State Estimation with a Kalman Filter

- When I drive into a tunnel, my GPS continues to show me moving forward, even though it isn’t getting any new position sensing data
  - How does it work?
- A Kalman filter produces estimate of system’s next state, given
  - noisy sensor data
  - control commands with uncertain effects
  - model of system’s (possibly stochastic) dynamics
  - estimate of system’s current state
- In our case, given
  - a blimp with (approximately known) dynamics
  - noisy sensor data
  - control commands
  - our current estimate of blimp’s state
- How should we predict the blimp’s next state?
  - How should we control blimp?
Kalman Filter

- Bayesian estimator, computes beliefs about state, assuming everything is linear and Gaussian
  - Gaussian is unimodal $\Rightarrow$ only one hypothesis
  - Example of a Bayes filter
- “Recursive filter,” since current state depends on previous state, which depends on state before that, and so on
- Two phases: prediction (not modified by data) and correction (data dependent)
- Produces estimate with minimum mean-squared error
- Even though the current estimate only looks at the previous estimate, as long as the actual data matches the assumptions (Gaussian etc), you can’t do any better, even if you looked at all the data in batch!
- Very practical and relatively easy to use technique!
Example

- Object falling in air
- We know the dynamics
  - Related to blimp dynamics, since drag and inertial forces are both significant
  - Dynamics same as driving blimp forward with const fan speed
- We get noisy measurements of the state (position and velocity)
- We will see how to use a Kalman filter to track it

![Position of object falling in air](image1)

**Position of object falling in air, Meas Nz Var= 0.0025 Proc Nz Var= 0.0001**

![Velocity of object falling in air](image2)

**Velocity of object falling in air**
Linear 1D Newtonian dynamics example
Object falling in air

State is \((x, v)\)

where \(v = \frac{dx}{dt}\)
Linear 1D Newtonian dynamics example
Object falling in air

\[ f = ma = m \frac{dv}{dt} \]

\[ f_a = -kv \text{ force due to drag} \]

(ideally we'd use \( v^2 \) instead of \( v \))

\[ f_g = -gm \]

\[ f = f_a + f_g = -kv - mg \]

\[ m \frac{dv}{dt} = -kv - mg \]

\[ \frac{dv}{dt} = -\frac{kv}{m} - g \]
How to solve the differential equation on the computer?

\[
\frac{dx}{dt} = v
\]

\[
\frac{\Delta x}{\Delta t} = v
\]

\[
x_t - x_{t-1} = v \Delta t
\]

\[
x_t = x_{t-1} + v \Delta t
\]

\[
\frac{dv}{dt} = -\frac{kv}{m} - g
\]

\[
\Delta v = -\frac{kv}{m} - g
\]

\[
v_t - v_{t-1} = -\frac{kv}{m} - g
\]

\[
v_t = v_{t-1} - \left(\frac{kv}{m} + g\right) \Delta t
\]
Object falling in air

Produced by iterating the difference equations on previous slide

At terminal velocity
In air, heavier objects do fall faster!

And they take longer to reach their terminal velocity

(Without air, all objects accelerate at the same rate)
Matlab code for modeling object falling in air

% falling.m
x0 = 0.0;
v0 = 0.0;
TMAX = 200;
x = zeros(1,TMAX);
V = zeros(1,TMAX);
g=9.8;
m=1.0;
k=10.0;
x(1) = x0;
v(1) = v0;
dt=0.01;
for t=2:TMAX,
    x(t) = x(t-1)+(v(t-1))*dt;
    v(t) = v(t-1)+(-(k/m)*(v(t-1))-g)*dt;
end
figure();
plot(x,'b'); hold on;
title(['Falling object k/m = ' num2str(k/m)]);
plot(v,'r');
legend('x','v'); hold off
Multi-dimensional Gaussians

\[ P(x \mid m, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(x - m)^2}{2\sigma^2}\right] \]

where

\[ Z(R) = \left( \det(R) / 2\pi \right)^{-1/2} \]

and \( R \) is the inverse of the covariance matrix.

