CALIBRATION PARAMETERS IMU

Panteleimon Manouselis, p.manouselis@student.utwente.nl, s3084493, M-ROB

I. INTRODUCTION

In this report, we present our analysis of IMU data to calculate the bias offset and scale factor error for the accelerometer and gyroscope. The goal of this analysis was to improve the accuracy of the IMU by identifying any errors in the accelerometer and gyroscope measurements and then recommending calibration parameters that should be used for the IMU to minimize those errors. Furthermore, outlier removal was applied as a preprocessing step to improve the quality of our data. We also aimed to estimate the run-to-run bias and scale factor instability. Based on our analysis, we provide recommendations for the calibration parameters (bias offset and scale factor error), which will be used to initialize the IMU data for the group work.

II. METHODS AND RESULTS

The first step was data preprocessing. The Z-scores outlier removal technique was employed and data points that were significantly different from the dataset were removed. The threshold was set to 2.5 by trial-and-error and the Z-score for each data point was computed by the following equation:

$$Z = \frac{data - \mu}{\sigma}$$

where μ is the mean value and σ is the standard deviation of the channel. The figure below shows an accelerometer channel before and after applying data preprocessing.

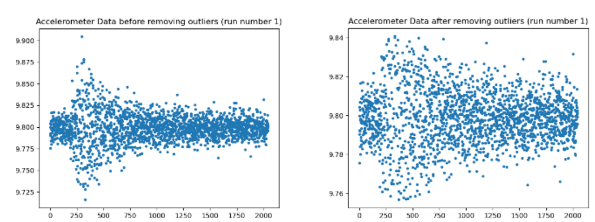

Figure 1: Accelerometer data before (left) and after (right) removing the outliers

To calculate the bias offset and scale factor error for the accelerometer and gyroscope, a six-position static test [1] was repeated five times and a series of IMU data was collected. To calculate the errors, we applied algorithm 1 (see APPENDIX A) to each run. For each run, algorithm 1 returns two vectors (b_a, b_g) and two matrices (S_a, S_g) which have the following form:

$$bias = \begin{bmatrix} b_X \\ b_Y \\ b_Z \end{bmatrix} \qquad S = \begin{bmatrix} S_X & 0 & 0 \\ 0 & S_Y & 0 \\ 0 & 0 & S_Z \end{bmatrix}$$

Indicatively, for experiment number 2 the calibration parameters (biases and scale factors) are presented:

$$b_a = \begin{bmatrix} 0.007536 \\ 0.009307 \\ -0.012125 \end{bmatrix}, S_a = \begin{bmatrix} 0.000413 & 0 & 0 \\ 0 & -0.000412 & 0 \\ 0 & 0 & 0.000415 \end{bmatrix}$$

$$b_g = \begin{bmatrix} -0.003720 \\ 0.000572 \\ -0.001235 \end{bmatrix}, S_g = \begin{bmatrix} -0.971483 & 0 & 0 \\ 0 & -0.995438 & 0 \\ 0 & 0 & -1.003476 \end{bmatrix}$$

Note: One can find the calculated calibration parameters, for each of the five experiments, by running the provided python script (see APPENDIX B). However, the above parameters are the ones I recommend using when initializing the IMU data for the group work. The equation we used to model the angular rate is:

$$\tilde{\omega} = \omega + b_g + S\omega + \varepsilon_g$$

where $\tilde{\omega}$ is the gyroscope measurement vector (deg/h), ω is the true angular rate velocity vector (deg/h), and ε_g is the gyro sensor noise. Notice that the matrix representing the gyro scale factor has values in its principal diagonal that are close to -1. Therefore, the terms ω and $S\omega$ almost cancel each other completely and our equation *in this case* can be simplified to $\tilde{\omega} \approx b_g + \varepsilon_g$. We are therefore, making no useful measurements with our gyroscope as we are mainly measuring errors and noise instead of the true angular rate velocity of the earth. One can therefore easily make two observations:

- Our gyroscope cannot detect the Earth’s rotation rate.
- Noise dominates our gyro measurements.

