form additional state estimates

Estimating UAV States

- Flight control (autopilot) algorithms use state feedbacks for control
  - We don't actually know truth -> We need measurements or estimates!
- Some states are directly measured (albeit with noise or bias):
  - Gyros measure body rates (p, q, r)
  - Magnetic compass can measure yaw (ψ)
  - GPS measures position (p_n, p_e, p_d) and inertial velocity (v_n, v_e, v_d)
    - But often at a low rate (e.g. 1Hz-5Hz)
- Some states can be derived easily from measurements:
  - Barometric altitude estimate from absolute pressure sensor (h)
  - Airspeed estimate from differential pressure sensor (V_a)
- Some states aren't directly observed:
  - Roll (ϕ) and Pitch (θ)*!!!!!*
    - Estimated by *fusing* gyro and accelerometer measurements

Discrete Time Models

We have discussed Continuous Time state space models in depth

$$\dot{\underline{x}} = A\underline{x} + B\underline{u}$$

Differential Equation

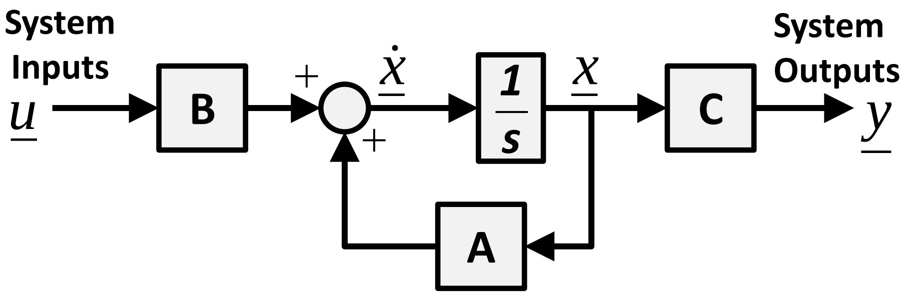
$$\underline{y} = C\underline{x} + D\underline{u}$$

$s:$ Derivative

$\frac{1}{s}:$ Integral

$$\Rightarrow \quad s\underline{x} = A\underline{x} + B\underline{u}$$

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



CT Block Diagram: Multiple-Input Multiple Output System (D is often zero)

Systems can be modeled as equivalent Discrete Time state space models

Discrete states: $\underline{x}[n] = \underline{x}(t_n),\ n = 1,2,3,\ldots$

$$\underline{x}[n+1] = \Phi\underline{x}[n] + \Gamma\underline{u}[n]$$

"Difference" Equation

$$\underline{y}[n] \quad = C\underline{x}[n] + D\underline{u}[n]$$

CT/DT Conversion details beyond our scope:

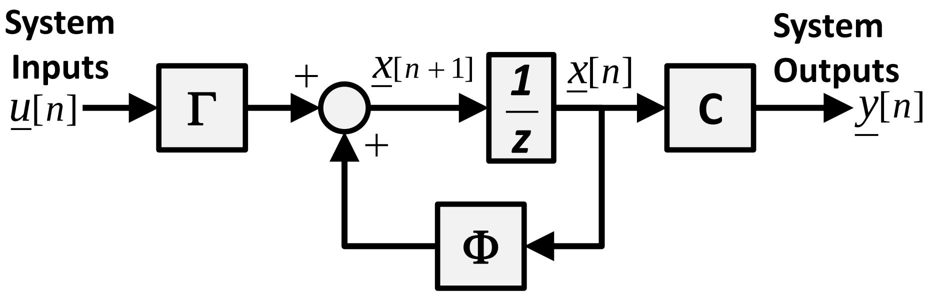
$$\{A,B\} \overset{\Delta t}{\leftrightarrow} \{\Phi,\Gamma\} \qquad \Delta t = t_{n+1} - t_n$$

$z:$ 1-step prediction

$\frac{1}{z}:$ 1-step delay

$$\Rightarrow \quad z\underline{x} = \Phi\underline{x} + \Gamma\underline{u}$$

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



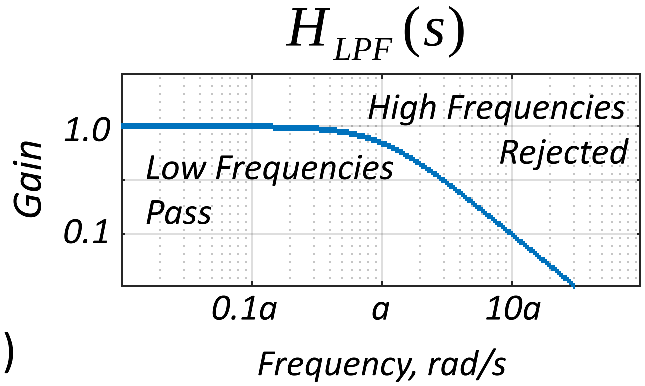
DT Block Diagram: Multiple-Input Multiple Output System (D is often zero)

Φ: State Transition Matrix

Maps states from time n to time n+1

Filtering Noisy Measurements

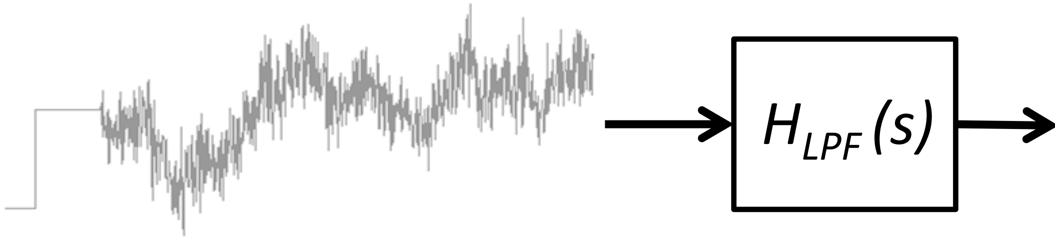
- Sensor measurements generally contain high frequency "noise" (Especially MEMS sensors)
- High frequency noise content needs to be filtered prior to use
  - Use a unity DC gain Low Pass Filter (LPF)

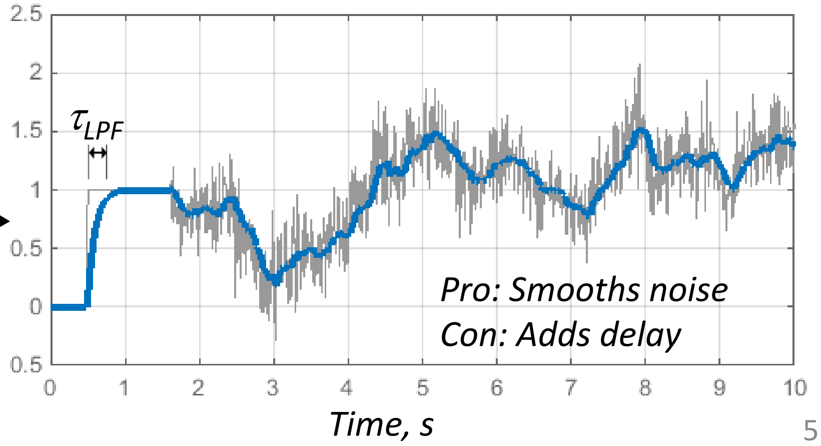


$$H_{LPF}(s) = \frac{a}{s+a} = \frac{1}{\tau_{LPF}\, s + 1}, \quad \tau_{LPF} = {}^{1}\!/_{a}$$

a : LPF bandwidth, rad/s
(LPF begins rejecting frequencies above *a* rad/s)

τ_{LPF}: LPF time constant, s
(LPF "averages" over τ_{LPF} seconds)





Pro: Smooths noise
Con: Adds delay

Low Pass Filter Implementation

Given a LPF: $H_{LPF}(s) = \frac{a}{s+a} = \frac{1}{\tau_{LPF} s + 1}, \quad \tau_{LPF} = {}^{1}\!/_{a}$

We can use the LPF on *raw* data to produce *filtered* data:

Laplace Domain: $Y_{filt}(s) = \frac{a}{s+a} Y_{raw}(s)$

Continuous Time Domain: $\dot{y}_{filt} = -a\, y_{filt} + a y_{raw}$

LPF implemented as a sampled-time discrete algorithm*:

$$y_{filt}[n+1] = \alpha_{LPF}\, y_{filt}[n] + \left(1 - \alpha_{LPF}\right) y_{raw}[n]$$

where:

$$\alpha_{LPF} = \exp\left(-aT_S\right) = \exp\left(-T_S \middle/ \tau_{LPF}\right)$$

a : LPF bandwidth, rad/s

τ_{LPF}: LPF time constant, s

T_S: Sample time for algorithm, s

Note: α_{LPF} acts as a weighting between the unfiltered data and the prior filtered result.

**Mathematical details for the conversion from a Laplace transfer function to a discrete time (z-Domain) difference equation are beyond the scope of the class. For those familiar, a “zero-order-hold” conversion was used.*

$$\frac{Y_{filt}(s)}{Y_{raw}(s)} = \frac{a}{s+a} \xrightarrow{zoh} \frac{Y_{filt}(z)}{Y_{raw}(z)} = \frac{1 - e^{-aT_S}}{z - e^{-aT_S}}$$

$$\Rightarrow Y_{filt}(z) = e^{-aT_S} z^{-1} Y_{filt}(z) + \left(1 - e^{-aT_S}\right) z^{-1} Y_{raw}(z)$$

$$\Rightarrow y_{filt}[n+1] = e^{-aT_S} y_{filt}[n] + \left(1 - e^{-aT_S}\right) y_{raw}[n]$$

State Estimates via Low Pass Filtering

We can estimate some states via filtering

Measurement Models:		**State Estimates:**
$y_{gyro,x} = p + \eta_{gyro,x}$	⇒	$\hat{p} = LPF\left(y_{gyro,x}\right)$
$y_{gyro,y} = q + \eta_{gyro,y}$	⇒	$\hat{q} = LPF\left(y_{gyro,y}\right)$
$y_{gyro,z} = r + \eta_{gyro,z}$	⇒	$\hat{r} = LPF\left(y_{gyro,z}\right)$
$y_{mag,2D} = \psi + \beta_{mag} + \eta_{mag}$	⇒	$\hat{\psi} = LPF\left(y_{mag,2D}\right)$
$y_{\text{absPress}} = P_0 e^{\frac{-Mg}{RT_{UAV}}\left(h_{Launch,ASL} + h\right)} + \beta_{\text{absPress}} + \eta_{\text{absPress}}$ *Using constant temperature assumption*	⇒	$\hat{h} = \frac{-RT_{UAV}}{Mg} \ln\left(\frac{LPF\left(y_{\text{absPress}}\right)}{P_{Launch}}\right)$ P_{Launch}: Pressure at Launch Alt
$y_{\text{diffPress}} = \frac{1}{2}\rho V_a^2 + \beta_{\text{diffPress}} + \eta_{\text{diffPress}}$	⇒	$\hat{V}_a = \sqrt{\frac{2}{\rho} LPF\left(y_{\text{diffPress}}\right)}$

Note 1: *LPF(y(t))* denotes a Low-Pass Filtering of *y(t)*

Note 2: Low-Pass Filtering does *not* remove biases

How do we estimate Roll and Pitch?

- None of our sensors directly measure roll or pitch, but two are related:
  - Gyros measure body rates (*p,q,r*) which are related to the derivative of roll and pitch
    - We could integrate *p* and *q* to estimate roll and pitch, but biases and errors would accrue
    - Essentially, gyros observe higher frequency roll & pitch dynamics
  - Non-maneuvering accelerometers can directly observe roll and pitch due to the orientation of gravity relative to the accelerometer triad

$$\begin{aligned} y_{accel,x} &= \dot{u} + qw - rv + g\sin\theta + \eta_{accel,x}\\ y_{accel,y} &= \dot{v} + ru - pw - g\cos\theta\sin\phi + \eta_{accel,y}\\ y_{accel,z} &= \underbrace{\dot{w} + pv - qu}_{\text{Dynamics}} \underbrace{- g\cos\theta\cos\phi}_{\text{Gravity Vector}} + \eta_{accel,z}\end{aligned} \xrightarrow{\text{During periods of constant velocity}} \begin{aligned}\theta &\cong \operatorname{asin}\left(\frac{y_{accel,x}}{g}\right)\\ \phi &\cong \operatorname{atan2}\left(-y_{accel,y}, -y_{accel,z}\right)\end{aligned}$$

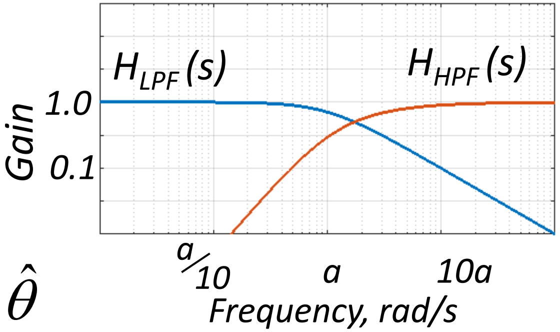
- Essentially, accelerometers observe lower frequency roll & pitch dynamics

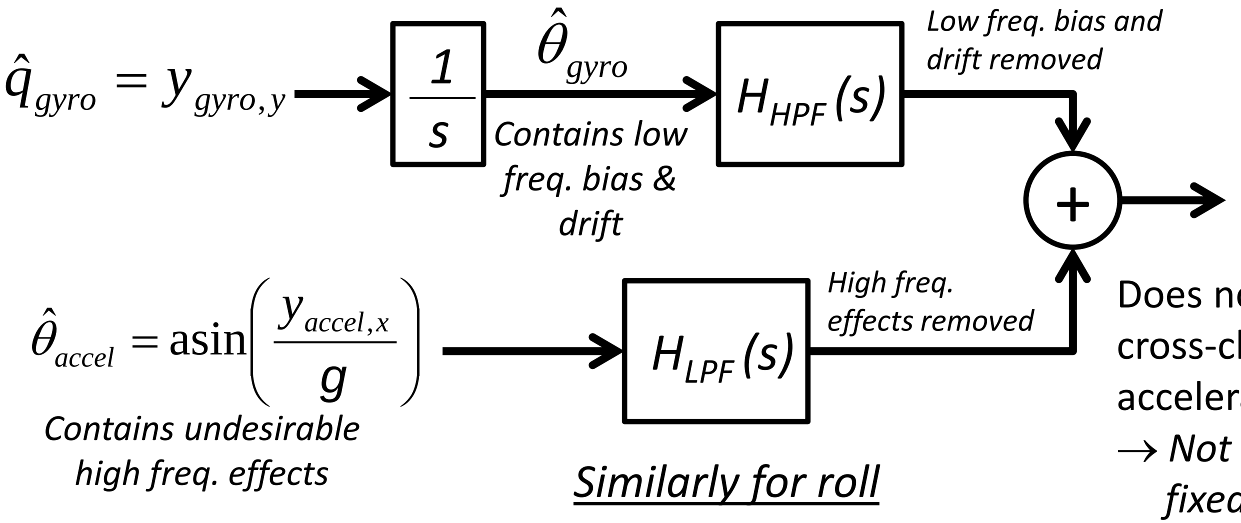
- To estimate roll and pitch, we must combine:
  - Low frequency accelerometer observations
  - High frequency gyro observations

How do we estimate Roll and Pitch?

