Sensor Fusion Example - Part 1 - Two Temperature Sensors

Our system consists of:

- 1 state variable (we want to know the temperature outside), and
- 2 (not quite accurate) sensors (two thermometers).

See my handwritten notes (pages 40-45), one day I will type them properly.

The source of this example is S. Levy's sensor fusion example (See his tutorial, https://simondlevy.github.io/ekf-tutorial/, part 14). Here, I have adopted BZARG's notation though (https://www.bzarg.com/p/how-a-kalman-filter-works-in-pictures/).

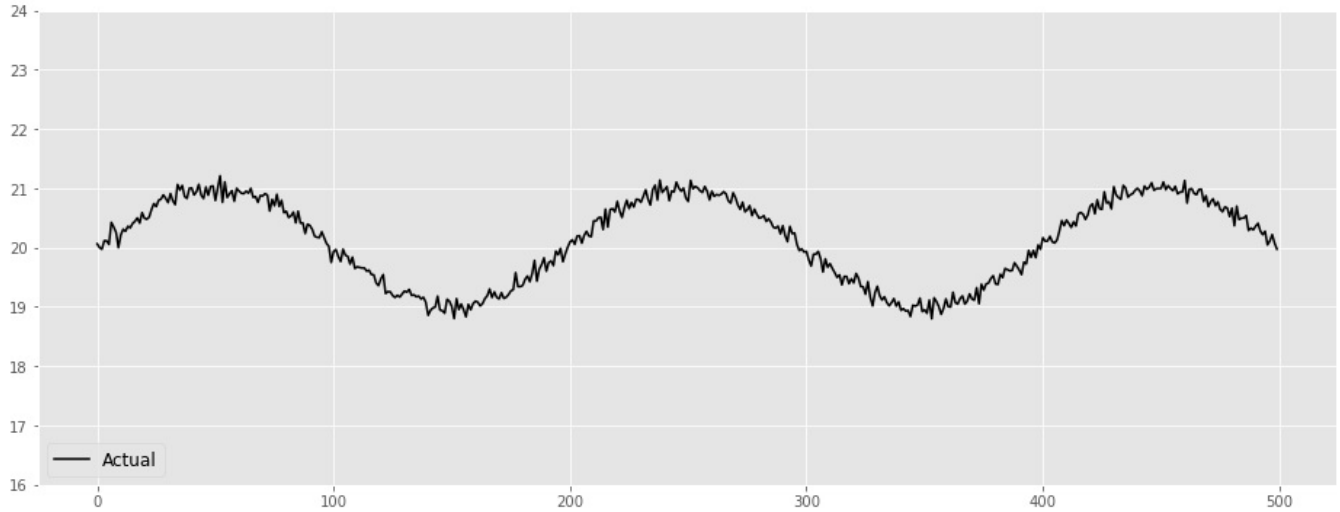
In [205…

```
# $Id: levy_sensor_fusion.ipynb,v 1.11 2024/02/01 23:39:46 ahmet Exp ahmet $

import numpy as np
import matplotlib.pyplot as plt

plt.style.use('ggplot')

np.random.seed(2023)
```

In [206…

```
# We pre-generate the samples and store in a NumPy array to allow us to run several
# experiments over the same signal for making some meaningful comparisons with
# different parameters.

# Here is the temperature signal we want to track.
# Note that x represents "the truth" here (hence no "hat"). Remember that state variables
# are not directly accessible to us.
# (We will try to extract (i.e. estimate) the true  values by using the outputs
# of "noisy" sensors.)
# Let's generate some samples of "the truth", they represent the outside temperature
N = 500 # number of "observations"
x = 20 + np.sin(5 * np.linspace(0, 1, N) * np.pi)

# Add some noise (temperature variation can not be an ideal sine wave)
# "Random" environmental fluctuations affect the temperature.
# Here, we are making an assumption that random environmental fluctuations are Gaussian.
noise_variance = 0.008
# noise mean is 0
w = np.random.normal(0, np.sqrt(noise_variance), N)
x = x + w
print(x[:5])

fig, ax = plt.subplots(figsize=(16, 6))
ax.plot(x, color='black', label='Actual')
ax.set_ylim([16,24])
ax.legend(fontsize='12', loc='lower left')
plt.show()
```

```
[20.06365402 20.00245087 19.97330615 20.11542726 20.11644562]
```

