Skoltech

Skolkovo Institute of Science and Technology

Extended Kalman filter for navigation and tracking

Experimental Data Processing

Assignment №9

Team №10

Submitted by:
Yunseok Park
Ilya Novikov
Ruslan Kalimullin

Instructor:
Tatiana Podladchikova

MA060238 (Term 1B, 2022-2023)

October 20th, 2022

Contents

1 Introduction

The objective of this laboratory work is to develop Extended Kalman filter for tracking a moving object when measurements and motion model are in different coordinate systems. This will bring about a deeper understanding of main difficulties of practical Kalman filter implementation for nonlinear models.

2 Work progress

Question 1 - 3
We generated true trajectory X_i of an object motion disturbed by normally distributed random acceleration:

$$\begin{aligned} x_i &= x_{i-1} + V_{i-1}^x T \\ V_i^x &= V_{i-1}^x \\ y_i &= y_{i-1} + V_{i-1}^y T \\ V_i^y &= V_{i-1}^y \end{aligned} \tag{1}$$

Following initial conditions are given to generate true trajectory:

1. size of trajectory $N = 500$
2. Interval between measurements: $T = 1$
3. Initial components of velocity: $V_x = 10$; $V_y = 10$
4. Initial coordinates: $x_0 = 1000$; $y_0 = 1000$
5. Variance of noise σ_i, $\sigma_a^2 = 0.3^2$ for both a_i^x, a_i^y

Second, generated true values of range D and azimuth β:

$$\begin{aligned} D_i &= \sqrt{x_i^2 + y_i^2} \\ \beta_i &= \arctan \frac{x}{y} \end{aligned} \tag{2}$$

Third, generated measurements D^m and β^m of range D and azimuth β with given values of variances $\eta_i^D = 50^2$ and $\eta_i^\beta = 0.004^2$:

$$\begin{aligned} D_i^2 &= D_i + \eta_i^D \\ \beta_i^2 &= \beta_i + \eta_i^\beta \end{aligned} \tag{3}$$

Question 4 - 8
Initial conditions for Extended Kalman filter algorithms (initial filtered estimate of state vector $X_{0,0}$ (Equation 4), initial filtration error covariance matrix $P_{0,0}$ (Equation 5)) are

given:

$$X_0 = \begin{bmatrix} D_i^m(1)\sin\beta_i^m(1) \\ 0 \\ D_i^m(1)\cos\beta_i^m(1) \\ 0 \end{bmatrix} \tag{4}$$

$$X_0 = \begin{bmatrix} 10^10 & 0 & 0 & 0 \\ 0 & 10^10 & 0 & 0 \\ 0 & 0 & 10^10 & 0 \\ 0 & 0 & 0 & 10^10 \end{bmatrix} \tag{5}$$

At every filtration step in the algorithm we linearized measurment equation by determining:

$$\frac{dh(\hat{X}_{i+1,i})}{dX_{i+1}} \tag{6}$$

Question 9

Developed Kalman filter algorithm to estimate state vector X_i (extrapolation and filtration).