- Option 1: Simple complementary filters
  - Combine:
    - Low-pass-filtered attitude estimate from accelerometers
    - High-pass-filtered attitude estimate (via integration) from gyro
  - Similar pitch and roll filters
  - Minimally suitable for quadcopter, not for fixed wing
    - e.g. Axial acceleration mucks up pitch estimate

$H_{HPF}(s) = 1 - H_{LPF}(s)$





$\hat{q}_{gyro} = y_{gyro,y}$

$\frac{1}{s}$

$\hat{\theta}_{gyro}$

Contains low freq. bias & drift

$H_{HPF}(s)$

Low freq. bias and drift removed

$+$

$\hat{\theta}$

$\hat{\theta}_{accel} = \operatorname{asin}\left(\frac{y_{accel,x}}{g}\right)$

Contains undesirable high freq. effects

$H_{LPF}(s)$

High freq. effects removed

Similarly for roll

Does not account for Coriolis effect, cross-channel effects, or translational accelerations

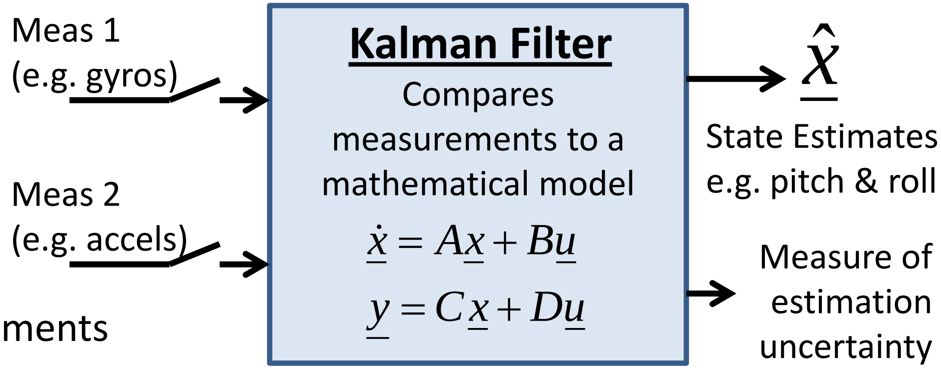
→ *Not valid for fixed wing vehicles*

Note: For a more complete closed-loop Complementary Filter sufficient for use in fixed-wing UAVs, see “AHRS Feedback Controller” in:

Barton, J. D., “Fundamentals of Small Unmanned Aircraft”, JHU/APL Tech Digest, V31-2, 2012

How do we estimate Roll and Pitch?

- Option 2: Kalman Filtering
  - A Kalman Filter is an *optimal* estimator
    - Compares measurements to a mathematical model of a system
    - Infers parameter (i.e. state) estimates from:
      - Indirect and/or noisy measurements
    - Recursive
      - Process measurements whenever they are available
      - Does not require storing previous measurements
    - Optimality?
      - Minimizes the mean square error of state estimates if system is linear and noise sources are Gaussian
  - What's it good for?
    - Estimating states given:
      - noisy measurements
      - *indirect* measurements
      - periodic or aperiodic measurements
    - Fusing multiple *types* of measurements
    - Predicting states *between* measurements
    - Providing a measure of estimation uncertainty



Meas 1
(e.g. gyros)

Meas 2
(e.g. accels)

Kalman Filter

Compares measurements to a mathematical model

$$\dot{\underline{x}} = A\underline{x} + B\underline{u}$$

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

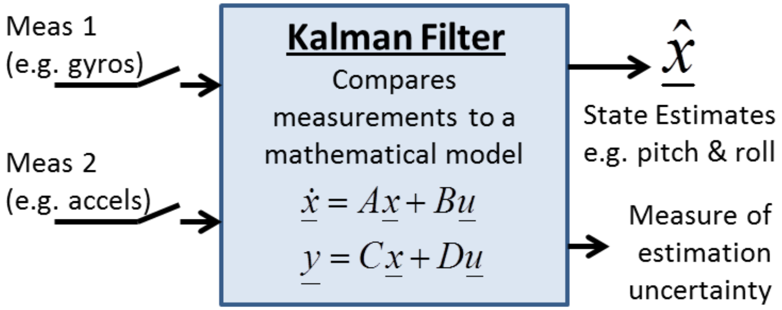
$\hat{\underline{x}}$

State Estimates
e.g. pitch & roll

Measure of estimation uncertainty

Outline for Kalman Filter Introduction

- Background:
  - Discrete form developed by Rudolph E. Kalman around 1960 in Baltimore, MD
  - Continuous form developed by Kalman and Bucy (JHU/APL) in 1961
  - "Extended" to non-linear systems by NASA (used for Apollo circumlunar navigation, etc.)
  - Used extensively in navigation/aerospace/RADAR/SONAR/tracking/financial applications
- Kalman Filtering is a very advanced topic
  - Relatively easy to implement, but mathematically intensive to derive/prove
- Introduction Outline
  - Discuss "Dynamic Observer Theory"
    - *A Kalman Filter is a "statistically optimal" Dynamic Observer*
  - Introduce necessary Statistics topics
    - *Scalar & Multivariate*
  - Present & Derive Linear Kalman Filter
    - *Variant: Continuous-Discrete Method*
  - Extend Linear KF to non-linear case
    - *EKF: Extended Kalman Filter*
  - Discuss EKF Tuning
- Next week: Apply EKF to UAV state estimation



Meas 1 (e.g. gyros)

Meas 2 (e.g. accels)

Kalman Filter

Compares measurements to a mathematical model

$$\dot{\underline{x}} = A\underline{x} + B\underline{u}$$

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

$\underline{\hat{x}}$

State Estimates e.g. pitch & roll

Measure of estimation uncertainty

Dynamic Observer Theory

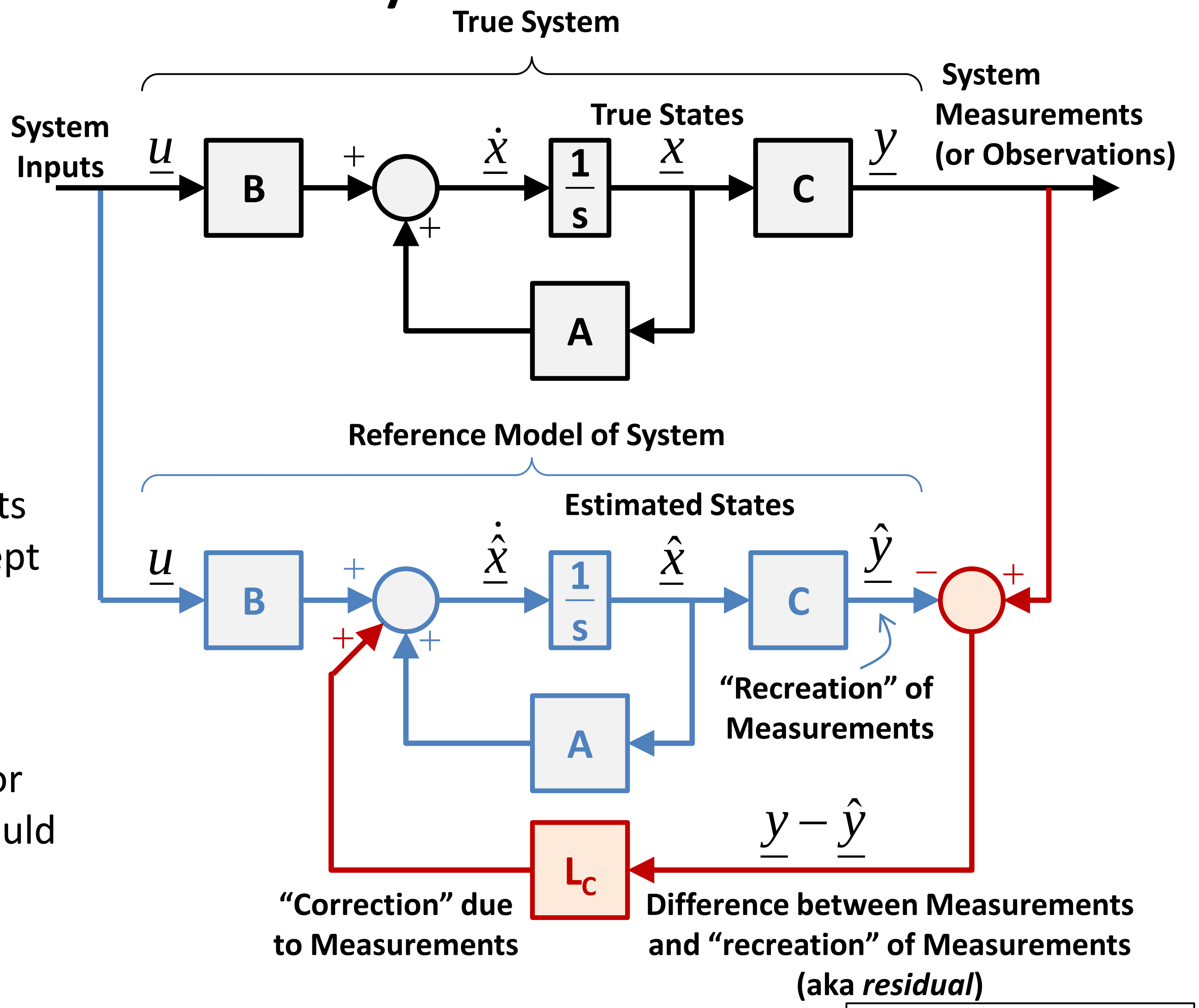
Given a state-space representation of a true system from "inputs" to "measurements":

$$\underline{\dot{x}} = A\underline{x} + B\underline{u} \qquad \underline{y} = C\underline{x}$$

An exact duplicate of the system (a reference model) should yield the same results given the same inputs (except for non-zero initial states):

$$\underline{\dot{\hat{x}}} = A\underline{\hat{x}} + B\underline{u} \qquad \underline{\hat{y}} = C\underline{\hat{x}}$$

But any model differences or unmodeled initial states would cause a divergence.


True System
System Inputs
$\underline{u}$
B
+
$\underline{\dot{x}}$
True States
$\frac{1}{s}$
$\underline{x}$
C
$\underline{y}$
System Measurements (or Observations)
+
A
Reference Model of System
Estimated States
$\underline{u}$
B
+
$\underline{\dot{\hat{x}}}$
$\frac{1}{s}$
$\underline{\hat{x}}$
C
$\underline{\hat{y}}$
−
+
+
+
A
"Recreation" of Measurements
$\underline{y} - \underline{\hat{y}}$
L_C
"Correction" due to Measurements
Difference between Measurements and "recreation" of Measurements (aka *residual*)

Dynamic Observer: Use a feedback gain (L_C) to "correct" the reference model based on measurements:

"Reference" model **"Correction" to model**

$$\underline{\dot{\hat{x}}} = A\underline{\hat{x}} + B\underline{u} \;+\; L_C\left(\underline{y} - \underline{\hat{y}}\right)$$

Desire: Choose L_C so that:

$$\underline{y} - \underline{\hat{y}} \rightarrow \underline{0}$$

A Kalman Filter is an "optimal" selection of "correction" gain L_C.