One-dimensional (scalar) Gaussian

Vector Gaussian

\[ P(x \mid m, R) = \frac{1}{Z(R)} \exp\left[-\frac{1}{2} (x - m)^T R (x - m) \right] \]
Multi-dimensional Gaussians, Covariance Matrices, Ellipses, and all that

In an N dimensional space \( \mathbf{x x}^T = R^2 \) is a sphere of radius \( R \)

Note that \( \mathbf{x x}^T = \langle \mathbf{x} \bullet \mathbf{x} \rangle = x_1^2 + x_2^2 + ... + x_N^2 = R^2 \)

Can write it more generally by inserting identity matrix \( \mathbf{x x}^T = \mathbf{x I x}^T = R^2 \)

If we replace \( \mathbf{I} \) by a more general matrix \( \mathbf{M} \), it will distort the sphere: for \( \mathbf{M} \) diagonal, it will scale each axis differently, producing an axis-aligned ellipsoid

We could also apply rotation matrices to a diagonal \( \mathbf{M} \) to produce a general, non-axis aligned ellipsoid

\( \Rightarrow \) The uncertainty “ball” of a multi-D Gaussian [e.g. the 1 std iso-surface] actually IS an ellipsoid!
Example

\[ A = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} \]

Blue: circle of radius 1 (e.g. 1 std iso-surface of uncorrelated uniform Gaussian noise)
Red: circle after transformation by A

\[ A = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix} \]
\[
A = \begin{pmatrix} 0.1 & 0 \\ 0 & 2 \end{pmatrix}
\]

With \( R = \begin{pmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{pmatrix} \) and \( \theta = -\frac{\pi}{16} \),

\[
A = R^T \begin{pmatrix} 0.1 & 0 \\ 0 & 2 \end{pmatrix} R = \begin{pmatrix} 0.172 & 0.363 \\ 0.363 & 1.93 \end{pmatrix}
\]
End
Kalman filter variables

$x$: state vector
$z$: observation vector
$u$: control vector

$A$: state transition matrix --- dynamics
$B$: input matrix (maps control commands onto state changes)
$P$: covariance of state vector estimate
$Q$: process noise covariance
$R$: measurement noise covariance
$H$: observation matrix
Kalman filter algorithm

Prediction for state vector and covariance:
\[
\bar{x} = Ax + Bu
\]
\[
\bar{P} = APA^T + Q
\]

Kalman gain factor:
\[
K = \bar{P}H^T (HH^T + R)^{-1}
\]

Correction based on observation:
\[
x = \bar{x} + K(z - H\bar{x})
\]
\[
P = \bar{P} - KH\bar{P}
\]

\(x\): state vector
\(z\): observation vector
\(u\): control vector
\(A\): state transition matrix --- dynamics
\(B\): control commands --> state changes
\(P\): covariance of state vector estimate
\(Q\): process noise covariance
\(R\): measurement noise covariance
\(H\): observation matrix
Need dynamics in matrix form

\[ x_t = x_{t-1} + v_{t-1} dt; \]
\[ v_t = v_{t-1} - \left( \frac{k}{m} v_{t-1} + g \right) dt; \]

Want \( A \) s.t.

\[
\begin{pmatrix}
    x_t \\
    v_t
\end{pmatrix} = \begin{pmatrix}
    A
\end{pmatrix} \begin{pmatrix}
    x_{t-1} \\
    v_{t-1}
\end{pmatrix}
\]

Try

\[
A \begin{pmatrix}
    x_{t-1} \\
    v_{t-1}
\end{pmatrix} = \begin{pmatrix}
    1 & \Delta t \\
    0 & 1 - \frac{k}{m} \Delta t
\end{pmatrix} \begin{pmatrix}
    x_{t-1} \\
    v_{t-1}
\end{pmatrix} = \begin{pmatrix}
    x_{t-1} + v_{t-1} \Delta t \\
    v_{t-1} - \frac{k}{m} v_{t-1} \Delta t
\end{pmatrix} = \begin{pmatrix}
    x_t \\
    v_t
\end{pmatrix}
\]

Hmm, close but gravity is missing...we can’t put it into \( A \) because it will be multiplied by \( v_t \). Let’s stick it into \( B \)!
Need dynamics in matrix form

Want

\[ x_t = x_{t-1} + v_{t-1} dt; \]
\[ v_t = v_{t-1} - \left( \frac{k}{m} v_{t-1} + g \right) dt; \]

