Homework 1 – IMU Data Extraction

Submitted in partial fulfillment of the requirements for the course of

ENPM809T – Autonomous Robotics

By

Adheesh Chatterjee

Course Faculty
Prof. Steven Mitchell

A. JAMES CLARK
SCHOOL OF ENGINEERING

INTRODUCTION

The aim of this project is to load data with Python/NumPy and plot/analyze it with Matplotlib. We use a ADXL327 3-axis accelerometer as part of the robot's navigational sensor suite. Each data packet contains the date and time of transmission along with a series of sensor readings that pertain to the robot's navigation. We then use the concept of moving averages to better understand the data and reduce noise

APPROACH

We first load the data into python using the numpy library and the 5th column of the raw data is then plotted using matplotlib. The plot is then annotated for better understanding

Then, 2-, 4-, 8-, 16-, and 64-, and 128-pt moving averages are calculated and the average data is then plotted on top of the original raw data. The mean and standard deviation of the averaged data is found the are also plotted to get a better understanding of the data.

Finally, all the resulting plots are saved and we interpret information from the sensor

CODE

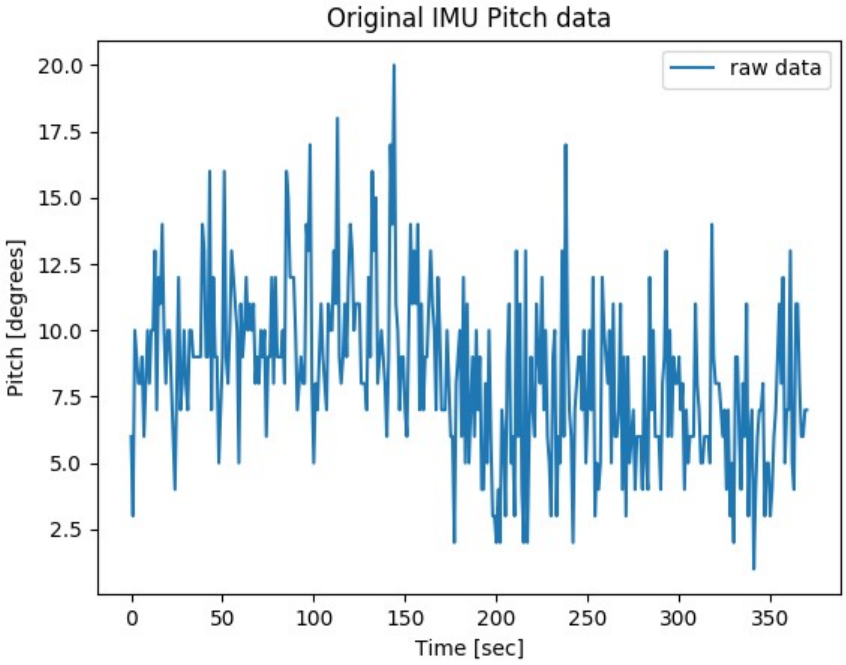
The entire repository containing the problem statement and the data can be viewed on https://github.com/adheeshc/IMU-Readings

