Indian Institute of Technology, Roorkee

Term paper on

Application of Kalman Filter to Smoothen GPS Navigational Data

By

Mohit Bakshi

10114028

B.Tech IV year

Computer Science and Engineering

IIT-Roorkee

Under the guidance of

Dr. Jayanta Kumar Ghosh

Civil Engineering Dept.

IIT-Roorkee

1 September, 2013

Abstract

Kalman Filter, developed by R.E. Kalman in 1960, has been a prime area of research since its inception. Providing a simple recursive solution to the problem of discrete data linear filtering, it has been of immense use in several areas, most notably, autonomous and assisted navigation.

The power of Kalman filter lies in several aspects: estimations of past, present, and even future states, even when the precise nature of the modeled system is unknown. As an optimal estimator for one-dimensional linear systems with Gaussian error statistics, it is also used for smoothing noisy data.

The purpose of this paper is to study in detail, the mathematics and intuition behind workings of Kalman Filter, state of the art technology employing such filters and to explore how it can be used to smoothen(reduce errors in) navigational data obtained from Global Positioning System(GPS).

Introduction

Overview

Also known as linear quadratic estimation (LQE), a Kalman Filter employs an algorithm that uses a series of measurements observed over time, containing noise, and produces estimates of unknown variables that tend to be more precise than those based on a single measurement alone. More formally, the Kalman filter operates recursively on streams of noisy input data to produce a statistically optimal estimate of the underlying system state

Working of Kalman Filter

The algorithm works in a two-step process

1. In the prediction step, the Kalman filter produces estimates of the current state variables, along with their uncertainties.
2. Once the outcome of the next measurement (necessarily corrupted with some amount of error, including random noise) is observed, these estimates are updated using a weighted average, with more weight being given to estimates with higher certainty.

The weights are calculated from the covariance, a measure of the estimated uncertainty of the prediction of the system's state. The result of the weighted average is a new state estimate that lies between the predicted and measured state, and has a better estimated uncertainty than either alone. This process is repeated every time step, with the new estimate and its covariance informing the prediction used in the following iteration.

Because of the algorithm's recursive nature, it can run in real time using only the present input measurements and the previously calculated state; no additional past information is required.

Assumptions

From a theoretical standpoint, the main assumption of the Kalman filter is that the underlying system is a linear dynamical system and that all error terms and measurements have a Gaussian distribution (often a multivariate Gaussian distribution).

Applications of Kalman Filters

The Kalman filter has numerous applications in technology. A common application is for guidance, navigation and control of vehicles, particularly aircraft and spacecraft. Furthermore, the Kalman filter is a widely applied concept in time series analysis used in fields such as signal processing and econometrics.

Kalman filters have been vital in the implementation of the navigation systems of U.S. Navy nuclear ballistic missile submarines, and in the guidance and navigation systems of cruise missiles such as the U.S. Navy's Tomahawk missile and the U.S. Air Force's Air Launched Cruise Missile. It is also used in the guidance and navigation systems of the NASA Space Shuttle and the attitude control and navigation systems of the International Space Station.

Details

The system on which we apply Kalman Filter evolves according to a process equation as follows:

Process Equation

$$x_t = A_t \times x_{t-1} + B_t \times u_t + w_t \quad (1)$$

- x_t is the state vector containing the terms of interest for the system(e.g., position, velocity, heading) at time t.
- u_t is the vector containing any control inputs (steering angle, throttle setting, braking force)
- F_t is the state transition matrix which applies the effect of each system state parameter at time t-1 on the system state at time t (e.g., the position and velocity at time t-1 both affect the position at time t)
- B_t is the control input matrix which applies the effect of each control input parameter in the vector u_t on the state vector (e.g., applies the effect of the throttle setting on the system velocity and position)
- w_t is the vector containing the process noise terms for each parameter in the state vector. The process noise is assumed to be drawn from a zero mean multivariate normal distribution with covariance given by the covariance matrix Q_t.

Measurement of the process parameters is modeled according to the equation :

Measurement Equation

$$z_t = H_t \times x_t + v_t \quad (2)$$

- z_t is the vector of measurements
- H_t is the transformation matrix that maps the state vector parameters into the measurement domain.
- v_t is the vector containing the measurement noise terms for each observation in the measurement vector. Like the process noise, the measurement noise is assumed to be zero mean Gaussian white noise with covariance R_t.

The Kalman filter algorithm involves two stages: prediction and measurement update.

The standard Kalman filter equations for the prediction stage are:

$$\widehat{x}_t = A_t \times \widehat{x}_{t-1} + B_t \times u_t \quad (3)$$

$$P_t = A_t \times P_{t-1} \times F_t^T + Q_t \quad (4)$$

- Q_t is the process noise covariance matrix associated with noisy control inputs.
- P_t is the covariance matrix associated with the prediction of state vector parameter.

The measurement update equations are given by:

$$K_t = P_t H_t^T \left(H_t P_t H_t^T + R_t\right)^{-1} \quad (5)$$

$$\widehat{x}_t = \widehat{x}_t + K_t(z_t - H\widehat{x}_t) \quad (5)$$
$$P_t = P_t - K_t H_t P_t \quad (7)$$

- K_t is known as the Kalman Gain. It is a factor that weighs how much of the measurement and the process estimate must be taken into account to predict the optimal estimate.

Algorithm Process Model

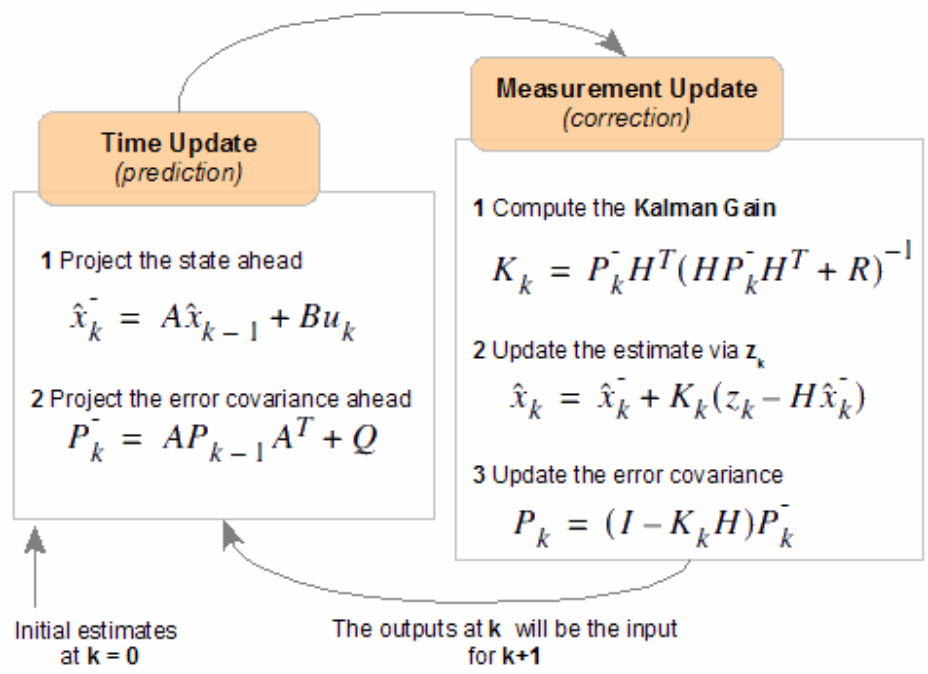


Fig. 1 [http://bilgin.esme.org/portals/0/images/kalman/iteration_steps.gif]

- *Time Update :* Taking the output of the previous state estimates we compute the state value at next time step using process equation. Here $\widehat{x_k^-}$ is the a priori estimate of the state vector. P_k^- is the a priori covariance of the estimate.
- *Measurement Update :* Using the a priori estimates of the state vector, we compute a factor called Kalman Gain. We use Kalman gain and the measurements obtained in this time step to obtain an optimal estimate of the state vector which we call a posteriori estimate.