Try \( Bu = \begin{pmatrix} 1 & 0 \\ 0 & -g \Delta t \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ -g \Delta t \end{pmatrix} \)

\[
\begin{pmatrix} x_t \\ v_t \end{pmatrix} = A \begin{pmatrix} x_{t-1} \\ v_{t-1} \end{pmatrix} + B \begin{pmatrix} 0 \\ 1 \end{pmatrix}
\]

\[
= \begin{pmatrix} 1 & \Delta t \\ 0 & 1 - \frac{k}{m} \Delta t \end{pmatrix} \begin{pmatrix} x_{t-1} \\ v_{t-1} \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 0 & -g \Delta t \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} x_{t-1} + v_{t-1} \Delta t \\ v_{t-1} - \left( \frac{k}{m} v_{t-1} + g \right) \Delta t \end{pmatrix}
\]

Now gravity is in there...we treated it as a control input.
Matlab for dynamics in matrix form

% falling_matrix.m: model of object falling in air, w/ matrix notation
x0 = 0.0; v0 = 0.0;
TMAX = 200;
x = zeros(2,TMAX);
g=9.8;
m=1.0;
k=10.0;
x(1,1) = x0; x(2,1) = v0;
dt=0.01;
u=[0 1]';
for t=2:TMAX,
    A=[[1      dt     ]; ... 
        [0 (1.0-(k/m)*dt)]]; 
    B=[[1    0     ]; ... 
        [0  -g*dt   ]]; 
    x(:,t) = A*x(:,t-1) + B*u;
end
% plotting
Matlab for Kalman Filter

function s = kalmanf(s)
    s.x = s.A*s.x + s.B*s.u;
    s.P = s.A * s.P * s.A' + s.Q;
    % Compute Kalman gain factor:
    K = s.P * s.H' * inv(s.H * s.P * s.H' + s.R);
    % Correction based on observation:
    s.x = s.x + K*(s.z - s.H*s.x);
    s.P = s.P - K*s.H*s.P;
end
return

The most computationally difficult step
Position of object falling in air, Meas Nz Var= 0.01 Proc Nz Var= 0.0001

Velocity of object falling in air

Observations, Kalman output, true dynamics
Calling the Kalman Filter (init)

\[ x_0 = 0.0; \ v_0 = 0.0; \]
\[ TMAX = 200; \]
\[ g=9.8; \]
\[ m=1.0; \ k=10.0; \]
\[ dt=0.01; \]

\[
\text{clear s} \quad \% \quad \text{Dynamics modeled by A}
\]
\[
\text{s.A} = \begin{bmatrix} 1 & dt \\ 0 & (1.0-(k/m)\times dt) \end{bmatrix}; \quad \ldots
\]

\[
\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}
\]
Calling the Kalman Filter (init)

% Measurement noise variance
MNstd = 0.4;
MNV = MNstd*MNstd;
% Process noise variance
PNstd = 0.02;
PNV = PNstd*PNstd;
% Process noise covariance matrix
s.Q = eye(2)*PNV;
% Define measurement function to return the state
s.H = eye(2);
% Define a measurement error
s.R = eye(2)*MNV; % variance

Q, H, and R are all diagonal!
Calling the Kalman Filter (init)

% Use control to include gravity
s.B = eye(2); % Control matrix
s.u = [0 -g*m*dt]'; % Gravitational acceleration
% Initial state:
s.x = [x0 v0]';
s.P = eye(2)*MNV;
s.detP = det(s.P); % Let's keep track of the noise by keeping detP
s.z = zeros(2,1);
Calling the Kalman Filter

% Simulate falling in air, and watch the filter track it
tru=zeros(TMAX,2); % true dynamics
tru(1,:)=[x0 v0];
detP(1,:)=s.detP;
for t=2:TMAX
    tru(t,:)=s(t-1).A*tru(t-1,:)' + s(t-1).B*s(t-1).u+PNstd *randn(2,1);
    s(t-1).z = s(t-1).H * tru(t,:) + MNstd*randn(2,1); % create a meas.
    s(t)=kalmanf(s(t-1)); % perform a Kalman filter iteration
    detP(t)=s(t).detP; % keep track of "net" uncertainty
end

The variable s is an object whose members are all the important data structures (A, x, B, u, H, z, etc)

tru: simulation of true dynamics. In the real world, this would be implemented by the actual physical system
Here we are assuming we can switch gravity back and forth (from -10 to +10 and back)
Blimp P control
using raw sensor data, no Kalman filter

Position of blimp (in air), Meas Nz Var= 25 Proc Nz Var= 0.09

Velocity of blimp (in air)

Det(P)

Commands

Servoing to setpoint -50
Mean Squared Error: 0.143
RMS: 0.379
Blimp P control using Kalman-filtered state estimate

Position of blimp (in air), Meas Nz Var= 25 Proc Nz Var= 0.09

Velocity of blimp (in air)

Det(P)

Commands

Servoing to setpoint -50
Mean Squared Error: 0.0373
RMS: 0.193
How to apply this for our blimps?