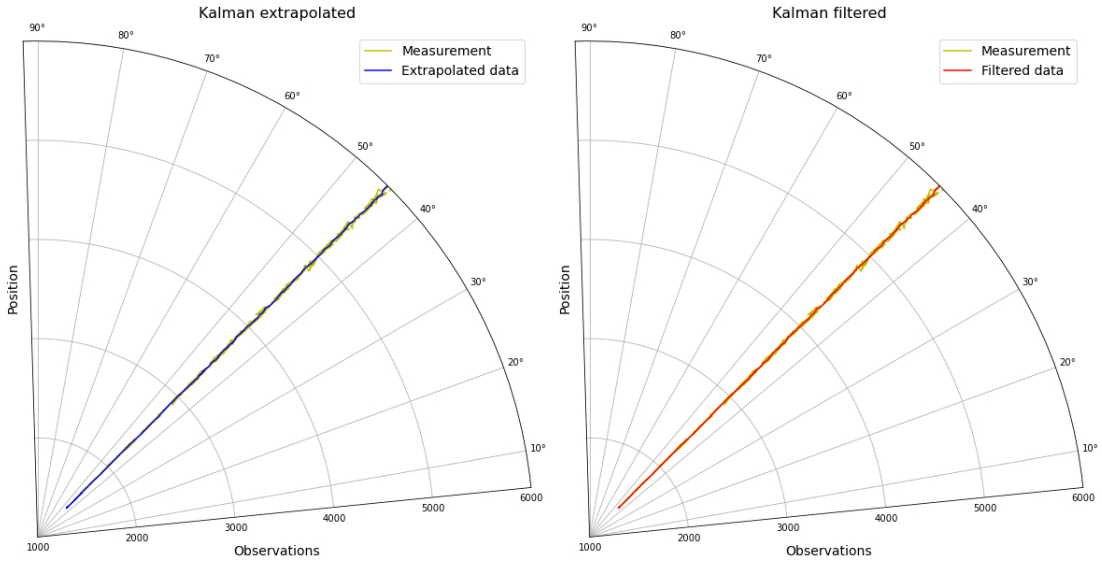


Figure 1: Extrapolated and filtered data

In this part, Kalman filter estimation and extrapolated estimation are plotted by calculating both range D and Azimuth β.

Question 10 - 11

In order to have a good sense of our results, and to check our accuracy, errors of extrapolation and filtration of both range D and azimuth β are plotted in Figure 2 and 3, respectively.

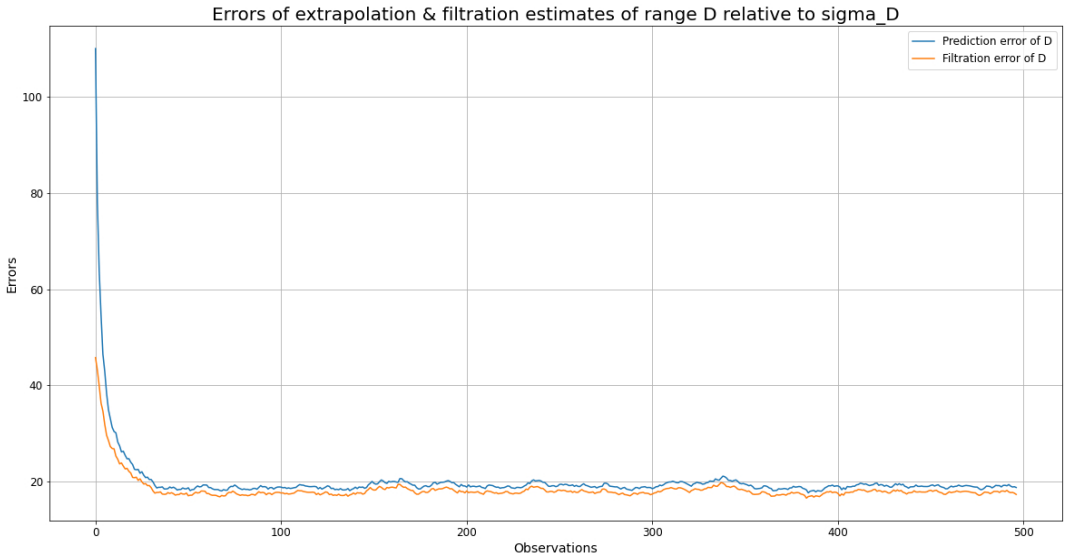


Figure 2: Error of range D

In Figure 2, it can be seen that both filtration and extrapolation errors of range D have the same trend. Visually we can see that filtration is slightly accurate than the extrapolation.

