Aerial Robotics Kharagpur Task Round Documentation Task 4 : TRACKING MULTIPLE OBJECTS

Nikhil Giri

***Abstract*— This Documentation gives a summary of how I approached task 4 given to me during ARK Software Selection Round**

From the time when I was first given these tasks to the time i am writing this documentation I have learned and explored a lot of things. I learnt a lots of algorithms and also tried implementing them. These algorithms are essential for drones to judge and correct their path on basis of algorithms they encounter.

During this task i had to study a lots of statistic and maths. I applied the required algorithm in various way. Here i have discussed the best one as of yet. Using Kalman Filter we estimate the path of the drone via its co-ordinates and then find location of different sensor depending on their measurement.

I. INTRODUCTION

In task 4 We are given 4 CSV files containing positional data at some time step from 6 different ground sensor. We have to assume one of them at origin and find position of the other sensors. Also using the measurement we have to find the path taken by the drone using Filters like Kalman Filter.

II. PROBLEM STATEMENT

In the task we assume that a drone is flying in a area with 6 ground stations with one sensor each which reports the location of the drone with respect to their ground station at that instant. The problem here is that we have no idea of the actual position of the ground station as well as drone. All the data we have is the position of the drone w.r.t each ground station at each time step. On top of this the sensors used in the ground station to provide measurement are noisy. Although through experiments we know that the noise is distributed according to Gaussian. The two objectives are :

- *To assume one of the ground station as origin and use its measurements to predict the drone's trajectory*
- *Determining the location of all ground station with respect to the one considered at origin*

These are the two objectives of this task. There are 4 different data available to us as 4 different csv files. CSV files "Data1.csv","Data2.csv" and "Data3.csv" have varying magnitude of noise and we have to predict the trajectory of the drone in all the three cases. In case of file "Data4.csv" , the problem is slightly difficult because due to server issues some of the values reported by ground stations were modified and were erroneous. Finally we are again supposed to predict the correct trajectory of drone after eliminating these erroneous value.

III. RELATED WORK

For this task , I learnt Gaussian mathematics, Statistics, G-H filters and Kalman Filters.

IV. INITIAL ATTEMPTS

To be fair I thought that this task was the toughest and Since the beginning of task i felt that I won't be able to do it. I referred many Kalman filter videos on from various sources but none of them made any sense and even if they did the clarity was not enough at all to apply it in my task. It was clear to me that small 10 minute video series wont be enough. So i started searching various written material and came across a Github page by Roger Labbe where he have provided a very informative course on Kalman Filters. After reading it until 7-8 chapters, I had enough clarity to apply the Kalman filter in our task. Though i did not went too deep in the mathematics of it but i got the basics of it.

My final approach was implementation of this Kalman filter.

V. FINAL APPROACH

Well Kalman Filter is essentially an concept rather than a Algorithm. There are various complexity of Kalman Filter which can be applied here . I also started with the basic Kalman filter on x axis and then build up to the final Filter which we will be discussing here.

Essentially what Kalman filter does is that it makes a prediction of the state of the system then it compares it with the measurement and then on basis of various factors such as belief in measurement i.e. how reliable is the sensor data, it estimates the a most probable position while updating its belief and then all previous estimates the Filter again predicts the position in next time step and the cycle goes on. Over time the filter prediction model becomes more and more accurate.

The Kalman filter used here is one dimensional that means for each of the 3 dimensions we have one observable variable and one hidden variable.

The observable variable is the position of the drone and the hidden variable is the velocity which is the first derivative of position. By including the velocity, we are trying to even optimize our estimation.

Lets Start Discussing the how we applied Kalman Filter :

1) Representation our system

We representation of our system is known as a state. This sate is what the Kalman Filter estimates for us. Our stare include the following things:

- X-coordinate of the system

- Derivative of X-coordinate(Velocity in X direction)
- Y-coordinate of the system
- Derivative of Y-coordinate(Velocity in Y direction)
- Z-coordinate of the system
- Derivative of Z-coordinate(Velocity in Z direction)

$$\text{State Matrix (X)} = \begin{bmatrix} X \\ X' \\ Y \\ Y' \\ Z \\ Z' \end{bmatrix}$$

2) Setting the dependency of one State variable on other

As we discussed our system predicts the State matrix for each time step by taking the previous position and other belief of system into consideration. For doing so we need two important things Process Model and State variance

Process Model is simple as our system is Newtonian the equations are :

$$X = X + \Delta t * X'$$

$$Y = Y + \Delta t * Y'$$

$$Z = Z + \Delta t * Z'$$

where as the derivatives remain constant as we are not considering acceleration in our state variable.

