Online regression using a Kalman Filter for Air Quality

see also 'Optimum Linear Estimation' at https://www.sciencedirect.com/topics/social-sciences/kalman-filter (https://www.sciencedirect.com/topics/social-sciences/kalman-filter)

- Replaces an offline multiple linear regression (MLR) batch run, runs recursively on an Arduino Atmega1284P as adaptive filter for the MLR regression coefficients
- Kalman Filter is derived from https://github.com/zziz/kalman-filter (https://github.com/zziz/kalman-filter)
- For theory, please read https://en.wikipedia.org/wiki/Kalman_filter (https://en.wikipedia.org/wiki/Kalman_filter)
- Key is to set the covariance of the process noise matrix to a zero matrix! See hint in 'Optimum Linear Estimation' at https://www.sciencedirect.com/topics/social-sciences/kalman-filter (https://www.sciencedirect.com/topics/social-sciences/kalman-filter).
- I send a big thank you to those who provided these great basis contributions
- Click on the button 'Re-start the kernel, and then re-run the whole notebook' above

Basic Kalman Filter class from https://github.com/zziz/kalman-filter (https://github.com/zziz/kalman-filter):

In [1]:
```
class KalmanFilter(object):
    def __init__(self, F = None, B = None, H = None, Q = None, R = None, P = None, x0 = None):

        if(F is None or H is None):
            raise ValueError("Set proper system dynamics.")

        self.n = F.shape[1]
        self.m = H.shape[1]

        self.F = F
        self.H = H
        self.B = 0 if B is None else B
        self.Q = np.eye(self.n) if Q is None else Q
        self.R = np.eye(self.n) if R is None else R
        self.P = np.eye(self.n) if P is None else P
        self.x = np.zeros((self.n, 1)) if x0 is None else x0

    def predict(self, u = 0):
        self.x = np.dot(self.F, self.x) + np.dot(self.B, u)             # Predicted (a priori) state
        self.P = np.dot(np.dot(self.F, self.P), self.F.T) + self.Q      # Predicted (a priori) estim
        return self.x

    def update(self, z):
        y = z - np.dot(self.H, self.x)                                  # Innovation or measurement
        S = self.R + np.dot(self.H, np.dot(self.P, self.H.T))           # Innovation (or pre-fit res
        #print("\nUpdate: self.H = ", self.H)
        #print("\nUpdate: self.P = ", self.P)
        #print("\nUpdate: self.R = ", self.R)
        K = np.dot(np.dot(self.P, self.H.T), np.linalg.inv(S))          # Optimal Kalman gain
        #print("\nUpdate: Kalman gain matrix K  = ", K)
        self.x = self.x + np.dot(K, y)
        I = np.eye(self.n)

        self.P = np.dot(np.dot(I - np.dot(K, self.H), self.P), (I - np.dot(K, self.H)).T) + np.dot(
```

Read historian.csv (same input file as 'Multiple linear regression for BME680 gas readings of a single sensor.ipynb' is using

In [2]:
```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

import pandas as pd
from datetime import datetime

import numpy as np


dateparse = lambda x: pd.datetime.strptime(x, '%d.%m.%Y %H:%M:%S,%f')

df0 = pd.read_csv("historian.csv", sep=';', thousands=".", decimal=",", skiprows = [0,1,2],dtype={'H

# print first 5 lines of the pandas dataframe

df0.head(19)

```

Out[2]:

	Datum	Mode	raw_gas_resistance	relative_humidity	temperature
0	2021-01-15 19:00:00.000	2	177160	28.9	24.6
1	2021-01-15 19:00:56.674	2	177160	28.9	24.6
2	2021-01-15 19:00:56.683	2	177160	29.6	24.6
3	2021-01-15 19:00:56.688	2	174900	29.6	24.6
4	2021-01-15 19:05:29.488	2	174900	29.6	24.6
5	2021-01-15 19:05:29.498	2	174900	30.1	24.6
6	2021-01-15 19:05:29.504	2	175280	30.1	24.6
7	2021-01-15 19:10:02.302	2	175280	30.1	24.6
8	2021-01-15 19:10:02.306	2	175280	29.7	24.6
9	2021-01-15 19:10:02.312	2	175860	29.7	24.6
10	2021-01-15 19:14:35.105	2	175860	29.7	24.6
11	2021-01-15 19:14:35.115	2	175860	29.9	24.6
12	2021-01-15 19:14:35.120	2	173780	29.9	24.6
13	2021-01-15 19:19:07.919	2	173780	29.9	24.6
14	2021-01-15 19:19:07.927	2	173780	30.3	24.6
15	2021-01-15 19:19:07.932	2	170320	30.3	24.6
16	2021-01-15 19:23:40.742	2	170320	30.3	24.6
17	2021-01-15 19:23:40.745	2	170320	30.7	24.6
18	2021-01-15 19:23:40.751	2	167060	30.7	24.6