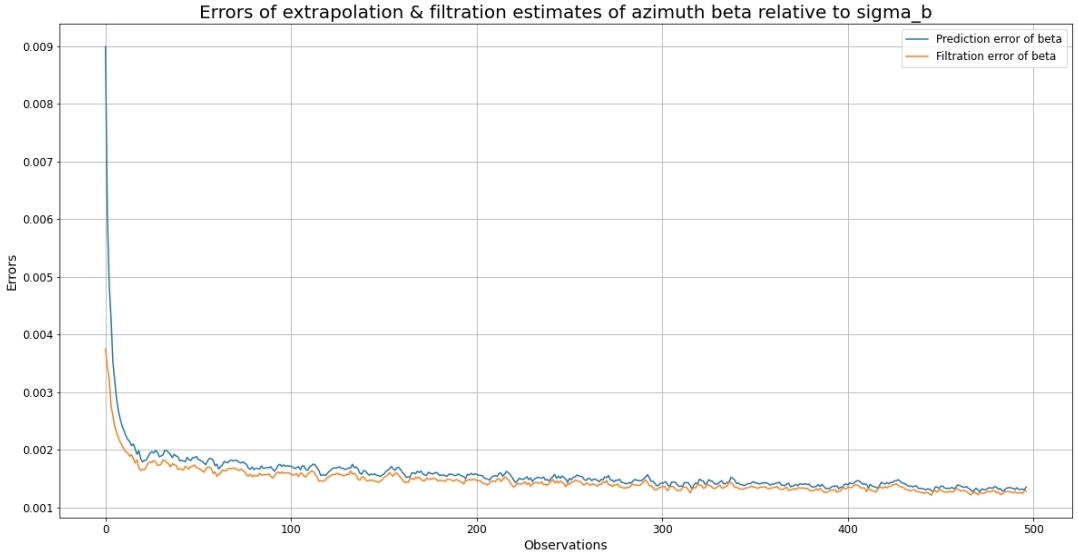


Figure 3: Error of azimuth β

In Figure 3, the first thing that stands out is the value of the errors of Azimuth β being so small. It can be explained by the distance being far enough that change in distance does not affect the azimuth angle. Therefore, it is normal to expect that less values. If we compare the estimations, it can be seen that just like in the range D, error of the β is less in filtration than in extrapolation as it is expected.

3 Conclusion

What we have learnt and tried:

1. How to work with close and far trajectories (deterministic) rather than random motion.
2. How initial position affects the estimation accuracy.
3. Importance of transformation of coordinates.
4. Effect of changing variances on estimation errors of range and azimuth.
5. How to decide the matrix R is well or ill-conditioned
6. When filter gain (K) diverges, there should be a distortion in linearity(due to linear dependence) or ill-conditioned(due to condition number) problem might be addressed.
7. How to understand significance of linearization errors.
8. How to connect linear/non-linear dependence of x and b with estimation errors while the observer is far away or quite close.

What we have reflected upon:

1. In what conditions filter stops working properly and diverges eventually.
2. How linearity affects the estimation errors and proper working of the filter.
3. Closer proximity to a moving object causes bigger estimation errors, hence the farther away the object from the observer makes better the linearization

Contribution of each members:

1. Ilya: wrote the code for the first case
2. Ruslan: wrote the code for the third case
3. Yunseok: wrote the code for the second case and report

In [1]:
```
import numpy as np
from numpy import linalg as LA
import matplotlib.pyplot as plt
import random
```

In [2]:
```
#Size of trajectory
n = 500
#Interval between measurements
T = 1
#Covariance
sigma_a = 0.3 ** 2
sigma_D = 50 ** 2
sigma_b = 0.004 ** 2
```

In [3]:
```
#Question 1
#Generation of true trajectory of an object motion disturbed by normally distributed random acceleration
def cartesian_trajectory(xy0 = [1000, 1000], v0 = [10, 10]):
    xy = np.zeros((2, n))
    v = np.zeros((2, n))
    a = np.zeros((2, n))
    a = np.random.normal(0, np.sqrt(sigma_a), n)
    xy[:, 0] = xy0
    v[:, 0] = v0
    for i in range(1, n):
        v[:, i] = v[:, i - 1] + a[i - 1] * T
        xy[:, i] = xy[:, i - 1] + v[:, i - 1] + (a[i - 1] * T ** 2) / 2
    return xy

c_traj = cartesian_trajectory()
```