Thus,

$$\text{Process Model (F)} = \begin{bmatrix} 1 & t & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & t & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & t \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Now, The co-variance matrix keeps relations or dependency between 2 state variables. 0 means independent variables. We will keep on updating it. The diagonal elements are variances where all other elements are co-variances.

$$\text{Co-Variance Matrix (P)} = \begin{bmatrix} 20 & 0 & 0 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 & 0 & 0 \\ 0 & 0 & 20 & 0 & 0 & 0 \\ 0 & 0 & 0 & 20 & 0 & 0 \\ 0 & 0 & 0 & 0 & 20 & 0 \\ 0 & 0 & 0 & 0 & 0 & 20 \end{bmatrix}$$

This is called setting our belief in the system, this value of 20 is just a random value i saw fitting. Over time in each iteration we will keep on updating. This is the reason the prediction gets better over time.

3) Creating Noise Co-variance

In the previous section We mentioned that the co variance matrix will change every time step but by how much? If it changes rapidly, change in velocity will result in overshooting and undershooting. if it adapts to change slowly, and there is some acceleration it will start lagging or leading.

Thus we create a Noise co-variance matrix to determine this change rate. An example of this will be :

$$\text{(Q)} = \begin{bmatrix} 0.25 & 0.5 & 0 & 0 & 0 & 0 \\ 0.5 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.25 & 0.5 & 0 & 0 \\ 0 & 0 & 0.5 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.25 & 0.5 \\ 0 & 0 & 0 & 0 & 0.5 & 1 \end{bmatrix} * \text{Q-var}$$

Where Q is the noise co-variance matrix. By changing value of this q-var we can find the optimal value

This ends our prediction Process

The final prediction equations are :

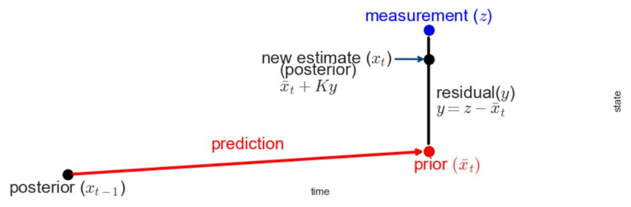
$$\bar{\mathbf{x}} = \mathbf{Fx}$$
$$\bar{\mathbf{P}} = \mathbf{FPF}^T + \mathbf{Q}$$

Lets not concern ourselves with the math regarding $F@P@F^T$ where '@' refers matrix multiplication, as even I did not get it. :(

Now lets move to the update Process

4) Update process

First of we need to understand how the update process works. In a time step we have 2 different values, i.e. The sensor measurement and the prediction so it is most probable that a really good estimate lies between then. So we take a factor this factor determine what estimation to take. Like if this factor equal to 0.5 we will take the mean i.e. middle value between these two. We call this factor Kalman Gain and denote it by 'K'



K denotes our trust in the measurement. If the trust is more the k will me near 1 i.e. we believe that the better estimate will be near measurement than prediction.

We can also say that K is the ratio of our trust in measurement/(Total trust in both the values)

Trust is measurement can be said equal to distrust in prediction . Thus, K can be represented as :

$$\mathbf{K} \approx \frac{uncertainty_{prediction}}{uncertainty_{prediction} + uncertainty_{measurement}}$$

Now we know that the difference between 2 values or the residual = measurement - prediction. But our prediction is has position as well as velocity whereas measurement has only position so we have to isolate position variables from the prediction or prior.

$$\text{(H)} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

thus residual $Y = z - H@X$, where, z is the measurement matrix.

Now we know that our filter have some noise so lets initiate some uncertainty in our sensor measurement:

(R) = $\begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix}$ * R-var where R-var is the the assume uncertainty in measurement. greater the value, more we force the filter to trust prediction

Lets calculate the total System uncertainty for denominator of our Kalman Gain:

System Uncertainty = $R + H@P@H^T$

That is uncertainty in measurement + uncertainty in prediction considered position only as measurement is position only.