In [3]:
```
# keep every 3rd row (CCU historian is tracking every change of a datapoint separately)
# each three consecutive entries in history.csv are identical; therefore we take every third entry (
df = df0[(df0.index % 3 == 0)]

df.head(7)
```

Out[3]:

	Datum	Mode	raw_gas_resistance	relative_humidity	temperature
0	2021-01-15 19:00:00.000	2	177160	28.9	24.6
3	2021-01-15 19:00:56.688	2	174900	29.6	24.6
6	2021-01-15 19:05:29.504	2	175280	30.1	24.6
9	2021-01-15 19:10:02.312	2	175860	29.7	24.6
12	2021-01-15 19:14:35.120	2	173780	29.9	24.6
15	2021-01-15 19:19:07.932	2	170320	30.3	24.6
18	2021-01-15 19:23:40.751	2	167060	30.7	24.6

Print values of Pandas dataframe. Please x-check if they meet your expectation!

In [4]:
```
df.values
```

Out[4]:
```
array([[Timestamp('2021-01-15 19:00:00'), 2, 177160, 28.9, 24.6],
       [Timestamp('2021-01-15 19:00:56.688000'), 2, 174900, 29.6, 24.6],
       [Timestamp('2021-01-15 19:05:29.504000'), 2, 175280, 30.1, 24.6],
       ...,
       [Timestamp('2021-01-19 16:41:43.161000'), 2, 112700, 38.5, 24.9],
       [Timestamp('2021-01-19 16:46:15.860000'), 2, 112860, 38.5, 24.9],
       [Timestamp('2021-01-19 16:50:48.569000'), 2, 113640, 38.6, 24.9]],
      dtype=object)
```

In [5]:
```
1 df.head(-1)
```

Out[5]:

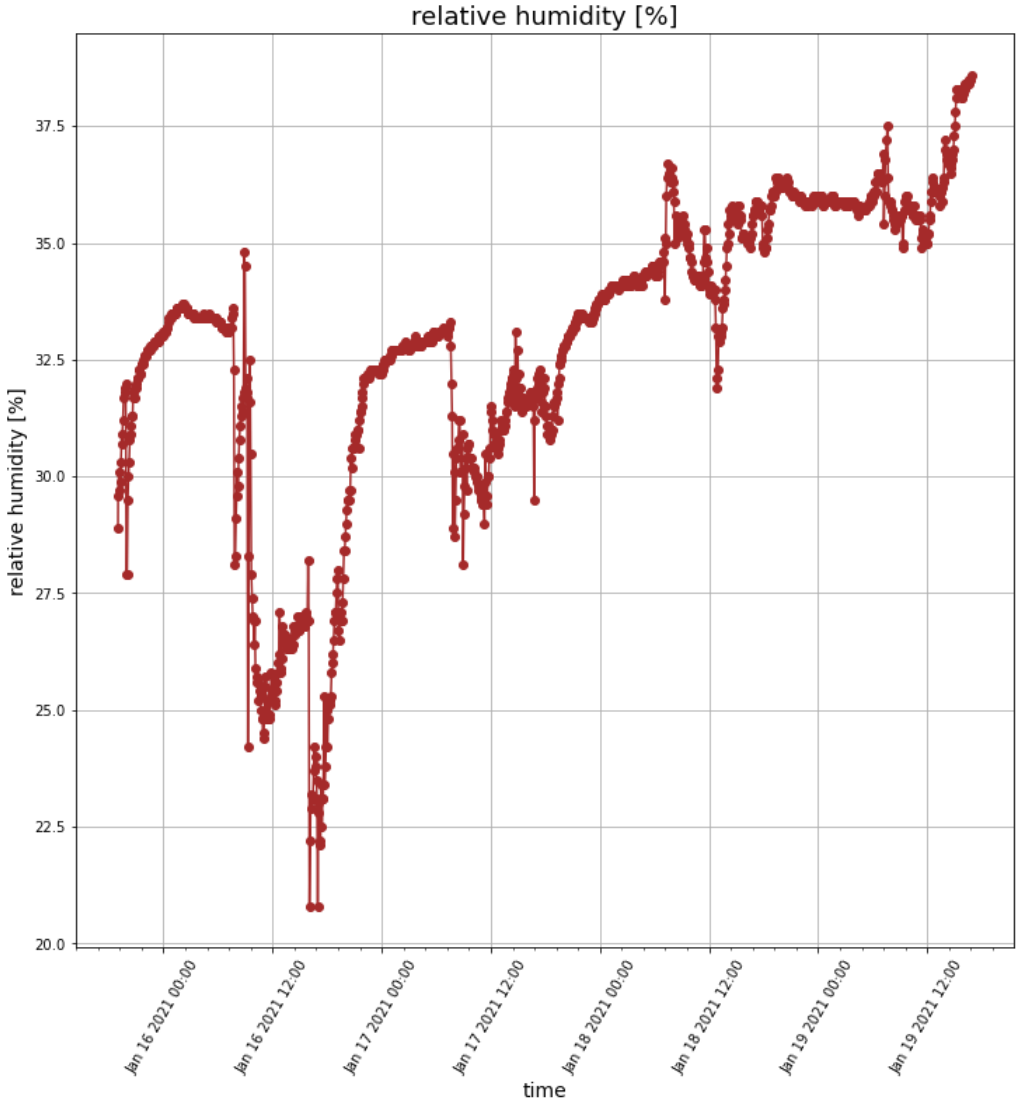
	Datum	Mode	raw_gas_resistance	relative_humidity	temperature
0	2021-01-15 19:00:00.000	2	177160	28.9	24.6
3	2021-01-15 19:00:56.688	2	174900	29.6	24.6
6	2021-01-15 19:05:29.504	2	175280	30.1	24.6
9	2021-01-15 19:10:02.312	2	175860	29.7	24.6
12	2021-01-15 19:14:35.120	2	173780	29.9	24.6
...	...	...	...	...	...
3660	2021-01-19 16:28:04.985	2	111520	38.4	24.9
3663	2021-01-19 16:32:37.689	2	112140	38.5	24.9
3666	2021-01-19 16:37:10.407	2	112460	38.4	24.9
3669	2021-01-19 16:41:43.161	2	112700	38.5	24.9
3672	2021-01-19 16:46:15.860	2	112860	38.5	24.9