In [4]:
```
#Question 2
#Generate true values of range D and azimuth beta
def true_p_trajectory(c_traj):
    xp = np.zeros((2, n))
    xp[0, :] = np.sqrt(c_traj[0, :] ** 2 + c_traj[1, :] ** 2) #D
    xp[1, :] = np.arctan(c_traj[0, :] / c_traj[1, :]) #beta
    return xp

p_traj = true_p_trajectory(c_traj)
```

In [5]:
```
def measurements(p_traj):
    zm = np.zeros((2, n))
    zm[0, :] = p_traj[0, :] + np.random.normal(0, np.sqrt(sigma_D), n) #Dm
    zm[1, :] = p_traj[1, :] + np.random.normal(0, np.sqrt(sigma_b), n) #betam
    return zm

zm = measurements(p_traj)
```

In [6]:
```
#Question 4 - 10

#Initial condition for Kalman filter
P0 = np.eye(4) * (10 ** 10)
X0 = np.array(([zm[0, 0] * np.sin(zm[1, 0]), 0, zm[0, 0] * np.cos(zm[1, 0]), 0]))
#Transition matrix
phi = np.array([[1, T, 0, 0], [0, 1, 0, 0], [0, 0, 1, T], [0, 0, 0, 1]])
#Input matrix
G = np.array([[(T * T) /2, 0], [T, 0], [0, (T * T) / 2], [0, T]])
#State noise covariance matrix Q
Q = G.dot(G.T) * sigma_a
#Measurement noise covariance matrix
R = np.array([[50 ** 2, 0], [0, 0.004 ** 2]])
```

In [7]:
```
def Kalman_filter(meas):
    #Initialization of arrays
    X_pred = np.zeros((4, n))
    P_pred = np.zeros((4, 4, n))
    X_filt = np.zeros((4, n))
    P_filt = np.zeros((4, 4, n))
    K = np.zeros((4, 2, n))
    pf = np.zeros((4, n))
    I = np.eye(4)
    dh = np.zeros((2, 4, n))

    # intial conditions
    X_filt[:, 0] = X0
    P_filt[:, :, 0] = P0

    for k in range(1, n):
        #Prediction
        X_pred[:, k] = phi.dot(X_filt[:, k - 1].reshape(4, 1)).reshape(4)
        P_pred[:, :, k] = (phi.dot(P_filt[:, :, k - 1])).dot(phi.T) + Q

        #Nonlinear function
```

