KALMAN FILTER

A Brief Introduction and q/kdb+ Implementation

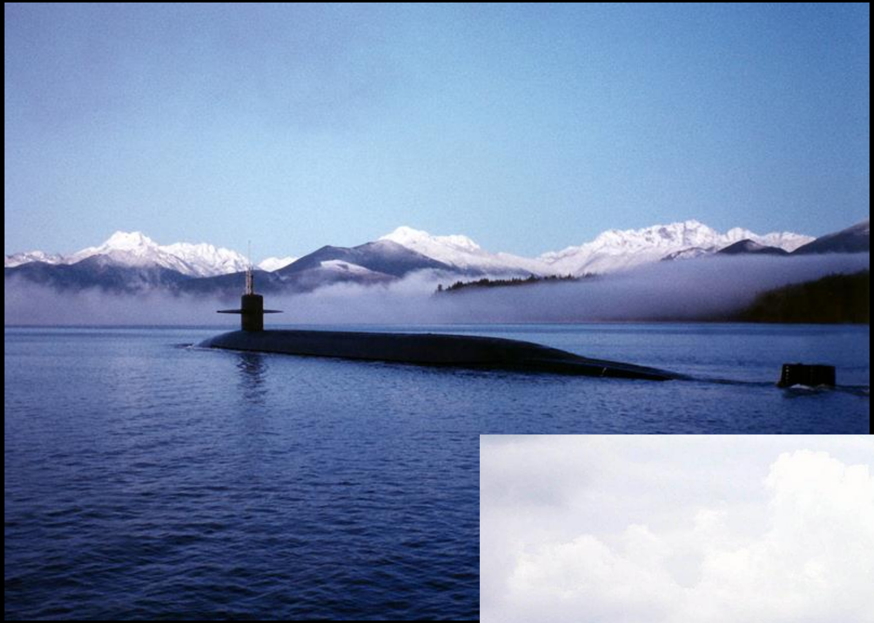
Mark Lefevre

THE KALMAN FILTER

- A statistical algorithm that produces accurate estimates of unknown state variables in the presence of noise and other inaccuracies
- Recursive and can be used in real-time
- Uses a 2 Step Process
  - **Predict** – estimate current **state variables**
  - **Update** – use next measurement to update estimate using a **weighted average**
- Noise Smoothing
- State Estimation
- Quick Convergence

APPLICATIONS OF KALMAN FILTERS

- Target tracking
  - Satellites, Missiles, Aircraft, Spacecraft, UAVs
- Guidance/Navigation (GPS, IM)
- Sensor fusion (ex. RADAR, LIDAR, LASER)
- Computer Vision
- Economics
- Finance

