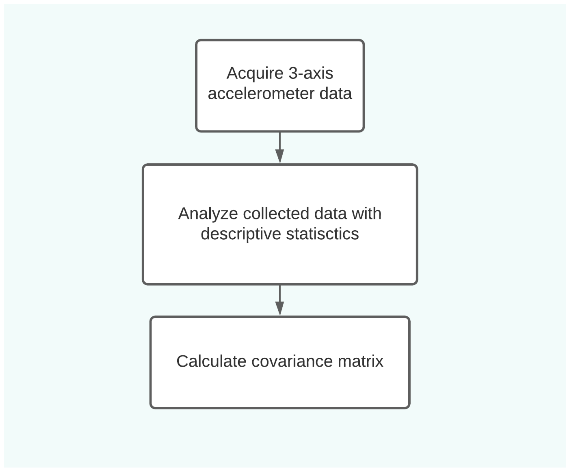
IMU Covariance Matrix - Assignment 2

Dimitriy Georgiev

March 27, 2021

1 Assignment Description

The assignment requires acquisition of data from a physical 3-axis accelerometer and calculating the associated covariance matrix based on the sensor readings.


Acquire 3-axis accelerometer data
Analyze collected data with descriptive statisctics
Calculate covariance matrix

2 Procedure

To collect the necessary data, I used a breakout board consisting of the LSM6DS3 6DoF inertial measurement unit from STMicroelectronics The setup is seen in Figure 1.

To communicate with the IMU, I used **I2C** digital interface. The sensor was configured with full-scale acceleration range of **±8 g** and output data rate of **104 Hz** according to the datasheet. (These settings were chosen in order to acquire more noise in the readings, otherwise noise is negligble if the scale is set in lower ranges). On the other hand, the sensor was by default in high-perfomance mode which implied that an internal anti-aliasing (low pass) filter was enabled. (These settings can be changed in the CTRL1_XL and CTRL7_G registers of the sensor). To configure the registers, I used SparkFun's LSM6DS3 Arduino Library. Finally, the microcontroller sampled the sensor at **100 Hz** and transmitted the acquired readings over USB to a computer where the values were stored in a .csv file via Python script. In total, 2000 values were recorded for each measurement axis of the accelerometer.

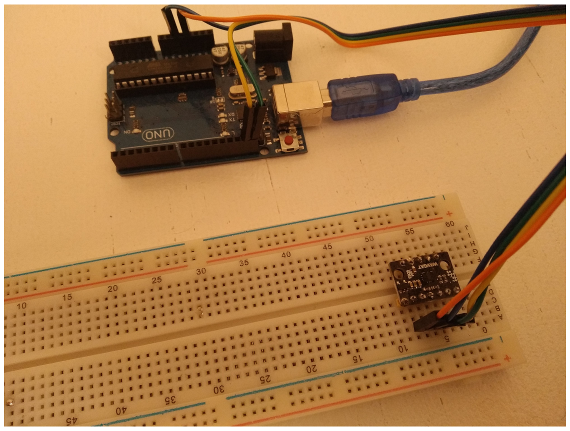


Figure 1. Setup - Arduino Uno and LSM6DS3 breakout board.

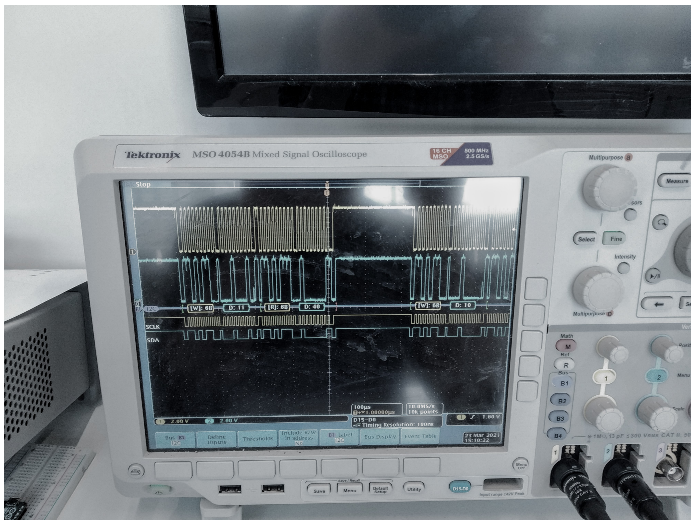


Figure 2. Used logic analyzer to practice and decode I2C.