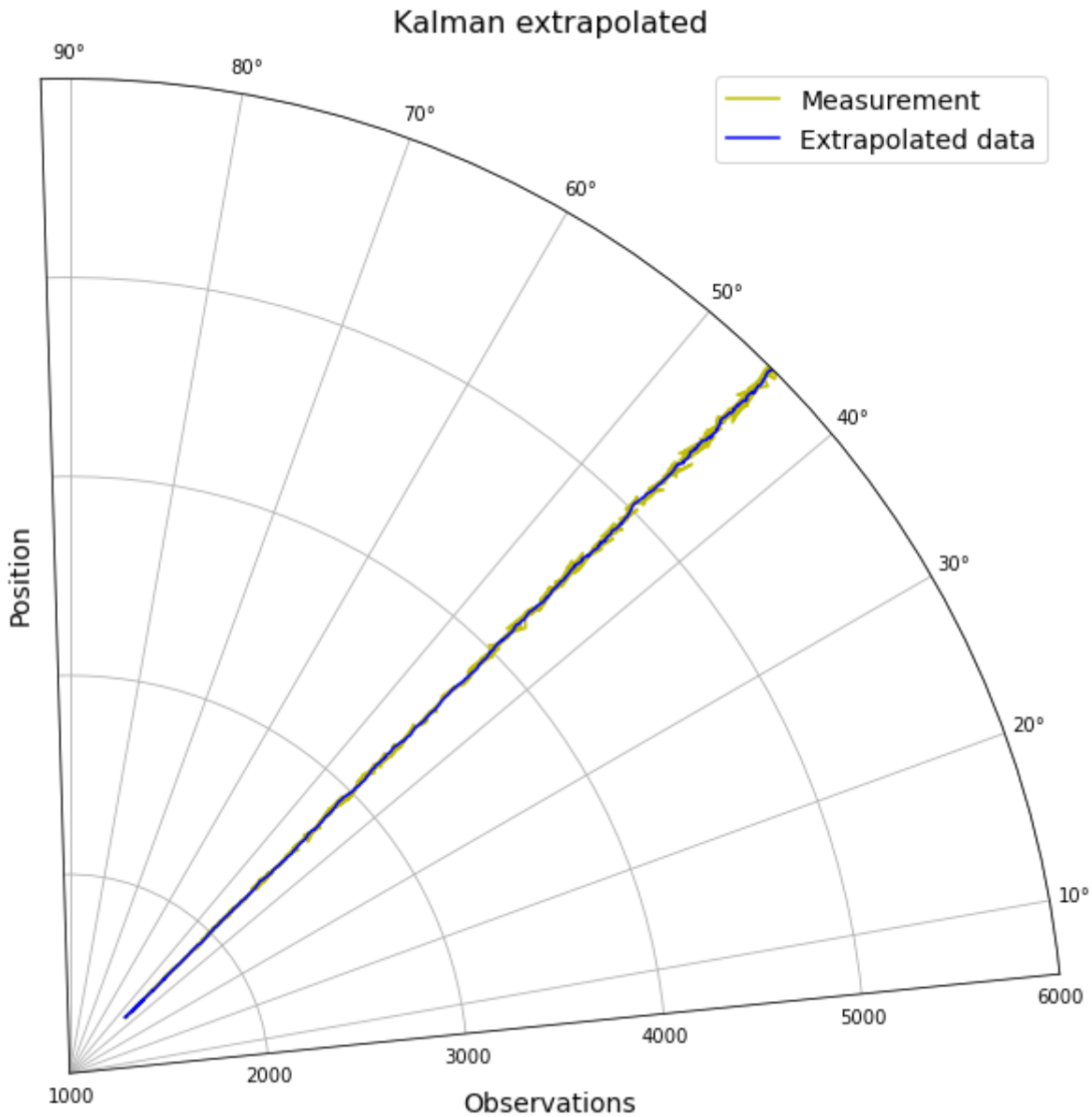
```
        dh[:, :, k] = np.array([[X_pred[0, k] / np.sqrt(X_pred[0, k] ** 2 + X_pred[2, k] ** 2), 0, X_pred[2, k] / np.sqrt(X_pred[
                                 [X_pred[2, k] / (X_pred[0, k] ** 2 + X_pred[2, k] ** 2), 0, -X_pred[0, k] / (X_pred[0, k] ** 2 +
        D = np.sqrt(X_pred[0, k] ** 2 + X_pred[2, k] ** 2)
        beta = np.arctan2(X_pred[0, k], X_pred[2, k])
        h = np.array([D, beta])

        #Filtration
        K[:, :, k] = np.dot(P_pred[:, :, k].dot(dh[:, :, k].T), np.linalg.inv((dh[:, :, k].dot(P_pred[:, :, k])).dot(dh[:, :, k]
        X_filt[:, k] = (X_pred[:, k].reshape(4, 1) + K[:, :, k].dot(meas[:, k].reshape(2, 1) - h.reshape(2, 1))).reshape(4)
        P_filt[:, :, k] = (I - K[:, :, k].dot(dh[:, :, k])).dot(P_pred[:, :, k])

    K = np.delete(K, 0, axis = 2)

    #Predicted
    pf[0, :] = np.sqrt(X_pred[0, :] ** 2 + X_pred[2, :] ** 2)
    pf[1, :] = np.arctan2(X_pred[0, :], X_pred[2, :])
    #Filtered
    pf[2, :] = np.sqrt(X_filt[0, :] ** 2 + X_filt[2, :] ** 2) #D
    pf[3, :] = np.arctan2(X_filt[0, :], X_filt[2, :]) #beta

    return pf

pf = Kalman_filter(zm)
```