Intuition [3]

The intuition behind these equations is shown below in figures.

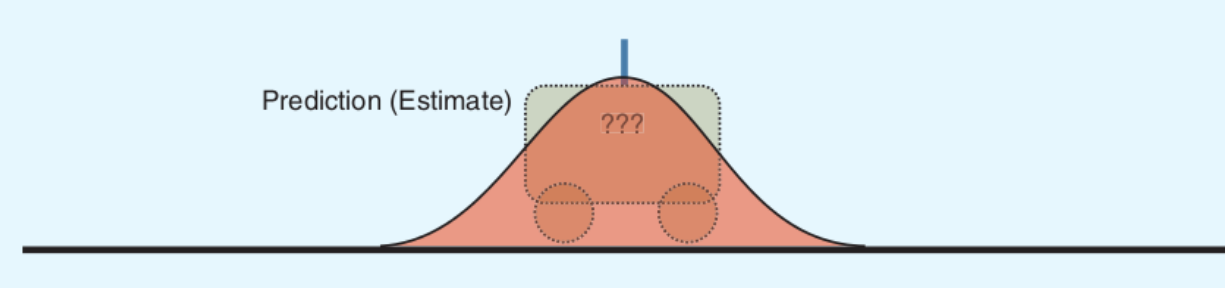


Fig 1. Here the prediction of the location of the object is shown. The curve corresponds to a gaussian distribution of the uncertainty in the position of the object.

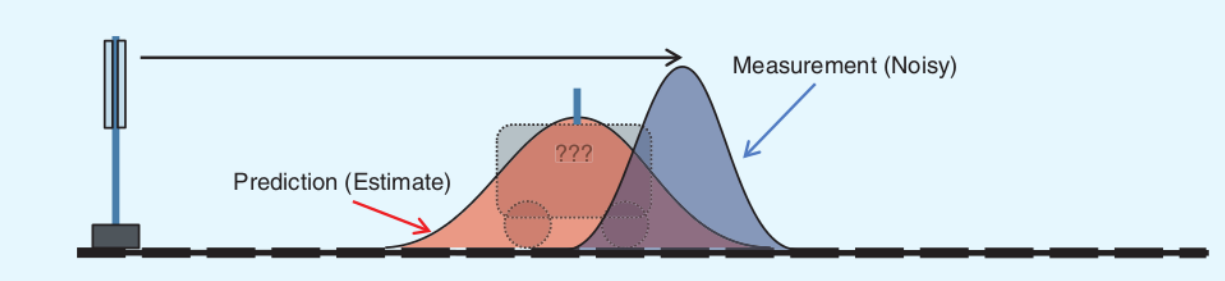


Fig 2. Here the measurement of the location of the object is shown. The gaussian curve in the measurement corresponds to the uncertainty in the measurement process due to noises.

Fig 3. By multiplying the probability distributions of the prediction and measurement processes, we can obtain a new gaussian distribution that corresponds to a better estimate of the location of the object.

An Example

Consider the measurement of resistance of a resistor using ohmmeter. Following conditions are assumed:

- The resistance does not change with time i.e the process of change of resistance has 0 control inputs. Since there are no factors that change the resistance, so there is no noise corresponding to the control inputs.
- The ohmmeter measures the resistance with a certain noise whose covariance remains fixed with time i.e the uncertainty in measurement does not change with time. The noise in measurement is taken from a gaussian normal distribution.

Let the resistance be 10 ohms and the color code on the resistor indicate an uncertainty in the indicated value of resistance with variance of 2 $ohms^2$.
Modeling the process equation :

$x_{k+1} = x_k$ [1]

Let ohmmeter measurement uncertainty be 1 ohm^2 about the actual value of the resistance.
Modeling the measurement equation :

$z_k = x_k + v_k$ [2]

covariance(v_k) = R_k = 1 ohm^2

Applying the Kalman Filter :
Initial estimates

$x_{-1} = 10$

$P_{-1} = 2$ (must be equal to the variance of the resistor value as indicated by color code)

At k=0,

Update

$x_0 = x_{-1} = 10$ ohms

$P_k = AP_{k-1}A^T+Q$

$\Rightarrow P_0 = 1.P_{-1}.1 + 0 = 2$ $ohms^2$

Measure ($z_0 = 10.5$ ohms)

$K_0 = \frac{P_0.H^T}{(HP_0H^T+R)} = \frac{2.1}{(1.2.1 + 1)} = 2/3$

$x_0 = x_0 + K_0(z_0-Hx_0) = 10 + ⅔ (10.5 - 1*10) = 10 ⅓$

$P_0 = (I - K_0H)P_0 = (1 - ⅔ * 1)2 = ⅔$

Note : Here we can see that the variance in the measurement(1 $ohms^2$) is less than that of the previous estimate(2 $ohms^2$). The Kalman Gain which is ⅔ is thus inclined so that measurement

has more weight in estimating the next state value. We can also see that the variance estimate reduces from 2 ohms^2 to ⅔ ohms^2.

At k=1,

Update

$x_1 = x_0 = 10\ ⅓$

$P_1 = P_0 = ⅔$

Measure (z_1 = 10.1 ohms)

$K_1 = (⅔ * 1)/(1*⅔*1 + 1) = ⅖$

$x_1 = x_1 + K_1(z_1 - Hx_1) = 10\ ⅓ + ⅖ * (10.1 - 1*10\ ⅓) = 10.24$

$P_1 = (1- ⅖)⅔ = ⅖$

Note : Here the estimate covariance is further reduced to ⅖, thus indicating that measurements are helping us in obtaining a stable value. The value of resistance too is approaching 10 and we keep continuing the process the value of both x and P will attain stability and will converge.

Applying Kalman Filter to Navigational Data

Here I consider a moving object, say a ballistic missile, whose actual coordinates we need to estimate. The process that governs the motion of the object is known via accelerometers, gyrometers, compass and other navigational devices.

The Problem - Suppose a ballistic missile is equipped with GPS, accelerometers, compass and other navigational devices whose data is transferred to a ground station via radio frequency. The initial velocity and position of missile (where it is launched from) is known with uncertainty. Suppose there is no air friction, so that the true path of the projectile is an undisturbed parabola. The measurements taken by sensors be associated with random errors(white gaussian noise). The GPS coordinates are measured after every interval of 1 second. Now the ground station need to estimate what path the missile took during flight. GPS data is converted into (x,y,z) coordinates before processing it in Kalman Filter.