1225 rows × 5 columns

```
1
```

Plot the calculated relative humidity

In [6]:
```
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from matplotlib.dates import DateFormatter
from matplotlib.ticker import (MultipleLocator, FormatStrFormatter, AutoMinorLocator)
fig, ax = plt.subplots(figsize=(12, 12))
plt.xticks(rotation=60)
ax.xaxis.set_major_formatter(DateFormatter('%b %d %Y %H:%M'))
ax.xaxis.set_minor_locator(AutoMinorLocator())
ax.plot_date(df['Datum'], df['relative_humidity'], linestyle='solid', color='brown')
plt.title('relative humidity [%]', fontsize=18)
plt.xlabel('time', fontsize=14)
plt.ylabel('relative humidity [%]', fontsize=14)
plt.grid(True)
plt.show()
```



Print again the first 5 lines of the Pandas dataframe. Check if a column for the relative humidity has been added.

In [7]:
```
df.head()
```

Out[7]:

	Datum	Mode	raw_gas_resistance	relative_humidity	temperature
0	2021-01-15 19:00:00.000	2	177160	28.9	24.6
3	2021-01-15 19:00:56.688	2	174900	29.6	24.6
6	2021-01-15 19:05:29.504	2	175280	30.1	24.6
9	2021-01-15 19:10:02.312	2	175860	29.7	24.6
12	2021-01-15 19:14:35.120	2	173780	29.9	24.6

Create a subset of the measurement data: 'raw_gas_resistance','temperature','relative_humidity'

In [8]:
```
my_observations = df[['raw_gas_resistance','temperature','relative_humidity']]
my_observations.head()
```

Out[8]:

	raw_gas_resistance	temperature	relative_humidity
0	177160	24.6	28.9
3	174900	24.6	29.6
6	175280	24.6	30.1
9	175860	24.6	29.7
12	173780	24.6	29.9

Create a numpy array of measurements for further processing

In [9]:
```
list_of_rows = [list(row) for row in my_observations.values]
```

Print the first four elements of list of lists i.e. rows

In [10]:
```
print(list_of_rows[:4])
```

```
[[177160.0, 24.6, 28.9], [174900.0, 24.6, 29.6], [175280.0, 24.6, 30.1], [175860.0, 24.6, 29.7]]
```