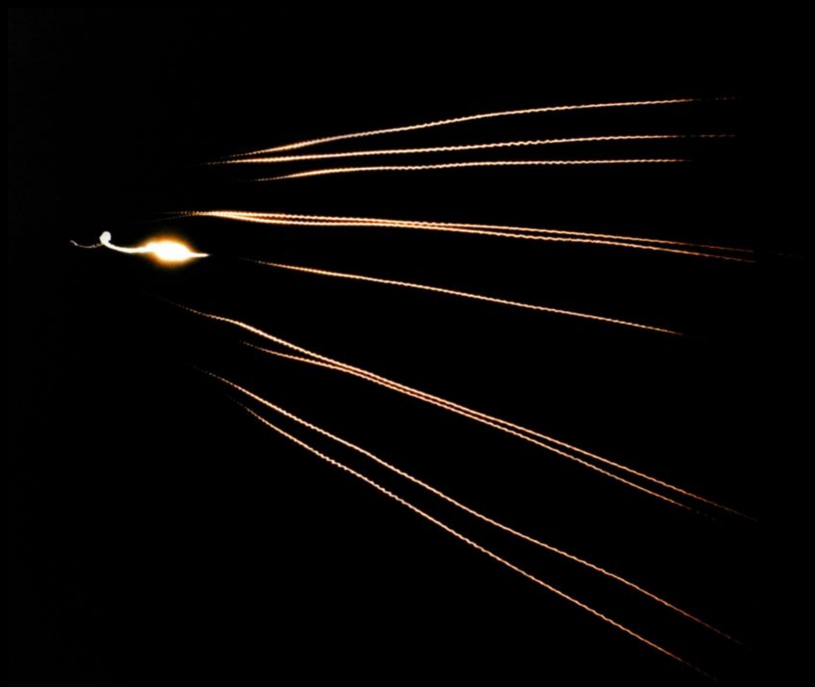


REALISTIC BALLISTIC TRAJECTORY

- Aerodynamic Drag

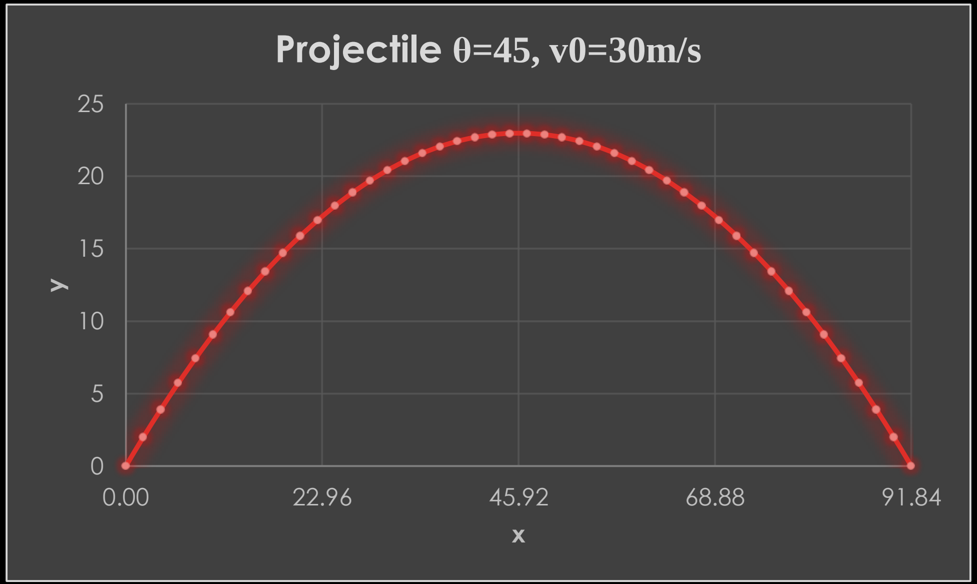
- Altitude Effects
    - Changes in Air Density
    - Decrease in Gravity
- Winds/Weather
- Coriolis Effect
- Earth Curvature
- Target Motion

- Internal Propulsion
- Multiple Stages/MIRV



SIMPLE PROJECTILE MOTION

- Instantaneous Launch
- No Aerodynamic Drag, Winds
- Only Acceleration due to Gravity
  - $g = -9.8\frac{m}{s^2}$



Equations

- Acceleration

$$a_x = 0$$
$$a_y = g$$

- Velocity

$$v_x = v_0 \cos\theta$$
$$v_y = v_0 \sin\theta + gt$$

Displacement

$$x = x_0 + v_0 t \cos\theta$$
$$y = y_0 + v_0 t \sin\theta + \frac{1}{2}gt^2$$

MEASUREMENT PROBLEM

- Actual state of the system is not directly observable
- Any measurement of the system outputs are unavoidably *noisy*
- **Process Model**

$$x_k = Ax_{k-1} + Bu_k + w_{k-1}$$

- **Measurement Model**

$$z_k = Hx_k + v_k$$

- **Process and Measurement Noise**

$$p(w) \sim N(0, Q)$$
$$p(v) \sim N(0, R)$$

STATE-SPACE MODEL