Example 1

The Model

Process

- State Vector : The state of the missile is known by 6 scalar quantities - (x,y,z) coordinates and (vx,vy,va) velocities in x,y and z directions. Therefore state vector is of the form -

$$[x\ y\ z\ vx\ vy\ vz]^T$$

- Control Vector : The controls of the missile must contain accelerometers and other navigational devices. For the sake of simplicity we consider only acceleration is the

control input. Therefore control vector is of the form

$$[ax\ ay\ az]^T = [\ 0\ 0\ -10]^T$$

therefore the acceleration in x and y direction is 0 while that in z direction is $-10 m/s^2$

- Process Noise : Since we assumed that there is no air friction, therefore the actual acceleration of the body is the one provided by the accelerometers.

$$Q = 0 \text{ (Covariance of noise)}$$

- Matrix A and B : Using equations of motion we can find A and B as follows -

$$A = \begin{vmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{vmatrix}_{[6 * 6]}$$

$$B = \begin{vmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0.5 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix}_{[6 * 3]}$$

Measurement

- Measurement Vector : Z is the converted GPS coordinates in (x,y,z) form

$$z = [mx\ my\ mz]^T$$

- Matrix H : H must project the actual (x,y,z) coordinates to the measurement quantities. Since we are assuming that the measurement scalars are the converted GPS coordinates, therefore we each of the measured position is same as the process position

$$H = \begin{vmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{vmatrix}_{[3 * 6]}$$

- Measurement Noise : We assume that GPS measurements contain only random noise i.e white gaussian noise, because Kalman Filter is capable of removing only gaussian noise. Therefore V_K is the measurement noise vector and each of its scalar is chosen randomly from a gaussian distribution with covariance mentioned below.

$$V = \begin{vmatrix} vx \\ vy \\ vz \end{vmatrix}_{[3 * 1]}$$

where each of vx, vy, vz is chosen randomly from gaussian noise distribution

- Noise Covariance R : We assume that covariance between any two of the x,y, and z coordinates is 0 i.e the measurement noise in x,y and z are independent of each other. Let variance in each coordinate be $10m^2$.

$$R = \begin{bmatrix} 10 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 10 \end{bmatrix}_{[3 * 3]}$$

Now we have defined our complete process and measurement equations.
We have obtained the constants - A, B, H, Q and R. Now we need to define the initial state estimates.
Let initial state estimate be

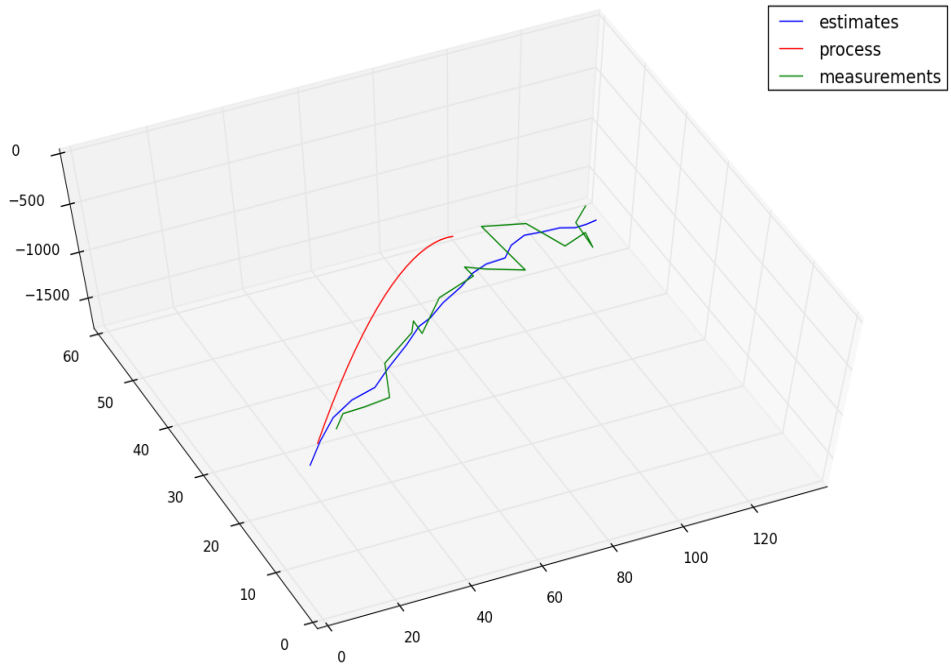
$$X_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 5 \\ 3 \\ 10 \end{bmatrix}$$

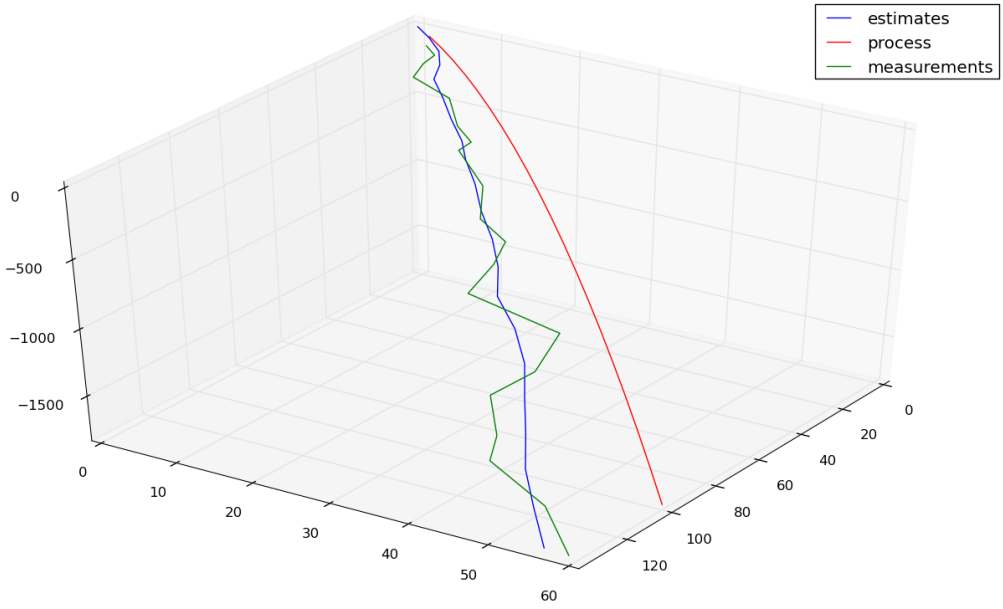
Therefore the initial speed in x direction is 50m/s, in y direction is 30m/s and in z direction is 100m/s.
Let the initial variance in estimate be

$$P_0 = \begin{bmatrix} 0.5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Therefore the uncertainty in any variable is independent from that in the other. The variance estimate in Position scalars (x,y,z) is 0.5 $meters^2$ and variance in speed scalars (vx, vy, vz) is 1 m/s^2.

<u>Running the Kalman Filter</u>
Using the above data, kalman filter was run for 20 observations.
I obtained graphs for estimates(the filtered output), process(the unfiltered output), and GPS measurements.


estimates
process
measurements
0
−500
−1000
−1500
60
50
40
30
20
10
0
0
20
40
60
80
100
120


estimates
process
measurements
0
−500
−1000
−1500
0
10
20
30
40
50
60
0
20
40
60
80
100
120

Observations

- We can observe that despite the erratic measurements(green), kalman filter creates a very good estimate of the projectile path.
- The process(red) plot deviates from the filtered estimate because the initial variance in the position and velocity is comparable to the actual position and velocity. Thus the process varies a lot.
- The plot of measurements is erratic because the variance in the noise of measurements if high - 10 m^2. Because of the high difference of variance in process and measurements we can see that Kalman Filter gives more weight to the process and less weight to the measurements, thus the path is approximately parabolic, instead of being erratic like measurements.

Example 2

Let the process model be same as above.

This time let us make the measurements more accurate(using a higher precision GPS instrument). So reducing the variance in the measurements,

R = | 2 0 0 |
| 0 4 0 |
| 0 0 3 |

This time the variance is dependent on the direction(usually this data will be obtained from GPS data).

