Jupyter notebook for proving the Kalman filter online regression

First it creates synthesized raw gas resitance, temperature, absolute humidity time series with selectable parameters and linear dependencies:

- gas_resistance_compensated = f1(g0, dg, f_g0); f1=g0+dg*np.power(np.sin(df['index']f_g02*math.pi),3)
- temperature = f2(T0, dT, f_T0); f2=T0+dT*np.power(np.cos(df['index']f_T02*math.pi),5)
- absolute_humidity = f3(aH0, dH, f_aH0); f3=aH0+dH*np.power(np.sin(df['index']f_aH02*math.pi),7)

The synthesized gas_resistance_raw is then calculated by the following linear equation:

- gas_resistance_raw = gas_resistance_compensated + alpha * temperature + beta * absolute_humidity

In Python Pandas sytax this translates into

df["gas_resistance_raw"] = df["gas_resistance_compensated"] + alpha*df["temperature"] + beta*df["absolute_humidity"]

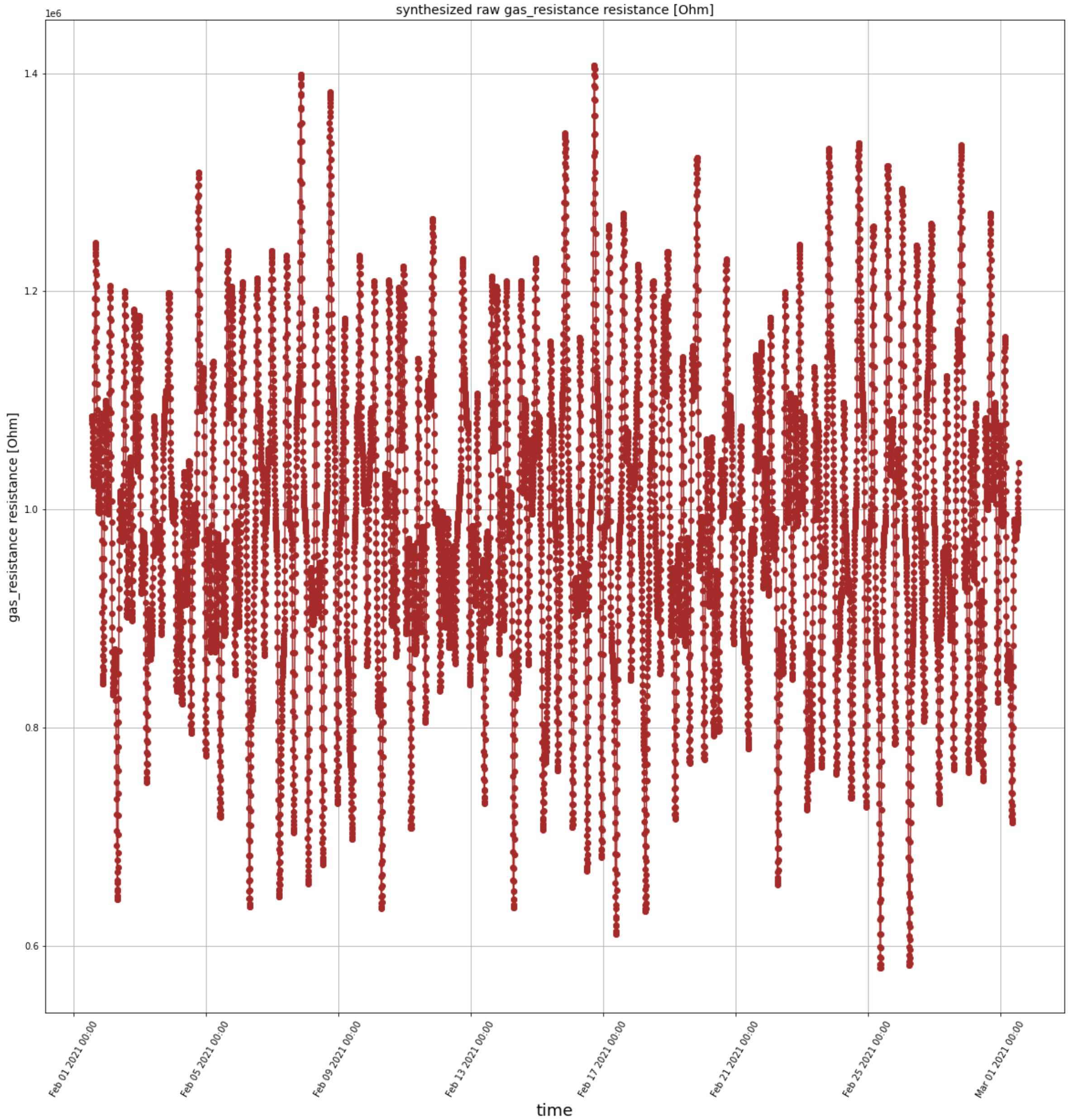
The synthesized triple ['gas_resistance_raw','temperature','absolute_humidity'] is then fed into the Kalman filter The Kalman filter is set up in such a way that is executing an online linear regression for estimation of the coefficients alpha and beta.