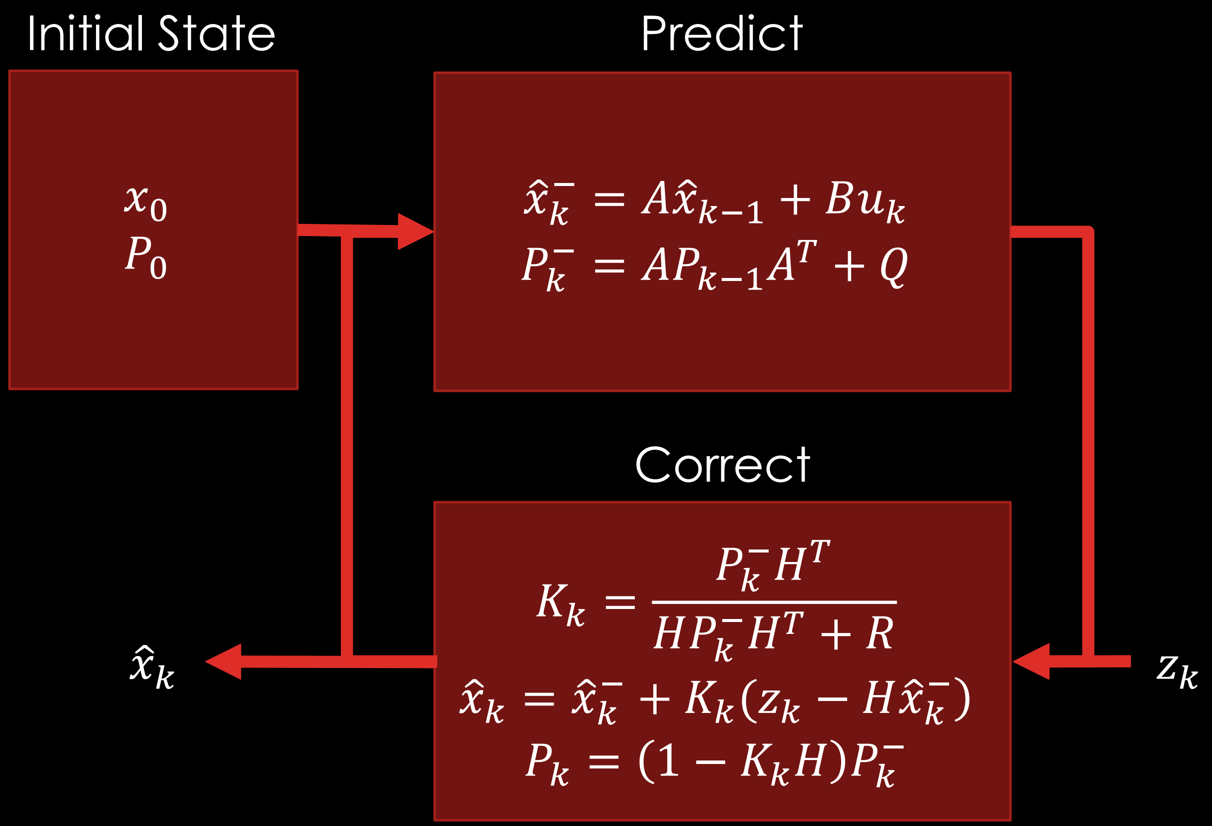
- Discretize Equations
  - Use matrix notation
    $$x_k = Ax_{k-1} + Bu_k$$
  - A, ***State Transition Model***
  - B, ***Control-Input Model***
  - x_k, current state
  - x_{k-1}, previous state
  - u_k, static vector
- State Example
  $$x_k = \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix}$$

```
/ create_stm: Create a state transition model for a projectile in 2D
/ [d]elta [t]ime is the change in time for each time step
create_stm:{[dt] 4 4#@[16#0.;*[5;til 4],2 7;:;#[4;1.],#[2;dt]]};

/ create_cim: Create a control-input model for a projectile in 2D
/   CIM accounts for the effect of acceleration on position and velocity
create_cim:{[dt] 4 2#@[8#0.;0 3 4 7;:;raze 2#'(0.5*dt*dt;dt)]};

/ create_initial_state: Create an initial state
/ [t]heta, angle at which projectile is fired in degrees
/ [v]elocity, initial velocity at which projectile is fired
create_initial_state:{[t;v]
    pi:acos -1;
    r:%[t;180]*pi;
    4 1#@[4#0.;2 3;:;(v*cos r;v*sin r)]
    };
```