DT Dynamic Observer

Continuous-Time Dynamic Observer:

Use a feedback gain (L_C) to "correct" the reference model based on measurements:

$$\dot{\hat{\underline{x}}} = A\hat{\underline{x}} + B\underline{u} \quad + \quad L_C\left(\underline{y} - \hat{\underline{y}}\right)$$

"Reference" model **"Correction" to model**

Discrete-Time Dynamic Observer:

Can implement equivalent in discrete time using gain L:

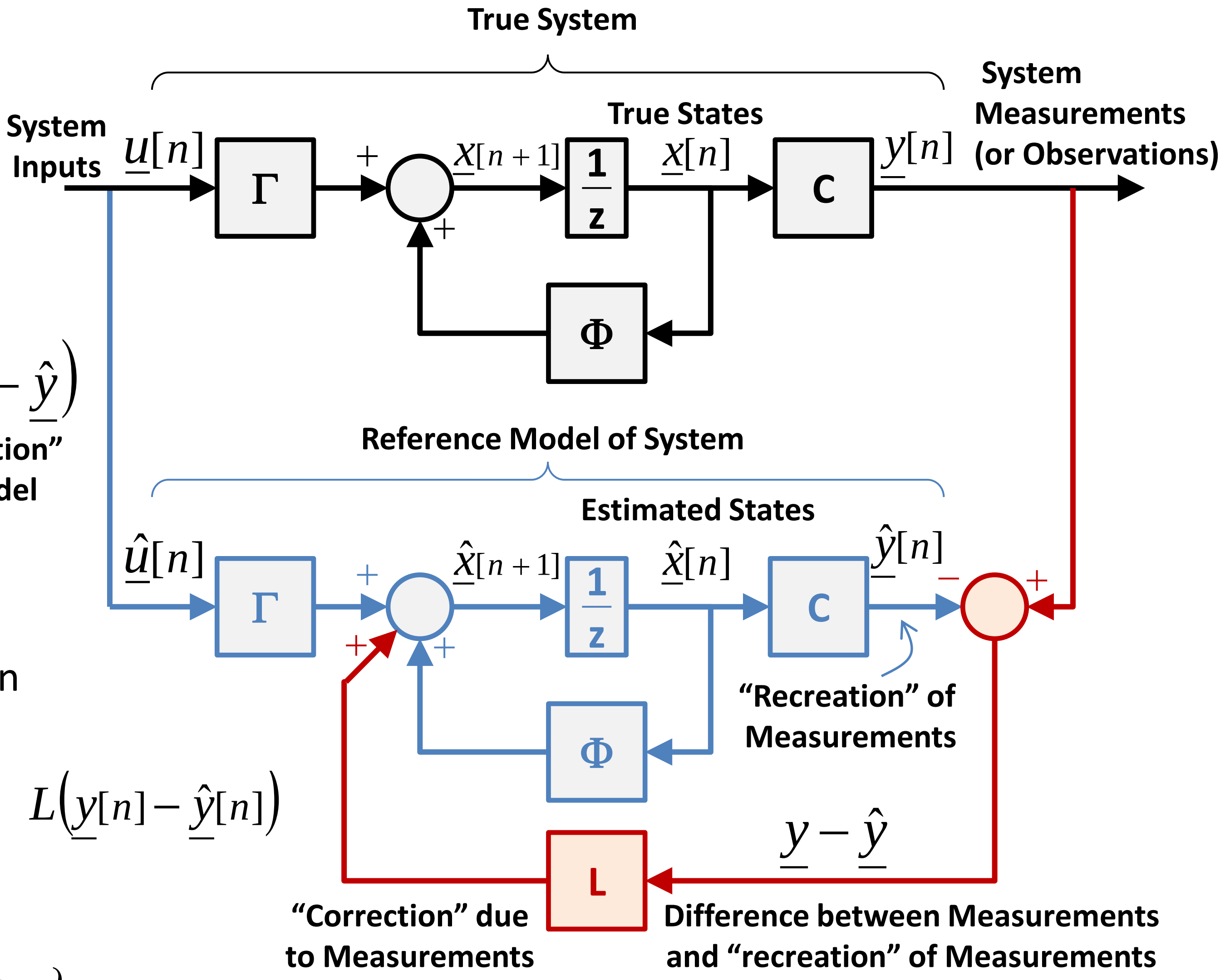
$$\hat{\underline{x}}[n+1] = \Phi\hat{\underline{x}}[n] + \Gamma\underline{u}[n] \quad + \quad L\left(\underline{y}[n] - \hat{\underline{y}}[n]\right)$$

CT/DT Conversion*:

$$\{A, B, L_C\} \underset{\Delta t}{\leftrightarrow} \{\Phi, \Gamma, L\}$$

Continuous Time Model **Discrete Time Model (function of *Δt*)**

Discrete-Time Equivalent


True System
System Inputs
$\underline{u}[n]$
Γ
+
+
$\underline{x}[n+1]$
$\frac{1}{z}$
True States
$\underline{x}[n]$
C
$\underline{y}[n]$
System Measurements (or Observations)
Φ
Reference Model of System
Estimated States
$\hat{\underline{u}}[n]$
Γ
+
+
+
$\hat{\underline{x}}[n+1]$
$\frac{1}{z}$
$\hat{\underline{x}}[n]$
C
$\hat{\underline{y}}[n]$
−
+
"Recreation" of Measurements
Φ
$\underline{y} - \hat{\underline{y}}$
L
"Correction" due to Measurements
Difference between Measurements and "recreation" of Measurements

*Details of CT/DT conversion are beyond scope of discussion

Continuous-Discrete Dynamic Observer Implementation

<u>Continuous-Time Version</u>

$$\dot{\hat{\underline{x}}} = A\hat{\underline{x}} + B\underline{u} \quad + \quad L_C\left(\underline{y} - \hat{\underline{y}}\right)$$

"Reference" **"Correction"**

<u>Discrete-Time Version</u>

$$\hat{\underline{x}}[n+1] = \Phi\hat{\underline{x}}[n] + \Gamma\underline{u}[n] \quad + \quad L\left(\underline{y}[n] - \hat{\underline{y}}[n]\right)$$

"Reference" **"Correction"**

- A convenient *hybrid* variant to use for sampled data systems is the Continuous-Discrete implementation:

<u>Between measurements</u>: $\dot{\hat{\underline{x}}} = A\hat{\underline{x}} + B\underline{u}$ *Propagate the reference model using the Continuous method*

<u>At measurements</u>: $\hat{\underline{x}}^{+} = \hat{\underline{x}}^{-} \quad + \quad L\left(\underline{y}[n] - C\hat{\underline{x}}^{-}\right)$ *"Correct" the system using the Discrete method*

Estimate at t_n *after* correction *(a posteriori)*

Estimate at t_n *before* correction *(a priori)*

Measurement at time t_n

Recreation of measurement using model

- Extending the Continuous-Discrete variant for a <u>non-linear system</u>:

<u>Between measurements</u>: $\dot{\hat{\underline{x}}} = f\left(\hat{\underline{x}}, \underline{u}\right)$

<u>At measurements</u>: $\hat{\underline{x}}^{+} = \hat{\underline{x}}^{-} \quad + \quad L\left(\underline{y}[n] - h(\hat{\underline{x}}^{-}, \underline{u}[n])\right)$

Dynamic Observer Example

Continuous-Discrete variant for a non-linear system:

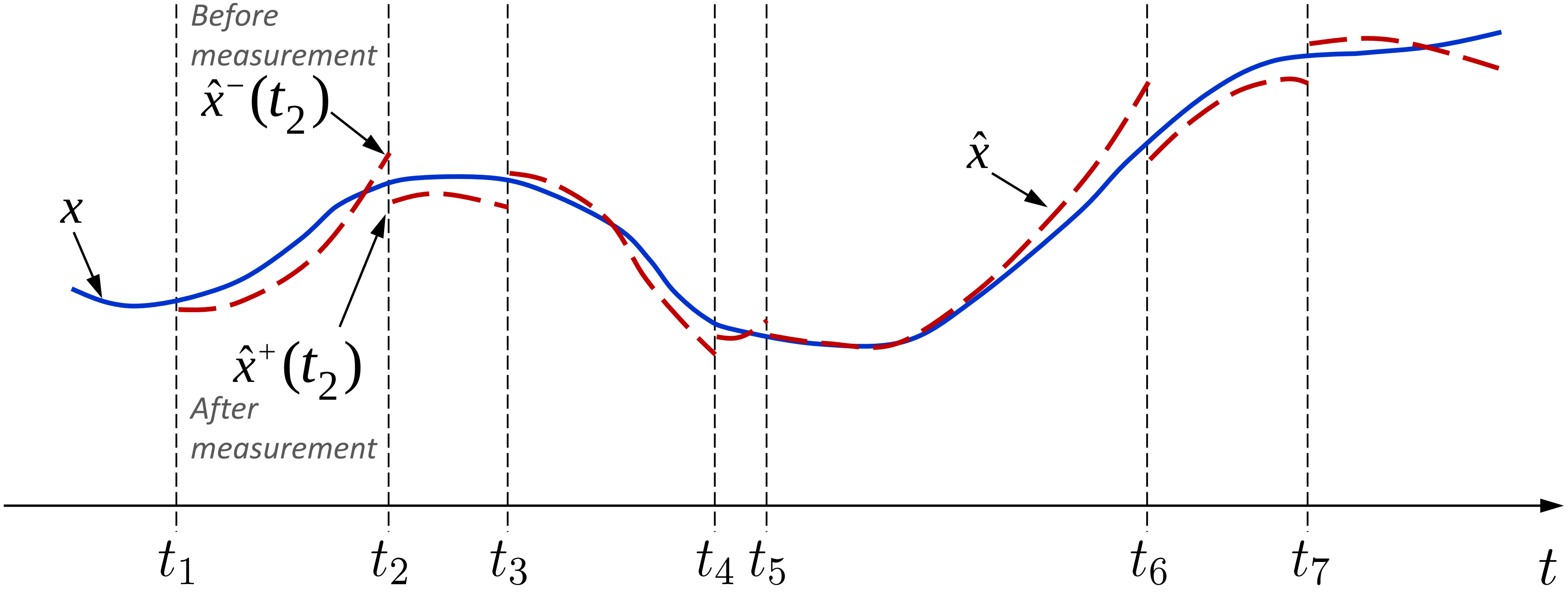
Between measurements: $\dot{\hat{\underline{x}}} = f\left(\hat{\underline{x}}, \underline{u}\right)$

At measurements: $\hat{\underline{x}}^{+} = \hat{\underline{x}}^{-} + L\left(\underline{y}[n] - h(\hat{\underline{x}}^{-}, \underline{u}[n])\right)$

Actual measurement at time t_n

Recreation of measurement using model

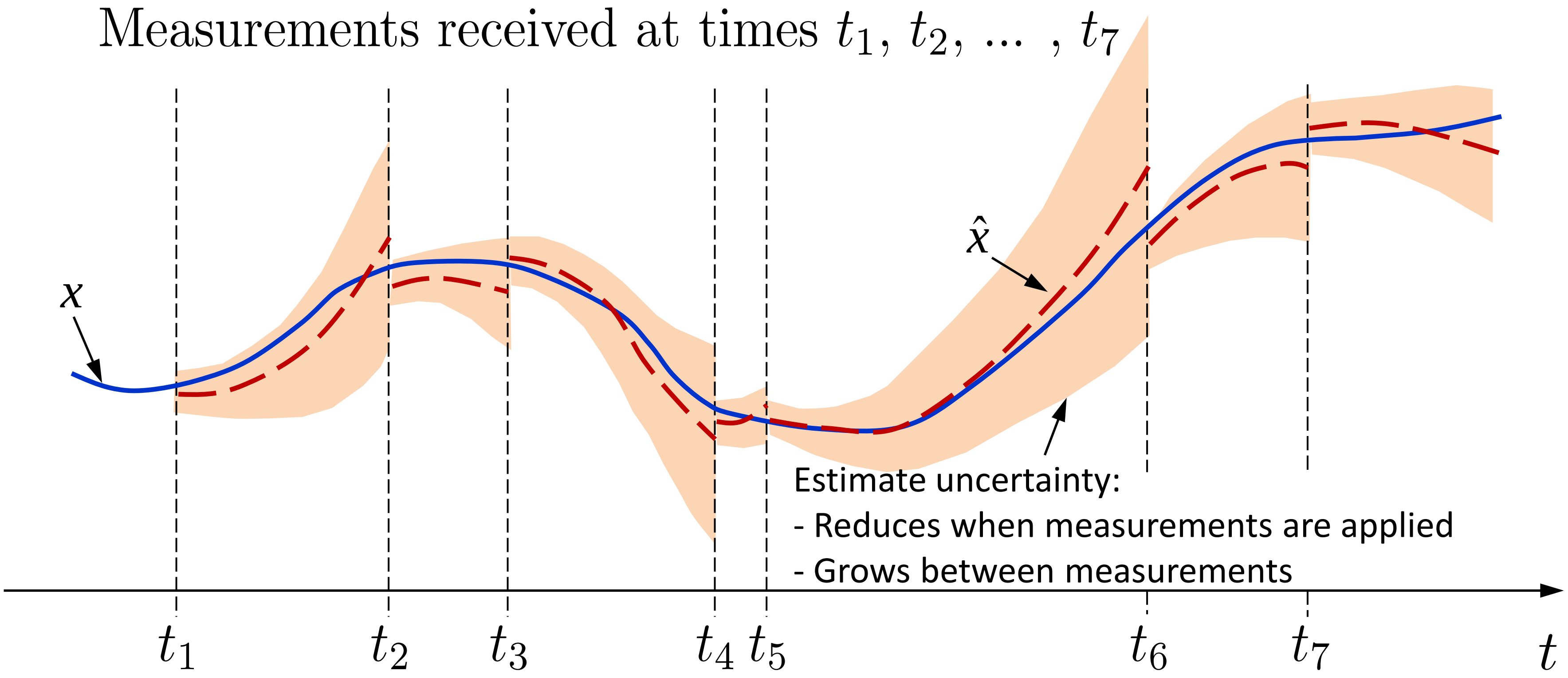
Measurements received at times t_1, t_2, ... , t_7



Kalman Filter

A Kalman Filter follows the same process as a Dynamic Observer, except:

- It determines the feedback gain matrix, *L*, that is statistically optimal at each time step (given specific assumptions)
- It has a built-in measure of estimation uncertainty
  - via an error covariance matrix



Measurements received at times t_1, t_2, ... , t_7

Estimate uncertainty:
- Reduces when measurements are applied
- Grows between measurements

Statistics Primer for Kalman Filtering

- Scalar Statistics
  - Consider two scalar random variables *x* and *y*

    Expected Value (aka mean or average): $E\{x\}=\mu_x$

    Variance: $Var\{x\} = E\{ (x- \mu_x)(x- \mu_x) \} = E\{x^2\} - \mu_x^{\,2} = \sigma_x^{\,2}$

    $Var\{x\} = E\{x^2\} \quad if\ \mu_x = 0$

    Standard Deviation: sqrt($Var\{x\}$) = σ_x

    Covariance: $Cov\{x,y\} = E\{ (x- \mu_x)(y- \mu_y) \} = E\{xy\} - E\{x\}E\{y\} = \rho_{xy}\sigma_x\, \sigma_y$

    ρ_{xy} *: Correlation coefficient between x and y,* $-1 \leq \rho_{xy} \leq 1$

    $Cov\{x,y\} = 0$ *if x and y are independent random variables*

  - Special case: Gaussian (aka normal) distribution
    - *Gaussian* variables have a probability distribution function (pdf) of: $f_X(x) = \dfrac{1}{\sqrt{2\pi}\sigma_X} e^{-\frac{(x-\mu_X)^2}{\sigma_X^2}}$