The fact that our gyro did not measure the earth’s rotation can be also seen in the data collected during the six-position static test. When the sensitive axis alternates between up and down we would normally expect a change of sign between consecutive angular rate measurements. However, no change of sign occurs for any of the X, Y, Z axis which indicates that the Earth’s rotation rate is not measured. In order to calibrate a gyroscope that cannot detect the Earth’s rotation rate, one should position the IMU on a table and rotate it at a constant rate of ω_t. Then the $\omega_e \sin(\phi)$ term is replaced with ω_t in the equation used for the scale factor error. This way one can calculate an improved approximation of the scale factor error (which is currently incorrectly calculated).

To estimate the run-to-run bias, we calculated the difference between consecutive bias offset vectors. Similarly, to evaluate the scale factor instability we computed the difference between consecutive scale factor error matrices. Indicatively, we present below the run-to-run bias and scale factor instability for both accelerometer and gyroscope data, between experiment 2 and experiment 3 (recommended calibration parameters).

$$b_a = \begin{bmatrix} -0.005349 \\ 0.001647 \\ -0.0001 \end{bmatrix}, S_a = \begin{bmatrix} 0.000053 & 0 & 0 \\ 0 & 0.000014 & 0 \\ 0 & 0 & 0.000028 \end{bmatrix}$$

$$b_g = \begin{bmatrix} -0.000048 \\ 0.000076 \\ -0.000222 \end{bmatrix}, S_g = \begin{bmatrix} -0.02887 & 0 & 0 \\ 0 & 0.00501 & 0 \\ 0 & 0 & 0.020748 \end{bmatrix}$$

APPENDIX A

BIAS OFFSET AND SCALE FACTOR ERROR ALGORITHM

Algorithm 1: Bias offset and scale factor error

Input:

$D = d_1, d_2, \dots, d_6$ Six files containing data from the six-position static test

Output:

b_a Accelerometer bias offset vector
s_a Accelerometer scale factor error matrix
b_g Gyroscope bias offset vector
s_g Gyroscope scale factor error matrix

Algorithm:

repeat three times:

1. Read two consecutive files and write them into dataframes
2. Select the sensitive axis (one channel from each dataframe) and calculate it's mean value.
3. Use equation $b_a = \frac{f_{up}+f_{down}}{2}$ to calculate the bias offset values of the accelerometer.
4. Use equation $S_a = \frac{f_{up}-f_{down}-2g}{2g}$ to calculate the scale factor values of the accelerometer.
5. Use equation $b_g = \frac{\omega_{up}+\omega_{down}}{2}$ to calculate the bias offset values of the gyroscope.
6. Use equation $S_g = \frac{\omega_{up}-\omega_{down}-2\omega_e \sin(\phi)}{2\omega_e si\, n(\phi)}$ to calculate the scale factor values of the gyroscope.
7. Store the values appropriately (list for biases and matrices for scale factor's

APPENDIX B

BIAS OFFSET AND SCALE FACTOR ERROR FOR EACH RUN

Run 1:

$$b_a = \begin{bmatrix} 0.005546 \\ 0.009344 \\ -0.012661 \end{bmatrix}, S_a = \begin{bmatrix} 0.000472 & 0 & 0 \\ 0 & -0.000427 & 0 \\ 0 & 0 & 0.000504 \end{bmatrix}$$

$$b_g = \begin{bmatrix} -0.003669 \\ 0.000563 \\ -0.001267 \end{bmatrix}, S_g = \begin{bmatrix} -0.989353 & 0 & 0 \\ 0 & -0.989081 & 0 \\ 0 & 0 & -0.989407 \end{bmatrix}$$