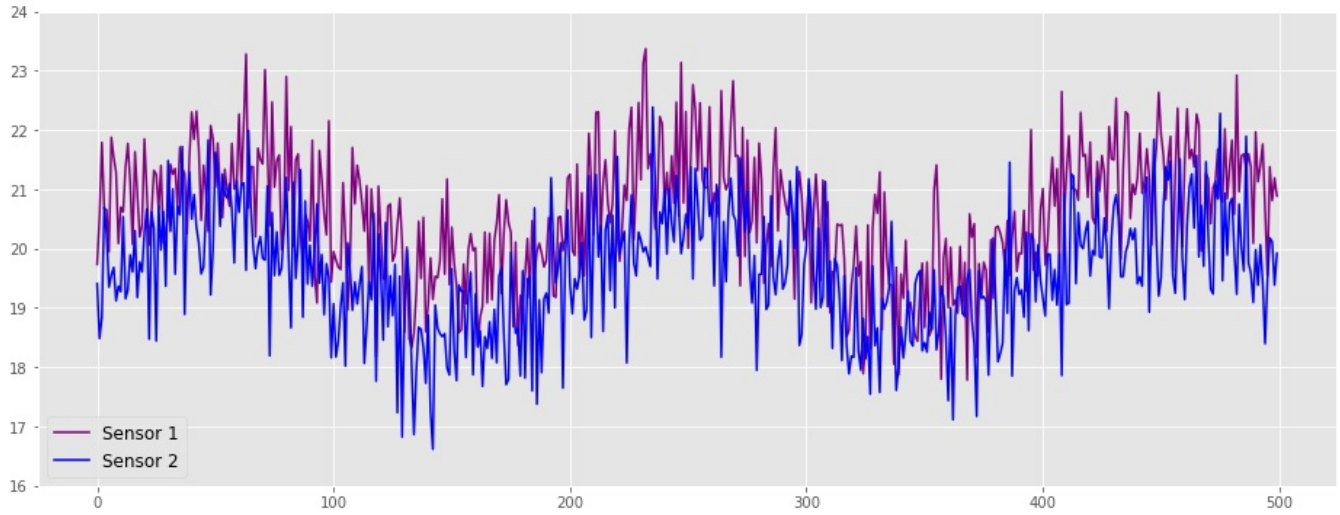


In [207…

```
# Generate a series of noisy sensor readings

# H is the mapping matrix between the sensor readings and state variables
# We assume both sensors produce values that need not to be scaled
H = np.array([[1],[1]])
```

```
BIAS1 = +0.5
BIAS2 = -0.5

# Covariance matrix of the sensor readings.
R = np.array([[0.64, 0], [0, 0.64]])

# We have two thermometers, we pre-generate two readings for each timestep.
z1 = x + BIAS1 + np.random.normal(0, np.sqrt(R[0,0]), N)
z2 = x + BIAS2 + np.random.normal(0, np.sqrt(R[1,1]), N)
print('z1 is:', type(z1).__name__,'and its shape is:', z1.shape)

z = np.vstack((z1,z2))
print('z is:', type(z).__name__,'and its shape is:', z.shape)

fig, ax = plt.subplots(figsize=(16, 6))
ax.plot(z[0,:], color='purple', label='Sensor 1')
ax.plot(z[1,:], color='blue', label='Sensor 2')
ax.set_ylim([16,24])
ax.legend(fontsize='12', loc='lower left')
plt.show()
```

```
z1 is: ndarray and its shape is: (500,)
z is: ndarray and its shape is: (2, 500)
```



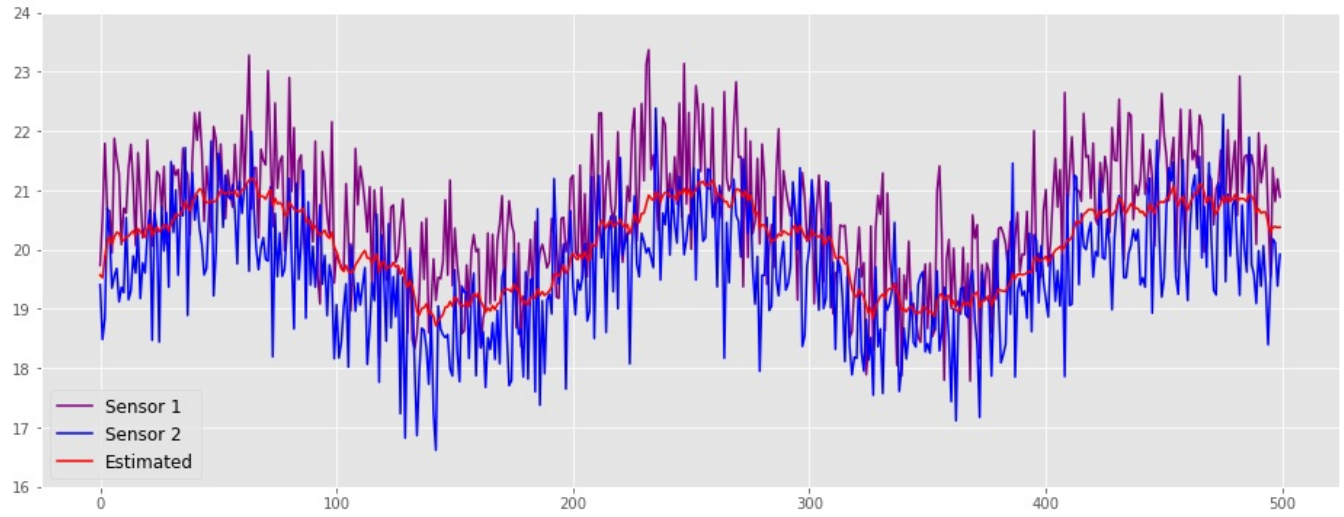
In [208…

```
# Let's run our Kalman Filter

F = np.array([[1]]) # F is a 1x1 matrix

# Covariance matrix of the process noise. In this example we have only one state variable
# (temperature), so, the covariance matrix here is actually a scalar, which is
# the variance of the process noise w (again a scalar).
Q = 0.01

# Covariance matrix of the state variables.
P = np.array([[1]]) # P is a 1x1 matrix
print('P is:', type(P).__name__,'and its shape is:', P.shape)

x_hat = np.zeros((1, N)) # x_hat is a 1 X N matrix
print('x_hat is:', type(x_hat).__name__,'and its shape is:', x_hat.shape)

# Assign the average of the first sensor readings as the initial estimate
x_hat[:,0] = (z[0,0] + z[1,0])/2

# Kalman Filter
for k in range(1, N):
    # Predict
    x_hat[:,k] = F @ x_hat[:,k-1] # System model: We estimate the current value same as
                                  # the previous value (see the matrix F above)
    P = (F @ P @ F.T) + Q

    # Update
    K = P @ H.T @ np.linalg.inv((H @ P @ H.T) + R)
    P = P - K @ H @ P
    x_hat[:,k] = x_hat[:,k] + K @ (z[:,k] - (H @ x_hat[:,k]))
```

```
P is: ndarray and its shape is: (1, 1)
x_hat is: ndarray and its shape is: (1, 500)
```