Also, let us increase the variance in the process i.e. now due to air-friction and atmospheric disturbances, the control inputs cannot be accurately measured. So,

Q = | 0 0 0 0 0 0 |
| 0 0 0 0 0 0 |
| 0 0 0 0 0 0 |
| 0 0 0 2 0 0 |
| 0 0 0 0 0.5 0 |
| 0 0 0 0 0 3 |

Thus air-friction brings about uncertainty in speed scalars of the process state.

Initial Estimates

Position,

X_0 = | 0 |
| 0 |
| 0 |
| 3 |
| 1 |

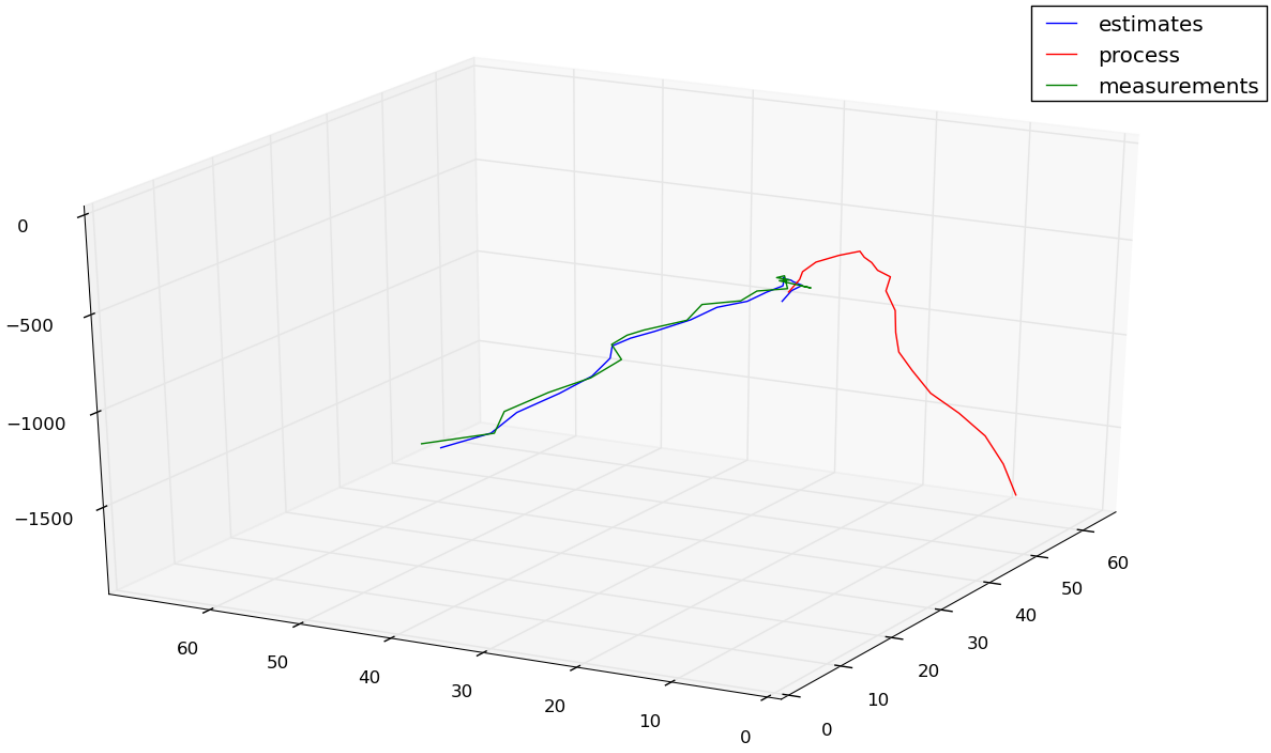
| 6 |

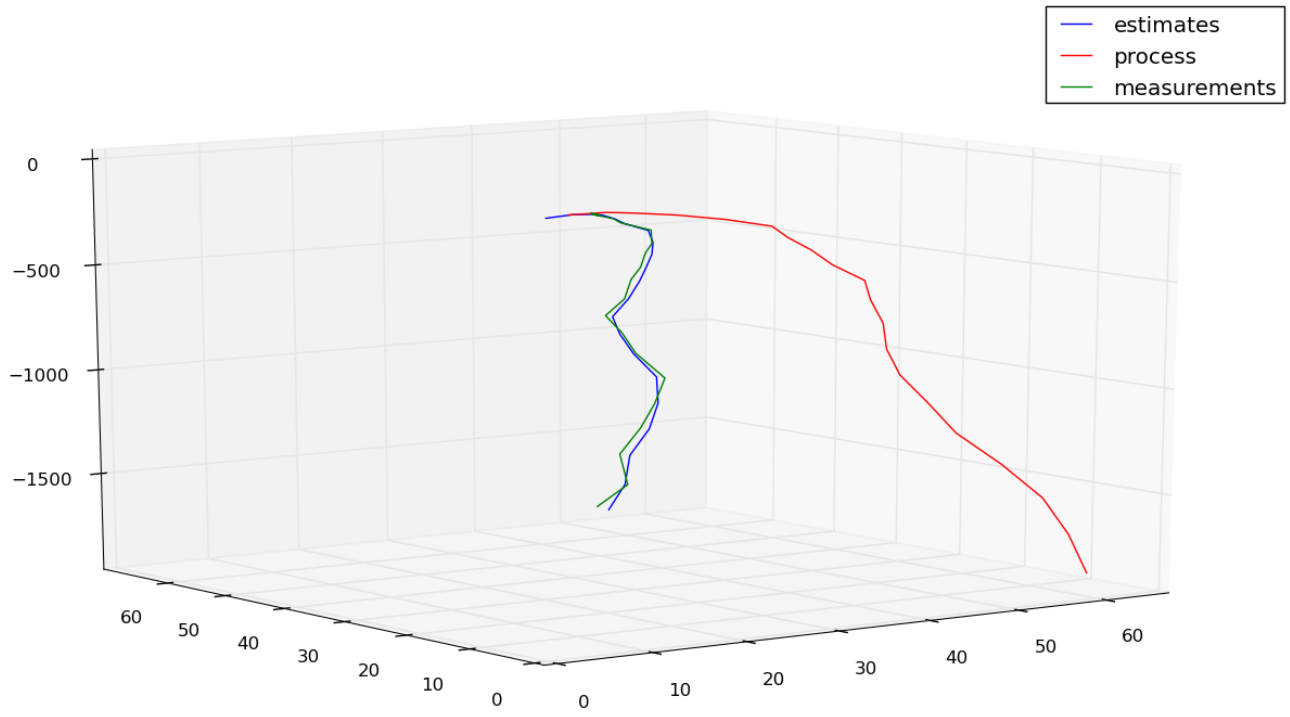
Covariance,

P_0 = | 0.1 0 0 0 0 0 |
| 0 0.1 0 0 0 0 |
| 0 0 0.1 0 0 0 |
| 0 0 0 0.5 0 0 |
| 0 0 0 0 0.5 0 |
| 0 0 0 0 0 0.5 |

Running the Kalman Filter

Outputs corresponding to 20 time steps are as follows -





Observations

- Here we observe that the measurements approximate the filtered output very well. This is because the uncertainty(variance) in measurements have decreased due to better GPS instruments.
- The process on the other hand is very different from the actual filtered output. This is because the initial uncertainty(variance) in the process was very high as compared to its estimate. Also addition of air-friction and other disturbances has changed the shape of the process output, so that it is different from parabola.
- Thus we can see that Kalman Filter automatically weighs in the measurements more than the process knowing that the uncertainty in process is high as compared to measurements.

Applying Kalman Filter to Static GPS Data

Now let us suppose that we are collecting GPS data of a point. Since we are static conditions now, the control inputs become 0.