In [8]:

```
def Errors(M):
    error_pred = np.zeros((2, n, M))
    error_filt = np.zeros((2, n, M))

    for i in range(M):
        c_traj = cartesian_trajectory()
        p_traj = true_p_trajectory(c_traj)
        zm = measurements(p_traj)
        pf = Kalman_filter(zm)

        error_pred[:, :, i] = (p_traj - pf[:2, :]) ** 2
        error_filt[:, :, i] = (p_traj - pf[2:, :]) ** 2

    Final_err_pred = np.sqrt(np.sum(error_pred, axis = 2) / (M - 1))
    Final_err_filt = np.sqrt(np.sum(error_filt, axis = 2) / (M - 1))

    return Final_err_pred, Final_err_filt

Final_err_pred, Final_err_filt = Errors(500)
```

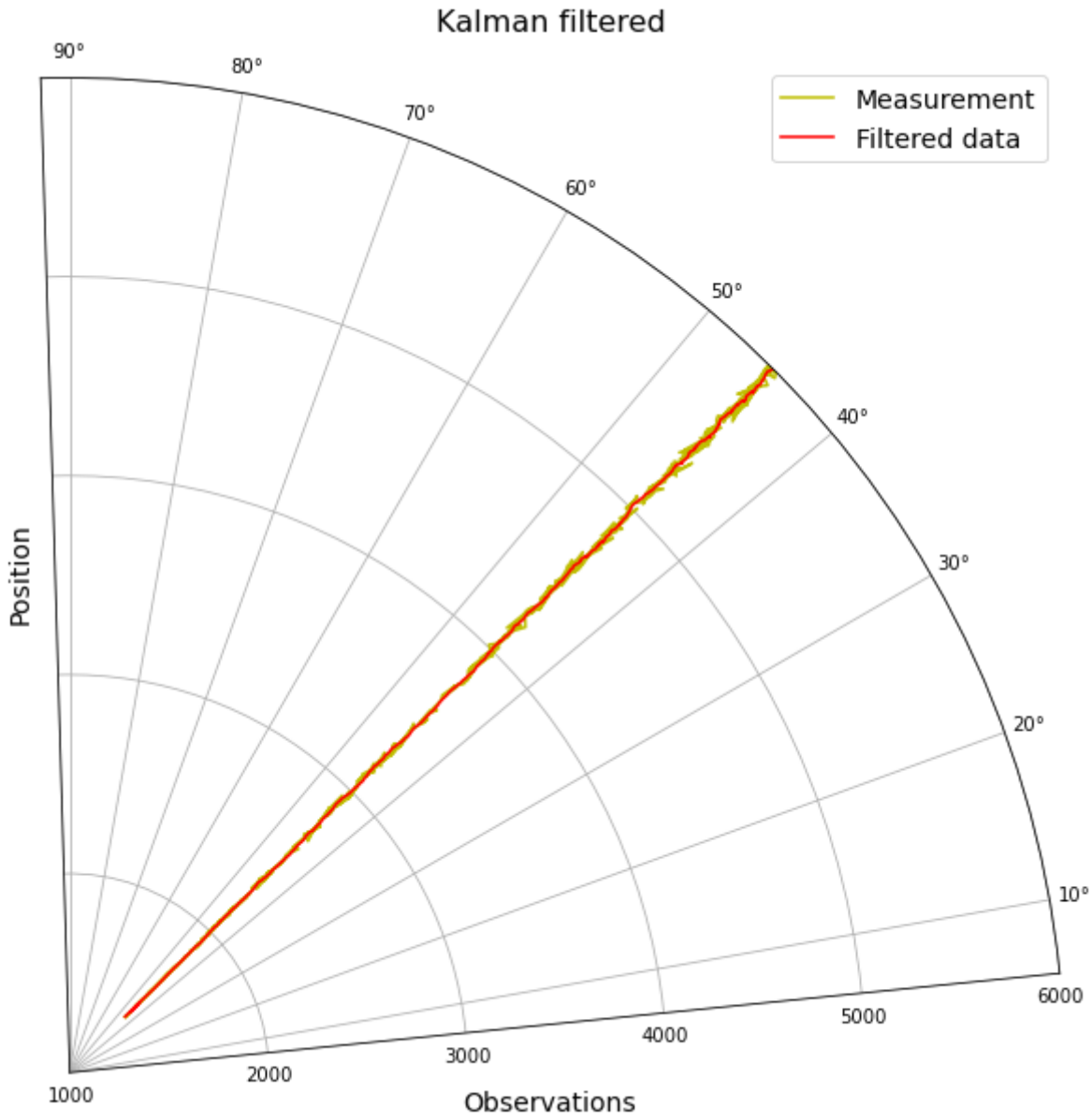
In [9]:

```
fig, ay = plt.subplots(figsize=(10, 10), subplot_kw={'projection': 'polar'})
ay.set_title("Kalman extrapolated", fontsize = 16)
ay.set_ylabel("Position", fontsize = 14)
ay.set_xlabel("Observations", fontsize = 14)
ay.plot(zm[1, 1:], zm[0, 1:], label='Measurement', c='y')
ay.plot(pf[1, 1:], pf[0, 1:], label='Extrapolated data', c='b')
plt.xlim(0.1, 1.6)
plt.ylim((1000,6000))
plt.legend(fontsize = 14,loc = 'best')
plt.savefig('K_extrapolated')
```