Given constant *a:*

$E\{ax\}=a\mu_x$

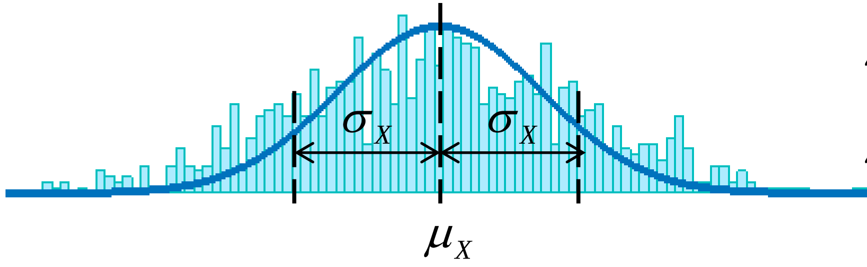
$E\{x+a\}=\mu_x+a$

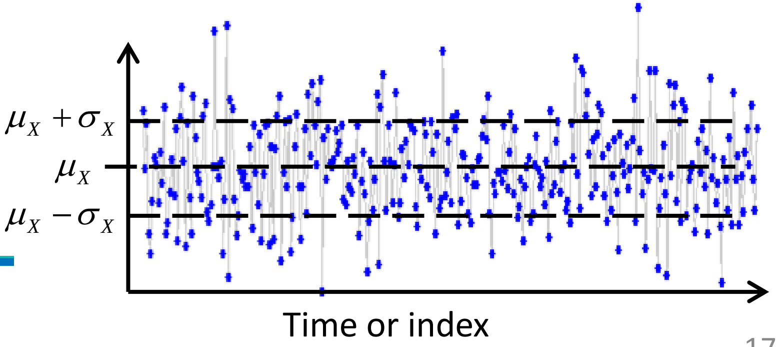
$E\{x+y\}=\mu_x+\mu_y$

$Var\{ax\}=a^2\sigma_x^{\,2}$

$Var\{x+a\}=\sigma_x^{\,2}$

Gaussian pdf compared with example histogram





Statistics Primer for Kalman Filtering

- Multivariate Statistics
  - Consider a random vector $\underline{x}$ consisting of n random variables

$$\underline{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \qquad E\{\underline{x}\} = \begin{bmatrix} E\{x_1\} \\ E\{x_2\} \\ \vdots \\ E\{x_n\} \end{bmatrix} = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_n \end{bmatrix} = \underline{\mu} \qquad \underline{\sigma} = \begin{bmatrix} \sqrt{Var\{x_1\}} \\ \sqrt{Var\{x_2\}} \\ \vdots \\ \sqrt{Var\{x_n\}} \end{bmatrix} = \begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_n \end{bmatrix}$$

Covariance Matrix:

$$Cov\{\underline{x},\underline{x}\} = E\{(\underline{x}-\underline{\mu})(\underline{x}-\underline{\mu})^T\} = \begin{bmatrix} \sigma_1^2 & \rho_{12}\sigma_1\sigma_2 & \dots & \rho_{1n}\sigma_1\sigma_n \\ \rho_{12}\sigma_1\sigma_2 & \sigma_2^2 & \dots & \rho_{2n}\sigma_2\sigma_n \\ \vdots & \vdots & \ddots & \vdots \\ \rho_{1n}\sigma_1\sigma_n & \rho_{2n}\sigma_2\sigma_n & \dots & \sigma_n^2 \end{bmatrix}$$

$-1 \le \rho_{xy} \le 1$

Covariance Matrix of $\underline{x}$

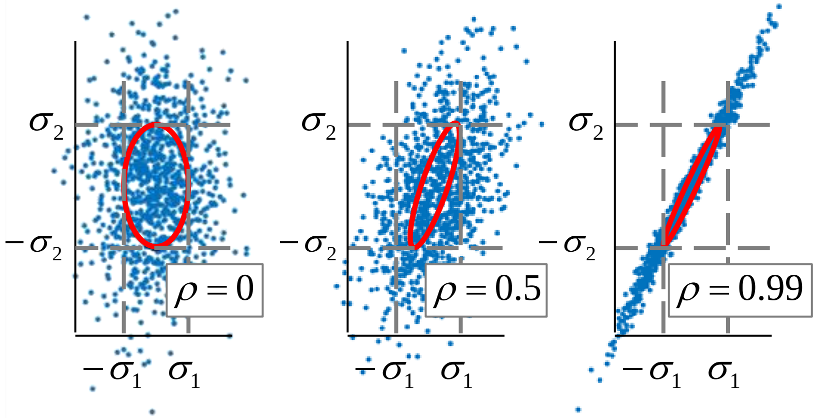
- Diagonal terms are variances of each variable in $\underline{x}$
- Off-diagonal terms are covariances between each variable in $\underline{x}$
- Covariance matrices are
  - Symmetric
  - Diagonals are > zero

- Special Cases
  - If $\underline{\mu} = \underline{0}$: $Cov\{\underline{x},\underline{x}\} = E\{\underline{x}\underline{x}^T\}$
  - If all variables in $\underline{x}$ are independent, then all off-diagonals are zero:

$$Cov\{\underline{x},\underline{x}\} = \begin{bmatrix} \sigma_1^2 & 0 & \dots & 0 \\ 0 & \sigma_2^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma_n^2 \end{bmatrix}$$

Example 2D Gaussian zero-mean distributions:

$$Cov\{\underline{x},\underline{x}\} = \begin{bmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{bmatrix}$$

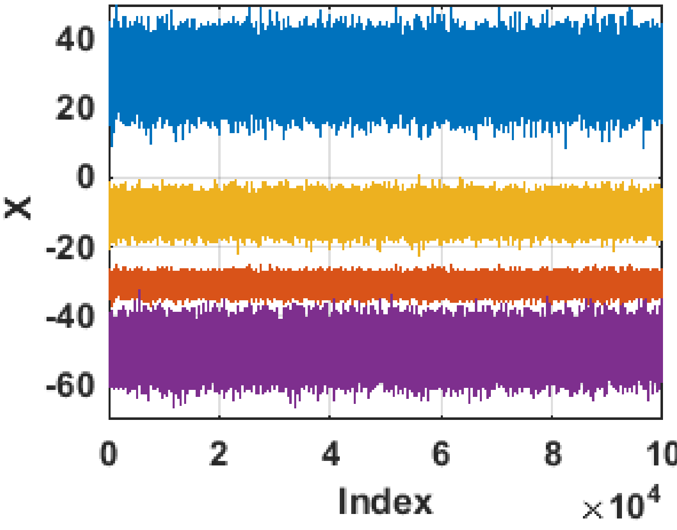


Statistics Primer for Kalman Filtering

- Multivariate Statistics Example in Matlab

```
% Define 4 random row vectors (of length N)
N=100000;
x1 = 5*randn(1,N) + 30;
x2 = .3*x1 - 40;
x3 = .5*(x2 - x1) + 2*randn(1,N) + 20;
x4 = 4*randn(1,N)-50;

% Make the 4-D multivariate random vector X
X=[x1; x2; x3; x4;];
```



x1: μ= 30, σ= 5

x3: μ= -10.5, σ= 2.65

x2: μ= -31, σ= 1.5

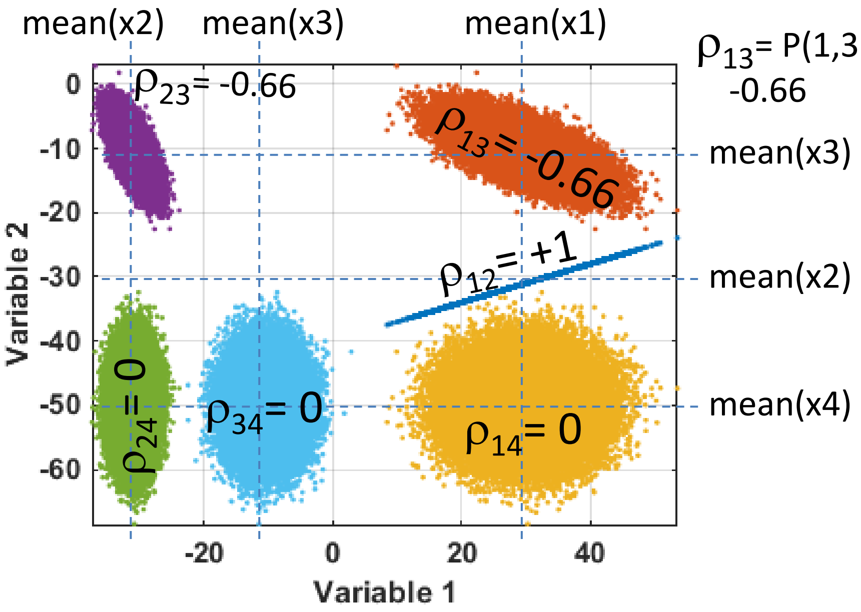
x4: μ= -50, σ= 4

```
mu = mean(X')
 29.9973  -31.0008  -10.4996  -49.9569

sigma = std(X')
 5.0025    1.5008    2.6529    3.9978

sigma_squared = sigma.^2   % variance
 25.0252    2.2523    7.0381   15.9823

P = cov(X')
   25.0252    7.5076   -8.7358   -0.0095
    7.5076    2.2523   -2.6207   -0.0028
   -8.7358   -2.6207    7.0381    0.0182
   -0.0095   -0.0028    0.0182   15.9823
```



ρ_{13}= P(1,3)/sqrt(P(1,1)*P(3,3))
-0.66

Linear Kalman Filter

System we want to estimate
(Linear with random variations)

$$\dot{\underline{x}} = A\underline{x} + B\underline{u} + \underline{\xi}$$

$$\underline{y}_i[n] = C_i\underline{x}[n] + \underline{\eta}_i[n]$$

Sampled measurement for sensor i at time t *t_n: $\underline{y}_i[n] = \underline{y}_i(t_n)$*

Reference model with state estimates:

$$\dot{\hat{\underline{x}}} = A\hat{\underline{x}} + B\underline{u}$$

$$\hat{\underline{y}}_i[n] = C_i\hat{\underline{x}}[n]$$

Error Covariance Matrix

$$P(t) \equiv E\{(\underline{x} - \hat{\underline{x}})(\underline{x} - \hat{\underline{x}})^T\}$$

Kalman Filter as a Dynamic Observer:

- L_i chosen at each measurement to minimize *trace(P)*

Process Noise Vector: $\underline{\xi}$ — *Represents modeling errors and unmodeled disturbances*

$$E\{\underline{\xi}\} = \underline{0}$$
$$Q \equiv E\{\underline{\xi}\,\underline{\xi}^T\}$$

Sampled Measurement Noise Vector: $\underline{\eta}_i[n]$ — *Represents noise in the sensor measurements*

$$E\{\underline{\eta}_i[n]\} = \underline{0}$$
$$R_i \equiv E\{(\underline{\eta}_i[n])(\underline{\eta}_i[n])^T\}$$

$\underline{\xi}$ & $\underline{\eta}_i[n]$ *are indep. & Gaussian*

Initialization: $\hat{\underline{x}} = \hat{\underline{x}}_0 \qquad P = P_0$

Tuning Parameters: $\hat{\underline{x}}_0, P_0, Q, R$

***State estimate propagation between measurements*:**

$$\dot{\hat{\underline{x}}} = A\hat{\underline{x}} + B\underline{u}$$ *Propagate estimates*

$$\dot{P} = AP + PA^T + Q$$ *Propagate uncertainties & add process noise*

***State estimate correction at measurement for sensor i*:**

$$L_i = P^- C_i^T\left(C_i P^- C_i^T + R_i\right)^{-1}$$ *Make Kalman Gain Matrix*

$$\hat{\underline{x}}^+ = \hat{\underline{x}}^- + L_i\left(\underline{y}_i[n] - C_i\hat{\underline{x}}^-\right)$$ *Correct estimates with measurement*

$$P^+ = (I - L_iC_i)P^-$$ *Reduce uncertainties due to info gained from meas.*

Linear Kalman Filter Understanding

System:

$$\dot{\underline{x}} = A\underline{x} + B\underline{u} + \underline{\xi}$$

$$\underline{y}_i[n] = C_i\underline{x}[n] + \underline{\eta}_i[n]$$

Covariance Matrices:

$$Q \equiv E\{\underline{\xi}\,\underline{\xi}^T\}$$

$$R_i \equiv E\{(\underline{\eta}_i[n])(\underline{\eta}_i[n])^T\}$$

$$P(t) \equiv E\{(\underline{x} - \hat{\underline{x}})(\underline{x} - \hat{\underline{x}})^T\}$$

State estimate propagation between measurements:

$$\dot{\hat{\underline{x}}} = A\hat{\underline{x}} + B\underline{u}$$

$$\dot{P} = AP + PA^T + Q$$

State estimate correction at measurement for sensor i:

$$L_i = P^- C_i^T \left(C_i P^- C_i^T + R_i\right)^{-1}$$

$$\hat{\underline{x}}^+ = \hat{\underline{x}}^- + L_i\left(\underline{y}_i[n] - C_i\hat{\underline{x}}^-\right)$$

$$P^+ = (I - L_i C_i)P^-$$

P(t) is the error covariance matrix, representing uncertainty

- *P* increases between measurements (increasing uncertainty)
- *P* decreases when measurements applied (gained information)

L is the Kalman Gain Matrix

- "Corrects" state estimate using a measurement
- Provides a "weighting" between estimates and measurements

$$\hat{\underline{x}}^+ = \hat{\underline{x}}^- + \underbrace{\left[P^- C_i^T (C_i P^- C_i^T + R_i)^{-1}\right]}_{\text{Kalman Gain Matrix, } L_i}\underbrace{\left(\underline{y}_i[n] - C_i\hat{\underline{x}}^-\right)}_{\text{Measurement } \textit{residual}}$$

Kalman Gain is:

- *Large* when $C_i P^- C_i^T >> R_i$ *(Filter trusts measurement over current estimate)*
- *Small* when $C_i P^- C_i^T << R_i$ *(Filter trusts current estimate over measurement)*