Convert the selection of measurements to a numpy array

In [11]:
```
np.array(list_of_rows)
measurements = np.array(list_of_rows)
print("number of measurement datapoints = ", len(measurements))
```

```
number of measurement datapoints =  1226
```

```
## Set the parameters of the Kalman filter

- Kalman filter with a zero covariance matrix for the process noise is well known a the recursive minimum least-square error (LMMSE) filter for a linear system with some assumptions on auto- and cross-correlations of process and measurement noise and initial state.

- observation vector y                          :   [raw_gas_resistance]; n=1; note: 'temperature' and 'rH' are NOT part of the observation vector!
- system state vector X                         :   [VOC_resistance, alpha_temperature, beta_rH]; m=3
- state transition matrix F                     :   identity matrix (m, n)
- observation transition matrix H               :   initial identidy matrix (1,m); then set to state dependant
- covariance matrix of the process noise Q      :   zero matrix (m,m)
- covariance matrix of the observation noise R :   matrix(1,1) with very small value
```

In [12]:
```
F = np.eye(3)
H = np.array([ [1, 1, 1] ]).reshape(1, 3)
# key ist to set Q to a zero matrix, in this case the Kalman filter works as an ordinary least squa
Q = np.array([ [0, 0, 0], [0, 0, 0], [0, 0, 0] ]).reshape(3, 3)
# set covariance of gas resistance measurements also to a very small value
R = np.array([ [0.000000001] ]).reshape(1, 1)

print("\nF = ",F)  # the state-transition model;
print("\nInitial H = ",H)  # the observation model;
print("\nQ = ",Q)  # covariance of the process noise
print("\nR = ",R)  # covariance of the observation noise

```

```
F =  [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]

Initial H =  [[1 1 1]]

Q =  [[0 0 0]
 [0 0 0]
 [0 0 0]]

R =  [[1.e-09]]
```

Initialize the Kalman filter

In [13]:
```
kf = KalmanFilter(F = F, H = H, Q = Q, R = R)
predictions = []
compensated_gas_resistance=[]
states=[]

#print("raw gas resistance measurements =", measurements[:,0])

print("dim measurements : ", measurements.shape)

last_index = len(measurements)

print ("last index of measurement array = ", last_index)


```

```
dim measurements :  (1226, 3)
last index of measurement array =  1226
```