It is also pasted below for further viewing

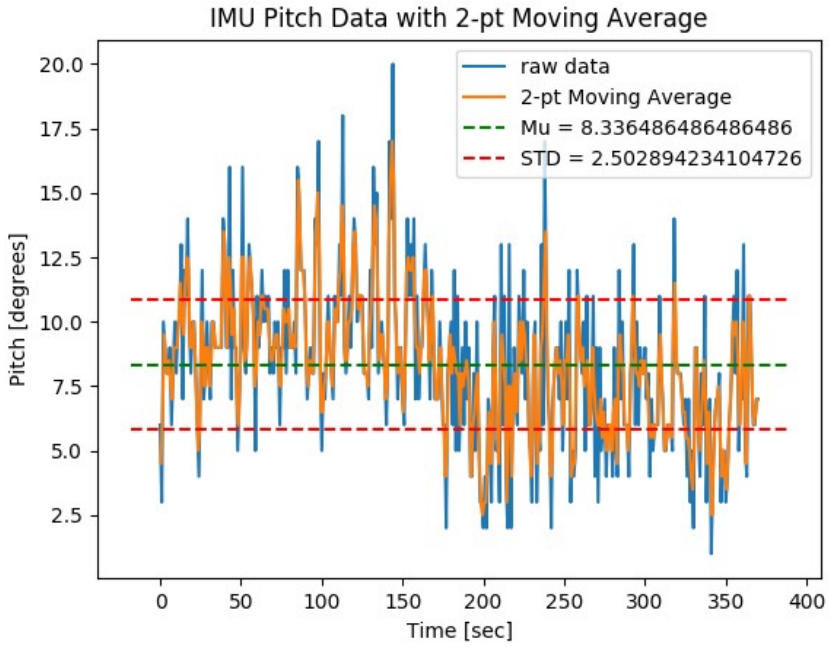
hw1.py

```python
# -*- coding: utf-8 -*-
#    ___      _ _               _
#  / _ \    | | |              | |
# / /_\ \ __| | |__   ___  ___ | |__
# |  _  |/ _` | '_ \ / _ \/ _ \/ __| '_ \
# | | | | (_| | | | |  __/  __/\__ \ | | |
# \_| |_/\__,_|_| |_|\___|\___||___/_| |_|
# Date:   2020-02-05 05:06:27
# Last Modified time: 2020-02-05 06:12:14

import numpy as np
import matplotlib.pyplot as plt

#Load data into python/numpy
data = np.loadtxt("../Init/imudata.txt",dtype={
    'names': ('date','time','x','y','pitch','col 6','col 7'),
    'formats' : ('S10','S10','f8','f8','f8','f8','f8')
    })
pitch_data=data['pitch']

#plot raw data for 5th column
x=np.arange(len(pitch_data))
plt.plot(x,pitch_data,label='raw data')
#label the axes and add title and legend
plt.xlabel('Time [sec]')
plt.ylabel('Pitch [degrees]')
plt.title('Original IMU Pitch data')
plt.legend(loc=1)
plt.savefig(f'../Output/Raw data')
plt.show()

#Calculating moving averages
moving_avg=[2,4,8,16,64,128]
for i in moving_avg:

    #plot raw data
    plt.plot(x,pitch_data,label='raw data')

    #Calculating the moving averages
    x_moving_average=[sum(range(j, j + i)) / i for j in range(len(pitch_data) - i + 1)]
    y_moving_average = [sum(pitch_data[j:j + i]) / i for j in range(len(pitch_data) - i + 1)]

    #plotting moving averages
    plt.plot(x_moving_average, y_moving_average,label=f'{i}-pt Moving Average')

    #calculating mu and sigma
    mu = float(np.mean(y_moving_average))
    sig = float(np.std(y_moving_average))
    print(f"i: {i} | mu: {mu} | sigma: {sig}")
    x_limits = plt.xlim()

    #plotting mu and sigma
    plt.plot(x_limits, (mu, mu), 'g--',label= f'Mu = {mu}')
    plt.plot(x_limits, (mu + sig, mu + sig), 'r--',label=f'STD = {sig}')
    plt.plot(x_limits, (mu - sig, mu - sig), 'r--')

    #adding labels, legend and title
    plt.title(f"IMU Pitch Data with {i}-pt Moving Average")
    plt.xlabel("Time [sec]")
    plt.ylabel("Pitch [degrees]")
    plt.legend(loc=1)
    plt.savefig(f'../Output/{i}-pt_Moving_Average')
    plt.show()
```