$\Rightarrow u_k = 0$

Process

$$A = \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

$$B = \begin{vmatrix} 0 \end{vmatrix}_{[3 * 3]}$$

$\Rightarrow X_k = X_{k-1}$

Thus the position according to the process does not change with time.

Noise in the process - since we are in static conditions, therefore the uncertainty with speed scalars must be 0 while that with position must also be 0.

$Q = 0_{[3 * 3]}$ - 3 * 3 zero matrix

Measurements

The measurements associated with GPS have uncertainty(random noise in measurements - with gaussian characteristics). Let covariance matrix of noise be

$$R = \begin{vmatrix} 10 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 10 \end{vmatrix}$$

Thus the variance in each direction is $10m^2$. (which is generally the case with medium precision GPS instruments).

Also,

$$H = \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix}_{[3 * 3]}$$

Initial Estimates

$$X_0 = \begin{vmatrix} 0 \\ 0 \\ 0 \end{vmatrix}$$

$$P_0 = \begin{vmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{vmatrix}$$

So, initially we have an uncertainty of $2m^2$ in the position of point in each direction.

Running the Kalman Filter

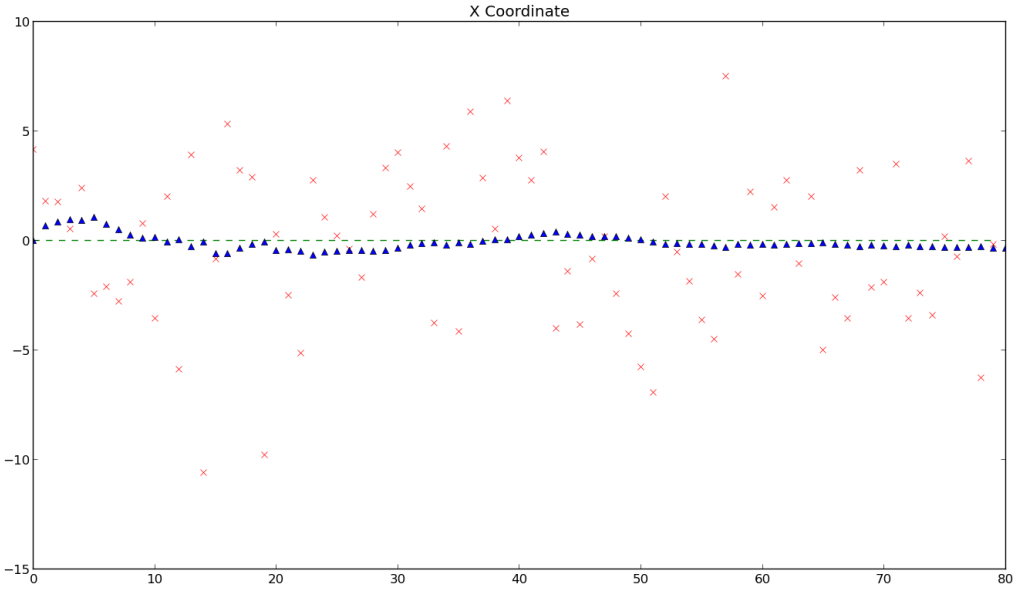
Kalman Filter was run for 80 time steps i.e 80 observations of the point were taken.