Run 2:

$$b_a = \begin{bmatrix} 0.007536 \\ 0.009307 \\ -0.012125 \end{bmatrix}, S_a = \begin{bmatrix} 0.000413 & 0 & 0 \\ 0 & -0.000412 & 0 \\ 0 & 0 & 0.000415 \end{bmatrix}$$

$$b_g = \begin{bmatrix} -0.003720 \\ 0.000572 \\ -0.001235 \end{bmatrix}, S_g = \begin{bmatrix} -0.971483 & 0 & 0 \\ 0 & -0.995438 & 0 \\ 0 & 0 & -1.003476 \end{bmatrix}$$

Run 3:

$$b_a = \begin{bmatrix} 0.002186 \\ 0.010954 \\ -0.012225 \end{bmatrix}, S_a = \begin{bmatrix} 0.000466 & 0 & 0 \\ 0 & -0.000398 & 0 \\ 0 & 0 & 0.000444 \end{bmatrix}$$

$$b_g = \begin{bmatrix} -0.003768 \\ 0.000648 \\ -0.001457 \end{bmatrix}, S_g = \begin{bmatrix} -1.000354 & 0 & 0 \\ 0 & -0.990428 & 0 \\ 0 & 0 & -0.982728 \end{bmatrix}$$

Run 4:

$$b_a = \begin{bmatrix} 0.003551 \\ 0.008711 \\ -0.009682 \end{bmatrix}, S_a = \begin{bmatrix} 0.000397 & 0 & 0 \\ 0 & -0.000462 & 0 \\ 0 & 0 & 0.000418 \end{bmatrix}$$

$$b_g = \begin{bmatrix} -0.003694 \\ 0.000530 \\ -0.001489 \end{bmatrix}, S_g = \begin{bmatrix} -1.00879 & 0 & 0 \\ 0 & -1.00375 & 0 \\ 0 & 0 & -0.98219 \end{bmatrix}$$

Run 5:

$$b_a = \begin{bmatrix} 0.002724 \\ 0.009674 \\ -0.012367 \end{bmatrix}, S_a = \begin{bmatrix} 0.000459 & 0 & 0 \\ 0 & -0.000459 & 0 \\ 0 & 0 & 0.000388 \end{bmatrix}$$

$$b_g = \begin{bmatrix} -0.003702 \\ 0.000610 \\ -0.001496 \end{bmatrix}, S_g = \begin{bmatrix} -0.984771 & 0 & 0 \\ 0 & -1.003194 & 0 \\ 0 & 0 & -0.992422 \end{bmatrix}$$

RUN-TO-RUN BIAS AND SCALE FACTOR INSTABILITY BETWEEN RUN 2,3 AND 4

Accelerometer (between run 2 and 3)

	X	Y	Z
Bias	-0.005349	0.001647	-0.0001
Scale	0.000053	0.000014	0.000028

Gyroscope (between run 2 and 3)

	X	Y	Z
Bias	-0.000048	0.000076	-0.000222
Scale	-0.02887	0.00501	0.020748

Accelerometer (between run 3 and 4)

	X	Y	Z
Bias	0.001364	-0.002243	0.002543
Scale	-0.000069	-0.000064	-0.000026

Gyroscope (between run 3 and 4)