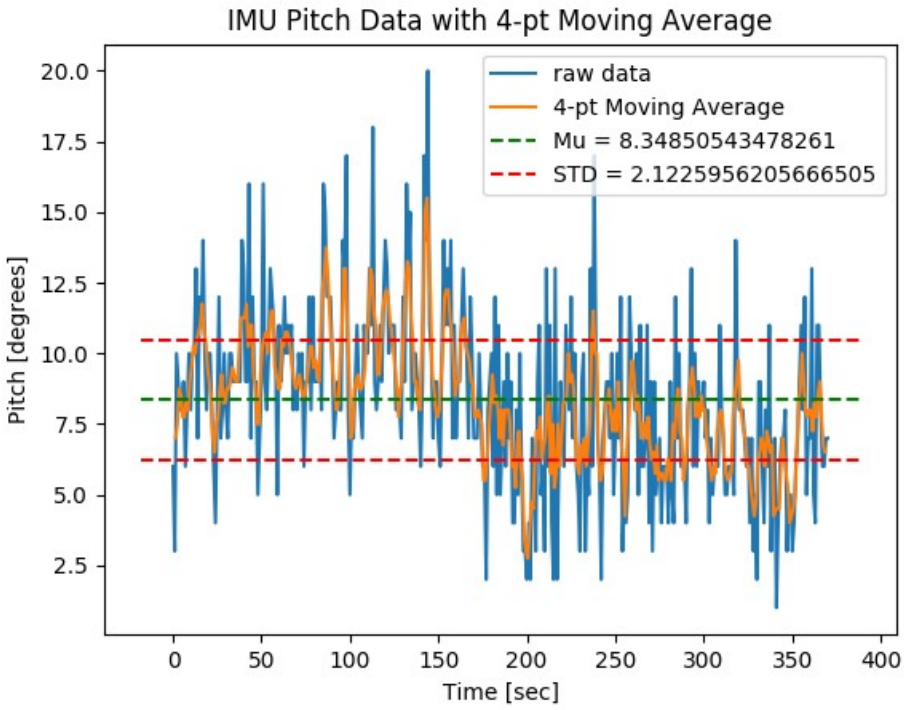
RESULTS



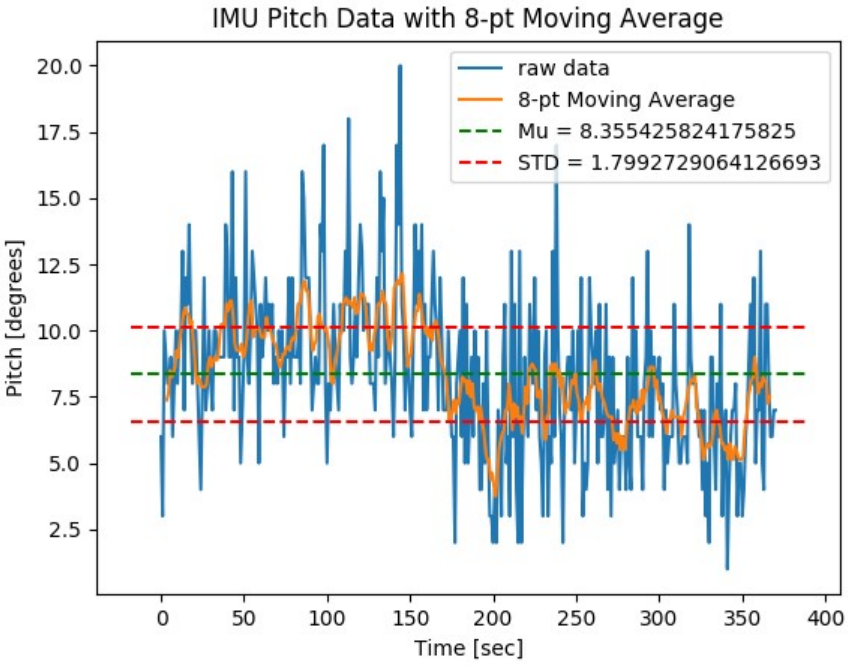
As we can see from the raw data, we get very noisy readings that need to be cleaned to get a better representation