(* Let's do it in Mathematica! *)
(* Here is the dynamics model *)

ClearAll["Global`*"

\[ A = \{\{1,\text{dt}\},\{0,1-((k/m)* \text{dt})\}\}; \]
\[ B=\{\{1,0\},\{0,-g \text{ dt}\}\}; \]
\[ \text{state} = \{x,v\}; \]
\[ \text{cmd} = \{0,1\}; \]
\[ A . \text{state} + B . \text{cmd} \quad (* \text{Dot is matrix multiply} *) \]

\[ \{\text{dt v+x,-dt g+(1-(\text{dt k})/m) v}\} \]
Definitions

g=.;
MNstd = 0.4;
MNV = MNstd*MNstd;
PNstd = 0.02;
PNV = PNstd*PNstd;
Q = {{PNV,0},{0,PNV}};
H = {{1,0},{0,1}};
R = {{MNV,0},{0,MNV}};
P = {{MNV,0},{0,MNV}};
The Kalman filter

\[
\text{PredState} = \text{Simplify}[A \cdot \text{state} + B \cdot \text{cmd}]; \\
\text{PredUncertainty} = \text{Simplify}[A \cdot P \cdot \text{Transpose}[A] + Q]; \\
\text{Kgain} = \text{Simplify}[\text{PredUncertainty} \cdot \text{Transpose}[H] \cdot \text{Inverse}[H \cdot \text{PredUncertainty} \cdot \text{Transpose}[H] + R]]; \\
\text{CorrectedState} = \text{Simplify}[\text{state} + \text{Kgain} \cdot (\{z_1, z_2\} - H \cdot \text{PredState})]; \\
\text{CorrectedUncertainty} = \text{Simplify}[\text{PredUncertainty} + \text{Kgain} \cdot H \cdot \text{PredUncertainty}];
\]
Result? Kind of a mess

\[
\text{In[23]} := \text{Rgain}
\]

\[
\text{Out[23]} := \begin{bmatrix}
0. - \frac{0.0513922}{0.102656 + dt^2 \left(0.025664 + \frac{0.051264 k^2}{m^2}\right) - \frac{0.102528 \, dt \, k \, m}{m}} - \\
0.102528 \, dt \, k \, m + 0.102656 \, n^2 + dt^2 \left(0.051264 \, k^2 + 0.025664 \, m^2\right)
\end{bmatrix}
\]

\[
\text{Out[23]} := \begin{bmatrix}
0. - \frac{0.0513922}{0.102656 + dt^2 \left(0.025664 + \frac{0.051264 k^2}{m^2}\right) - \frac{0.102528 \, dt \, k \, m}{m}} - \\
0.102528 \, dt \, k \, m + 0.102656 \, n^2 + dt^2 \left(0.051264 \, k^2 + 0.025664 \, m^2\right)
\end{bmatrix}
\]

\[
\text{Out[23]} := \begin{bmatrix}
0. - \frac{0.0513922}{0.102656 + dt^2 \left(0.025664 + \frac{0.051264 k^2}{m^2}\right) - \frac{0.102528 \, dt \, k \, m}{m}} - \\
0.102528 \, dt \, k \, m + 0.102656 \, n^2 + dt^2 \left(0.051264 \, k^2 + 0.025664 \, m^2\right)
\end{bmatrix}
\]

\[
\text{Out[23]} := \begin{bmatrix}
0. - \frac{0.0513922}{0.102656 + dt^2 \left(0.025664 + \frac{0.051264 k^2}{m^2}\right) - \frac{0.102528 \, dt \, k \, m}{m}} - \\
0.102528 \, dt \, k \, m + 0.102656 \, n^2 + dt^2 \left(0.051264 \, k^2 + 0.025664 \, m^2\right)
\end{bmatrix}
\]