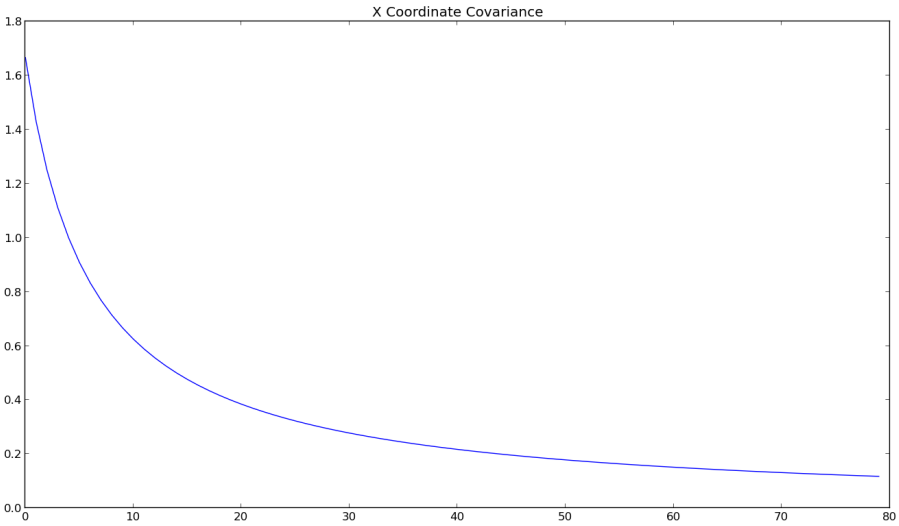
Figures -

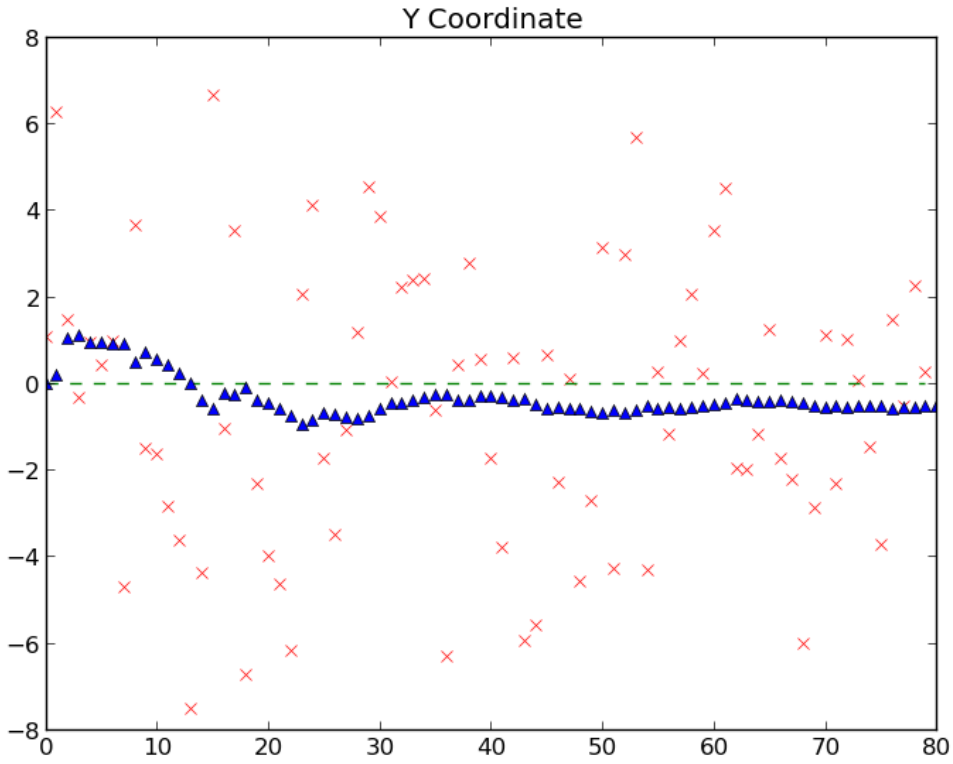
'x' represent measurement

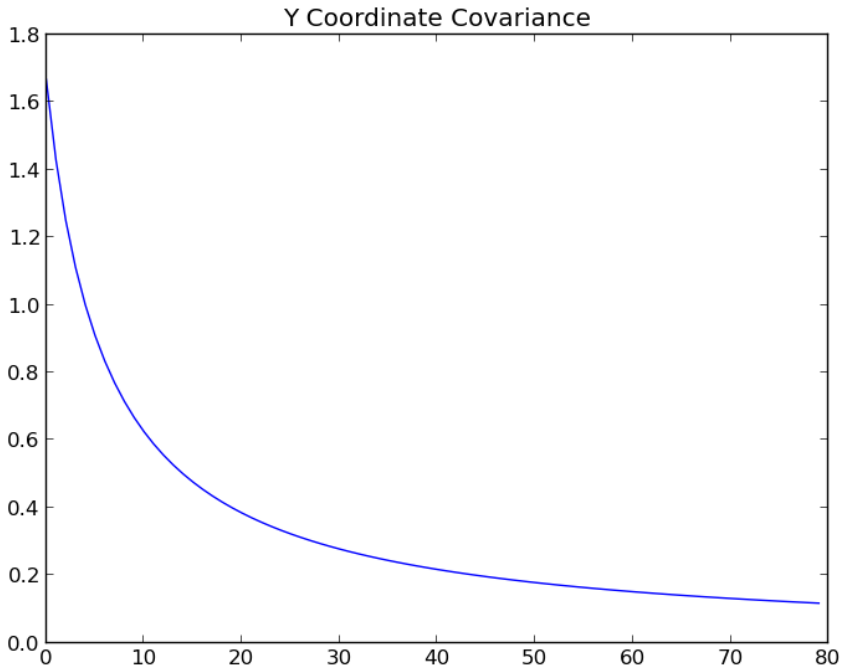
'triangles(^)' represent filter estimate

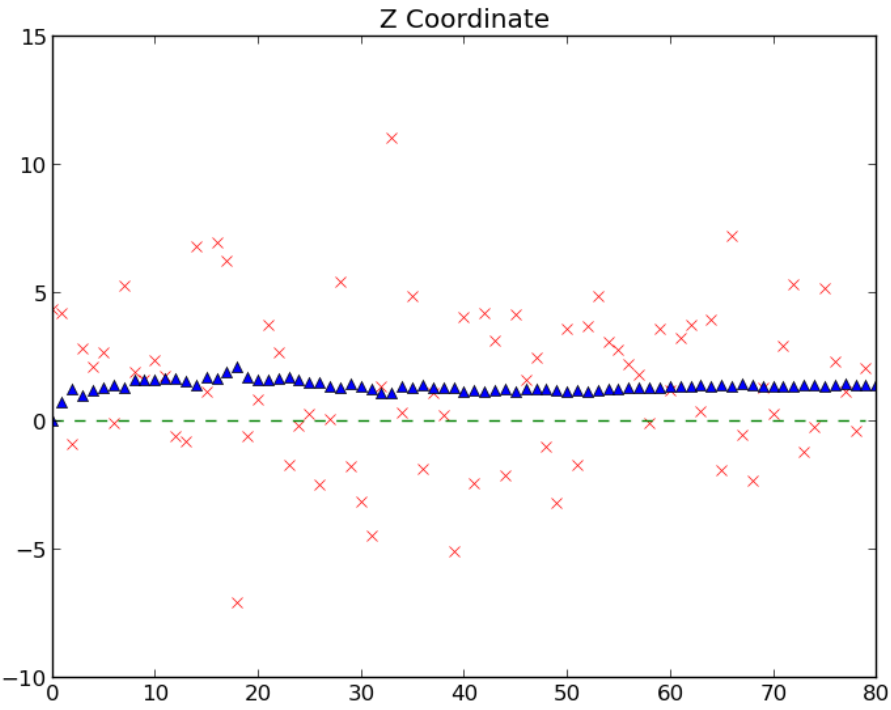
'----' represent the actual position

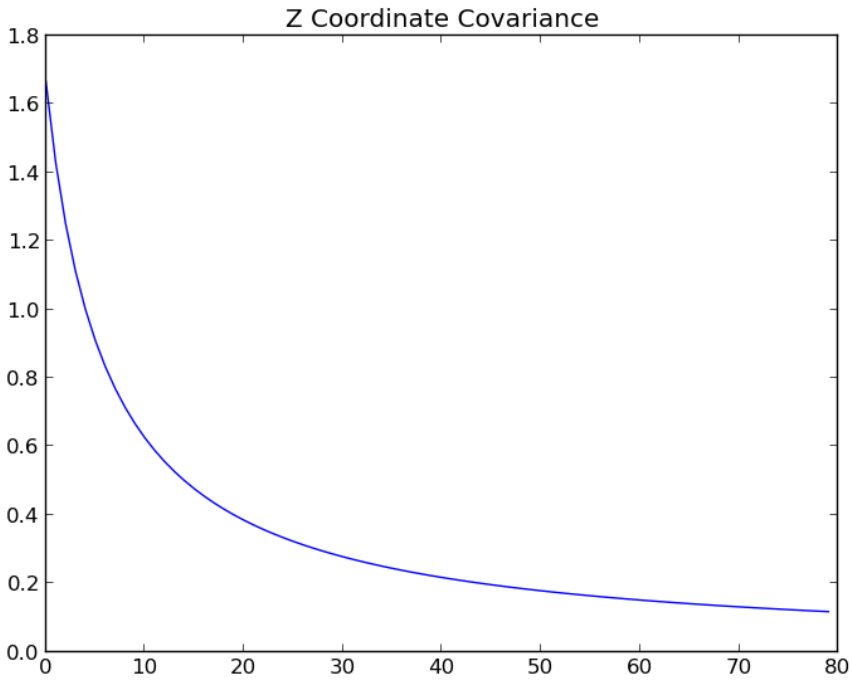



X Coordinate Covariance
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
60
70
80


Y Coordinate
8
6
4
2
0
−2
−4
−6
−8
0
10
20
30
40
50
60
70
80


Y Coordinate Covariance
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
60
70
80


Z Coordinate
15
10
5
0
−5
−10
0
10
20
30
40
50
60
70
80



Observations

- We can see that the measurements represented by 'x' are randomly distributed about the mean position 0, chosen from gaussian distribution with variance $10m^2$.
- For each of the x,y,z coordinates we can see that the covariance is decreasing with each observations, hence improving the certainty with which we can locate the point.
- The estimates of x,y and z are also settling down to their correct values.

Conclusion

We can see that applying Kalman Filter to a process that can be defined by a linear dynamic system can improve the process estimates significantly. The Kalman Filter was applied on simulated navigational GPS data and the behavior of Kalman Filter was found to be correct and optimal. Similarly it was found that applying Kalman Filter to static GPS data also improves the estimates as uncertainty decreases with each observation.

Future Work

- Due to lack of time and resources, I could not apply the filter on real GPS data. This can be easily done as the code I have created is generic and can be easily used in future.
- Though it is easy to obtain GPS data of a body in motion, getting process data such as acceleration and direction is difficult task as it requires sophisticated sensors. The availability of such sensors in the future can help in creating a real navigational GPS data situation instead of a simulated one.
- The GPS data is generally available in RINEX format. The software must be extended to process RINEX files instead of text files to make it more user friendly.
- Converting WGS-84 and UTM coordinates to standard coordinates used in navigational techniques and devices is another interesting challenge for future work.