	X	Y	Z
Bias	0.000074	-0.000118	-0.000032

Scale	-0.008436	-0.013322	0.000538

APPENDIX C

PYTHON SCRIPT

```
# import warnings to supress warning caused by bug in catboost 1.0.6
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt

import numpy as np
import os
import pandas as pd
import scipy.constants as sc
from math import sin
from math import radians

# Constants
lat = 52.239 # latitude of citadel where data was recorded

# Path were the .txt files are saved
desktop = os.path.join(os.path.join(os.environ['USERPROFILE']), 'Desktop')
bd = str(desktop)+'/RPCN imu calibration txt/'

# Going to that directory
os.chdir(bd)

# Listing all the files in the directory (30 txt files)
entries = os.listdir(bd)


# Preprocessing
# Remove outliers from data by using the Z-Score method
def detect_and_remove_outliers(data, threshold=2.5):
  # Calculate the mean and standard deviation of the data
  mean = np.mean(data)
  std = np.std(data)
  # Calculate the Z-scores of the data
  z_scores = (data - mean) / std
  # Identify the data points with a Z-score above the threshold
  outliers = np.abs(z_scores) > threshold
  # Remove the outliers from the data
  cleaned_data = data[~outliers]
  return cleaned_data

# For accelerometer
bias_error_list_acc = []
scale_error_list_acc = []

# For gyro
bias_error_list_gyro = []
scale_error_list_gyro = []
for i in range(0, 30, 6):
    # Axis 1
    # txt to array
    df1 = pd.read_csv(entries[i], sep="    ", header=0, skiprows=12, usecols=[0, 1, 2, 3, 4, 8, 9, 10, 
14, 15, 16])
    # Visualize the data before removing the outliers
    plt.figure()
    plt.plot(df1["Acc_Z"], '.')
    plt.title('Accelerometer Data before removing outliers (run number ' +str(i%5 +1)+')')
```

```
    # Remove the outliers from the data
    df1["Acc_Z"] = detect_and_remove_outliers(df1["Acc_Z"])

    # Visualize the data after removing the outliers
    plt.figure()
    plt.plot(df1["Acc_Z"], '.')
    plt.title('Accelerometer Data after removing outliers (run number ' +str(i%5 +1)+')')

    mean_Acc_Z = df1["Acc_Z"].mean() # F_up_Z

    # Second file
    df2 = pd.read_csv(entries[i+1], sep="  ", header=0, skiprows=12, usecols=[0, 1, 2, 3, 4, 8, 9, 10,
14, 15, 16])
    df2["Acc_Z"] = detect_and_remove_outliers(df2["Acc_Z"])
    mean_Acc_Z_2 = df2["Acc_Z"].mean() # F_down_Z

    # Calculating Systematic Bias offset
    bias_error_Z = (mean_Acc_Z + mean_Acc_Z_2)/2

    # Calculating Systematic Scale Factor Error
    scale_error_Z = (mean_Acc_Z - mean_Acc_Z_2 - 2*sc.g)/(2*sc.g)


    # Axis 2
    df1 = pd.read_csv(entries[i+2], sep="  ", header=0, skiprows=12, usecols=[0, 1, 2, 3, 4, 8, 9, 10,
14, 15, 16])
    df1["Acc_Y"] = detect_and_remove_outliers(df1["Acc_Y"])
    mean_Acc_Y = df1["Acc_Y"].mean() # F_up_Z

    df2 = pd.read_csv(entries[i+3], sep="  ", header=0, skiprows=12, usecols=[0, 1, 2, 3, 4, 8, 9, 10,
14, 15, 16])
    df2["Acc_Y"] = detect_and_remove_outliers(df2["Acc_Y"])
    mean_Acc_Y_2 = df2["Acc_Y"].mean() # F_down_Z

    # Calculating Systematic Bias offset