Run the Kalman filter

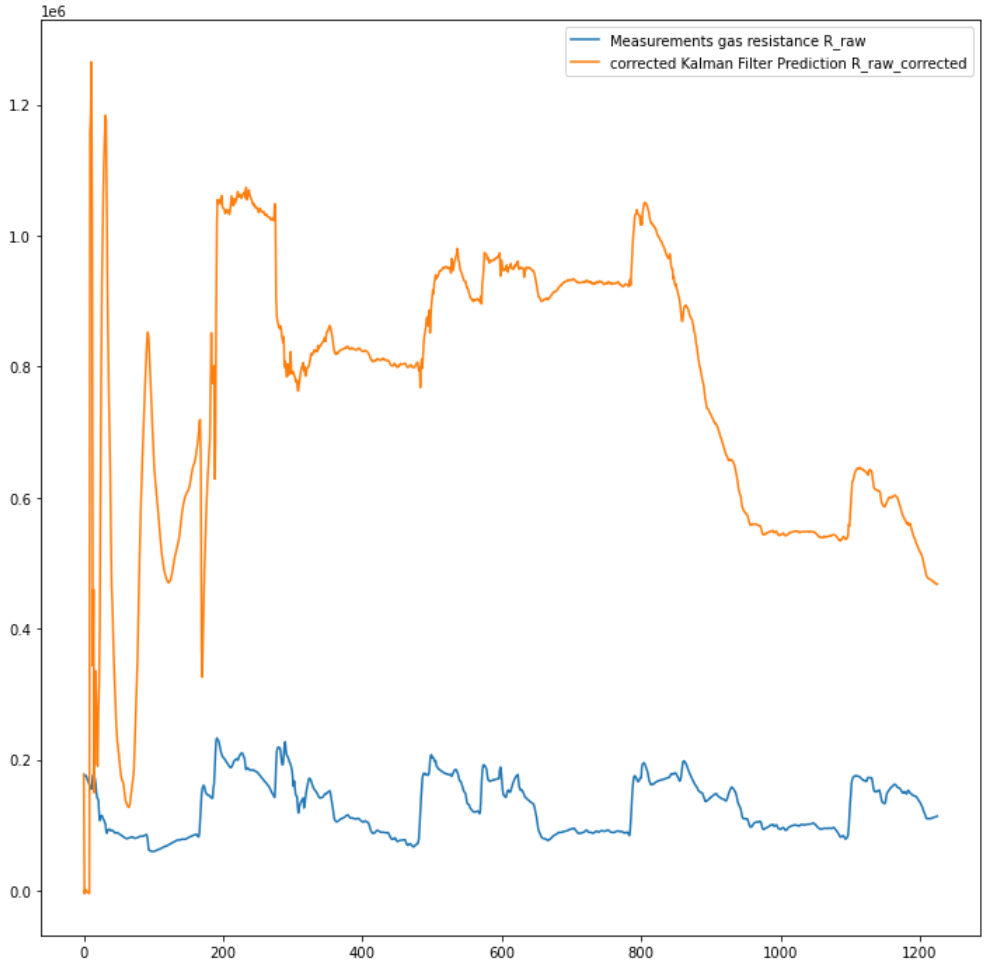
In [14]:
```
it = 0  # iteration index
#print("\nState vector kf.x= ", kf.x)
for z in measurements:
    zg = z[0] # raw_gas_resistance
    # make observation model matrix state dependant
    H = np.array([[1, z[1], z[2]]]).reshape(1, 3)
    # z[1]: measured temperature T
    # z[2]: measured relative humidity rH
    # estimated state vector x:
    # x[0]: estimated VOC resistance
    # x[1]: estimated regression coefficient for T temperature dependency
    # x[2]: estimated regression coefficient for rH relative humidity dependency
    kf.H = H
    it = it + 1
    #print("\nState vector kf.x= ", kf.x)
    #print results for the last sample of the measurement sequence
    if ((it == last_index)):  # print results of last measurement index
        print ("\nIteration index = ", it)
        print ("\n")
        print("\nState vector kf.x= ", kf.x)
        print("\nObservation vector z = ", z)
        print("\nObservation transition matrix kf.H = ", kf.H)
        print("\nKalman filter prediction = ", kf.predict())
        print("\nKalman filter update = ",np.dot(H,  kf.predict()))
        print ("\n\n")
    predictions.append(np.dot(H,  kf.predict()))

    compensated_gas_resistance.append(zg-kf.predict()[1,0]*z[1]-kf.predict()[2,0]*z[2])
    print("\nraw gas resistance                = ",zg)
    print("\ntemperature coefficent prediction = ",kf.predict()[1,0])
    print("\ntemperature                       = ",z[1])
    print("\ntemperature compensation          = ",-kf.predict()[1,0]*z[1])
    print("\nhumidity coefficent prediction    = ",kf.predict()[2,0])
    print("\relative humidity                 = ",z[2])
    print("\nhumidity compensation             = ",-kf.predict()[2,0]*z[2])
    #print("\nKalman state prediction           = ",kf.predict())
    #print("\ntmperature coefficent prediction = ",kf.predict()[1,0])
    #print("\ncompensated gas resistance       = ",zg-kf.predict()[1,0]*z[1]-kf.predict()[2,0]*z[2],
    states.append(kf.x)
    kf.update(zg)  #only zg raw_gas_resistance is an observation variable!
```

```
raw gas resistance                =  177160.0

temperature coefficent prediction =  0.0

temperature                       =  24.6

temperature compensation          =  -0.0

humidity coefficent prediction    =  0.0
elative humidity                 =  28.9

humidity compensation             =  -0.0

raw gas resistance                =  174900.0

temperature coefficent prediction =  3023.6067075053434

temperature                       =  24.6
```

Plot the results of the Kalman filter

Plot measured gas resistance versus corrected gas resistance (compensation of temperature and humidity interference)

In [15]:
```
import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize=(12, 12))
ax.plot(range(len(measurements)), measurements[:,0], label = 'Measurements gas resistance R_raw')
ax.plot(range(len(predictions)), compensated_gas_resistance[:], label = 'corrected Kalman Filter Pre
ax.legend()
plt.show()
```




```
## Plot alpha (temperature coefficient) and beta (rH coefficient) regression coefficients
```

In [16]:
```
import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize=(12, 12))
ax.plot(range(len(predictions)), np.array(states)[:,1], label = 'alpha (temperature coefficient)')
ax.plot(range(len(predictions)), np.array(states)[:,2], label = 'beta (relative humidity coefficient
ax.legend()
plt.show()
```

Regression results of the recursive minimum least-square error (LMMSE) Kalman filter

In [17]:
```
print("\nNumber of captured data points used for online regression using Kalman filter  = %11d" % le

print("\n\nLinear regression coefficient of temperature interference alpha_LMMSE          = %11.3lf"
print("Linear regression coefficient of relative humidity interference beta_LMMSE     = %11.3lf" % k
print("\n\n")
```

```
Number of captured data points used for online regression using Kalman filter  =        1226


Linear regression coefficient of temperature interference alpha_LMMSE          =   -3580.083
Linear regression coefficient of relative humidity interference beta_LMMSE     =   -6835.842
```

You are done! Congratulations!