3 Implementation

3.1 Include Libraries

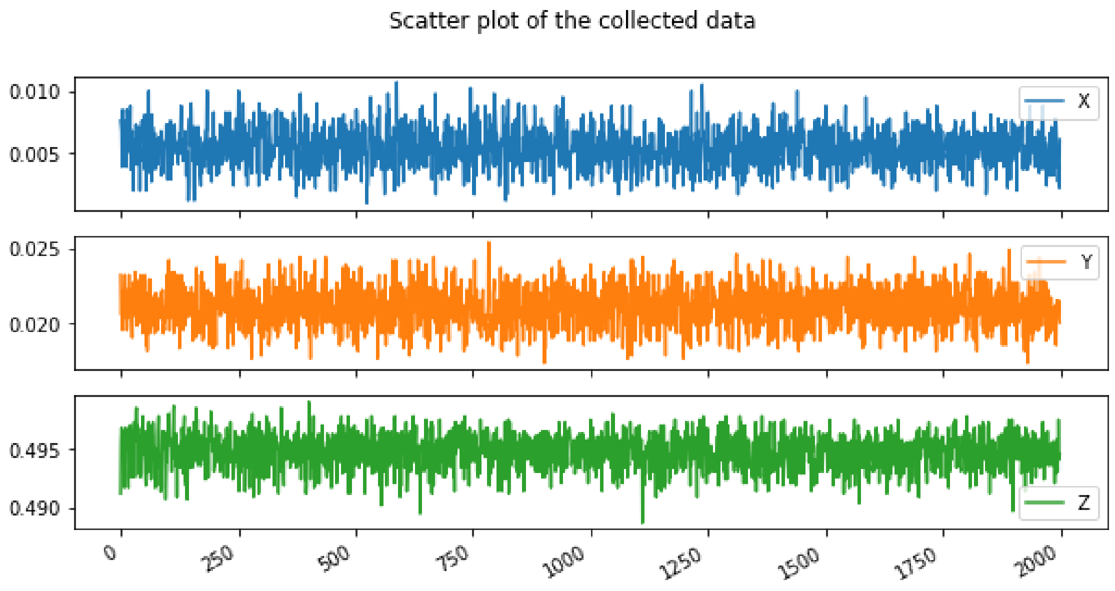
[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

3.2 Import Data with Pandas

[2]:
```
imu_table = pd.read_csv("data/imu_data_combined.csv")

# Display last 10 values in the dataset
imu_table.tail(10)
```

```
[2]:            X       Y       Z
     1990  0.0039  0.0188  0.4943
     1991  0.0073  0.0185  0.4929
     1992  0.0063  0.0207  0.4958
     1993  0.0056  0.0215  0.4934
     1994  0.0054  0.0203  0.4948
     1995  0.0027  0.0215  0.4931
     1996  0.0041  0.0205  0.4975
     1997  0.0039  0.0215  0.4948
     1998  0.0022  0.0205  0.4941
     1999  0.0061  0.0200  0.4946
```

```
[17]: axes = imu_table.plot.line(subplots=True, title="Scatter plot of the collected␣
      ↪data", figsize=[10,5])
      plt.show()
```



4 Descriptive Statistics

4.1 Measures of Central Tendency

Mean

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

```
[4]: mean_x = imu_table['X'].sum() / imu_table['X'].size
     mean_y = imu_table['Y'].sum() / imu_table['Y'].size
     mean_z = imu_table['Z'].sum() / imu_table['Z'].size
```

```
print(" Mean of X: ", mean_x, "\n",
      "Mean of Y: ", mean_y, "\n",
      "Mean of Z: ", mean_z)
```

```
 Mean of X:  0.00542455
 Mean of Y:  0.0210529
 Mean of Z:  0.4947642
```

4.2 Measures of Variability

Sample Variance

$$s^2 = \frac{1}{(n-1)} \sum_{i=1}^{n} (x_i - \bar{x})^2$$

We divide the sum by $n-1$ according to Bessel's correction.

```
[5]: # Delta Degrees of Freedom (ddof). The divisor used in calculations is N - ddof,␣
     ↪where N represents the number of elements.
     variance_x = imu_table['X'].var(ddof=1)
     variance_y = imu_table['Y'].var(ddof=1)
     variance_z = imu_table['Z'].var(ddof=1)
     print(" Variance of X: ", variance_x, "\n",
           "Variance of Y: ", variance_y, "\n",
           "Variance of Z: ", variance_z)
```

```
 Variance of X:  2.3846045997999e-06
 Variance of Y:  1.7937184492246122e-06
 Variance of Z:  2.1364866033016636e-06
```