A KALMAN FILTER



- Choice an Initial State
- Predict
  1. Project the state
  2. Project the error covariance
- Correct
  1. Compute the Kalman Gain
  2. Update estimate with measurement
  3. Update the error covariance

DISCRETE KALMAN FILTER

- Predictor-Corrector Estimator
- Maintains 2 statistical moments (state, error covariance)
- Minimizes Error Covariance
- Define *a priori* and *a posteriori* Estimate Errors

$$e_k^- \equiv x_k - \hat{x}_k^-$$
$$e_k \equiv x_k - \hat{x}_k$$

- Estimate *a priori* and *a posteriori* Covariances

$$P_k^- = E\left[e_k^- e_k^{-T}\right]$$
$$P_k = E\left[e_k e_k^{-T}\right]$$

- Compute *a posteriori* State Estimate

$$\hat{x}_k = \hat{x}_k^- + K\left(z_k - H\hat{x}_k^-\right)$$

KALMAN GAIN

- Kalman Gain is Chosen to Minimize *a posteriori* Error Covariance

$$K_k = \frac{P_k^- H^T}{H P_k^- H^T + R}$$

- Blending Factor which Weights the *a priori* Estimated State and the Measurement
  - Value is between 0 and 1
- As Measurement Error Covariance *R* tends to 0
  - Residual has more weight
- As *a priori* Estimated Error Covariance P_k^- tends to 0
  - *a priori* Estimated State has more weight

Q/KDB+ CODE

```
/ Projectile Motion Parameters

theta:45;

v0:300;

g:-9.8;

tof:(2*v0*sin acos[-1]*theta%180)%neg g;

nsteps:1000;

dt:tof%nsteps;


stm:create_stm[dt];

cim:create_cim[dt];

is:create_initial_state[45;v0];

results:{flip `x`y`vx`vy!flip x} raze each

    nsteps {[stm;cim;x] (stm$x)+cim$2 1#0 -
9.8}[stm;cim;]\ is;

results:`t xcols update t:0+dt*til 1+nsteps from
results;
```

```
/ Kalman Filter Matrices/Helper Functions

create_Q:{[dt;v] G:4 1#raze 2#'(0.5*dt*dt;dt);
Q:*[v;v]*G$flip G};

create_P:{4 4#16#x,4#0.};

create_H:{2 4#rotate[-2;8#x,4#0.]};

create_R:{2 2#4#x,2#0.};

create_I:{4 4#16#1.,4#0.};

Q:create_Q[dt;1.];

P0:create_P[10.];

H:create_H[1.];

ra:5.0;

R:create_R[ra*ra];

I:create_I[];

u:2 1#0.,g;
```

Q/KDB+ CODE

```
/ Predict

predState:{[A;x;B;u] (A$x)+B$u};

predErrCov:{[A;P;Q] Q+(A$P)$flip A};

predict:{[A;x;B;u;P;Q]
(predState[A;x;B;u];predErrCov[A;P;Q])};

/ Create a projection (only x (state variables) and P
(error covariance matrix) change each iteration)

predict_p:predict[stm;;cim;u;;Q];


kf:{[x;P;Z]

    aposteriori:correct_p[;;Z] . reverse
apriori:predict_p . (x;P);

    apriori,aposteriori

    };
```

```
/ Correct

computeKalmanGain:{[P;H;R] (P$flip H)$inv R+(H$P)$flip
H};

updateEstimate:{[prevxhat;K;z;H] prevxhat+K$z-
H$prevxhat};

updateErrCov:{[I;K;H;P] (I-K$H)$P};

correct:{[prevP;H;R;prevxhat;Z;I]

    K:computeKalmanGain[prevP;H;R];

    xhat:updateEstimate[prevxhat;K;Z;H];

    P:updateErrCov[I;K;H;prevP];

    (K;xhat;P)

    };

/ Create a projection (only prevP (a priori error
covariance estimate),

/     prevxhat (a priori state estimate), Z (new
measurement) change each iteration)

correct_p:correct[;H;R;;;I];
```