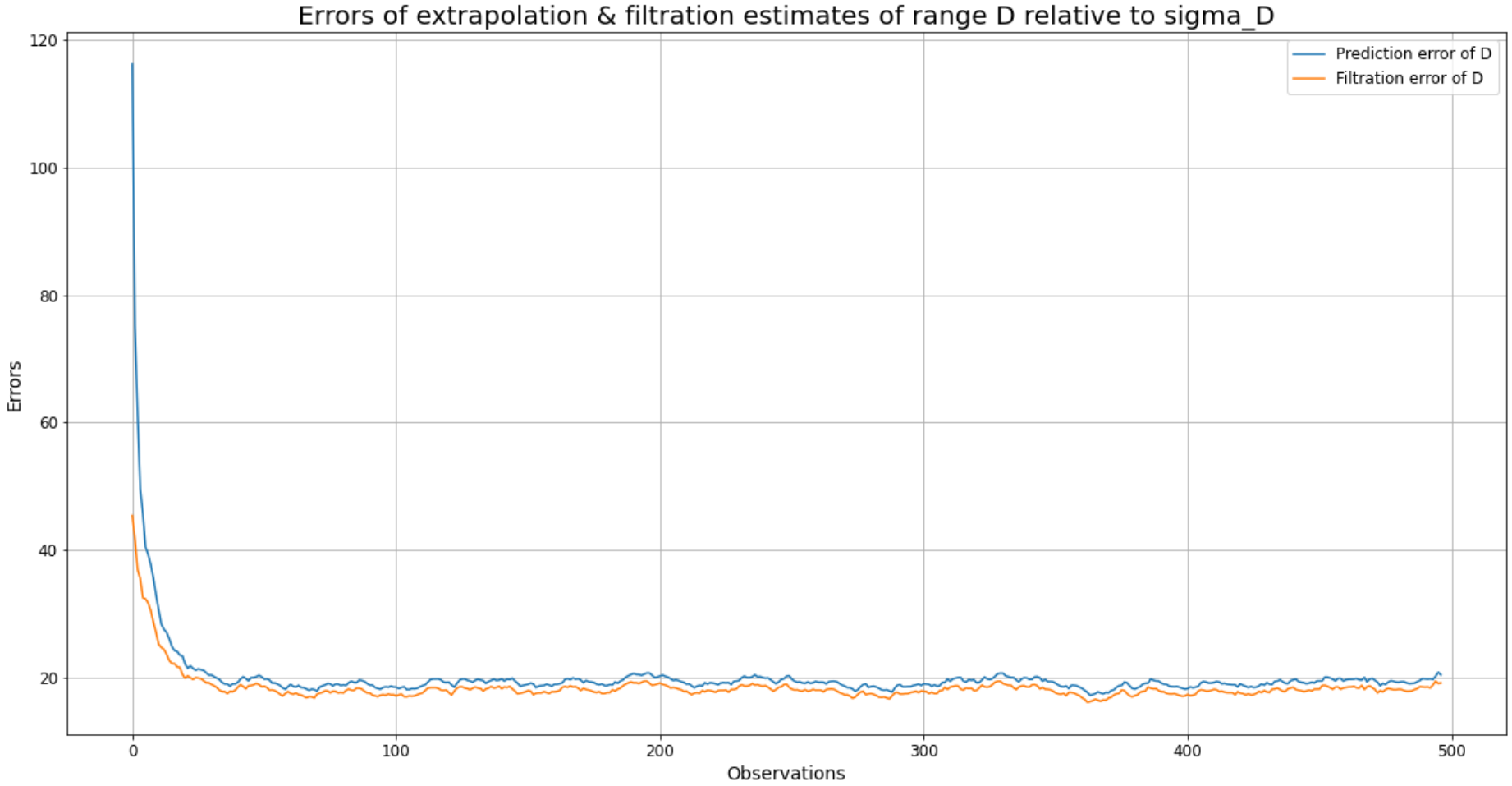


In [10]:
```
fig, az = plt.subplots(figsize=(10, 10), subplot_kw={'projection': 'polar'})
az.set_title("Kalman filtered", fontsize = 16)
az.set_ylabel("Position", fontsize = 14)
az.set_xlabel("Observations", fontsize = 14)
az.plot(zm[1, 1:], zm[0, 1:], label='Measurement', c='y')
```

```
az.plot(pf[3, 1:], pf[2, 1:], label='Filtered data', c='r')
plt.xlim(0.1, 1.6)
plt.ylim((1000,6000))
plt.legend(fontsize = 14,loc = 'best')
plt.savefig('K_filtered')
```



In [11]:
```
fig, a = plt.subplots(figsize=(20,10))
a.set_title("Errors of extrapolation & filtration estimates of range D relative to sigma_D", fontsize = 20)
a.set_xlabel("Observations", fontsize = 14)
a.set_ylabel("Errors", fontsize = 14)
a.plot(Final_err_pred[0, 3:], label = "Prediction error of D")
a.plot(Final_err_filt[0, 3:], label = "Filtration error of D")
a.tick_params(labelsize = 12)
a.legend(fontsize = 12)
a.grid()
plt.savefig('D')
```



In [12]:
```
fig, b = plt.subplots(figsize=(20,10))
b.set_title("Errors of extrapolation & filtration estimates of azimuth beta relative to sigma_b", fontsize = 20)
```

```
b.set_xlabel("Observations", fontsize = 14)
b.set_ylabel("Errors", fontsize = 14)
b.plot(Final_err_pred[1, 3:], label = "Prediction error of beta")
b.plot(Final_err_filt[1, 3:], label = "Filtration error of beta")
b.tick_params(labelsize = 12)
b.legend(fontsize = 12)
b.grid()
plt.savefig('b')
```