Now , Finally our ***estimate will be prediction + residual* Kalman Gain*** as discussed and evident earlier. Finally, the thing we have been waiting for: Code implementation

```python
###Imports###

filename  = "Data4.csv"
zs_x = np.array(pd.read_csv(filename,usecols=[0]),
    dtype= np.float32)
zs_y = np.array(pd.read_csv(filename,usecols=[1]),
    dtype= np.float32)
zs_z = np.array(pd.read_csv(filename,usecols=[2]),
    dtype= np.float32)
zs1 = np.array([x * 1000 for x in zs_x])
zs2 = np.array([x * 1000 for x in zs_y])
zs3 = np.array([x * 1000 for x in zs_z])


dt = 1
X = np.array([[zs1[0], 2,zs2[0],2,zs3[0],2]]).T
P = np.diag([10, 10,10,10,10,10])
F = np.array([[1,dt,0,0,0,0],
              [0, 1,0,0,0,0],
              [0,0,1,dt,0,0],
              [0,0,0,1,0,0],
              [0,0,0,0,1,dt],
              [0,0,0,0,0,1]])

H = np.array([[1.,0,0,0,0,0],
              [0,0,1.,0,0,0],
              [0,0,0,0,1.,0]])

#R-var = 5000
R = np.array([[5000,0,0],
              [0,5000,0],
              [0,0,5000]])


Q_var = 0.02
Q = np.array([
             [.25*dt**4,.5*dt**3,0,0,0,0],
             [.5*dt**3,dt**2,0,0,0,0],
             [0,0,.25*dt**4,.5*dt**3,0,0],
             [0,0,.5*dt**3,dt**2,0,0],
             [0,0,0,0,.25*dt**4,5*dt**3],
             [0,0,0,0,.5*dt**3,dt**2]
             ])*Q_var


x_pos = [X[0]]
y_pos = [X[2]]
z_pos = [X[4]]

###KALMAN FILTER LOOP####
for i in range(zs_x.size):
    z = [zs1[i],zs2[i],zs3[i]]

    X = F @ X
    P = F @ P @F.T + Q

    S = H @ P @ H.T + R
    K = P @ H.T @ inv(S)
    Y = z - H@X

    X += K @ Y
    P = P - K @ H @ P

    #estimate positions
    x_pos.append(X[0])
    y_pos.append(X[2])
    z_pos.append(X[4])
```

Listing 1. Kalman Filter

Further we can smoothen the curve by taking average of consecutive 10 positions.

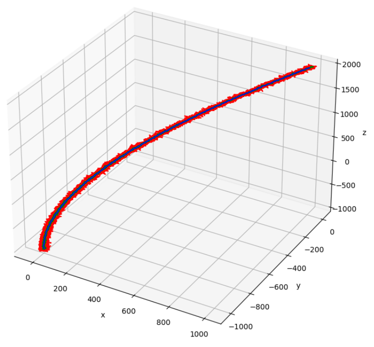
Note : I have multiplied all measurement by 1000 for ease in calculation and understanding

For DATA 4

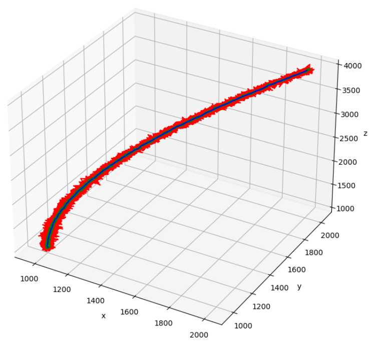
We have to reject some values, the mathematical way of it to see if a value lies beyond 3 standard variance away from the prediction and if so then reject it. but i could not implement it. So i tried other approach, the drone can't fly at high speeds such as 100m/s i.e. 360 km/hr so if the difference between two consecutive value is more than 100 then we reject it. As simple as that!!

VI. Results and Observation

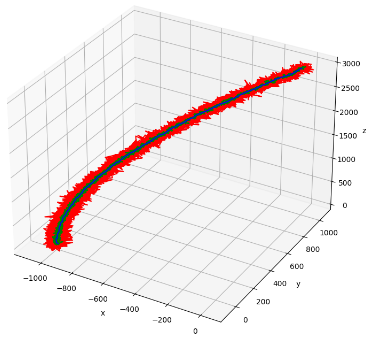
The results are :



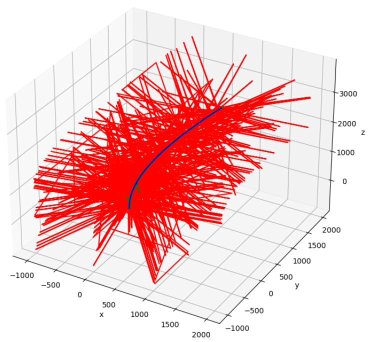
Data 1 :



Data 2 :



Data 3 :



Data 4 :

Here Red is the given data, Green is 1D Kalman Filter, Blue is 1D kalman Filter with smoothing

Now for Part 2 of the task

All this graph is assuming Ground Station 1 As origin Now position of other ground station

Ground Station No.	X	Y	Z
2	100	100	0
3	400	800	0
4	-500	-900	0
5	700	900	0
6	500	-900	0

VII. Future Work

By playing with Q-var we can see that there is definitely some accelerations so in future i would like to create one with 2nd derivative as well. i couldn't learn but will definitely learn EKF and particle Filters and implement them.

Conclusion

Kalman Filter is a powerful tool and is very necessary for drones. the math and statistics were tough parts but once done with simulation as a lot of fun.

References

[1] Roger Labble, "Kalman-and-Bayesian-Filters-in-Python",Github, 2020.
[2] Matlab,"Understanding Kalman Filter",Youtube,2017
[3] Michel van Biezen,"Special Topic -Kalman Filter", 2016