Q/KDB+ CODE

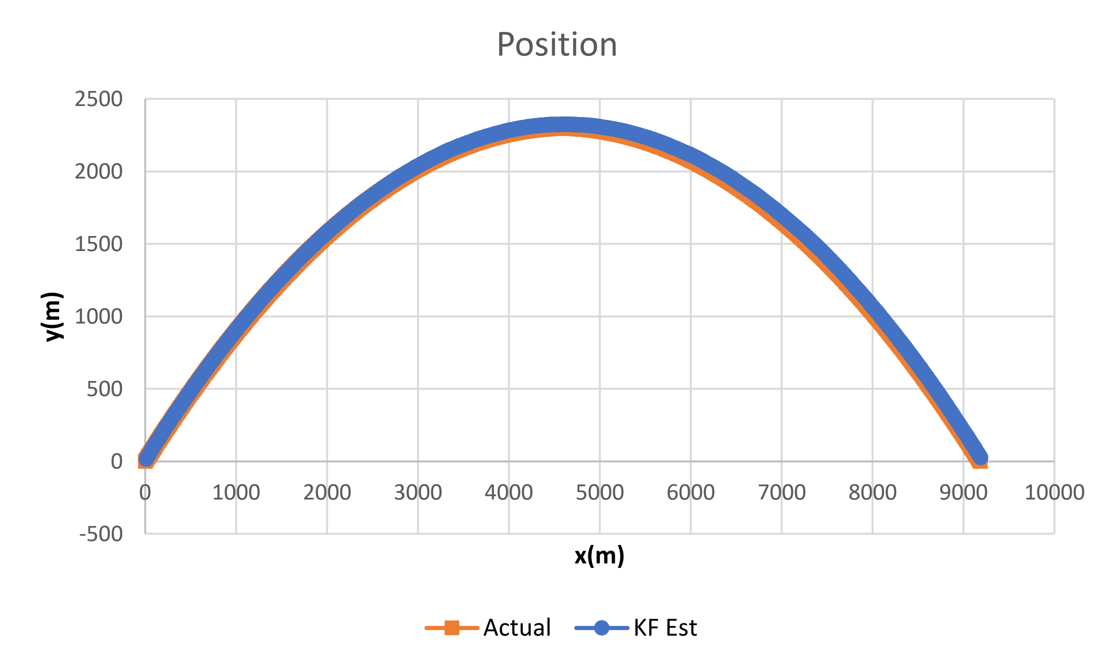
```
/ Create noisy measurements
/ Use Psaris' bm implementation for normal variates
\l stat.q
update vxn:vx+sqrt[R[0;0]]*.stat.bm
nsteps?1f,vyn:vy+sqrt[R[1;1]]*.stat.bm nsteps?1f from
`results where i>0;
history:();
x:create_initial_state[55;1.75*v0];
P:P0;
measurements:select vxn,vyn from results where i>0;
while[0<count measurements;
    Z:2 1#value measurements[0];
    history:history,enlist res:kf[x;P;Z];
    measurements:1_measurements;
    x:res 3; P:res 4;
    ];
```

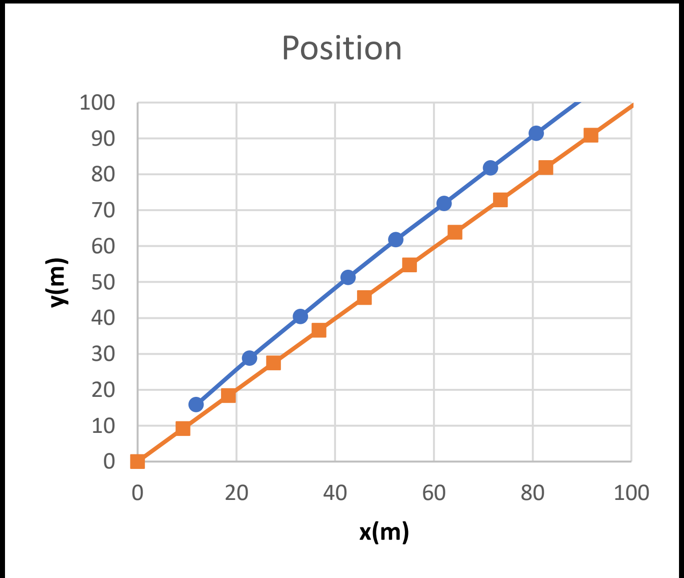
t	x	y	vx	vy	vxn	vyn
0	0	0	212.132	212.132	0n	0n
0.04329225	9.183673	9.17449	212.132	211.7078	211.5433	210.1631
0.0865845	18.36735	18.33061	212.132	211.2835	209.3	202.8508
0.1298768	27.55102	27.46837	212.132	210.8592	212.5902	201.0393
0.173169	36.73469	36.58776	212.132	210.435	200.4968	217.8728
0.2164613	45.91837	45.68878	212.132	210.0107	215.6915	202.8268
0.2597535	55.10204	54.77143	212.132	209.5864	206.9672	212.3463
0.3030458	64.28571	63.83571	212.132	209.1622	224.7602	219.5127
0.346338	73.46939	72.88163	212.132	208.7379	223.8947	203.4936