\[
\text{Out[23]} := \begin{bmatrix}
0. - \frac{0.0513922}{0.102656 + dt^2 \left(0.025664 + \frac{0.051264 k^2}{m^2}\right) - \frac{0.102528 \, dt \, k \, m}{m}} - \\
0.102528 \, dt \, k \, m + 0.102656 \, n^2 + dt^2 \left(0.051264 \, k^2 + 0.025664 \, m^2\right)
\end{bmatrix}
\]

\[
\text{Out[23]} := \begin{bmatrix}
0. - \frac{0.0513922}{0.102656 + dt^2 \left(0.025664 + \frac{0.051264 k^2}{m^2}\right) - \frac{0.102528 \, dt \, k \, m}{m}} - \\
0.102528 \, dt \, k \, m + 0.102656 \, n^2 + dt^2 \left(0.051264 \, k^2 + 0.025664 \, m^2\right)
\end{bmatrix}
\]

\[
\text{Out[23]} := \begin{bmatrix}
0. - \frac{0.0513922}{0.102656 + dt^2 \left(0.025664 + \frac{0.051264 k^2}{m^2}\right) - \frac{0.102528 \, dt \, k \, m}{m}} - \\
0.102528 \, dt \, k \, m + 0.102656 \, n^2 + dt^2 \left(0.051264 \, k^2 + 0.025664 \, m^2\right)
\end{bmatrix}
\]

\[
\text{Out[23]} := \begin{bmatrix}
0. - \frac{0.0513922}{0.102656 + dt^2 \left(0.025664 + \frac{0.051264 k^2}{m^2}\right) - \frac{0.102528 \, dt \, k \, m}{m}} - \\
0.102528 \, dt \, k \, m + 0.102656 \, n^2 + dt^2 \left(0.051264 \, k^2 + 0.025664 \, m^2\right)
\end{bmatrix}
\]

\[
\text{Out[23]} := \begin{bmatrix}
0. - \frac{0.0513922}{0.102656 + dt^2 \left(0.025664 + \frac{0.051264 k^2}{m^2}\right) - \frac{0.102528 \, dt \, k \, m}{m}} - \\
0.102528 \, dt \, k \, m + 0.102656 \, n^2 + dt^2 \left(0.051264 \, k^2 + 0.025664 \, m^2\right)
\end{bmatrix}
\]

\[
\text{Out[23]} := \begin{bmatrix}
0. - \frac{0.0513922}{0.102656 + dt^2 \left(0.025664 + \frac{0.051264 k^2}{m^2}\right) - \frac{0.102528 \, dt \, k \, m}{m}} - \\
0.102528 \, dt \, k \, m + 0.102656 \, n^2 + dt^2 \left(0.051264 \, k^2 + 0.025664 \, m^2\right)
\end{bmatrix}
\]

etc
Put in numerical values for constants

\[
\begin{align*}
    m &= 1; \\
    k &= 1; \\
    dt &= 0.01; \\
    K_{\text{gain}} &= \{\{0.500637, 0.00249354\}, \{0.00249354, 0.495599\}\} \\
    \text{CorrectedState} &= \{x + 0.500637 (-0.01 v - x + z_1) + 0.00249354 (-v + 0.01 (g + v) + z_2), \\
    &\quad v + 0.00249354 (-0.01 v - x + z_1) + 0.495599 (-v + 0.01 (g + v) + z_2)\} \\
\end{align*}
\]

*The filter is a linear combination of previous state estimates, and sensor values (z1 and z2)!*
Extensions

- Extended Kalman Filter (EKF)
  - Non-linear generalization…assumes linearization around the currently estimated state
  - Standard for GPS, robot navigation, etc

- Information Filter
  - A variant that allows N previous sensor values to be processed in one filter update step

- Unscented Kalman Filter (UKF)…the unscented Kalman Filter does not stink!
  - Use a sampling of points to represent distributions
Extra examples…various noise settings
Position of object falling in air, Meas Nz Var= 0.16 Proc Nz Var= 0.0004

Velocity of object falling in air

Det(P)
Process noise $Q = 0.0$
Process noise $Q = 0.0$
Position of object falling in air $\sigma^2 = 1$

Velocity of object falling in air

Process noise $Q = 0.0$
Position of object falling in air $\sigma^2 = 0.01$

Velocity of object falling in air

Process noise $Q = 0.02$