    bias_error_Y = (mean_Acc_Y + mean_Acc_Y_2)/2

    # Calculating Systematic Scale Factor Error
    scale_error_Y = (mean_Acc_Y - mean_Acc_Y_2 - 2*sc.g)/(2*sc.g)


    # Axis 3
    df1 = pd.read_csv(entries[i+4], sep="  ", header=0, skiprows=12, usecols=[0, 1, 2, 3, 4, 8, 9, 10,
14, 15, 16])
    df1["Acc_X"] = detect_and_remove_outliers(df1["Acc_X"])
    mean_Acc_X = df1["Acc_X"].mean() # F_up_Z

    df2 = pd.read_csv(entries[i+5], sep="  ", header=0, skiprows=12, usecols=[0, 1, 2, 3, 4, 8, 9, 10,
14, 15, 16])
    df2["Acc_X"] = detect_and_remove_outliers(df2["Acc_X"])
    mean_Acc_X_2 = df2["Acc_X"].mean() # F_down_Z

    # Calculating Systematic Bias offset
    bias_error_X = (mean_Acc_X + mean_Acc_X_2)/2

    # Calculating Systematic Scale Factor Error
    scale_error_X = (mean_Acc_X - mean_Acc_X_2 - 2*sc.g)/(2*sc.g)

    # Adding erros in vectors
    bias_error_acc = pd.Series(data=[bias_error_X, bias_error_Y, bias_error_Z], index=["X", "Y", "Z"])
    bias_error_list_acc.append(bias_error_acc)
    scale_error_acc = pd.Series(data=[scale_error_X, scale_error_Y, scale_error_Z], index=["X", "Y",
"Z"])
    scale_error_list_acc.append(scale_error_acc)



    # For gyro
    # Axis 1
    df1 = pd.read_csv(entries[i], sep="    ", header=0, skiprows=12, usecols=[0, 1, 2, 3, 4, 8, 9, 10,
14, 15, 16])
```

```
    df1["Gyr_Z"] = detect_and_remove_outliers(df1["Gyr_Z"])
    mean_Gyr_Z = df1["Gyr_Z"].mean() # F_up_Z

    df2 = pd.read_csv(entries[i+1], sep="  ", header=0, skiprows=12, usecols=[0, 1, 2, 3, 4, 8, 9, 10,
14, 15, 16])
    df2["Gyr_Z"] =detect_and_remove_outliers(df2["Gyr_Z"])
    mean_Gyr_Z_2 = df2["Gyr_Z"].mean() # F_down_Z

    # Calculating Systematic Bias offset
    bias_error_Z_gyr = (mean_Gyr_Z + mean_Gyr_Z_2)/2

    # Calculating Systematic Scale Factor Error
    scale_error_Z_gyr = (mean_Gyr_Z - mean_Gyr_Z_2 -
2*(15.04/3600)*sin(radians(lat)))/(2*(15.04/3600)*sin(radians(lat)))


    # Axis 2
    df1 = pd.read_csv(entries[i+2], sep="  ", header=0, skiprows=12, usecols=[0, 1, 2, 3, 4, 8, 9, 10,
14, 15, 16])
    df1["Gyr_Y"] = detect_and_remove_outliers(df1["Gyr_Y"])
    mean_Gyr_Y = df1["Gyr_Y"].mean() # F_up_Z

    df2 = pd.read_csv(entries[i+3], sep="  ", header=0, skiprows=12, usecols=[0, 1, 2, 3, 4, 8, 9, 10,
14, 15, 16])
    df2["Gyr_Y"] = detect_and_remove_outliers(df2["Gyr_Y"])
    mean_Gyr_Y_2 = df2["Gyr_Y"].mean() # F_down_Z

    # Calculating Systematic Bias offset
    bias_error_Y_gyr = (mean_Gyr_Y + mean_Gyr_Y_2)/2

    # Calculating Systematic Scale Factor Error
    scale_error_Y_gyr = (mean_Gyr_Y - mean_Gyr_Y_2 -
2*(15.04/3600)*sin(radians(lat)))/(2*(15.04/3600)*sin(radians(lat)))


    # Axis 3
    df1 = pd.read_csv(entries[i+4], sep="  ", header=0, skiprows=12, usecols=[0, 1, 2, 3, 4, 8, 9, 10,
14, 15, 16])
    df1["Gyr_X"] = detect_and_remove_outliers(df1["Gyr_X"])
    mean_Gyr_X = df1["Gyr_X"].mean() # F_up_Z

    df2 = pd.read_csv(entries[i+5], sep="  ", header=0, skiprows=12, usecols=[0, 1, 2, 3, 4, 8, 9, 10,
14, 15, 16])
    df2["Gyr_X"] = detect_and_remove_outliers(df2["Gyr_X"])