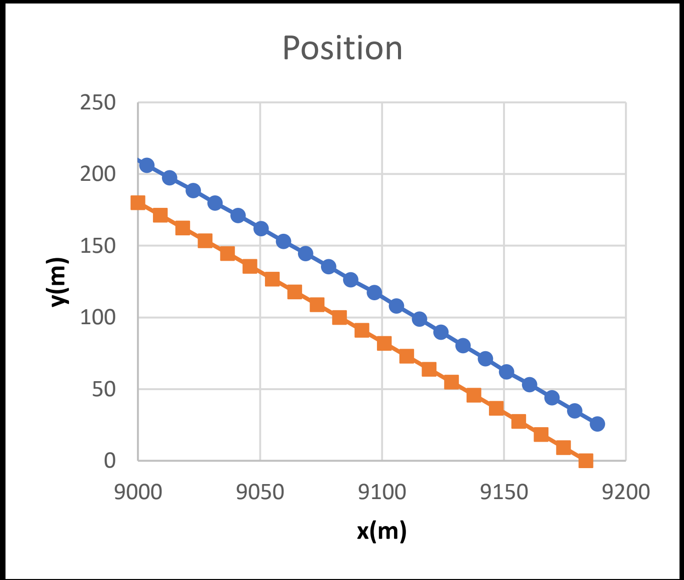
KALMAN FILTER EXAMPLE
POSITION ESTIMATES

Actual Parameters	
theta (d)	45j
theta (r)	0.785398
v0	300j
g	-9.8
v0x	212.132
v0y	212.132
tof	43.29225
nsteps	1000j
dt	0.043292

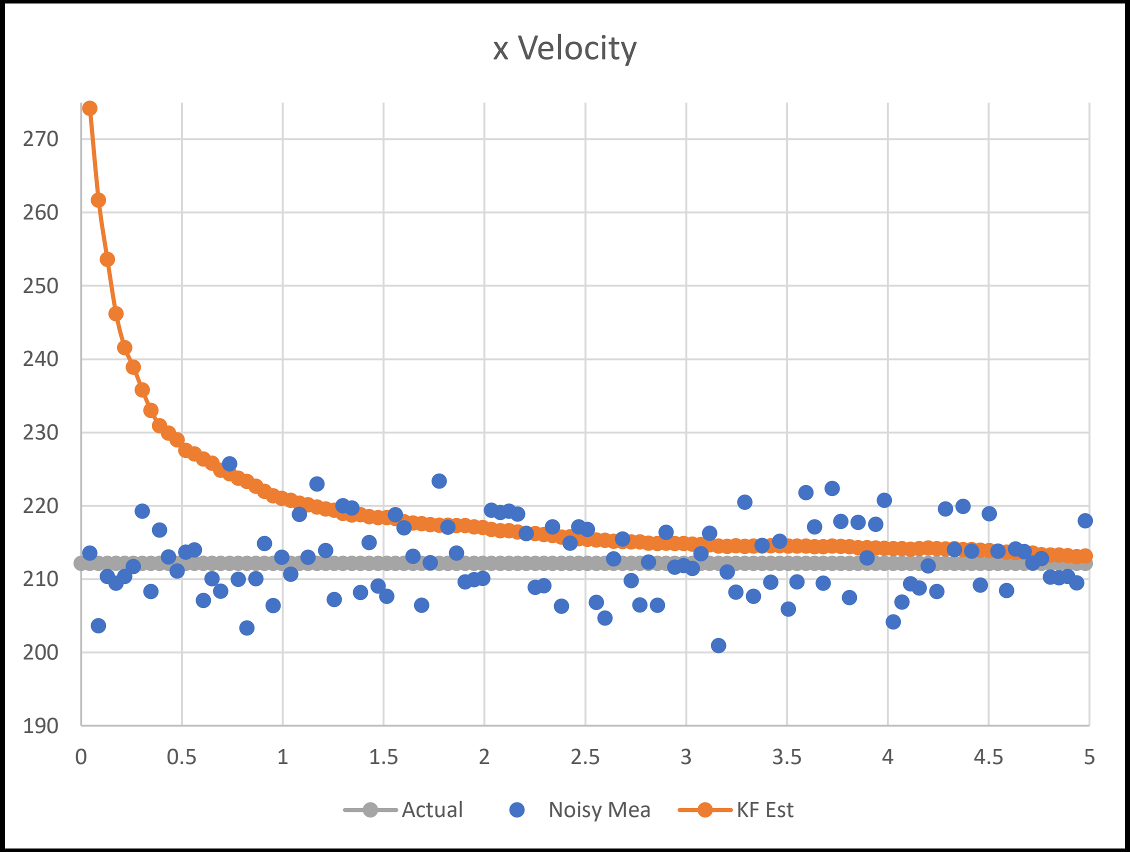
Initial State (v0=525, theta=55)	
x	0
y	0
vx	301.1276
vy	430.0548