Please check at the 'Results' section at the bottom of the notebook the estimated alpha and beta coefficients.

There may be a small deviating between defined and estimated regression coefficients if the number of ventilation cycles is quite small, e.g. few days only. In reality we get one or two major ventilation cycles each day.

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
from datetime import datetime
import pandas as pd
import math
import matplotlib.dates as mdates
from matplotlib.dates import DateFormatter
from matplotlib.ticker import (MultipleLocator, FormatStrFormatter,
                               AutoMinorLocator)
from pandas.plotting import register_matplotlib_converters

register_matplotlib_converters()

number_of_days   = 28 # duration of the synthesized data set used for the online regi
time_step        = 4  # in minutes; sampling time of sensor

number_of_points = int(number_of_days * 24 * 60 / time_step) + 1  # cxalculated; do n

g0               = 250000          # base value of gas_resistance_raw
dg               = g0/2            # modulation of gas_resistance_raw
f_g0             = 1               # frequency of gas_resistance_raw in days

T0               = 23              # base value of temperature
dT               = 6.567           # modulation of temperature
f_T0             = 2.37            # frequency of temperature in days

aH0              = 9.3             # base value of absolute_humidity
dH               = 5.5             # modulation of absolute_humidity
f_aH0            = 2.2557          # frequency of absolute_humidity in days


# change the cooefficients 'alpha' and 'beta' here and check at the end of the notebo
alpha            = 15345            # linear dependency coefficient of temperature
beta             = 41080            # linear dependency coefficient of absolute_humi

fig, ax = plt.subplots(figsize=(20, 20))
plt.xticks(rotation=60)
ax.xaxis.set_major_formatter(DateFormatter('%b %d %Y %H:%M'))

# you may choose another start date in 'pd.date_range('2021-02-01 12:59:59.50', perio

start_date  = '2021-02-01 12:59:59.50'

df=pd.DataFrame({"time"                       : pd.date_range(start_date, periods=numb
                "index"                       : np.linspace(0,number_of_days, num=numb

df["index"] = df["index"].astype(np.float64)

df=pd.DataFrame({"time"                       : pd.date_range(start_date, periods=numb
                "index"                       : np.linspace(0,number_of_days, num=numb
                "gas_resistance_compensated" : g0+dg*np.power(np.sin(df['index']*f_g0
                "temperature"                 : T0+dT*np.power(np.cos(df['index']*f_T0
                "absolute_humidity"           : aH0+dH*np.power(np.sin(df['index']*f_a

df["gas_resistance_raw"] =  df["gas_resistance_compensated"] + alpha*df["temperature"

ax.plot_date(df['time'], df['gas_resistance_raw'], linestyle='solid', color='brown')
plt.title('synthesized raw gas_resistance resistance [Ohm]', fontsize=14)
plt.xlabel('time', fontsize=18)
plt.ylabel('gas_resistance resistance [Ohm]', fontsize=14)
plt.grid(True)

plt.show()
```



Multilinear Regression (MLR) for comparison

In [2]:

```python
from sklearn import linear_model
import statsmodels.api as sm

X = df[['temperature','absolute_humidity']] # here we have 2 variables for multiple r
Y = df['gas_resistance_raw']