Linear Kalman Filter Understanding

System:

$$\dot{\underline{x}} = A\underline{x} + B\underline{u} + \underline{\xi}$$

$$\underline{y}_i[n] = C_i\,\underline{x}[n] + \underline{\eta}_i[n]$$

Covariance Matrices:

$$Q \equiv E\{\underline{\xi}\,\underline{\xi}^T\}$$

$$R_i \equiv E\{(\underline{\eta}_i[n])(\underline{\eta}_i[n])^T\}$$

$$P(t) \equiv E\{(\underline{x} - \underline{\hat{x}})(\underline{x} - \underline{\hat{x}})^T\}$$

State estimate propagation between measurements:

$$\dot{\underline{\hat{x}}} = A\underline{\hat{x}} + B\underline{u}$$

$$\dot{P} = AP + PA^T + Q$$

State estimate correction at measurement for sensor i:

$$L_i = P^- C_i^T \left(C_i P^- C_i^T + R_i\right)^{-1}$$

$$\underline{\hat{x}}^+ = \underline{\hat{x}}^- + L_i\left(\underline{y}_i[n] - C_i\,\underline{\hat{x}}^-\right)$$

$$P^+ = \left(I - L_i C_i\right)P^-$$

Scalar Correction Example

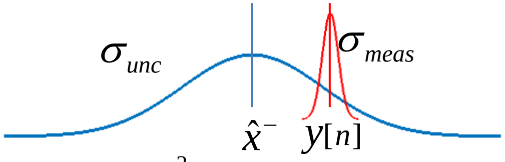
- *x* is a temperature we want to estimate
- *P* is our estimation uncertainty: $P = \sigma_{unc}^2$
- *y* is a noisy thermometer measurement: *y = x + noise,* with meas. uncertainty: $R = \sigma_{meas}^2$ $\quad Note:\ C = 1$

Scalar values, so:

$$\hat{x}^+ = \hat{x}^- + \frac{P^- C^T}{C\,P^- C^T + R}\left(y[n] - C\hat{x}^-\right)$$

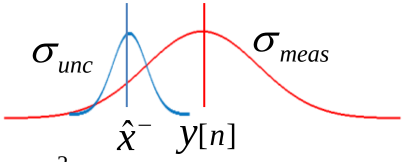
$$\hat{x}^+ = \hat{x}^- + \frac{\sigma_{unc}^2}{\sigma_{unc}^2 + \sigma_{meas}^2}\left(y[n] - \hat{x}^-\right)$$

Case 1: $\sigma_{meas}^2 << \sigma_{unc}^2$



$$\Rightarrow \frac{\sigma_{unc}^2}{\sigma_{unc}^2 + \sigma_{meas}^2} \approx 1 \;\Rightarrow\; \hat{x}^+ \approx y[n]$$

Case 2: $\sigma_{meas}^2 >> \sigma_{unc}^2$



$$\Rightarrow \frac{\sigma_{unc}^2}{\sigma_{unc}^2 + \sigma_{meas}^2} \approx 0 \;\Rightarrow\; \hat{x}^+ \approx \hat{x}^-$$

➡ *L* provides a weighting specifying how much it trusts the measurement versus the current state estimate!

Linear Kalman Filter Derivation Outline

System:

$$\dot{\underline{x}} = A\underline{x} + B\underline{u} + \underline{\xi}$$

$$\underline{y}_i[n] = C_i\,\underline{x}[n] + \underline{\eta}_i[n]$$

Covariance Matrices:

$$Q \equiv E\{\underline{\xi}\,\underline{\xi}^T\}$$

$$R_i \equiv E\{(\underline{\eta}_i[n])(\underline{\eta}_i[n])^T\}$$

$$P(t) \equiv E\{(\underline{x} - \hat{\underline{x}})(\underline{x} - \hat{\underline{x}})^T\}$$

***State estimate propagation between measurements*:**

$$\dot{\hat{\underline{x}}} = A\hat{\underline{x}} + B\underline{u}$$

$$\boxed{\dot{P} = AP + PA^T + Q}$$

***State estimate correction at measurement for sensor i*:**

$$\boxed{L_i = P^- C_i^T\left(C_i P^- C_i^T + R_i\right)^{-1}}$$

$$\hat{\underline{x}}^+ = \hat{\underline{x}}^- \;+\; L_i\left(\underline{y}_i[n] - C_i\,\hat{\underline{x}}^-\right)$$

$$\boxed{P^+ = (I - L_i C_i)P^-}$$

We introduced the Kalman Filter as a special (optimal) case of a dynamic observer:

Continuous-Discrete Dynamic Observer:

***Between measurements*:**

$$\dot{\hat{\underline{x}}} = A\hat{\underline{x}} + B\underline{u}$$

Propagate the reference model using the Continuous method

***At measurements*:**

$$\hat{\underline{x}}^+ = \hat{\underline{x}}^- \;+\; L_i\left(\underline{y}_i[n] - C_i\,\hat{\underline{x}}^-\right)$$

"Correct" the system using the Discrete method

We'll use the following outline to derive the remaining Linear Kalman Filter matrix equations:

1. Derive *P(t)* propagation: $\boxed{\dot{P} = AP + PA^T + Q}$
   Method: Differentiating *P(t)*

2. Solve for general equation of *P* after an update:
   $$P^+ = P^- - P^- C^T L^T - LCP^- + LCP^- C^T L^T + LRL^T$$

3. Derive Kalman Gain: $\boxed{L_i = P^- C_i^T\left(C_i P^- C_i^T + R_i\right)^{-1}}$
   Method: Find *L(t)* which minimize trace of *P(t)*

4. Plug *L* into *P*+: $\boxed{P^+ = (I - L_i C_i)P^-}$

Linear Kalman Filter Derivation

System:

$$\dot{\underline{x}} = A\underline{x} + B\underline{u} + \underline{\xi}$$

$$\underline{y}_i[n] = C_i\underline{x}[n] + \underline{\eta}_i[n]$$

Covariance Matrices:

$$Q \equiv E\{\underline{\xi}\,\underline{\xi}^T\}$$

$$R_i \equiv E\{(\underline{\eta}_i[n])(\underline{\eta}_i[n])^T\}$$

$$P(t) \equiv E\{(\underline{x}-\hat{\underline{x}})(\underline{x}-\hat{\underline{x}})^T\}$$

State estimate propagation between measurements:

$$\dot{\hat{\underline{x}}} = A\hat{\underline{x}} + B\underline{u}$$

$$\boxed{\dot{P} = AP + PA^T + Q}$$

State estimate correction at measurement for sensor i:

$$L_i = P^- C_i^T\left(C_i P^- C_i^T + R_i\right)^{-1}$$

$$\hat{\underline{x}}^+ = \hat{\underline{x}}^- + L_i\left(\underline{y}_i[n] - C_i\hat{\underline{x}}^-\right)$$

$$P^+ = (I - L_i C_i)P^-$$

Step 1: Derivation of *P* propagation *between* measurements

- Method: Differentiate *P(t)*: $\dot{P} = AP + PA^T + Q$

$$\text{Let } \tilde{\underline{x}} \equiv \underline{x} - \hat{\underline{x}}$$

$$\dot{\tilde{\underline{x}}} = \dot{\underline{x}} - \dot{\hat{\underline{x}}}$$

$$\dot{\tilde{\underline{x}}} = [A\underline{x} + B\underline{u} + \underline{\xi}] - [A\hat{\underline{x}} + B\underline{u}]$$

$$\dot{\tilde{\underline{x}}} = A\tilde{\underline{x}} + \underline{\xi}$$

Solving $\dot{\tilde{\underline{x}}} = A\tilde{\underline{x}} + \underline{\xi}$ (from Diff Eqs)

$$\tilde{\underline{x}}(t) = e^{At}\tilde{\underline{x}}_0 + \int_0^t e^{A(t-\tau)}\underline{\xi}(\tau)d\tau$$

Solving $E\{\tilde{\underline{x}}\,\underline{\xi}^T\}$

$$E\{\tilde{\underline{x}}\,\underline{\xi}^T\} = E\left\{\begin{matrix} e^{At}\tilde{\underline{x}}_0\underline{\xi}^T \\ + \int_0^t e^{A(t-\tau)}\underline{\xi}(\tau)\underline{\xi}^T(\tau)d\tau \end{matrix}\right\}$$

$$= \int_0^t e^{A(t-\tau)}Q\delta(t-\tau)d\tau$$

$$= \tfrac{1}{2}Q$$

$$\dot{P} = \frac{d}{dt}P = \frac{d}{dt}E\{\tilde{\underline{x}}\tilde{\underline{x}}^T\}$$

$$= E\{\dot{\tilde{\underline{x}}}\tilde{\underline{x}}^T + \tilde{\underline{x}}\dot{\tilde{\underline{x}}}^T\}$$

$$= E\{A\tilde{\underline{x}}\tilde{\underline{x}}^T + \underline{\xi}\tilde{\underline{x}}^T + \tilde{\underline{x}}\tilde{\underline{x}}^T A^T + \tilde{\underline{x}}\underline{\xi}^T\}$$

$$= AP + PA^T + \underbrace{E\{\underline{\xi}\tilde{\underline{x}}^T\}}_{\frac{1}{2}Q} + \underbrace{E\{\tilde{\underline{x}}\underline{\xi}^T\}}_{\frac{1}{2}Q}$$

$$\Rightarrow \quad \dot{P} = \underbrace{AP + PA^T}_{P\text{ propagation due to state dynamics}} + \underbrace{Q}_{P\text{ growth due to model uncertainty}}$$

Linear Kalman Filter Derivation

System:

$$\dot{\underline{x}} = A\underline{x} + B\underline{u} + \underline{\xi}$$

$$\underline{y}_i[n] = C_i\underline{x}[n] + \underline{\eta}_i[n]$$

Covariance Matrices:

$$Q \equiv E\{\underline{\xi}\,\underline{\xi}^T\}$$

$$R_i \equiv E\{(\underline{\eta}_i[n])(\underline{\eta}_i[n])^T\}$$

$$P(t) \equiv E\{(\underline{x}-\hat{\underline{x}})(\underline{x}-\hat{\underline{x}})^T\}$$

State estimate propagation between measurements:

$$\dot{\hat{\underline{x}}} = A\hat{\underline{x}} + B\underline{u}$$

$$\dot{P} = AP + PA^T + Q$$

State estimate correction at measurement for sensor i:

$$L_i = P^-C_i^T\left(C_iP^-C_i^T + R_i\right)^{-1}$$

$$\hat{\underline{x}}^+ = \hat{\underline{x}}^- + L_i\left(\underline{y}_i[n] - C_i\hat{\underline{x}}^-\right)$$

$$P^+ = \left(I - L_iC_i\right)P^-$$

Step 2: Solve for general equation for *P* after an update

- Method: Expand $P^+ = E\{(\underline{x}-\hat{\underline{x}}^+)(\underline{x}-\hat{\underline{x}}^+)^T\}$

$$\begin{aligned}
Let\ \ \tilde{\underline{x}}^+ &= \underline{x} - \hat{\underline{x}}^+ \\
&= \underline{x} - \hat{\underline{x}}^- - L\left(\underline{y}[n] - C\hat{\underline{x}}^-\right) \\
&= \underline{x} - \hat{\underline{x}}^- - L\left(C\underline{x} + \underline{\eta} - C\hat{\underline{x}}^-\right) \\
&= \tilde{\underline{x}}^- - LC\tilde{\underline{x}}^- - L\underline{\eta}
\end{aligned}$$

$$E\{\tilde{\underline{x}}^-\tilde{\underline{x}}^{-T}\} = P^-$$

$$E\{\underline{\eta}\,\underline{\eta}^T\} = R$$

$$E\{\tilde{\underline{x}}^-\underline{\eta}^T\} = 0$$

$$E\{\underline{\eta}\,\tilde{\underline{x}}^{-T}\} = 0$$

$\tilde{\underline{x}}^-$ & $\underline{\eta}$ are independent

$$\begin{aligned}
Let\ \ P^+ &= E\{\tilde{\underline{x}}^+\tilde{\underline{x}}^{+T}\} \\
&= E\{(\tilde{\underline{x}}^- - LC\tilde{\underline{x}}^- - L\underline{\eta})(\tilde{\underline{x}}^- - LC\tilde{\underline{x}}^- - L\underline{\eta})^T\} \\
&= E\{\tilde{\underline{x}}^-\tilde{\underline{x}}^{-T} - \tilde{\underline{x}}^-\tilde{\underline{x}}^{-T}C^TL^T - \tilde{\underline{x}}^-\underline{\eta}^TL^T \\
&\qquad - LC\tilde{\underline{x}}^-\tilde{\underline{x}}^{-T} + LC\tilde{\underline{x}}^-\tilde{\underline{x}}^{-T}C^TL^T + LC\tilde{\underline{x}}^-\underline{\eta}^TL^T \\
&\qquad - L\underline{\eta}\,\tilde{\underline{x}}^{-T} + L\underline{\eta}\,\tilde{\underline{x}}^{-T}C^TL^T + L\underline{\eta}\,\underline{\eta}^TL^T\}
\end{aligned}$$

$$\Rightarrow\ P^+ = P^- - P^-C^TL^T - LCP^- + LCP^-C^TL^T + LRL^T$$

Linear Kalman Filter Derivation

System:

$$\dot{\underline{x}} = A\underline{x} + B\underline{u} + \underline{\xi}$$

$$\underline{y}_i[n] = C_i\underline{x}[n] + \underline{\eta}_i[n]$$

Covariance Matrices:

$$Q \equiv E\{\underline{\xi}\,\underline{\xi}^T\}$$

$$R_i \equiv E\{(\underline{\eta}_i[n])(\underline{\eta}_i[n])^T\}$$

$$P(t) \equiv E\{(\underline{x}-\hat{\underline{x}})(\underline{x}-\hat{\underline{x}})^T\}$$