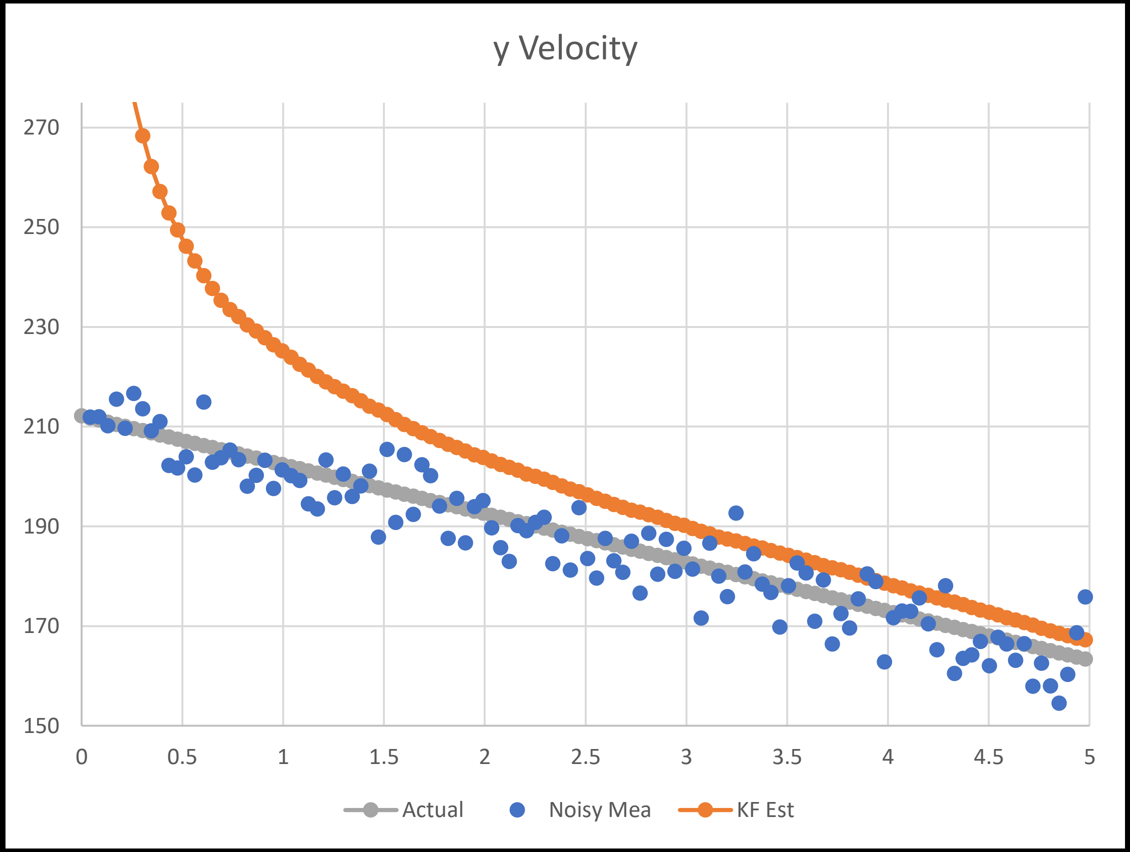




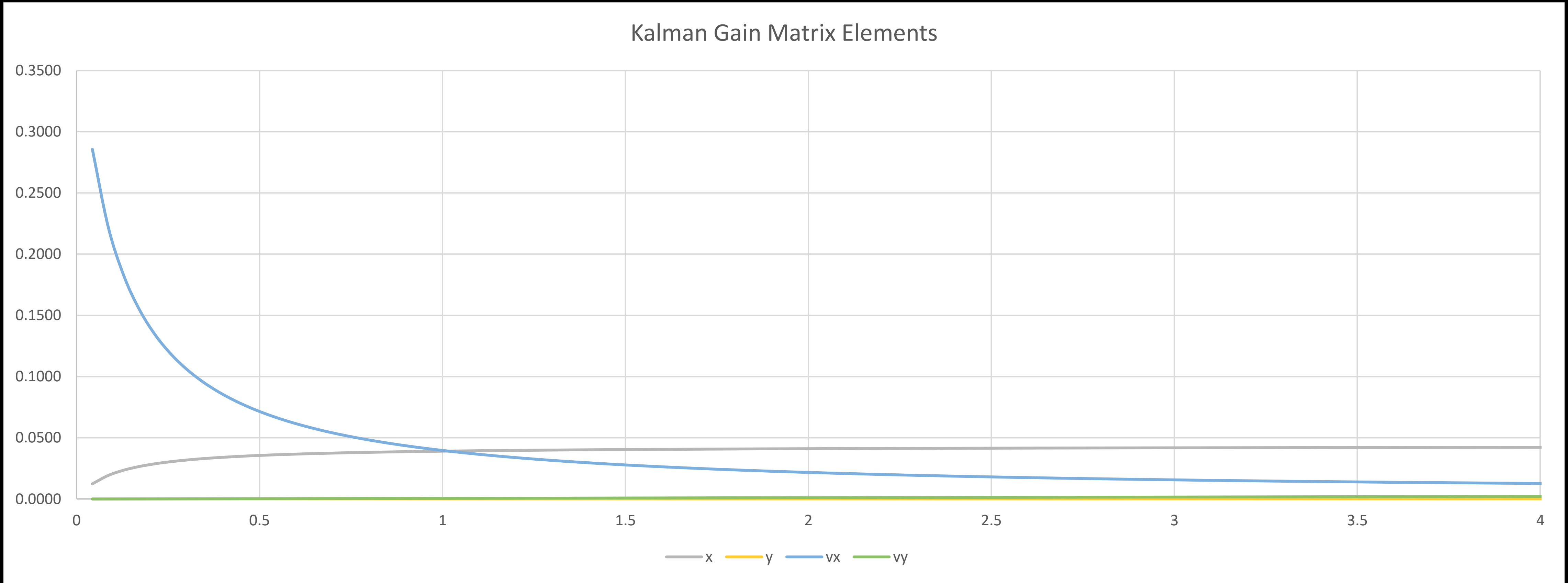


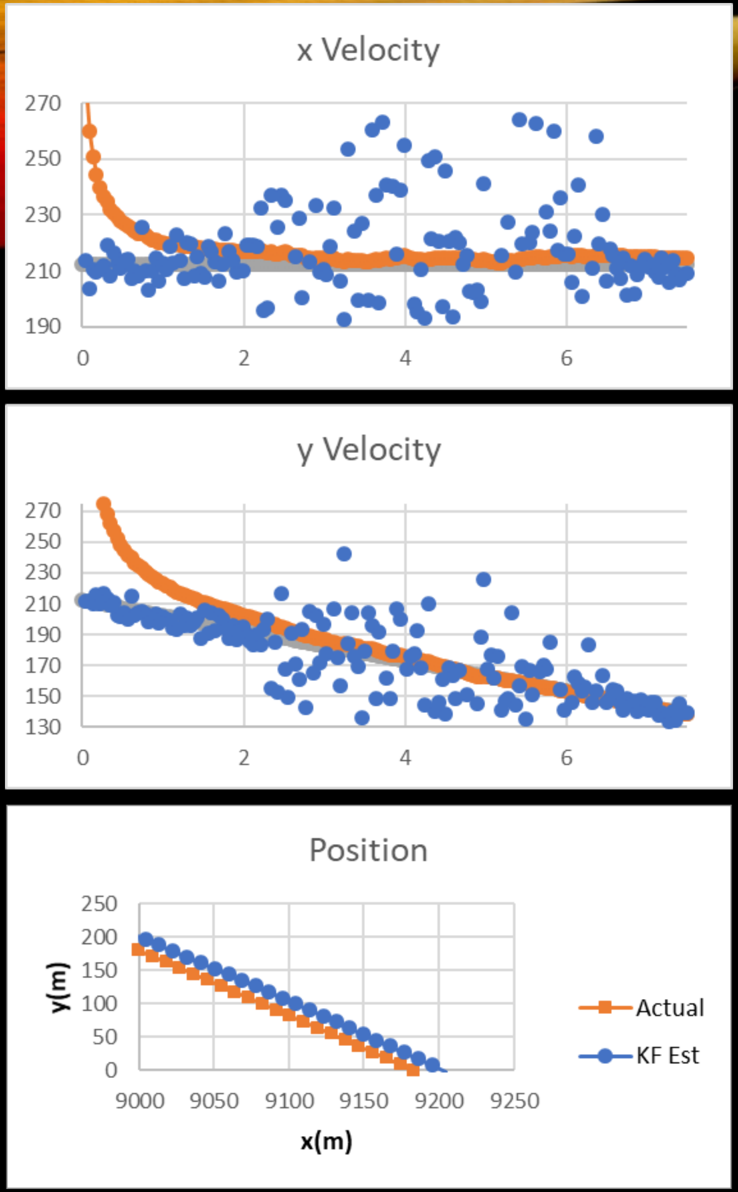
KALMAN FILTER EXAMPLE
VELOCITY ESTIMATES