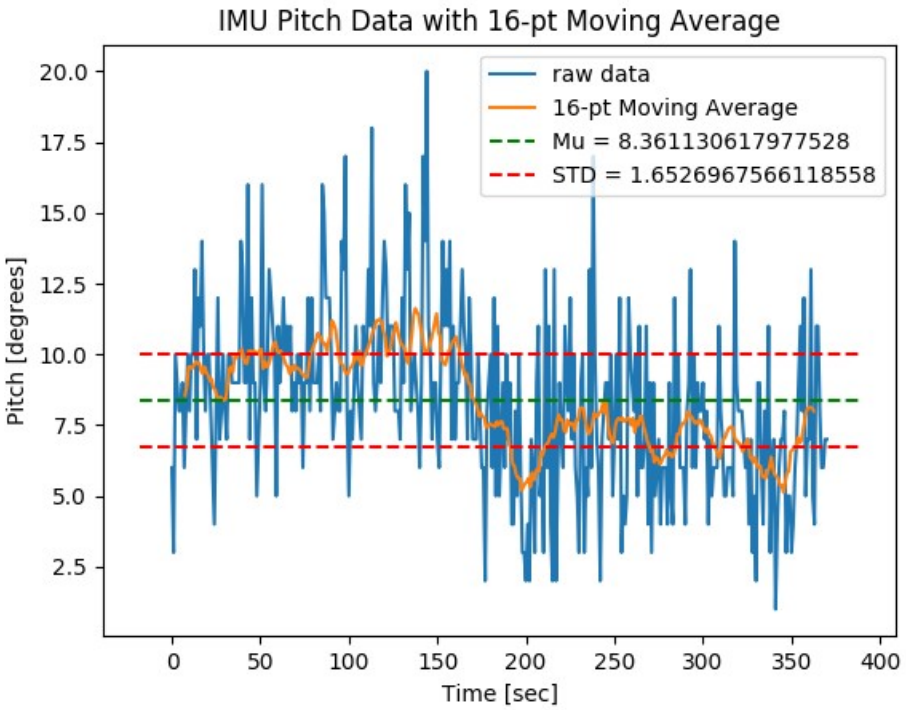
As we see from the 2-pt moving average, we barely remove any noise and the processed data is pretty much the same



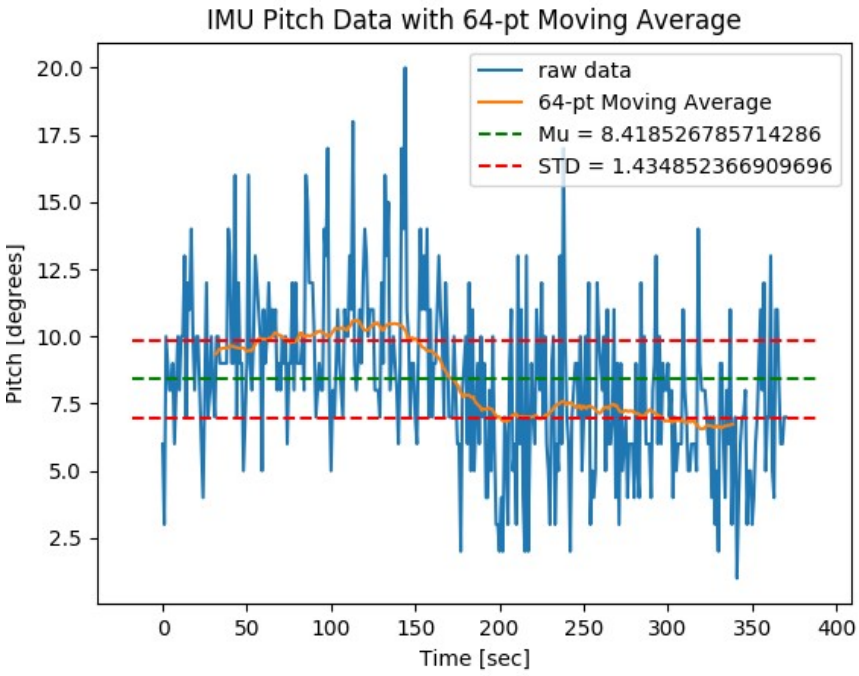
The 4-pt moving average, gives us better readings, but it is still very sparsely distributed

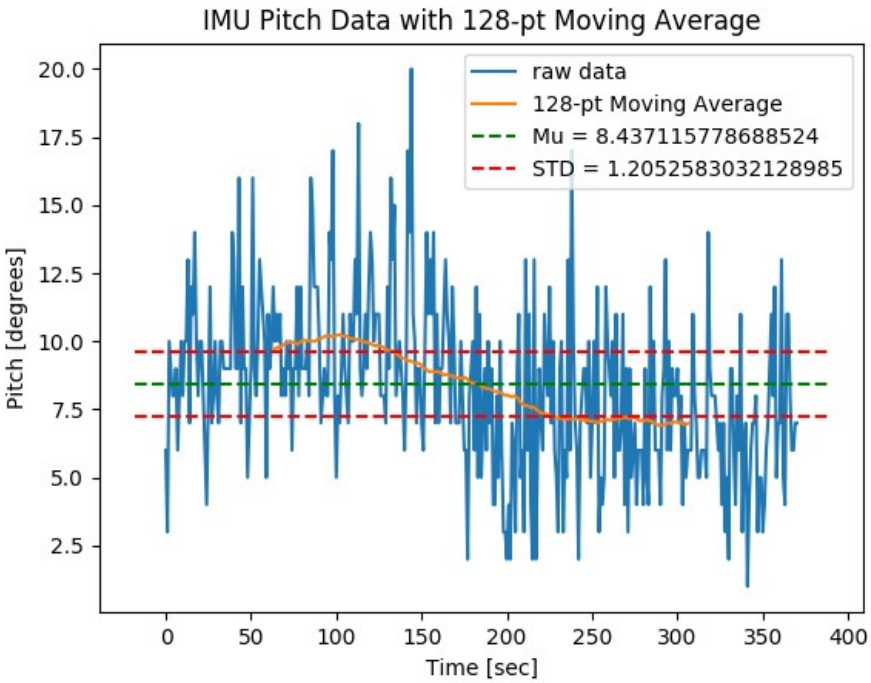


The 8-point moving average begins to show the low-frequency noise in the sensor, but the high-frequency noise is still visible



I feel the 16-pt moving average filter does a good job of removing a lot of the noise but there are still too many spikes appearing too close together, for our current application, this could mean sudden turns, but this is highly improbably, hence it must be noise





The 64-pt moving average filter in my opinion is best suited to the problem, as it removes most of the noise from the data while preserving some peaks which may represent measurements which the 128-pt filter completely removes. I feel the 128-pt filter may be over-smoothing the data and we are losing out on some information

Although the raw data we see now does look very similar to a sinusoidal wave, we cannot overlook the fact that in reality, there might have been some random movements, thus we cannot get a perfect sinusoidal wave.

Hence a 64-pt filter might be the best among all the tested filters here.

```
i: 2 | mu: 8.336486486486486 | sigma: 2.502894234104726
i: 4 | mu: 8.34850543478261 | sigma: 2.1225956205666505
i: 8 | mu: 8.355425824175825 | sigma: 1.7992729064126693
i: 16 | mu: 8.361130617977528 | sigma: 1.6526967566118558
i: 64 | mu: 8.418526785714286 | sigma: 1.434852366909696
i: 128 | mu: 8.437115778688524 | sigma: 1.2052583032128985
```

The mean and standard deviation of the moving average filters are tabulated above. Although we see the means converge to around 8.4, the standard deviations reduce considerably from 2.5 to half of that at 1.2

This makes sense as per the central limit theorem. Essentially, what happens is that lets say for the 2pt moving average filter, the left most and right most points are only used once and the all other points are used twice to create a moving average. This concept is extended for higher order filters. This filter although serves reasonably well here, can be significantly affected by outliers and bad measurements (lots of noise).