# with sklearn
regr = linear_model.LinearRegression()
regr.fit(X, Y)

print('Intercept: \n', regr.intercept_)
print('Coefficients: \n', regr.coef_)

X = sm.add_constant(X)
model = sm.OLS(Y, X).fit()
predictions = model.predict(X)

print_model = model.summary()
print(print_model)
print(model.rsquared)

print("\n\n\nResults of multilinear regression (MLR):\n")
print("\n\nset temperature coefficent 'alpha' of synthesis               = %11.1lf" %
print("\ntemperature coefficent 'alpha' of MLR prediction              = %11.1lf" %
print("\nprediction error of MLR temperature coefficent 'alpha'        = %11.2lf %%"

print("\n\nset absolute humidity coefficent 'beta' of synthesis          = %11.1lf" %
print("\nabsolute humidity coefficent 'beta' of MLR prediction         = %11.1lf" %
print("\nprediction error of MLR absolute humditiy coefficent 'beta'   = %11.2lf %%"
print("\n")
```

```
Intercept: 
 252989.60151164245
Coefficients: 
 [15173.35598905 41183.18128458]
                            OLS Regression Results                            
==============================================================================
Dep. Variable:     gas_resistance_raw   R-squared:                       0.727
Model:                            OLS   Adj. R-squared:                  0.727
Method:                 Least Squares   F-statistic:                 1.340e+04
Date:                Sun, 04 Jun 2023   Prob (F-statistic):               0.00
Time:                        13:27:50   Log-Likelihood:            -1.2675e+05
No. Observations:               10081   AIC:                         2.535e+05
Df Residuals:                   10078   BIC:                         2.535e+05
Df Model:                           2                                         
Covariance Type:            nonrobust                                         
=====================================================================================
                        coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------
const               2.53e+05   5636.078     44.888      0.000    2.42e+05    2.64e+05
temperature        1.517e+04    213.843     70.956      0.000    1.48e+04    1.56e+04
absolute_humidity  4.118e+04    276.880    148.740      0.000    4.06e+04    4.17e+04
==============================================================================
Omnibus:                      505.605   Durbin-Watson:                   0.001
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              199.909
Skew:                          -0.000   Prob(JB):                     3.89e-44
Kurtosis:                       2.310   Cond. No.                         203.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specif
ied.
0.7267139975934515



Results of multilinear regression (MLR):



set temperature coefficent 'alpha' of synthesis               =     15345.0

temperature coefficent 'alpha' of MLR prediction              =     15173.4

prediction error of MLR temperature coefficent 'alpha'        =        1.12 %
```

```
set absolute humidity coefficent 'beta' of synthesis          =     41080.0

absolute humidity coefficent 'beta' of MLR prediction         =     41183.2

prediction error of MLR absolute humditiy coefficent 'beta'   =       -0.25 %
```

Please check whether the R-squared (uncentered) of the multiple linear regression above is sufficiently good (should be > 0.7): R-squared (also called coefficient of determination) is the portion of variance in the dependent variables that can be explained by the independent variables. Hence, as a rule of thumb for interpreting the strength of a relationship based on its R-squared value is:

```
if R-squared value < 0.3 this value is generally considered as None or very weak
effect size
if R-squared value 0.3 < r < 0.5 this value is generally considered as weak or lo
w effect size
if R-squared value 0.5 < r < 0.7 this value is generally considered as moderate e
ffect size
if R-squared value 0.7 < r < 1.0 this value is generally considered as strong eff
ect size
```

If R-squared value is < 0.3, the collected history may be too short. Please try to collect datapoints for a longer timeframe!

In [3]:
```
print("\n\nR-squared (uncentered) of the multiple linear regression      = %11.2lf\r
```

```
R-squared (uncentered) of the multiple linear regression      =        0.73
```

Kalman Filter

In [4]:
```
class KalmanFilter(object):
    def __init__(self, F = None, B = None, H = None, Q = None, R = None, P = None, x0