State estimate propagation between measurements:

$$\dot{\hat{\underline{x}}} = A\hat{\underline{x}} + B\underline{u}$$

$$\dot{P} = AP + PA^T + Q$$

State estimate correction at measurement for sensor i:

$$L_i = P^-C_i^T\left(C_iP^-C_i^T + R_i\right)^{-1}$$

$$\hat{\underline{x}}^+ = \hat{\underline{x}}^- + L_i\left(\underline{y}_i[n] - C_i\hat{\underline{x}}^-\right)$$

$$P^+ = (I - L_iC_i)P^-$$

Step 3: Derivation of the <u>Kalman Gain Matrix</u>

- <u>Method:</u> Solve for $L_i(t)$ which minimizes trace of $P(t)$:

$$L_i = P^-C_i^T\left(C_iP^-C_i^T + R_i\right)^{-1}$$

Specifically: Solve for L such that:

$$\frac{\partial}{\partial L}tr\left(P^+\right) = 0$$

trace: Sum of diagonals of matrix

From previous slide:

$$P^+ = P^- - P^-C^TL^T - LCP^- + LCP^-C^TL^T + LRL^T$$

$$\frac{\partial}{\partial L}tr\left(P^+\right) = \cancelto{0}{\tfrac{\partial}{\partial L}tr(P^-)} - \cancelto{P^-C^T}{\tfrac{\partial}{\partial L}tr(P^-C^TL^T)}$$

$$- \underset{P^-C^T}{\tfrac{\partial}{\partial L}tr(LCP^-)} + \underset{2LCP^-C^T}{\tfrac{\partial}{\partial L}tr(LCP^-C^TL^T)} + \underset{2LR}{\tfrac{\partial}{\partial L}tr(LRL^T)}$$

$$\tfrac{\partial}{\partial L}tr\left(P^+\right) = -2P^-C^T + 2LCP^-C^T + 2LR$$

$$\Longrightarrow\ 0 = -2P^-C^T + 2LCP^-C^T + 2LR$$

$$\Longrightarrow\ 2P^-C^T = 2L(CP^-C^T + R)$$

$$\Longrightarrow\ L = P^-C^T(CP^-C^T + R)^{-1}$$

Matrix Rules

$tr(ABC) = tr(C^TB^TA^T)$

$\tfrac{\partial}{\partial A}tr(BAD) = D^TB^T$

if $B = B^T$

then $\tfrac{\partial}{\partial A}tr(ABA^T) = 2AB$

Linear Kalman Filter Derivation

System:

$$\dot{\underline{x}} = A\underline{x} + B\underline{u} + \underline{\xi}$$

$$\underline{y}_i[n] = C_i\underline{x}[n] + \underline{\eta}_i[n]$$

Covariance Matrices:

$$Q \equiv E\{\underline{\xi}\,\underline{\xi}^T\}$$

$$R_i \equiv E\{(\underline{\eta}_i[n])(\underline{\eta}_i[n])^T\}$$

$$P(t) \equiv E\{(\underline{x}-\hat{\underline{x}})(\underline{x}-\hat{\underline{x}})^T\}$$

***State estimate propagation between measurements*:**

$$\dot{\hat{\underline{x}}} = A\hat{\underline{x}} + B\underline{u}$$

$$\dot{P} = AP + PA^T + Q$$

***State estimate correction at measurement for sensor i*:**

$$L_i = P^-C_i^T\left(C_iP^-C_i^T + R_i\right)^{-1}$$

$$\hat{\underline{x}}^+ = \hat{\underline{x}}^- + L_i\left(\underline{y}_i[n] - C_i\hat{\underline{x}}^-\right)$$

$$\boxed{P^+ = (I - L_iC_i)P^-}$$

Step 4: Derivation of *P* update *at* measurements:

- Method: Plug *L* back into *P*+:

$$\left.\begin{aligned} L &= P^-C^T(CP^-C^T+R)^{-1} \\ P^+ &= P^- - P^-C^TL^T - LCP^- \\ &\quad + LCP^-C^TL^T + LRL^T \end{aligned}\right\} \longrightarrow P^+ = (I - L_iC_i)P^-$$

Starting from general *P*+ eqn:

$$P^+ = P^- - P^-C^TL^T - LCP^- + \overbrace{LCP^-C^TL^T + LRL^T}^{L[CP^-C^T+R]L^T}$$

Plug in *L*:

$$\rightarrow P^+ = P^- - P^-C^T(CP^-C^T+R)^{-1}CP^- - P^-C^T(CP^-C^T+R)^{-1}CP^-$$
$$+ P^-C^T(CP^-C^T+R)^{-1}\underbrace{[CP^-C^T+R](CP^-C^T+R)^{-1}}_{I}CP^-$$

$$\begin{aligned} P^+ = P^- &- P^-C^T(CP^-C^T+R)^{-1}CP^- \\ &- P^-C^T(CP^-C^T+R)^{-1}CP^- \\ &+ P^-C^T(CP^-C^T+R)^{-1}CP^- \end{aligned}$$

$$P^+ = P^- - P^-C^T(CP^-C^T+R)^{-1}CP^-$$

$$P^+ = (I - \underbrace{P^-C^T(CP^-C^T+R)^{-1}}_{L}C)P^- \longrightarrow P^+ = (I - L_iC_i)P^-$$

P (uncertainty) decreases due to gained information

EKF: Extended Kalman Filter (non-linear)

Linear Kalman Filter theory is extended to non-linear systems

- Use non-linear functions to propagate and reconstruct measurements
- Use Jacobians to make linear matrices (*A & C*)
  - Used for covariance propagation and Kalman gain

Covariance Matrices:

$$Q \equiv E\{\underline{\xi}\,\underline{\xi}^T\}$$

$$R_i \equiv E\{(\underline{\eta}_i[n])(\underline{\eta}_i[n])^T\}$$

$$P(t) \equiv E\{(\underline{x}-\underline{\hat{x}})(\underline{x}-\underline{\hat{x}})^T\}$$

	Linear System	**Non-Linear System**
System we want to estimate	$\underline{\dot{x}} = A\underline{x} + B\underline{u} + \underline{\xi}$ $\underline{y}_i[n] = C_i\,\underline{x}[n] + \underline{\eta}_i[n]$	$\underline{\dot{x}} = f(\underline{x},\underline{u}) + \underline{\xi}$ $\underline{y}_i[n] = h_i(\underline{x}[n],\underline{u}) + \underline{\eta}_i[n]$
State estimate propagation between measurements	$\dot{\underline{\hat{x}}} = A\underline{\hat{x}} + B\underline{u}$ $\dot{P} = AP + PA^T + Q$	$\dot{\underline{\hat{x}}} = f(\underline{\hat{x}},\underline{u})$ $A = \frac{\partial}{\partial \underline{x}} f(\underline{\hat{x}},\underline{u})$ $\dot{P} = AP + PA^T + Q$
State estimate correction at measurement for sensor i *Before update:* $\underline{\hat{x}}^-, P^-$ *After update:* $\underline{\hat{x}}^+, P^+$	$L_i = P^- C_i^T (C_i P^- C_i^T + R_i)^{-1}$ $\underline{\hat{x}}^+ = \underline{\hat{x}}^- + L_i(\underline{y}_i[n] - C_i\underline{\hat{x}}^-)$ $P^+ = (I - L_i C_i)P^-$	$C_i = \frac{\partial}{\partial \underline{x}} h_i(\underline{\hat{x}}^-, \underline{u}[n])$ $L_i = P^- C_i^T (C_i P^- C_i^T + R_i)^{-1}$ $\underline{\hat{x}}^+ = \underline{\hat{x}}^- + L_i(\underline{y}_i[n] - h_i(\underline{\hat{x}}^-, \underline{u}[n]))$ $P^+ = (I - L_i C_i)P^-$

EKF Algorithm Implementation

Derive model of system to estimate:

Setup:

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

$$\underline{y}_i[n] = h_i(\underline{x}[n],\underline{u}) + \underline{\eta}_i[n]$$

Meas. from sensor i

Select tuning parameters:

- *$\underline{x}_0$: Initial state estimate*
- *P_0: Initial state uncertainty covariance matrix*
- *Q: Process noise cov. matrix*
- *R: Measurement noise cov. matrix*

Covariance Matrices:

$$P(t) \equiv E\{(\underline{x}-\hat{\underline{x}})(\underline{x}-\hat{\underline{x}})^T\}$$

$$Q \equiv E\{\underline{\xi}\,\underline{\xi}^T\}$$

$$R_i \equiv E\{(\underline{\eta}_i[n])(\underline{\eta}_i[n])^T\}$$

Initialization: $\hat{\underline{x}} = \hat{\underline{x}}_0 \qquad P = P_0$

Repeat each time step

Propagate reference model to current time: $t_{current} = t_{prev} + \Delta t$

```
for counter=1 to N do:                                          // N small Euler integration sub-steps is more accurate
    $\hat{\underline{x}} = \hat{\underline{x}} + \left(\frac{\Delta t}{N}\right) f(\hat{\underline{x}},\underline{u})$          // State propagation
    $A = \frac{\partial}{\partial \underline{x}} f(\hat{\underline{x}},\underline{u})$          // Compute Jacobian of f() about current estimate
    $P = P + \left(\frac{\Delta t}{N}\right)\{AP + PA^T + Q\}$          // Covariance propagation (dynamics & process noise)
    $P = real(\frac{1}{2}P + \frac{1}{2}P^T)$          // Make sure P stays real and symmetric
end for
```

Correct estimates with measurements: $\underline{y}_i[n]$

```
if measurement received from sensor i
    $C_i = \frac{\partial}{\partial \underline{x}} h_i(\hat{\underline{x}},\underline{u}[n])$          // Compute Jacobian of h_i() about current estimate
    $L_i = PC_i^T\left(C_i P C_i^T + R_i\right)^{-1}$          // Compute Kalman Gain Matrix
    $\hat{\underline{x}} = \hat{\underline{x}} + L_i\left(\underline{y}_i[n] - h_i(\hat{\underline{x}},\underline{u}[n])\right)$          // Correct estimates with measurement
    $P = (I - L_i C_i)P$          // Decrease P due to gained information
    $P = real(\frac{1}{2}P + \frac{1}{2}P^T)$          // Make sure P stays real and symmetric
end if
```

Actual measurements *Recreation of measurements from estimates*

Intentionally Complicated EKF Algorithm Example

Particle rotating about center on a string

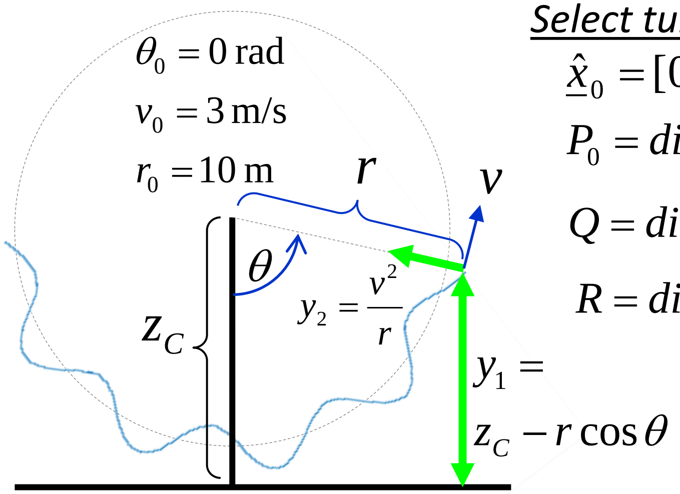
- Radius varies sinusoidally in time, plus random variations
- Particle tangential velocity varies randomly

Derive model of system to estimate:

$$\underline{x} = \begin{bmatrix} \theta \\ v \\ r \end{bmatrix} \qquad \dot{\underline{x}} = \begin{bmatrix} \dot\theta \\ \dot v \\ \dot r \end{bmatrix} = \begin{bmatrix} v/r \\ 0 \\ 3\cos(3t) \end{bmatrix} + \underline{\xi}$$

θ: angle, rad
v: tangential velocity, m/s
r: radius, m

$$\underline{y}[n] = \begin{bmatrix} z_C - r\cos\theta \\ v^2/r \end{bmatrix} + \underline{\eta}[n]$$



Select tuning parameters:

$$\hat{\underline{x}}_0 = [0.1 \quad 3.2 \quad 9.1]^T$$

$$P_0 = diag\left([0.05^2 \quad 0.1^2 \quad 0.5^2]\right)$$

$$Q = diag\left([0.0^2 \quad 0.02^2 \quad 0.1^2]\right)$$

$$R = diag\left([0.1^2 \;\; 0.01^2]\right)$$

Two measurements:
y_1: height
y_2: radial acceleration

Propagate reference model to current time:

```
for counter=1 to N do:
    $\hat{\underline{x}} = \hat{\underline{x}} + \left(\frac{\Delta t}{N}\right) f(\hat{\underline{x}}, \underline{u})$
    $A = \frac{\partial}{\partial \underline{x}} f(\hat{\underline{x}}, \underline{u})$
    $P = P + \left(\frac{\Delta t}{N}\right)\{AP + PA^T + Q\}$
    $P = real(\frac{1}{2}P + \frac{1}{2}P^T)$
end for
```

Correct estimates with measurements:

```
if measurement received from sensor i
    $C_i = \frac{\partial}{\partial \underline{x}} h_i(\hat{\underline{x}}, \underline{u}[n])$
    $L_i = PC_i^T\left(C_i P C_i^T + R_i\right)^{-1}$
    $\hat{\underline{x}} = \hat{\underline{x}} + L_i\left(\underline{y}_i[n] - h_i(\hat{\underline{x}}, \underline{u}[n])\right)$
    $P = (I - L_i C_i)P$
    $P = real(\frac{1}{2}P + \frac{1}{2}P^T)$
end if
```