Sample Standard Deviation

$$s = \sqrt{\frac{1}{(n-1)} \sum_{i=1}^{n} (x_i - \bar{x})^2}$$

```
[6]: std_x = imu_table['X'].std(ddof=1)
     std_y = imu_table['Y'].std(ddof=1)
     std_z = imu_table['Z'].std(ddof=1)
     print(" Standard deviation of X: ", std_x, "\n",
           "Standard deviation of Y: ", std_y, "\n",
           "Standard deviation of Z: ", std_z)
```

```
 Standard deviation of X:  0.0015442165003003627
 Standard deviation of Y:  0.0013392977447993452
 Standard deviation of Z:  0.0014616725362753668
```

4.3 Measures of Correlation

Covariance Matrix

$$Q = \begin{pmatrix} CoVar(X,X) & CoVar(X,Y) & CoVar(X,Z) \\ CoVar(Y,X) & CoVar(Y,Y) & CoVar(Y,Z) \\ CoVar(Z,X) & CoVar(Z,Y) & CoVar(Z,Z) \end{pmatrix}$$

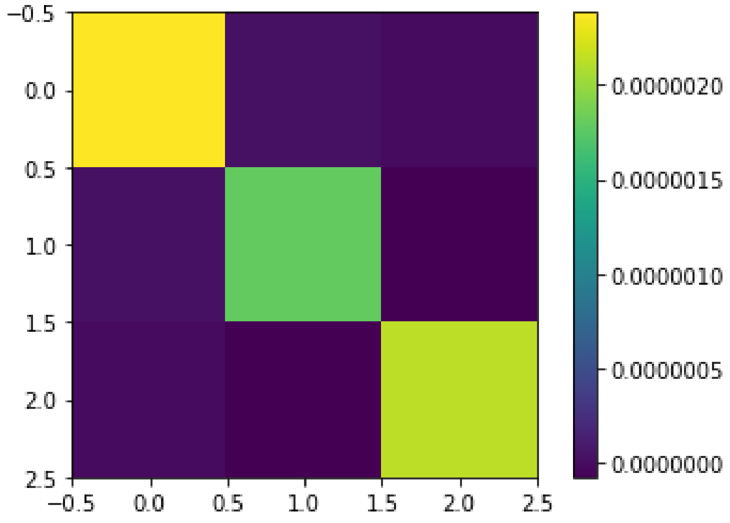
$$CoVar(X,Y) = \frac{1}{(n-1)} \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})$$

If we fill in Y=X, we get the formula for the Variance of X.

[7]:
```
# Get covariance matrix
imu_table.cov()
```

[7]:
```
              X             Y             Z
X  2.384605e-06  4.206234e-08 -4.163182e-10
Y  4.206234e-08  1.793718e-06 -7.247242e-08
Z -4.163182e-10 -7.247242e-08  2.136487e-06
```

[8]:
```
# Plot a heatmap of covariance matrix
plt.imshow(imu_table.cov())
plt.colorbar()
plt.show()
```



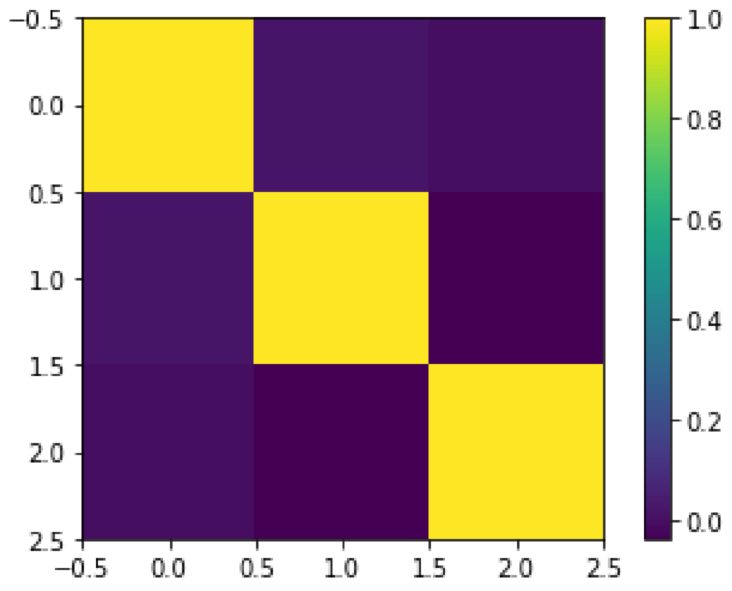
Correlation Coefficient *Pearson correlation coefficient between variables X and Y:*