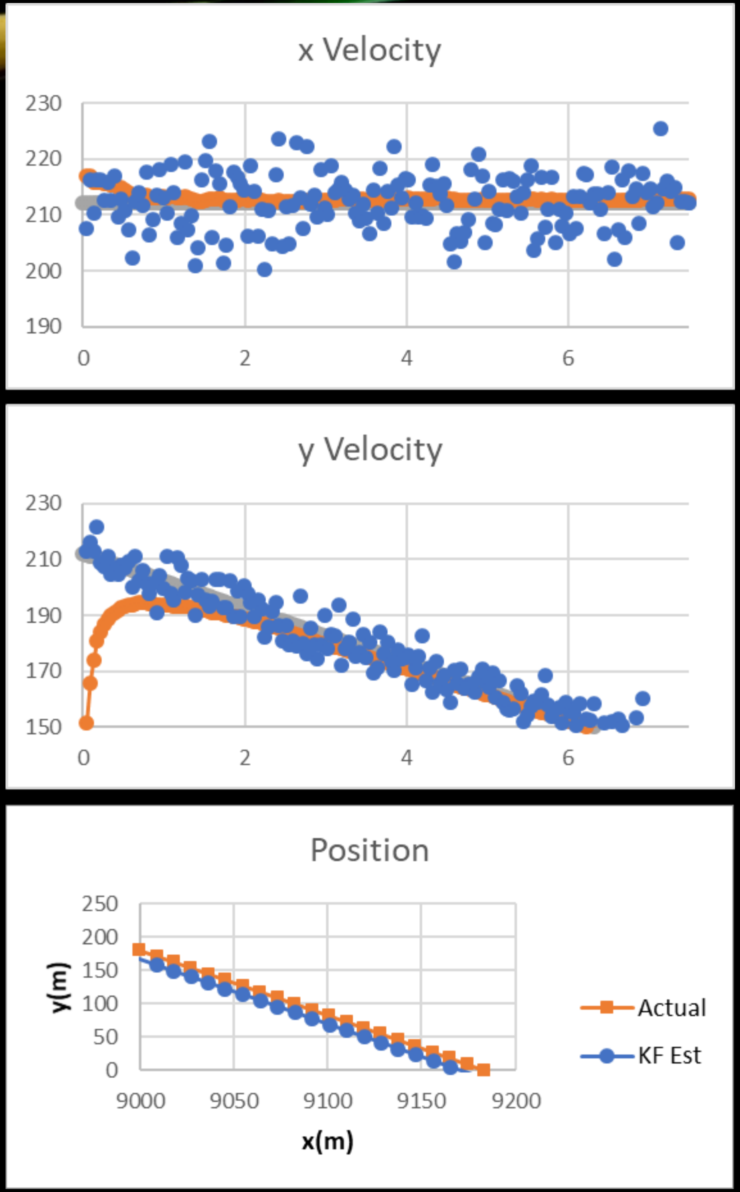


KALMAN FILTER EXAMPLE
KALMAN GAIN

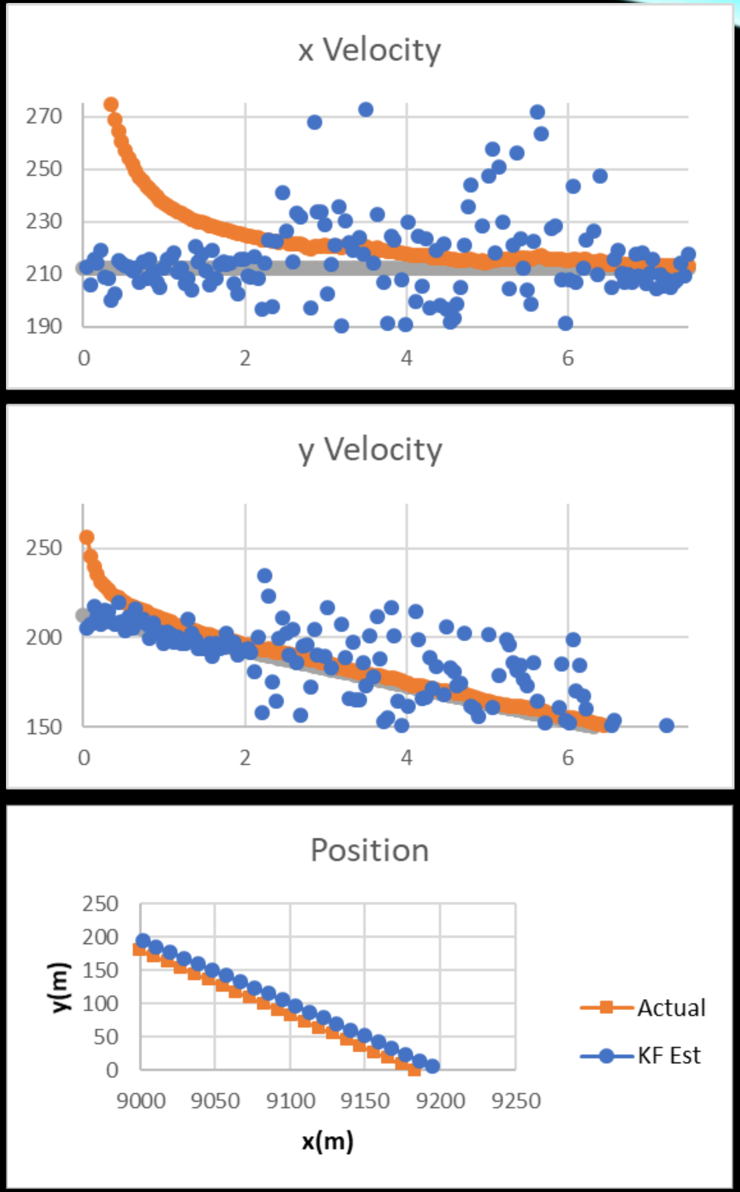




Ex2: Noise burst of 5X noise



Ex3: Slower, Flatter estimate
(30 degrees and 255m/s)



Ex4: Faster, flatter estimate
(30 degrees and 555m/s)
combined with a 5X noise burst

EXTENSIONS

- Extended Kalman Filter (EKF)
  - Standard KF applied to first order Taylor’s approximation on non-linear state-space model
- Unscented Kalman Filter (UKF)
  - Uses **Sigma Points** to take additional weighted points on source Gaussian and map them to target Gaussian through a non-linear function
- Central Kalman Filter (CKF)
  - Very computational expensive
- Distributed Kalman Filter (DKF)
  - Uses distributed microfilters and a consensus filter
- Ensemble Kalman Filter (EnKF)

SUMMARY

- Given a linear dynamical system, a Kalman Filter finds an optimal estimate of the state vector
  - Optimal in a least squares sense
- Memory efficient
- Extremely Fast
- In addition to state, provides estimation quality information
- Robust