References

1. Wikipedia - Kalman Filter
2. Welch, Greg, and Gary Bishop. "An introduction to the Kalman filter." (1995).
3. Faragher, Ramsey. "Understanding the Basis of the Kalman Filter Via a Simple and Intuitive Derivation [Lecture Notes]." *Signal Processing Magazine, IEEE* 29.5 (2012): 128-132.
4. Julier, Simon J., and Jeffrey K. Uhlmann. "New extension of the Kalman filter to nonlinear systems." *AeroSense 97*. International Society for Optics and Photonics, 1997.
5. Evensen, Geir. "The ensemble Kalman filter: Theoretical formulation and practical implementation." *Ocean dynamics* 53.4 (2003): 343-367.
6. BROWN, Stanley A. "Introduction to random signals & applied kalman filtering with matlab exercises & solutions 3e sol." (1997).

Software Report

Software Capabilities

- The Kalman Filter code runs the kalman filter on a process whose parameters are to be provided.
- Once given data of *constants, measurements(optional) and control input*, the code automatically finds all time step estimates of the process, kalman filter estimates - state vectors and covariance matrices.
- The software provides the capability to run a simulation by independently generating random measurements whose characteristics are provided in the *constants* file. If the process measurements are available, then simulation mode can be switched off.
- The software also provides capability to generate graphs. Both 3-d(for trajectories) and 1-d(for positions/state vector variables) plots are supported.
- In 3-d mode, both scatter and line plots can be generated.

How to use the code

- The code is written in python. Therefore basic understanding of the language is required.
- numpy library has been used extensively. Understanding of the library can help in better understanding and usage of the code.
- *kf.py*
    - This file contains the code for kalman filter
    - Run it using
        - >>> *python kf.py* on linux terminal
        - Use Python IDE on windows system
    - Inputs required
        - *constants*
            - Create a file named *constants* and store it in the same directory as kf.py.
            - There are 7 constants required to run a kalman filter
                - A
                - B
                - H
                - Q
                - R
                - X_0
                - P_0
            - Store each of these constant matrices/vectors in numpy matrix form, separated by newlines.
        - *measurements (this file is optional - not required in simulation)*
            - This file stores the process measurements.

            - Measurements in successive time steps are to be stored on newline.
            - The measurement vector Z must be stored in numpy matrix form.
        - *controls*
            - This file stores the control inputs given at each time step.
            - Each time step control input is separated by a newline.
            - The control vector must be stored in numpy matrix form
    - Outputs created
        - *estimates*
            - The kalman filter outputs. Stored in numpy matrix form
        - *process*
            - The process that is extrapolated from initial estimate and process equation. State vector and Covariance matrix stored in numpy form
        - *gen_measurements (created only in simulation mode)*
            - contains the randomly generated measurements for the process.
            - Stored in numpy matrix form
        - *plot_file_covar , plot_file_estimates , plot_file_meas , plot_file_proc*
            - These files store data for generating graphs
    - Simulation vs Experiment
        - Upon running the file, user is asked if measurements are to be generated.
        - For simulation type - y
        - For experiment(measurements stored in *measurements* file) type - n
- *plot3d.py*
    - The code uses *matplotplot* library
    - This file takes the *plot_file_xxx* files generated above and plots a 3d graph for the first 3 variables in the state vector.
    - Thus it is advised that the first 3 variable be position scalars in case graphs are to be generated.
    - Upon running the code, user is asked to choose between 2 options - scatter/line
        - Choose scatter to plot positions.
        - Choose line to plot trajectories.
- *plotstatic.py*
    - The code uses matplotlib
    - This file takes the *plot_file_xxx* files generated above and plots a 2d graphs for the first 3 variables in the state vector and their corresponding variances.
    - Thus it is advised that the first 3 variables in the state vector be position scalars.
    - First 3 graphs correspond to the variables, next 3 graphs correspond to their covariances.

Understanding the code

- Kalman Filter class is created. It stores constants - A,B,H,Q,R,X_0,P_0 and some time varying variables such as the current_state estimate and current_process_estimate
- getConstants function retrieves constants from *constants file* and creates numpy matrices
- Kalman filter object *kf* is initialized with these constants.
- *run* function in kf class runs the code
- It retrieves measurements file and control files.
- It iterates over time steps by reading a line from control file. Therefore the kalman filter is run as long as the control inputs is being retrieved from the file.
- It calls *process* function to evaluate the process state vector as it should have been if kalman filter was not run.
- Then the *step* function is called. This function steps over one time step and provides with the new current_state_estimate and new current_covariace_estimate.
- *output* function is called and provided with the results of *step* function to write down the outputs into output_files. It also creates data for plotting.

---------- Code ----------

---- file kf.py ---------- containes kalman filter code

```
import numpy
import math
import ast

f = open('estimates','w')                  ### Kalman Filter estimates go into this file
proc_file = open('process','w')            ### Process estimates go into this file
gen_meas = open('gen_measurements','w')
plt = open('plot_file_estimates','w')      ### Plotting data for kalman filter estimates
plt_meas = open('plot_file_meas','w')      ### Plotting data for measurements
plt_proc = open('plot_file_proc','w')      ### Plotting data for process
plt_cov = open('plot_file_covar','w')      ### Plotting data for covariance in variables

class KalmanFilter():
    def __init__(self, _A, _B, _H, _Q, _R, _X0, _P0):
        ### Constants
        self.A = _A
        self.B = _B
        self.H = _H
        self.Q = _Q
        self.R = _R
        self.n = _A.shape[0]      ### size of state vector
        self.m = _H.shape[0]      ### size of measurement vector
        ### Time Step Variables
        self.curr_state = _X0   # Current Kalman Filter estimate
        self.curr_proc_state = _X0 # Current Process estimate
        self.curr_variance = _P0   # Current Covariance estimate
```

```
        self.i = 0                  # time step
        self.K = 0                  # Current Kalman Gain

    def step(self, controls, measures):    # finds values for next time step
        global b,gen_meas
        self.i = self.i + 1
        ### Time update
        pred_X = (self.A * self.curr_state) + (self.B * controls)
        pred_P = ((self.A * self.curr_variance) * numpy.transpose(self.A)) + self.Q
        if b == 'y':
            mean = numpy.zeros(self.m)
            v = numpy.random.multivariate_normal(mean,self.R)
            v = numpy.transpose(numpy.matrix(v))
            measures = (self.H * pred_X) + v
            gen_meas.write(str(measures))
            gen_meas.write("\n\n")
            x = float(measures[0][0])
            y = float(measures[1][0])
            z = float(measures[2][0])
            plt_meas.write(str(x)+" "+str(y)+" "+str(z))
            plt_meas.write("\n")
        ### Measurement Update
        Ki = (pred_P * numpy.transpose(self.H)) * numpy.linalg.inv(self.H * pred_P *
numpy.transpose(self.H) + self.R) ## Kalman Gain
        self.K = Ki
        self.curr_state = pred_X + Ki * (measures - (self.H * pred_X))
        I = numpy.identity(self.n)
        self.curr_variance = (I - Ki * self.H) * pred_P
        return (self.i, self.curr_state, self.curr_variance)

    def process(self, controls):
        proc_X = (self.A * self.curr_proc_state) + (self.B * controls)
        mean = numpy.zeros((self.n))
        w = numpy.random.multivariate_normal(mean,self.Q)
        w = numpy.transpose(numpy.matrix(w))
        proc_X = proc_X + w
        self.curr_proc_state = proc_X
        x = float(proc_X[0][0])
        y = float(proc_X[1][0])
        z = float(proc_X[2][0])
        plt_proc.write(str(x)+" "+str(y)+" "+str(z))
        plt_proc.write("\n")

    def output(self,i,X,P):
        global f
        global proc_file
        f.write(str(i))
        f.write("\n")
        f.write(str(X))
        f.write("\n")
```

```
    f.write(str(P))
    f.write("\n\n")
    proc_file.write(str(self.i)+"\n")
    proc_file.write(str(self.curr_proc_state))
    proc_file.write("\n\n")
    ### PLOTTING X,Y,Z
    x = float(X[0][0])
    y = float(X[1][0])
    z = float(X[2][0])
    plt.write(str(x)+" "+str(y)+" "+str(z))
    plt.write("\n")
    ### PLotting covariances in each direction
    # X
    p = numpy.array(P[0][0])
    print p
    x_cov = float(p[0][0])
    # Y
    p = numpy.array(P[1][0])
    print p
    y_cov = float(p[0][1])
    # Z
    p = numpy.array(P[2][0])
    print p
    z_cov = float(p[0][2])
    print x_cov, y_cov, z_cov
    plt_cov.write(str(x_cov)+" "+str(y_cov)+" "+str(z_cov))
    plt_cov.write("\n")


def run(self):                          ## run Kalman Filter
    # write initial estimates into the files
    x = float(self.curr_state[0][0])
    y = float(self.curr_state[1][0])
    z = float(self.curr_state[2][0])
    plt.write(str(x)+" "+str(y)+" "+str(z))
    plt.write("\n")

    global b
    i = 0
    const = open('constants','r')
    if b == 'n':
        meas = open('measurements','r')
    control = open('controls','r')
    l = control.readline()
    while l:
        ### get control inputs
        cont = numpy.matrix(ast.literal_eval(l))
        ### get measurements input
        if b == 'n':
            l = meas.readline()
```

```
                mes = numpy.matrix(ast.literal_eval(l))
            elif b == 'y':                              # if measurements are to be
generated randomly
                mes = numpy.zeros(shape=(self.m,1))
                mes = numpy.matrix(mes)
            self.process(cont)
            (j,X,P) = self.step(cont,mes)
            self.output(j,X,P)
            l = control.readline()

def getConstants():
    const_file = open('constants','r')
    l = const_file.readline()
    _A = numpy.matrix(ast.literal_eval(l))
    l = const_file.readline()
    _B = numpy.matrix(ast.literal_eval(l))
    l = const_file.readline()
    _H = numpy.matrix(ast.literal_eval(l))
    l = const_file.readline()
    _Q = numpy.matrix(ast.literal_eval(l))
    l = const_file.readline()
    _R = numpy.matrix(ast.literal_eval(l))
    l = const_file.readline()
    _X = numpy.matrix(ast.literal_eval(l))
    l = const_file.readline()
    _P = numpy.matrix(ast.literal_eval(l))
    return (_A,_B,_H,_Q,_R,_X,_P)

print "Do you wish to generate random measurements(y/n): "
b = raw_input()  ### b = y if measurements are to be generated randomly
(A,B,H,Q,R,X,P) = getConstants()
proc_file.write(str(0)+"\n")
proc_file.write(str(X))
proc_file.write("\n\n")
kf = KalmanFilter(A,B,H,Q,R,X,P)
kf.run()
```