$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2}\sqrt{\sum(y_i - \bar{y})^2}} = \frac{CoVar(X,Y)}{s_x.s_y}$$

[9]:
```
# Get correlation coefficient matrix
imu_table.corr()
```

[9]:
```
          X         Y         Z
X  1.000000  0.020338 -0.000184
Y  0.020338  1.000000 -0.037021
Z -0.000184 -0.037021  1.000000
```

[10]:
```
# Plot a heatmap of correlation coefficient matrix
plt.imshow(imu_table.corr())
plt.colorbar()
plt.show()
```

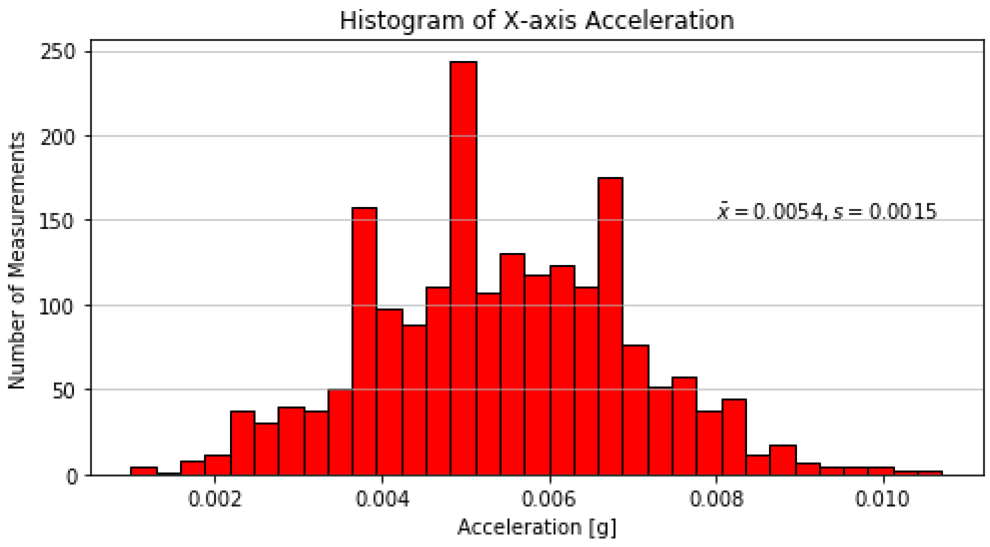


4.4 Histograms

A histogram divides the variable into bins, counts the data points in each bin, and shows the bins on the x-axis and the counts on the y-axis. In this case, the bins are intervals of acceleration measured in g's and the count is the number of measurements falling into that interval.

Distribution of Acceleration X-axis

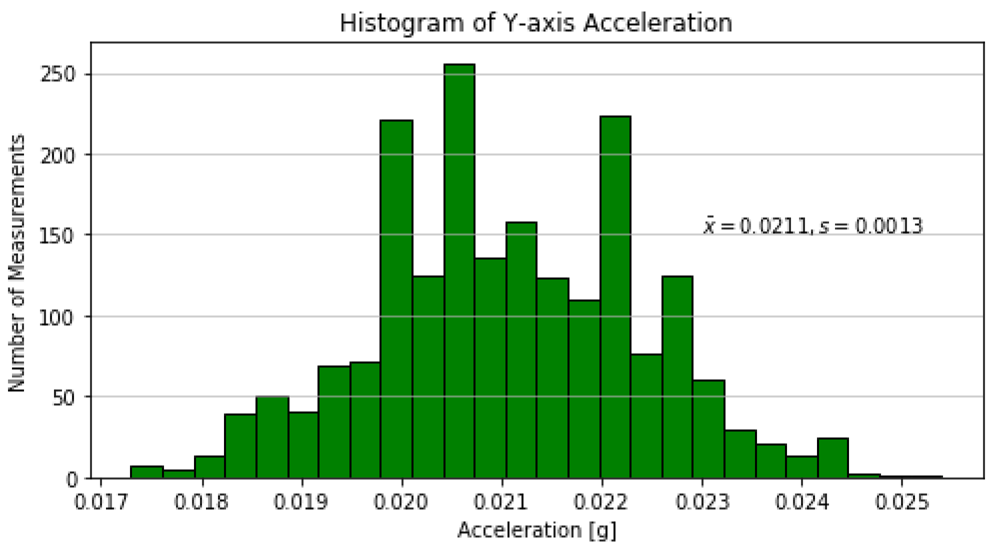
[11]:
```
fig = plt.figure(figsize=(8,4))

# matplotlib histogram
n_measurements, bins, patches = plt.hist(imu_table['X'], color = 'red',␣
 ↪edgecolor = 'black',
         bins = 'auto')
plt.grid(axis='y', alpha=0.75)
plt.xlabel('Acceleration [g]')
plt.ylabel('Number of Measurements')
plt.title('Histogram of X-axis Acceleration')
plt.text(0.008, 150, r'$\bar{x}=%.4f, s=%.4f$' % (mean_x, std_x))
plt.show()
```