        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
        self.P = np.dot(np.dot(self.F, self.P), self.F.T) + self.Q      # Predicted
        return self.x

    def update(self, z):
        y = z - np.dot(self.H, self.x)                                  # Innovation
        S = self.R + np.dot(self.H, np.dot(self.P, self.H.T))           # Innovation
        #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 Ka
        #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)

```

In [5]:
```
my_observations = df[['gas_resistance_raw','temperature','absolute_humidity']]
my_observations.head()
```

Out[5]:

	gas_resistance_raw	temperature	absolute_humidity
0	1.085750e+06	29.567000	9.300000
1	1.085320e+06	29.538962	9.300000
2	1.084043e+06	29.455468	9.300000
3	1.081952e+06	29.318364	9.300002
4	1.079097e+06	29.130659	9.300013

In [6]:
```
list_of_rows = [list(row) for row in my_observations.values]
print(list_of_rows[:4])
```

```
[[1085749.615, 29.567, 9.3], [1085320.0299324945, 29.538961578920173, 9.3000000008048],
[1084043.4715432667, 29.45546783939964, 9.300000102457007], [1081952.2814001064, 29.318
363719058752, 9.300001734807228]]
```

In [7]:
```
np.array(list_of_rows)
measurements = np.array(list_of_rows)
print("number of measurement datapoints = ", len(measurements))
```

```
number of measurement datapoints =  10081
```

In [8]:

```
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 an ordinary
q0 = 0.0
Q = np.array([ [q0, q0, q0], [q0, q0, q0], [q0, q0, q0] ]).reshape(3, 3)
# set covariance of gas_resistancet resistance measurements also to a very small valu
R = np.array([ [0.0001] ]).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 =  [[0.0001]]
```

In [9]:

```
kf = KalmanFilter(F = F, H = H, Q = Q, R = R)
predictions = []
raw_gas=[]
compensated_gas_resistance_resistance=[]
states=[]

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

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

last_index = len(measurements)

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

```
dim measurements :  (10081, 3)
last index of measurement array =  10081
```

In [10]:

```
it = 0  # iteration index
#print("\nState vector kf.x= ", kf.x)
for z in measurements:
    zg = z[0] # raw_gas_resistance_resistance
    raw_gas.append(zg)
    # make observation model matrix state dependant
    H = np.array([[1, z[1], z[2]]]).reshape(1, 3)
    # z[1]: measured temperature
    # z[2]: calculated absolute humidity absolute_humidity(T, rH)
    # estimated state vector x:
    # x[0]: estimated VOC resistance
    # x[1]: estimated regression coefficient for T temperatureerature dependency
    # x[2]: estimated regression coefficient for absolute_humidity aboslute humidity
    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 == 1) or (it == last_index)):  # print results of first and last measure
        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("\nraw gas = ",zg)
        print ("\n\n")
    predictions.append(np.dot(H,  kf.predict()))

    compensated_gas_resistance_resistance.append(zg-kf.predict()[1,0]*z[1]-kf.predict
    #compensatedensated_gas_resistance_resistance.append(-kf.predict()[1,0]*z[1]-kf.
    #compensatedensated_gas_resistance_resistance.append(-kf.predict()[1,0]*z[1])
    #rint("\nraw gas_resistance resistance                = ",zg)
    #print("\ntemperatureerature coefficent prediction = ",kf.predict()[1,0])
    #print("\ntemperatureerature                        = ",z[1])
    #print("\ntemperatureerature compensatedensation          = ",-kf.predict()[1,0]
    #print("\nhumidity coefficent prediction    = ",kf.predict()[2,0])
    #print("\nabsolute humidity                 = ",z[2])
    #print("\nhumidity compensatedensation              = ",-kf.predict()[2,0]*z[2])
    #print("\nKalman state prediction          = ",kf.predict())
    #print("\ntemperature coefficent prediction = ",kf.predict()[1,0])
    #print("\ncompensatedensated gas_resistance resistance       = ",zg-kf.predict()
    states.append(kf.x)
    kf.update(zg)  #only zg raw_gas_resistance_resistance is an observation variable.
```

```
Iteration index =  1



State vector kf.x=  [[0.]
 [0.]
 [0.]]

Observation vector z =  [1.08574961e+06 2.95670000e+01 9.30000000e+00]

Observation transition matrix kf.H =  [[ 1.     29.567   9.3   ]]

Kalman filter prediction =  [[0.]
 [0.]
 [0.]]

Kalman filter update =  [[0.]]

raw gas =  1085749.615




Iteration index =  10081



State vector kf.x=  [[252989.43801795]
 [ 15173.35966598]
```