---------- *kf.py ends here* ----------

----- file *plot3d.py* ------ *plots 3d graphs*

```
import matplotlib as mpl
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import matplotlib.pyplot as pl

def plotxyz():
    global b
    g = open('plot_file_estimates','r')
    l = g.readline()
    x = []
```

```
y = []
z = []
while l:
    lst = l.split(' ')
    x.append(lst[0])
    y.append(lst[1])
    z.append(lst[2])
    l = g.readline()
x = map(float,x)
y = map(float,y)
z = map(float,z)
fig = pl.figure()
ax = fig.gca(projection='3d')
if b == "scatter":
    ax.scatter(x, y, z, label='estimates',color='blue')
elif b == "line":
    ax.plot(x, y, z, label='estimates',color='blue')
ax.legend()

g= open('plot_file_proc','r')
l = g.readline()
x = []
y = []
z = []
while l:
    lst = l.split(' ')
    x.append(lst[0])
    y.append(lst[1])
    z.append(lst[2])
    l = g.readline()
x = map(float,x)
y = map(float,y)
z = map(float,z)
ax = fig.gca(projection='3d')
if b == "scatter":
    ax.scatter(x, y, z, label='process',color='green')
elif b == "line":
    ax.plot(x, y, z, label='process',color='green')
ax.legend()
g= open('plot_file_meas','r')
l = g.readline()
x = []
y = []
z = []
while l:
    lst = l.split(' ')
    x.append(lst[0])
    y.append(lst[1])
    z.append(lst[2])
    l = g.readline()
```

```
    x = map(float,x)
    y = map(float,y)
    z = map(float,z)
    ax = fig.gca(projection='3d')
    if b == "scatter":
        ax.scatter(x, y, z, label='measurements',color='red')
    elif b == "line":
        ax.plot(x, y, z, label='measurements',color='red')
    ax.legend()
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('z')
    pl.show()

print "Do you want a line plot(trajectories) or a scatter plot(positions) [Enter
scatter/line]?"
b = raw_input()
plotxyz()
```

---------- *plot3d.py ends here* ----------

-------- *file plotstatic.py* ---------- *creates 2d graphs for static processes*

```
import matplotlib as mpl
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import matplotlib.pyplot as pl

def plotxyz():
    g = open('plot_file_estimates','r')
    l = g.readline()
    x_est = []
    y_est = []
    z_est = []
    while l:
        lst = l.split(' ')
        x_est.append(lst[0])
        y_est.append(lst[1])
        z_est.append(lst[2])
        l = g.readline()
    x_est = map(float,x_est)
    y_est = map(float,y_est)
    z_est = map(float,z_est)
    g= open('plot_file_proc','r')
    l = g.readline()
    x_proc = []
    y_proc = []
    z_proc = []
    while l:
        lst = l.split(' ')
        x_proc.append(lst[0])
```

```
    y_proc.append(lst[1])
    z_proc.append(lst[2])
    l = g.readline()
x_proc = map(float,x_proc)
y_proc = map(float,y_proc)
z_proc = map(float,z_proc)
g= open('plot_file_meas','r')
l = g.readline()
x_mes = []
y_mes = []
z_mes = []
while l:
    lst = l.split(' ')
    x_mes.append(lst[0])
    y_mes.append(lst[1])
    z_mes.append(lst[2])
    l = g.readline()
x_mes = map(float,x_mes)
y_mes = map(float,y_mes)
z_mes = map(float,z_mes)
pl.title('X Coordinate')
pl.plot(x_mes,'rx')
pl.plot(x_proc,'g--')
pl.plot(x_est,'b^')
pl.show()
pl.title('Y Coordinate')
pl.plot(y_mes,'rx')
pl.plot(y_proc,'g--')
pl.plot(y_est,'b^')
pl.show()
pl.title('Z Coordinate')
pl.plot(z_mes,'rx')
pl.plot(z_proc,'g--')
pl.plot(z_est,'b^')
pl.show()
g = open('plot_file_covar','r')
l = g.readline()
x_cov = []
y_cov = []
z_cov = []
while l:
    lst = l.split(' ')
    x_cov.append(lst[0])
    y_cov.append(lst[1])
    z_cov.append(lst[2])
    l = g.readline()
x_cov = map(float,x_cov)
y_cov = map(float,y_cov)
z_cov = map(float,z_cov)
pl.title('X Coordinate Covariance')
```

```
    pl.plot(x_cov)
    pl.show()
    pl.title('Y Coordinate Covariance')
    pl.plot(y_cov)
    pl.show()
    pl.title('Z Coordinate Covariance')
    pl.plot(z_cov)
    pl.show()


plotxyz()
```

---------- *file plotstatic.py ends here* ----------