    mean_Gyr_X_2 = df2["Gyr_X"].mean() # F_down_Z

    # Calculating Systematic Bias offset
    bias_error_X_gyr = (mean_Gyr_X + mean_Gyr_X_2)/2

    # Calculating Systematic Scale Factor Error
    scale_error_X_gyr = (mean_Gyr_X - mean_Gyr_X_2 -
2*(15.04/3600)*sin(radians(lat)))/(2*(15.04/3600)*sin(radians(lat)))

    # Adding erros in vectors
    bias_error_gyr = pd.Series(data=[bias_error_X_gyr, bias_error_Y_gyr, bias_error_Z_gyr], index=["X",
"Y", "Z"])
    bias_error_list_gyro.append(bias_error_gyr)
    scale_error_gyr = pd.Series(data=[scale_error_X_gyr, scale_error_Y_gyr, scale_error_Z_gyr],
index=["X", "Y", "Z"])
    scale_error_list_gyro.append(scale_error_gyr)


# Create a DataFrame from the lists
print("bias error Accelerometer")
print(bias_error_list_acc)
print("\n")

print("scale factor error Accelerometer")
print(scale_error_list_acc)
print("\n")
print("bias error Gyroscope")
```

```
print(bias_error_list_gyro)
print("\n")
print("scale factor error Gyroscope")
print(scale_error_list_gyro)
print("\n")

# Run-to-run bias offset
run_to_run_bias_acc = pd.DataFrame(columns=["X", "Y", "Z"], index=[1, 2, 3, 4])
run_to_run_bias_gyro = pd.DataFrame(columns=["X", "Y", "Z"], index=[1, 2, 3, 4])

# Run-to-run scale factor (Scale factor instability)
Scale_factor_instability_acc = pd.DataFrame(columns=["X", "Y", "Z"], index=[1, 2, 3, 4])
Scale_factor_instability_gyro = pd.DataFrame(columns=["X", "Y", "Z"], index=[1, 2, 3, 4])

for i in range(len(bias_error_list_acc) - 1):
    # X - axis
    run_to_run_bias_acc.iloc[i, 0] = bias_error_list_acc[i + 1][0] - bias_error_list_acc[i][0]
    run_to_run_bias_gyro.iloc[i, 0] = bias_error_list_gyro[i+1][0]-bias_error_list_gyro[i][0]

    Scale_factor_instability_acc.iloc[i, 0] = scale_error_list_acc[i + 1][0] -
scale_error_list_acc[i][0]
    Scale_factor_instability_gyro.iloc[i, 0] = scale_error_list_gyro[i+1][0]-scale_error_list_gyro[i][0]

    # Y - axis
    run_to_run_bias_acc.iloc[i, 1] = bias_error_list_acc[i + 1][1] - bias_error_list_acc[i][1]
    run_to_run_bias_gyro.iloc[i, 1] = bias_error_list_gyro[i + 1][1] - bias_error_list_gyro[i][1]

    Scale_factor_instability_acc.iloc[i, 1] = scale_error_list_acc[i + 1][1] -
scale_error_list_acc[i][1]
    Scale_factor_instability_gyro.iloc[i, 1] = scale_error_list_gyro[i + 1][1] -
scale_error_list_gyro[i][1]

    # Z - axis
    run_to_run_bias_acc.iloc[i, 2] = bias_error_list_acc[i + 1][2] - bias_error_list_acc[i][2]
    run_to_run_bias_gyro.iloc[i, 2] = bias_error_list_gyro[i + 1][2] - bias_error_list_gyro[i][2]

    Scale_factor_instability_acc.iloc[i, 2] = scale_error_list_acc[i + 1][2] -
scale_error_list_acc[i][2]
    Scale_factor_instability_gyro.iloc[i, 2] = scale_error_list_gyro[i + 1][2] -
scale_error_list_gyro[i][2]

# We only use data from runs (2, 3 and 4)
run_to_run_bias_acc = run_to_run_bias_acc.iloc[[1,2]]
run_to_run_bias_gyro = run_to_run_bias_gyro.iloc[[1,2]]

Scale_factor_instability_acc = Scale_factor_instability_acc.iloc[[1,2]]