Repeat each time step

$$\dot{\hat\theta} = \frac{\hat v}{\hat r}$$

$$f(\hat{\underline{x}}, \underline{u}) = \begin{bmatrix} \hat{x}_2/\hat{x}_3 \\ 0 \\ 3\cos(3t) \end{bmatrix} \Rightarrow A = \begin{array}{c} \\ \\ \end{array}\begin{array}{ccc} & \frac{\partial f_1}{\partial \hat{x}_2} & \frac{\partial f_1}{\partial \hat{x}_3} \\ \left[\begin{matrix} 0 \\ 0 \\ 0 \end{matrix}\right. & \begin{matrix} 1/\hat{x}_3 \\ 0 \\ 0 \end{matrix} & \left.\begin{matrix} -\hat{x}_2/\hat{x}_3^2 \\ 0 \\ 0 \end{matrix}\right] \end{array}$$

$$h(\hat{\underline{x}}, \underline{u}) = \begin{bmatrix} z_C - \hat{x}_3\cos\hat{x}_1 \\ \hat{x}_2^2/\hat{x}_3 \end{bmatrix} \Rightarrow C = \begin{bmatrix} \hat{x}_3\sin\hat{x}_1 & 0 & -\cos\hat{x}_1 \\ 0 & 2\hat{x}_2/\hat{x}_3 & -\hat{x}_2^2/\hat{x}_3^2 \end{bmatrix}$$

If y_1 and y_2 are available at different times, separate:

$$h_1(\hat{\underline{x}}, \underline{u}) = [z_C - \hat{x}_3\cos\hat{x}_1] \quad C_1 = [\hat{x}_3\sin\hat{x}_1 \quad 0 \quad -\cos\hat{x}_1] \quad R_1 = (0.1^2)$$

$$h_2(\hat{\underline{x}}, \underline{u}) = [\hat{x}_2^2/\hat{x}_3] \quad C_2 = [0 \quad 2\hat{x}_2/\hat{x}_3 \quad -\hat{x}_2^2/\hat{x}_3^2] \quad R_2 = (0.01^2)$$

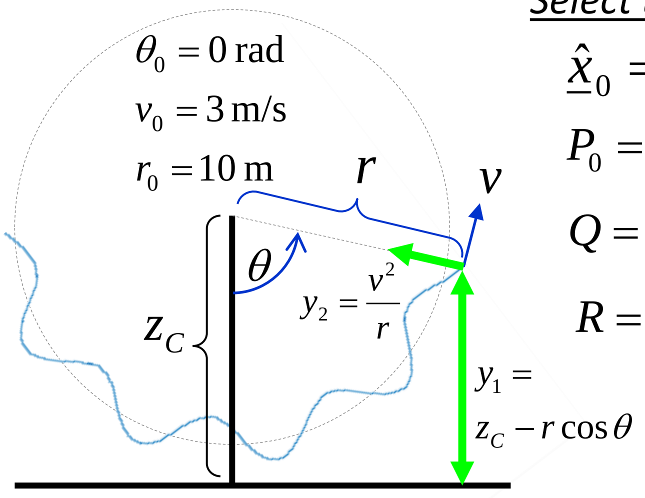
Intentionally Complicated EKF Algorithm Example

Derive model of system to estimate:

$$\underline{x} = \begin{bmatrix} \theta \\ v \\ r \end{bmatrix} \qquad \dot{\underline{x}} = \begin{bmatrix} \dot\theta \\ \dot v \\ \dot r \end{bmatrix} = \begin{bmatrix} v/r \\ 0 \\ 3\cos(3t) \end{bmatrix} + \underline{\xi}$$

θ: angle, rad
v: tangential velocity, m/s
r: radius, m

$$\underline{y}[n] = \begin{bmatrix} z_C - r\cos\theta \\ v^2/r \end{bmatrix} + \underline{\eta}[n]$$



$\theta_0 = 0$ rad

$v_0 = 3$ m/s

$r_0 = 10$ m

Select tuning parameters:

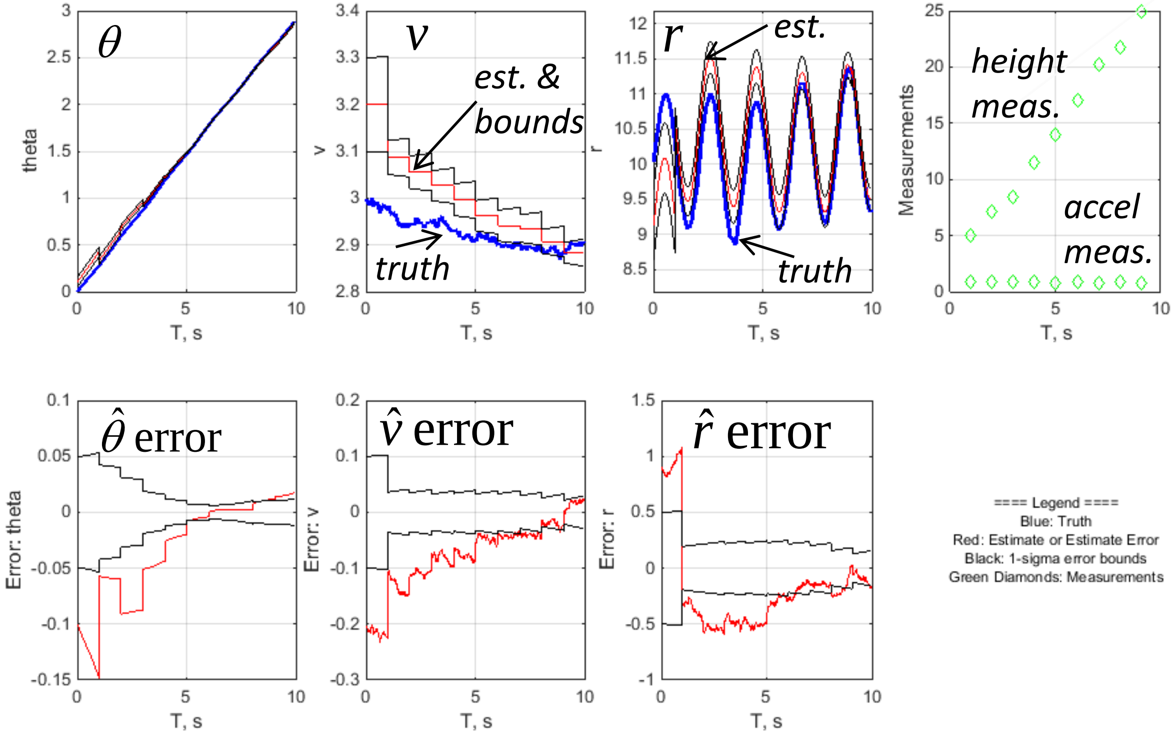
$$\hat{\underline{x}}_0 = [0.1 \quad 3.2 \quad 9.1]^T$$

$$P_0 = diag\left([0.05^2 \quad 0.1^2 \quad 0.5^2]\right)$$

$$Q = diag\left([0.0^2 \quad 0.02^2 \quad 0.1^2]\right)$$

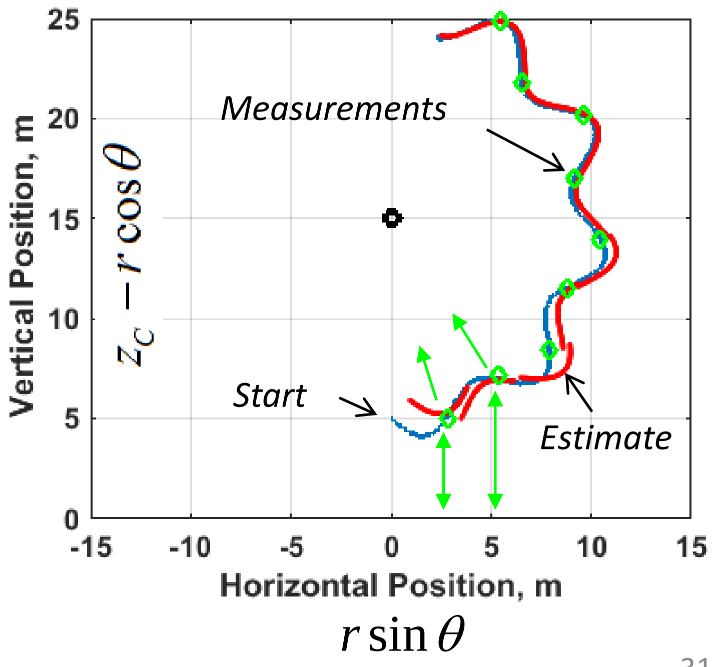
$$R = diag\left([0.1^2 \quad 0.01^2]\right)$$

Two measurements:
y_1: height
y_2: radial acceleration



==== Legend ====
Blue: Truth
Red: Estimate or Estimate Error
Black: 1-sigma error bounds
Green Diamonds: Measurements

Measurements at 1 Hz



EKF Practical Issues

- Extended Kalman Filters are a powerful tool, but:
  - Require a reasonable mathematical model that can be updated with and linearized about current state estimates
  - Require some understanding of system & modeling uncertainties
    - System statistics can be non-Gaussian, but not "too" non-Gaussian
  - Non-linear systems can't be "too" non-linear
  - States must be "observable" (directly or indirectly) by the measurements
  - Requires some engineering intuition for tuning parameters (P_0, Q, R)
    - Discussed on next slide
  - Standard (presented) formulation can result in numerical instabilities
    - Always make sure P stays symmetric and real!
    - Other formulations are preferred (Square-Root EKF, UD-Factorization, etc.)
  - Other variants exist:
    - Ensemble Kalman Filter, Unscented Kalman Filter (UKF), Particle Filter (PF)

EKF Tuning Advice for *P_0*, *Q* and *R*

- General Advice:
  - Tuning trades: convergence rate versus stability
    - If matrix diagonals are too large: Filter may not converge quickly (or at all)
    - If matrix diagonals are too small: Filter may diverge from truth!!!!
  - Covariance matrices must be symmetric with non-negative diagonals
  - Generally a good idea to make diagonal elements greater than zero
    - Gives the filter some wiggle-room to learn estimates
  - Often, *P_0*, *Q* and *R* are initialized to be diagonal matrices
    - If you know initial correlations, use them. But…
    - Determining statistical correlations is hard! Let the filter do it!

Initial Error Cov. Matrix

$$P_0 = \begin{bmatrix} \sigma_{x1}^2 & & 0 \\ & \sigma_{x2}^2 & \\ & & \ddots & \\ 0 & & & \sigma_{xn}^2 \end{bmatrix}_{n \text{ x } n}$$

Even if you know an initial state perfectly, make *P_0* diagonals be non-zero

- If P diags are too small, EKF will ignore measurements

Process Noise Cov. Matrix

$$Q = \begin{bmatrix} \sigma_{\xi 1}^2 & & 0 \\ & \sigma_{\xi 2}^2 & \\ & & \ddots & \\ 0 & & & \sigma_{\xi n}^2 \end{bmatrix}_{n \text{ x } n}$$

If model truly follows actual system dynamics, isn't too non-linear, and doesn't randomly vary:

$$Let \quad \sigma_{\xi k} \rightarrow 0 \ \ (e.g. \ \ 10^{-?})$$

Otherwise: increase diagonals to give EKF wiggle room and avoid divergence

Measurement Noise Cov. Matrix

$$R = \begin{bmatrix} \sigma_{\eta 1}^2 & & 0 \\ & \sigma_{\eta 2}^2 & \\ & & \ddots & \\ 0 & & & \sigma_{\eta m}^2 \end{bmatrix}_{m \text{ x } m}$$

Measurement statistics can often by estimated.

- Start with meas. statistics
- Increase if necessary
- Increase if measurements have unaccounted for bias, etc.

Lecture 10 Homework, 1/8

For problems 1-4, we'll be using LPFs to estimate states from noisy measurements. For each:

- **Appropriate LPF time constants can be found in "init_uavsim_params"**
- **Turn off all winds!!**
- **Keep the "Feedback Switch" set to "truth"**
- **Run the simulation for 60 seconds**
- **Use the resulting waypoint trajectory from the previous homework**

1) "uavsim_estimates" has a subfunction called "LPF". Modify "LPF" to emulate a sampled-time low-pass filter. Mapping from LPF difference equation to function inputs are provided below. Turn in the resulting "LPF" implementation code.

$$\underbrace{y_{filt}[n+1]}_{\substack{\texttt{y} \\ \text{Filter output}}} = \alpha_{LPF}\,\underbrace{y_{filt}[n]}_{\texttt{y\_prev}} + (1-\alpha_{LPF})\,\underbrace{y_{raw}[n]}_{\substack{\texttt{u} \\ \text{Filter input}}}$$

Lecture 10 Homework, 2/8

2) **The output of "uavsim_estimates" is a vector of resulting estimated states, which are needed for autopilot control. After implementing the LPF subfunction, a number of noisy measurements will be appropriately filtered. Do the following:**
   - **In "uavsim_estimates", populate p_hat, q_hat, r_hat and psi_hat with the outputs of the respective low-pass filters. As a result, they will be part of the "uavsim_estimates" output vector.**
   - **In "uavsim_logging", enable output logging of state estimates (set "log_estimates=true;")**
   - **In "uavsim_logging", find the "if log_estimates" portion and add the following variables to the output log: p_est_dps, q_est_dps, r_est_dps, and yaw_est_deg. (Make sure to "wrap" the filtered yaw value between -180° and 180°.)**

   Run the simulation and provide the following:

   a) **Provide the LPF time constants (as specified in "init_uavsim_params") for gyro, accelerometer, and magnetometer measurements.**
   b) **Plot a comparison of: p_dps, p_gyro_dps, and p_est_dps. The estimated body rate should follow the gyro measurement, but filter out the noise. Zoom appropriately to show an example of the filtering.**
   c) **Plot a comparison of: q_dps, q_gyro_dps, and q_est_dps. Zoom appropriately.**
   d) **Plot a comparison of: r_dps, r_gyro_dps, and r_est_dps. Zoom appropriately.**
   e) **Plot a comparison of: yaw_deg, yaw_mag_deg, and yaw_est_deg. Show that the LPF removes much of the high frequency noise, but does not remove the magnetometer bias.**

Lecture 10 Homework, 3/8

3) **In "uavsim_estimates", estimate altitude using barometric pressure. Do the following:**
   - **As discussed in the previous lecture, barometric altitude is estimated by comparing measured pressure with a "reference" pressure, e.g. the measured pressure at launch. We'll assume the UAV had an accurate measure of pressure at launch. Mimic this by computing true P_launch, which is a function of P.h0_ASL.**
   - **A significant cause of barometric altitude error is due to gradual changes in atmospheric pressure during flight. We mimicked this using a (constant) bias in our static pressure measurement. Use P_launch and the low-pass-filtered static pressure measurement to estimate barometric altitude (using the "constant temperature" equation). Hint: In Matlab, the natural log is called via the function log( ).**
   - **In "uavsim_estimates", set h_hat equal to the estimated barometric altitude (h_baro). As a result, it will be logged in "uavsim_logging".**

   Run the simulation and compare alt_m with the resulting alt_baro_m. Do not be surprised if the altitude estimate is in error by 5m or more. In the plot, point out the resulting bias and any residual measurement noise.