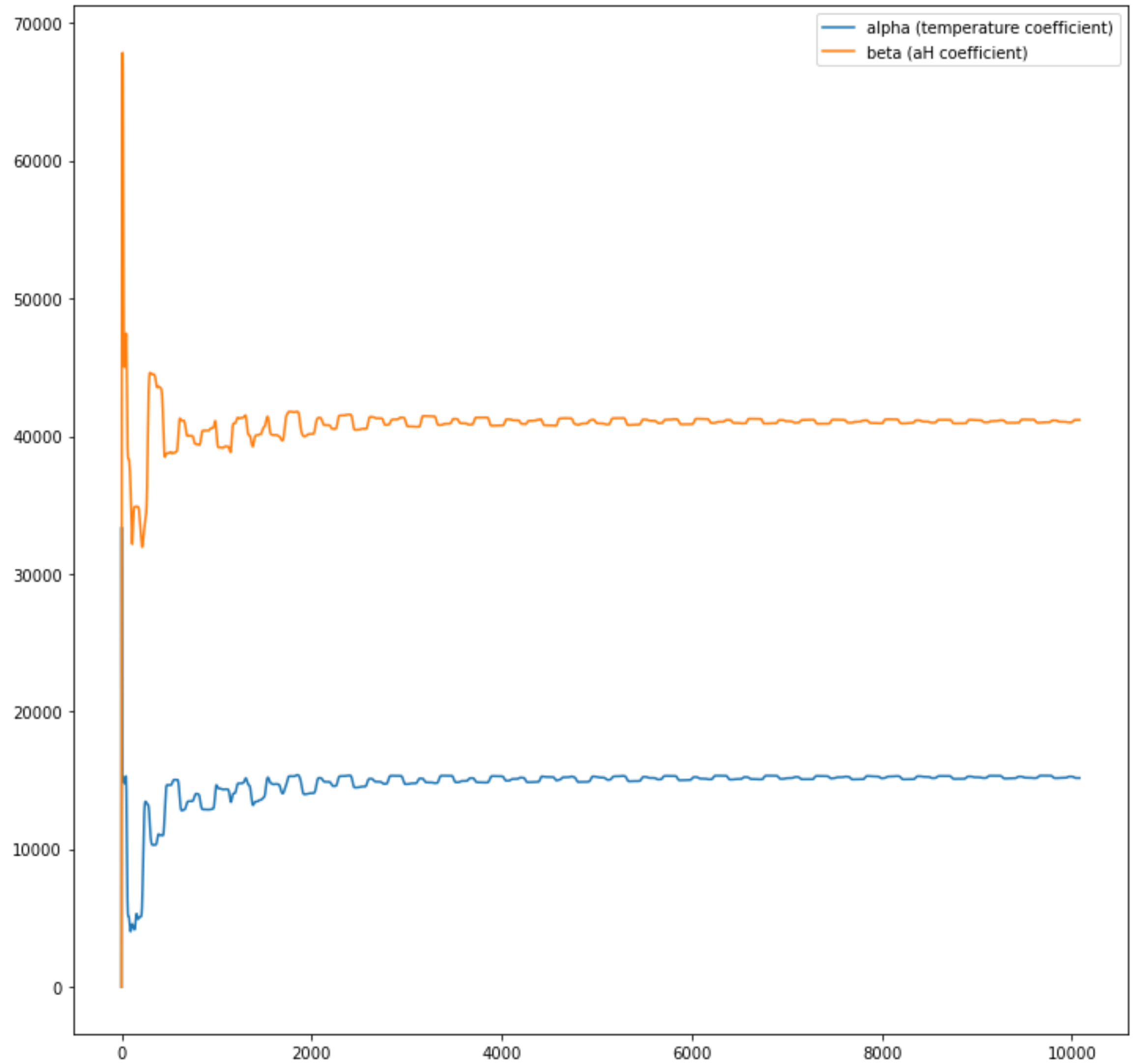
```
 [ 41183.19260825]]

Observation vector z =  [1.04272019e+06 2.23089507e+01 1.09637132e+01]

Observation transition matrix kf.H =  [[ 1.         22.30895069 10.96371322]]

Kalman filter prediction =  [[252989.43801795]
 [ 15173.35966598]
 [ 41183.19260825]]

Kalman filter update =  [[1043011.88385973]]

raw gas =  1042720.1874265161
```