The standard methods of tuning the number of bins are the normal reference rules according to Scott and Freedman & Diaconis which proceed by assuming the data is close to normally-distributed, and intended to minimize the difference between the histogram and the underlying distribution of data. In this case, the underlying algorithm chose Freedman & Diaconis's.

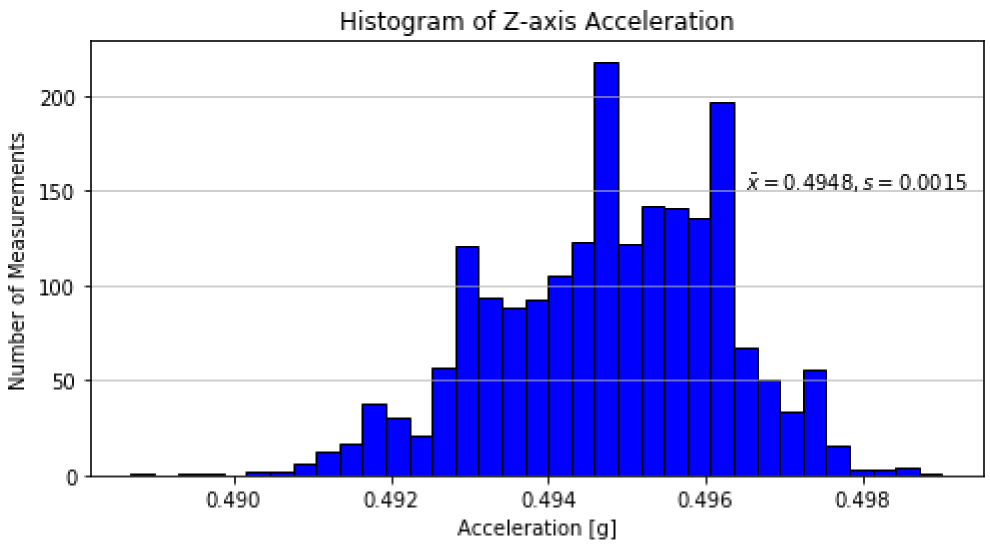
Distribution of Acceleration Y-axis

[12]:
```
fig = plt.figure(figsize=(8,4))
n_measurements, bins, patches = plt.hist(imu_table['Y'], color = 'g', edgecolor␣
 ↪= 'black',
        bins = 'auto')
plt.grid(axis='y', alpha=0.75)
plt.xlabel('Acceleration [g]')
plt.ylabel('Number of Measurements')
plt.title('Histogram of Y-axis Acceleration')
plt.text(0.023, 150, r'$\bar{x}=%.4f, s=%.4f$' % (mean_y, std_y))
plt.show()
```



Distribution of Acceleration Z-axis

```
[13]: fig = plt.figure(figsize=(8,4))
n_measurements, bins, patches = plt.hist(imu_table['Z'], color = 'blue',␣
 ↪edgecolor = 'black',
          bins = 'auto')
plt.grid(axis='y', alpha=0.75)
plt.xlabel('Acceleration [g]')
plt.ylabel('Number of Measurements')
plt.title('Histogram of Z-axis Acceleration')
plt.text(0.4965, 150, r'$\bar{x}=%.4f, s=%.4f$' % (mean_z, std_z))
plt.show()
```


Histogram of Z-axis Acceleration
Number of Measurements
Acceleration [g]
$\bar{x} = 0.4948, s = 0.0015$