4) **In "uavsim_estimates", estimate pitot-based airspeed from the filtered differential pressure measurement. Set Va_hat = Va_pitot. Run the simulation and compare airspeed_mps with airspeed_pitot_mps. Do not be surprised if the estimate is off by 2 m/s or more. In the plot, point out the resulting bias and any residual measurement noise.**

Enable display of estimates by modifying uavsim_display:

```
plot_estimates = 1;
```

Blue: truth
Red: command
Green: estimate

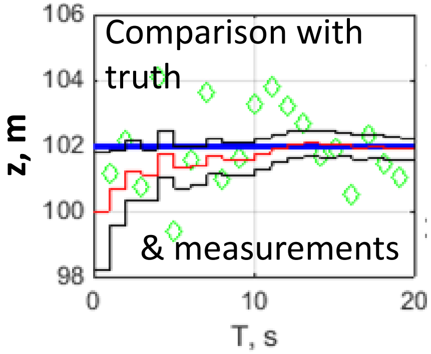
Lecture 10 Homework, 4/8

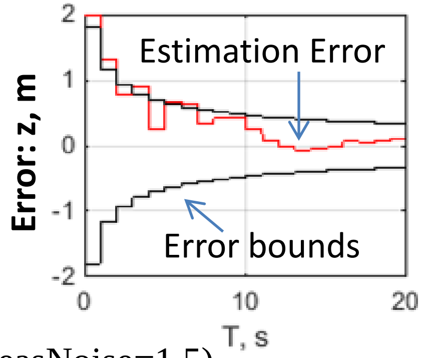
For problems 5-9, and continuing into next week's homework, you will be using a provided skeleton routine (ekf_driver.m) to investigate how an Extended Kalman Filter estimates states using direct or indirect measurements. In order to use the routine, you'll need to supply the EKF logic (see Problem 5). Each problem will build off of the result from the problem before it, starting with a simple scalar & linear model to estimate, and ending (next week) with a multi-state non-linear model with a non-linear measurement. Although the problems build off of each other, it is strongly suggested that you rename the "ekf_driver" routine appropriately before beginning each new problem.

> **Note: With absolutely no loss of correctness or generality, we will implement the described non-linear *Extended* Kalman Filter variant (as opposed to the linear Kalman Filter), even for completely linear and simple systems.**

5) **The provided routine "ekf_driver.m" is pre-prepared for Problem 6, except for the actual implementation of the EKF logic. You will need to fill in the portions that say "<code here>". Implement the appropriate EKF propagation and correction logic and provide a print-out of the resulting "EKF Processing" section of "ekf_driver.m".**

Lecture 10 Homework, 5/8





6) Consider the following very simple estimation problem, which uses an EKF to estimate a constant, given noisy measurements: A UAV is actually at a constant altitude (z) of 102m, but its autopilot initially thinks the altitude is 100m, with a 1-sigma certainty of 1.8m. Every second, the UAV makes a noisy altitude measurement. (Measurements are unbiased, but have a Gaussian noise with a standard deviation of 1.5m.) How does the UAV's altitude estimate improve over 20 seconds of measurements?

Mathematical translation for ekf_driver implementation:

- True state vector is scalar: x=[altitude]=102m.
- True state is absolutely constant, so:
  - xdot = fTrue(x,…) = 0
  - the true process noise is zero: sigma_trueProcessNoise=0
- Measurements every 1 second (dtMeas=1) with a noise std.dev. of 1.5m (sigma_trueMeasNoise=1.5)
- Variable to estimate is altitude: xhat=[altitudeEstimate]. Initially, xhat=100m.
- EKF designer knows the initial estimate may be off by 1.8m (1-sigma): sigma_ekfInitUnceratainty=1.8
- EKF designer knows that the altitude is truly unchanging: f=0 & sigma_ekfProcessNoise=0;
  - Jacobian of "f=0" wrt xhat is zero: A=0
- EKF designer knows true measurement noise: sigma_ekfMeasNoise=sigma_trueMeasNoise
- Measurements are directly of the only state in xhat, so measurement recreation is: h=xhat(1)
  - Jacobian of "h=xhat(1)" wrt xhat is: C=1

Analyze ekf_driver.m and run it a few times to understand how it works and what the EKF is doing. (If your output isn't roughly like the provided example above, check your EKF code!)

a) Provide an example print-out of the resulting figure, noting: how well the EKF estimates the true state (it should converge over time) and how well the EKF error bounds match the actual estimation error.

b) How does the gain *L* vary over time, and does it make sense? (Think about weightings.)

Lecture 10 Homework, 6/8

7) **The previous "constant parameter estimation" problem resulted in a number of specified truth parameters as well as EKF parameters that an EKF designer would need to know or choose. We'll now explore how some of these parameters affect state estimation, and what is the effect when parameters are improperly chosen. For each of the following questions, begin with the original parameter values (provided here for convenience) and vary one parameter at a time. For each, provide a short description of the effect changing each parameter has on state estimation.**

Original parameter values:

```
sigma_trueProcessNoise = 0;        sigma_ekfInitUncertainty = 1.8;
sigma_trueMeasNoise = 1.5;         sigma_ekfProcessNoise = 0;
dtMeas=1;                          sigma_ekfMeasNoise = sigma_trueMeasNoise;
```

Defines P_0

Defines Q

Defines R

a) **What is the effect of increasing** dtMeas **to 5 (a measurement every 5 seconds)?**

b) **What is the effect of decreasing** dtMeas **to 0.01 (e.g. continuous measurements)?**

c) **What is the effect of increasing the initial uncertainty** (sigma_ekfInitUncertainty) **to 10? (Think about this in terms of how much the filter "trusts" the very first measurement compared to its initial estimate.)**

d) **What is the effect of decreasing the initial uncertainty** (sigma_ekfInitUncertainty) **to 0.1? (Note: keep actual initial error at 2 meters. This is akin to saying something wrong, but loudly and with a lot of authority.)**

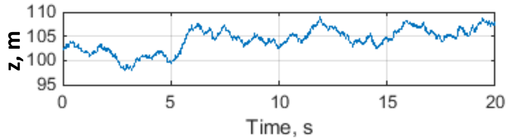
e) **What if we don't know the actual measurement noise statistics? Try overestimating the measurement noise statistics by setting** sigma_ekfMeasNoise = 5*sigma_trueMeasNoise.

f) **Similarly, try underestimating the measurement noise statistics by setting** sigma_ekf_MeasNoise = 0.1*sigma_trueMeasNoise.

Lecture 10 Homework, 7/8

8) **Now let's try to understand what "process noise" is. Inevitably, when simplifying some real-world phenomenon into a mathematical model, we probably ignored some higher-order effects. "Process noise" in this context is a statistical measure of how much the actual phenomenon might differ from our mathematical model. For the "constant parameter estimation" problems above, we are modeling truth as a pure constant value (i.e. it has a derivative of zero). What if it is actually a slowly varying parameter (like a slow random walk)? We can still use an EKF, we just need to tell the EKF that truth might deviate a little from our mathematical model! We account for this deviation using the process noise covariance matrix Q. (Prior to starting this problem, make sure to revert any parameters you modified in the last problem to their original values.) Note: to add random process noise to our "truth" model, which we are propagating with Euler integration, ekf_driver does the following:**

$$\dot{x} = f(x,u) + \frac{\sigma_{\xi}}{\sqrt{\Delta t}}(randn) \Longrightarrow$$



a) **Let's start with a well-modeled situation… Let's add process noise to our truth, making it a slowly varying parameter, and we'll properly account for that process noise in our EKF. Specifically, set:** sigma_trueProcessNoise=2 & sigma_ekfProcessNoise=2.
How well does the EKF estimate this slowly varying parameter? How well does the EKF "know" its resulting estimation uncertainty? What happens between measurements?

b) **Given the process noise from (a), what now happens if we have more continuous measurements (**dtMeas=0.01**)? (Revert back to** dtMeas=1 **after answering this question.)**

c) **What if we don't properly account for true process noise in our EKF?**
Set sigma_trueProcessNoise=2 & sigma_ekfProcessNoise=0**. Explain the result.**

d) **What if we account for a lot of process noise that doesn't actually exist?**
Set sigma_trueProcessNoise=0 & sigma_ekfProcessNoise=2**. Explain the result.**

Lecture 10 Homework, 8/8

9) Now, let's add a second state to our EKF and estimate a dynamic (but still linear) system. Consider a vertically-launched projectile with no drag or other disturbances. What comes up must come down. We know it is launching at 85 m/s, give or take 1 m/s (1-sigma). It's launch altitude is zero. We are tracking the projectile's altitude (z) with a radar co-located with the launcher that provides altitude measurements with a 1-sigma noise of 1.5 m, once each second. Our reference and measurement models are:

$$\underline{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} z \\ \dot{z} \end{bmatrix} \qquad \ddot{z} = -g \;\Rightarrow\; \dot{\underline{x}} = f(\underline{x}, g) = \frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \dot{z} \\ \ddot{z} \end{bmatrix} = \begin{bmatrix} \dot{z} \\ -g \end{bmatrix} = \begin{bmatrix} x_2 \\ -g \end{bmatrix}$$

$$\begin{aligned} y_{meas} &= z + \eta_{meas} \\ &= x_1 + \eta_{meas} \end{aligned}$$

Modify a version of "ekf_driver" to emulate this system:

- **Initializations**: x=[0; 85+1*randn]. xhat=[0; 85].
- **Add a g=9.81 m/s^2 sim parameter, and modify other numeric parameters appropriately.**
- **Modify** fTrue=[], f=[], hTrue=[], and h=[]
- **Determine and implement appropriate matrices for A and C (both are constant)**
- **Add the line:** xNames{end+1}='zdot, m/s';

```
sigma_trueProcessNoise = [0; 0];
sigma_trueMeasNoise = ?;
dtMeas = ?;

sigma_ekfInitUncertainty = [0.5; ?];
sigma_ekfProcessNoise = [.001; .001];
sigma_ekfMeasNoise = ?;
```

Defines P$_0$

Defines Q

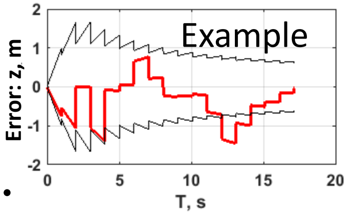
Defines R

"Tuning" Notes:
- Positive P diagonals are desirable, so choosing 0.5m initial *z* uncertainty.
- Model matches truth, but it is desirable to have non-zero diagonal elements in Q. Choosing small "process" noise.

Run the EKF driver. It should stop when altitude is less than zero.

a) Turn in the resulting code (no need to turn in the plotting portion)

b) Turn in an example output plot, noting how well the EKF estimated z and zdot.



c) Explain how the EKF can provide an estimate for a state that it didn't directly measure (i.e. zdot).

d) From the final P matrix, determine the final estimate uncertainties and correlation coefficient (ρ). Does ρ make sense? Explain.

“Hint” email sent to previous class

All,
I definitely realize that the EKF problems are challenging. Let me see if I can make the waters a little less murky with some hints:

- In #5, the only ekf_driver code you will change is modifying the <code here> statements

- In 6-8, you'll adjust the specified parameters and describe the results. (The purpose of these questions is to hopefully gain a little intuition about P, Q, and R (from a scalar perspective).

- Problem 9 requires some real thinking... We are changing from a scalar system to a 2-state system. In addition to modifying the parameters, you'll also need to modify the code lines defining: fTrue, hTrue, f, h, A, and C.

- For this problem, there are 2 states, and 1 measurement. Think about what that means for the sizes of fTrue, hTrue, f, h, A, and C.

- Make sure that fTrue and hTrue are functions of the true states, x(1) & x(2). (They define the true system)

- Make sure that f, h, A & C are functions of the estimated states, xhat(1) and xhat(2). (They are part of the EKF algorithm that is estimating the true system. The EKF algorithm doesn't know truth, it just has an estimate of truth.)

- A is a Jacobian linearization of f with respect to the vector xhat. As an example: For some entirely different 3-state system that isn't relevant to our homework, let's say that the mathematical model for the time derivative of xhat = [xhat(1); xhat(2); xhat(3)] were defined by:

```
          [          xhat(2)+83              ]
    f =   [              0                   ]
          [ xhat(3)*sin(4*xhat(1))   ]
                                        where: xhat_dot = f(xhat)
```

A is the Jacobian of "f" with respect to each element of xhat. Specifically, A(i,j) is the derivative of "f(i)" with respect to "xhat(j)".

Thus, A would be a 3x3 matrix:

```
       [                 0                          1               0              ]
  A =  [                 0                          0               0              ]
       [ 4*xhat(3)*cos(4*xhat(1))        0       sin(4*xhat(1))   ]
```

Why:

A(1,2) = Derivative of "f(1)" wrt "xhat(2)". A(1,2)=1

A(3,1) = Derivative of "f(3)" wrt "xhat(1)". A(3,1) = 4*xhat(3)*cos(4*xhat(1))

etc.

(To be clear: The above example is NOT relevant to our homework problems.)

- Similar process for C, except that C will be mxn, where m is the number of measurements and n is the number of states:

C(i,j) is the derivative of "h(i)" with respect to "xhat(j)".