Results

Plot alpha (temperature coefficient) and beta (aH coefficient) regression coefficients

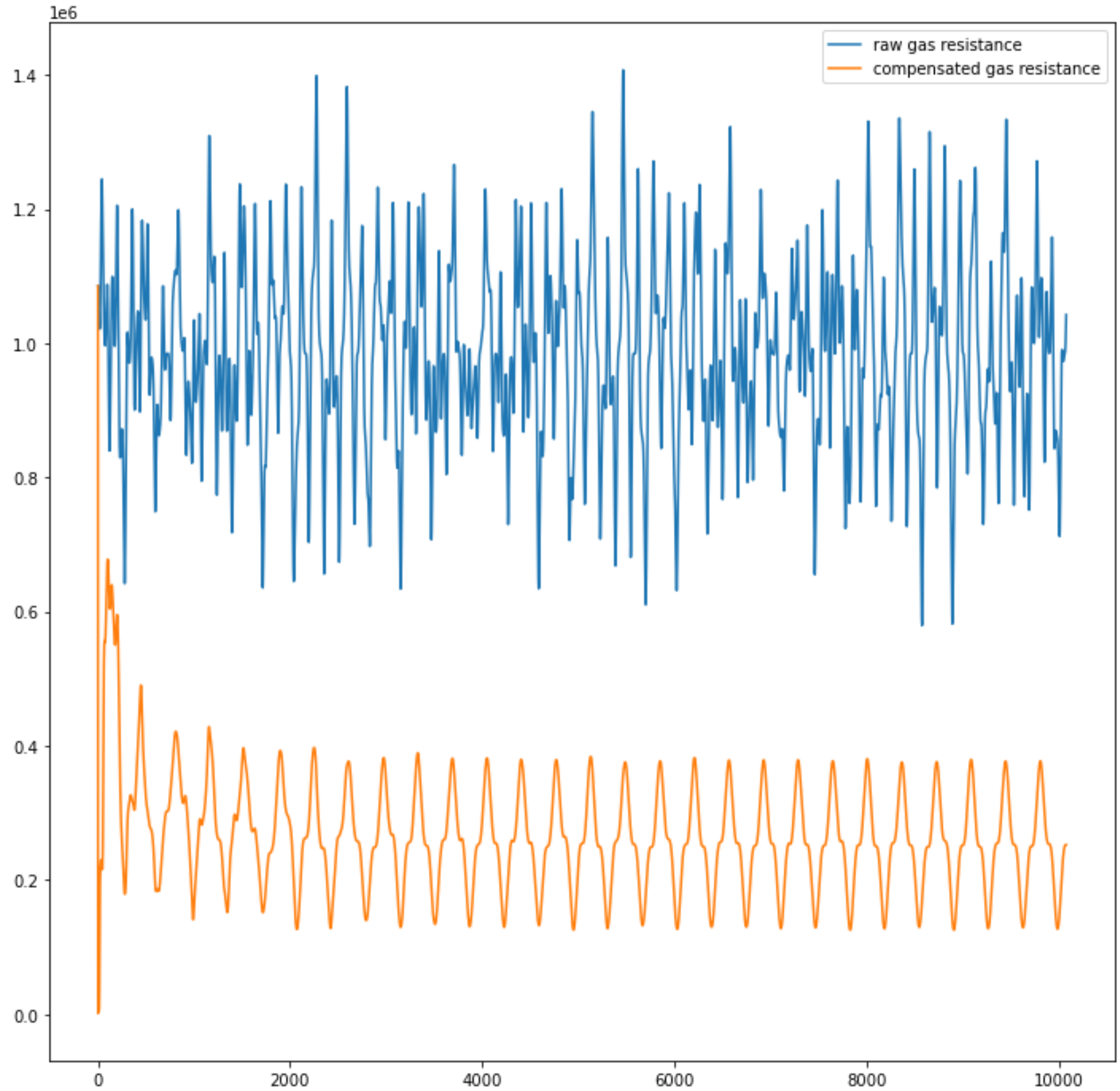
In [11]:

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



Plot compenasted gas resistance

In [12]:
```
import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize=(12, 12))
ax.plot(range(len(raw_gas)), np.array(raw_gas), label = 'raw gas resistance')
ax.plot(range(len(compensated_gas_resistance_resistance)), np.array(compensated_gas_r
ax.legend()
plt.show()
```



Summary of results

In [13]:
```
print("number of days of synthesized data = ", number_of_days)
print("number of measurement datapoints   = ", len(measurements))
```

```
number of days of synthesized data =  28
number of measurement datapoints   =  10081
```

Online regression with Kalman filter

In [14]:

```
print("\n\n\nResults of an online regression using a Kalman filter:\n")
print("\n\nset temperature coefficent 'alpha'                              = %11.1lf" % alp
print("\ntemperature coefficent 'alpha' prediction                      = %11.1lf" % kf.pr
print("\nprediction error of temperature coefficent 'alpha'             = %11.2lf %%" % ((
```

```
Results of an online regression using a Kalman filter:


set temperature coefficent 'alpha'                         =     15345.0

temperature coefficent 'alpha' prediction                  =     15173.4

prediction error of temperature coefficent 'alpha'         =        1.12 %
```

In [15]:

```
print("\n\nset absolute_humidity coefficent 'beta'                         = %11.1lf" % bet
print("\nabsolute_humidity coefficent 'beta' prediction                 = %11.1lf" % kf.pr
print("\nprediction error of absolute_humidity coefficent 'beta'        = %11.2lf %%" % ((
```

```
set absolute_humidity coefficent 'beta'                    =     41080.0

absolute_humidity coefficent 'beta' prediction             =     41183.2

prediction error of absolute_humidity coefficent 'beta'    =       -0.25 %
```

Classical multilinear regression (see above)

In [16]:

```
print("\n\n\nResults of multilinear regression (MLR):\n")
print("\n\nset temperature coefficent 'alpha' of synthesis                   = %11.1lf" %
print("\ntemperature coefficent 'alpha' of MLR prediction                  = %11.1lf" % r
print("\nprediction error of MLR temperature coefficent 'alpha'            = %11.2lf %%"

print("\n\nset absolute humidity coefficent 'beta' of synthesis              = %11.1lf" %
print("\nabsolute humidity coefficent 'beta' of MLR prediction             = %11.1lf" % r
print("\nprediction error of MLR absolute humditiy coefficent 'beta'       = %11.2lf %%"
print("\n")
```

```
Results of multilinear regression (MLR):


set temperature coefficent 'alpha' of synthesis                =     15345.0

temperature coefficent 'alpha' of MLR prediction               =     15173.4

prediction error of MLR temperature coefficent 'alpha'         =        1.12 %

set absolute humidity coefficent 'beta' of synthesis           =     41080.0

absolute humidity coefficent 'beta' of MLR prediction          =     41183.2

prediction error of MLR absolute humditiy coefficent 'beta'    =       -0.25 %
```

Outcomes:

1. The online regression with Kalman filter and the offline multilinear regression (MLR) are resulting in identical regression coefficients for the same synthesized data set
2. Both methods are quite accurately predicting the synthesized parameters if data squence is long enough, i.e. several (>>14) days
3. The Kalman filter can easily be realized in a micro controller since it requires to store just one state in the RAM

memory

Please play with the parameter 'number_of_days' above

Increase set value of 'number_of_days above' and check the influence on the accuracy of the estimation.

- We can see that we need at least 14 days in order to get a reasonably low error of the estimation.
- After 4 days the estimation error is still significantly high!
- The estimation accuracy will improve when the Kalman filter will run for a longer time, e.g. for >> 14 days
- Since we have an online regression, the Kalman filter will be able to adapt to slowly changing paramaters, e.g. in the different seasons or aging of the sensor

Done