Scale_factor_instability_gyro = Scale_factor_instability_gyro.iloc[[1,2]]

print("run to run bias error Accelerometer")
print(run_to_run_bias_acc)
print("\n\n")

print("run to run bias error Gyroscope")
print(run_to_run_bias_gyro)
print("\n\n")

print("Scale factor instability Accelerometer")
print(Scale_factor_instability_acc)
print("\n\n")

print("Scale factor instability Gyroscope")
print(Scale_factor_instability_gyro)
print("\n\n")

# Define a function to calculate the non-orthogonality matrix of a gyroscope
def calc_non_orthogonality_matrix(measurements):
  # Extract the measured outputs for each axis
  x = measurements[0]
  y = measurements[1]
  z = measurements[2]
```

```
  # Calculate the non-orthogonality matrix using the formula above
  N = [[(x**2 + y**2 + z**2 - x*y - x*z - y*z) / 2, (y*z - x**2) / 2, (x*z - y**2) / 2],
       [(y*z - x**2) / 2, (x**2 + y**2 + z**2 - x*y - x*z - y*z) / 2, (x*y - z**2) / 2],
       [(x*z - y**2) / 2, (x*y - z**2) / 2, (x**2 + y**2 + z**2 - x*y - x*z - y*z) / 2]]

  return N

plt.show()

x = 0
while x < 1 or x > 5:
    print("For what run of the experiment (1-5) should I calculate the calibration parameters?")
    x = input()
    x = int(x)

i = (x-1)*6
df1 = pd.read_csv(entries[i], sep="    ", header=0, skiprows=12, usecols=[0, 1, 2, 3, 4, 8, 9, 10, 14,
15, 16])
gyr_z_mean = df1["Gyr_Z"].mean()  # F_up_Z
acc_z_mean = df1["Acc_Z"].mean()

df2 = pd.read_csv(entries[i + 2], sep="    ", header=0, skiprows=12, usecols=[0, 1, 2, 3, 4, 8, 9, 10,
14, 15, 16])
gyr_y_mean = df2["Gyr_Y"].mean() # F_up_Y
acc_y_mean = df2["Acc_Y"].mean()

df3 = pd.read_csv(entries[i+4], sep="  ", header=0, skiprows=12, usecols=[0, 1, 2, 3, 4, 8, 9, 10, 14,
15, 16])
gyr_x_mean = df3["Gyr_X"].mean()  # F_up_X
acc_x_mean = df3["Acc_X"].mean()

measurements_gyro = [gyr_x_mean, gyr_y_mean, gyr_z_mean]
non_orthogonality_matrix_gyr = calc_non_orthogonality_matrix(measurements_gyro)
measurements_acc = [acc_x_mean, acc_y_mean, acc_z_mean]
non_orthogonality_matrix_acc = calc_non_orthogonality_matrix(measurements_acc)

# Final calibration parameters
# Gyro
print("\nCalibration parameters for gyroscope should be:")
print("Instrument Bias Error (used run number " + str(x) + ")")
print(bias_error_list_gyro[x-1])
print("\nGyroscope Scale Error (used run number " + str(x) + ")")
print(scale_error_list_gyro[x-1])
print("\nNon-orthogonality matrix of gyroscope (used run number " + str(x) + ")")
print(non_orthogonality_matrix_gyr)

# Accelerometer
print("\nCalibration parameters for accelerometer should be:")
print("Instrument Bias Error (used run number " + str(x) + ")")
print(bias_error_list_acc[1])
print("\nAccelerometer Scale Error (used run number " + str(x) + ")")
print(scale_error_list_acc[1])
print("\nNon-orthogonality matrix of accelerometer (used run number " + str(x) + ")")
print(non_orthogonality_matrix_acc)
```

References

[1] A. Noureldin, T. B. Karamat and J. Georgy, Fundamentals of Inertial Navigation, Satellite-based Positioning and their Integration, London : Springer, 2013.