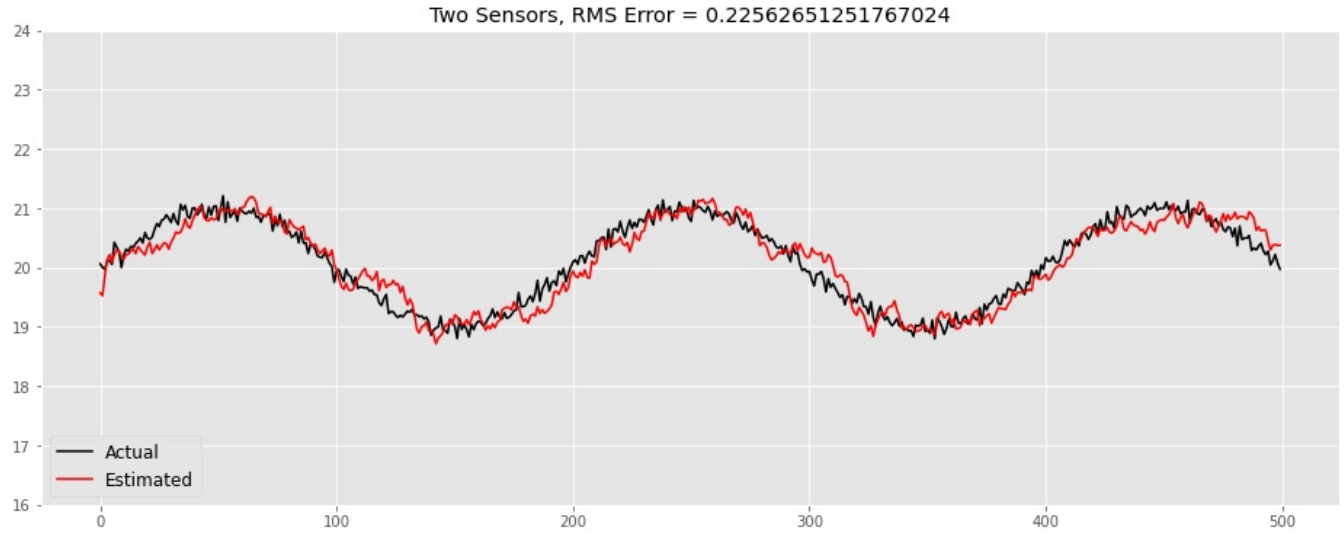
In [209…

```
fig, ax = plt.subplots(figsize=(16, 6))
ax.plot(z[0,:], color='purple', label='Sensor 1')
ax.plot(z[1,:], color='blue', label='Sensor 2')
ax.plot(x_hat[0,:], color='red', label='Estimated')
ax.set_ylim([16,24])
ax.legend(fontsize='12', loc='lower left')
plt.show()
```



In [210…

```
# Let's compare real values and their estimates
x_hat = np.reshape(x_hat, N) # x is a vector of [N,], x_hat is a [1,N] matrix,
                             # (if I don't do this, RMSE calculation goes rogue.)
rmse = np.sqrt(np.sum((x - x_hat)**2) / len(x))

fig, ax = plt.subplots(figsize=(16, 6))
ax.set_title('Two Sensors, RMS Error =' + ' '+str(rmse))
ax.plot(x, color='black', label='Actual')
ax.plot(x_hat, color='red', label='Estimated')
ax.set_ylim([16,24])
ax.legend(fontsize='12', loc='lower left')
plt.show()
```



In [ ]: