ADC controls the stepper motors with PWM

Yu Che Lin
Electrical Engineering Department, College of Engineering
San Jose State University, San Jose, CA 94112
E-mail: s9883024@gmail.com

Abstract

This report is writing for implementing A/D conversion for sensor interface and stepper motor drive integration test. In this project, it will integrate ADC sensor input to drive the stepper motor. By using the ADC data, this data will transfer to the microcontroller through I2C. Based on this data, the microcontroller will produce the PWM signal to control stepper motor. This report will provide not only hardware structure but also software designs the whole process to explain how to successfully realize this project. In the hardware part, it will base on ARM system with the raspberry pi environment. In the software side, this project will use python to be the programming language. In the last, providing how to validate the calculated result.

1. Introduction

Embedded systems are where the software meets the physical world. As people put tiny computers into all sorts of systems (door locks, pacemakers, etc.), how to implement the software is truly, terrifyingly important. Writing software for these things is more difficult than computer software because the systems have so few resources. Therefore, it needs more people to study this field together.

To connect to the physical world, ADC (Analog to Digital converters) is an essential role. In the real world, most data are characterized by analog signals. In order to manipulate the data using a microprocessor, we need to convert the analog signals to the digital signals, so that the microprocessor will be able to read, understand and manipulate the data. As a result, understanding how to use ADC is the first step for became an embedded engineer.

In addition, Pulse width modulation (PWM) also an important element for mastering embedded systems. PWM is employed in a wide variety of applications, ranging from measurement and communications to power Control and conversion. A lot of applications are based on using PWM, such as a 3D printer or fan control. PWM main use is for controlling DC motors, but it can also be used to control valves, pumps, hydraulics, and other mechanical parts. Especially in this paper place emphasis on PWM using in the stepper motor. It will show detail on how to control stepper motor by using PWM.

As mentioned above, the application of ADC and PWM is very important to be a good embedded engineer. But there is currently no good guide to help novices into this field. Therefore, this article will simply lead people on how to build an environment and implement basic applications step by step, in order to guide more people to do more research in this field.

2. Methodology

This project divided into three major parts. First, it gets the sample data from ADC and compares this data between itself and the data measured by a potential meter in the real world. The data which get from ADC were not such perfect. It had some deviation. Therefore, it needs to do some compensation with ADC data to make sure the data close to the ideal value. It this part, the most difficult thing is how to get the data from ADC. In the beginning, it hard to find the direction. After sever research, python will be a good solution for getting the data. The detail of python code will explain in section-3.2.

Secondly, after getting data from ADC, the next step is using FFT program to validate the ADC data by computing its power spectrum. In this verification, it will change different sample rate to present the result.

Finally, it integrates the ADC input with the PWM motor driver for stepper motor speed control. When the potentiometer increases its output voltage, the stepper motor will increase its speed. When the input voltage reaches the 3.3 ADC limit, the motor reaches the maximum speed. As the input voltage reaches 0.0 ADC, the motor should stop. Also, it will read LSM303 sensor input to CPU and display its data. LSM303 will present x-axis, y-axis and heading angle.

2.1. Objectives and Technical Challenges

Many systems design will encounter the subtleties in ADC specifications that often lead to less-than-desired system performance. [1] This article explains how to compensate an ADC based on the system requirements and describes the various sources of error when making an ADC measurement.

Using a 16-bit-resolution analog-to-digital converter (ADC) does not necessarily mean your system will have 16-bit accuracy. Sometimes, much to the surprise and

consternation of engineers, a data-acquisition system will exhibit much lower performance than expected. When this is discovered after the initial prototype run, a mad scramble for a higher-performance ADC ensues, and many hours are spent reworking the design as the deadline for preproduction builds fast approaches. Therefore, compensation is quite essential for getting the data from ADC in order to apply the perfect data to produce PWM signals.

2.2. Problem Formulation and Design

This project needs use Fourier transform to change the ADC data to frequency domain. Thus, the Fourier transform is an essential formula. Moreover, it will focus on fast Fourier transform (FFT) [2]. A fast Fourier transform (FFT) is an algorithm that computes the discrete Fourier transform (DFT) of a sequence, or its inverse (IDFT). Fourier analysis converts a signal from its original domain (often time or space) to a representation in the Frequency domain and vice versa. Fast Fourier transforms are widely used for applications in engineering, science, and mathematics.

$$A_k = \sum_{n=0}^{N-1} e^{-i\frac{2\pi}{N}kn} a_n$$

Formula 1. DFT formula

To more detail about FFT, it provides the DFT formula as formula 1. When a signal is discrete and periodic, we don't need the continuous Fourier transform. Instead, we use the discrete Fourier transform or DFT. Suppose our signal is a for n = 0 ... N − 1, and $a_n = a_{n+jN}$ for all n and j. The FFT is a fast algorithm for computing the DFT. If we take the 2-point DFT and 4-point DFT and generalize them to 8-point, 16-point, ..., 2r-point, we get the FFT algorithm. In section 4, will use FFT to do the testing and verification.

Also, LSM303 only can provide x-axis, y-axis and z-axis data but not angle information. To convert the data readings into a 0-360° degrees compass heading, we can use the arctan function to compute the angle of the vector defined by the Y and X axis readings. The result will be in radians, so we multiply by 180 degrees and divide by Pi to convert that to degrees. Formula 2. shows the detail of the calculation. For normalize to 0-360°, if angle small than 0 degree. It needs plus 180°.

$$\frac{((\tan^{-1}((y-axis)/(x-axis)))*180)}{\pi}$$

Formula 2. Heading angle formula

3. Implementation

This section will give the detailed about one-to-one correspondence description of this project's design and how to solve the problem and to achieve provide the high-quality PWM signals:

1. This paragraph will provide system block diagrams and circuit schematics for the hardware design;
2. Second paragraph will give the guide how to use python to realize our goal and present it flow chart and pseudo code description for the step-by-step discussion of the software design.
3. Provide the formula and method to implement validation.

3.1. Hardware Design

In this paragraph, it places the emphasis on hardware design which includes bill of material, system block diagram, and schematic design.

First of all, was listed the material it wanted, which including their electrical characteristics for building schematic design.

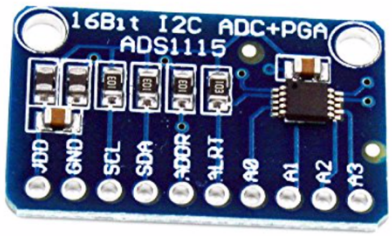
ARM microcontroller: Raspberry pi is an ARM microcontroller. It provides 1.4GHz 64-bit quad-core processor, dual-band wireless LAN, Bluetooth 4.2/BLE, faster Ethernet, HDMI display and Power-over-Ethernet support. Moreover, it supports GPIO, UART, SPI and I2C interface. I2C is an essential interface in this project's implementation. Since raspberry pi not only provide different kinds of the interface but also good to get started, it is a good choice for the beginner to do their project.

Item	Electrical characteristics
Power consumption	2500 mA
Input power range	5V

Fig.1. The electrical characteristics of raspberry pi

ADC Converter: Since raspberry pi does not support the analog-to-digital converter, this project chooses ADS1115 as an ADC converter. The ADS1115 provides

16-bit precision at 860 samples/second over I2C. As a result, it can provide the sample data thought I2C. Furthermore, since it supports from 2V to 5V power, it can measure a large range of signals and its super easy to use. It is a great general purpose 16-bit converter.



Item	Electrical characteristics
Power consumption	300 μA
Input power range	2.2V to 5V
Data Rate	8 SPS to 860 SPS

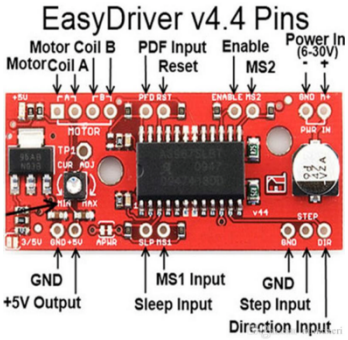
Fig.2. The electrical characteristics of ADS1115

Meter Board: LSM303 is accelerometer board in this project. Its I2C interface is compatible with both 3.3v and 5v processors and the two pins can be shared by other I2C devices. It can detect the change in capacitance on each axis is converted to an output voltage proportional to the acceleration on that axis. Thus, it can use the I2C to get the data which this project's need.

Item	Electrical characteristics
Power consumption	150 mA
Input power range	3.3V to 5V
SCL clock frequency	400 K Hz

Fig.3. The electrical characteristics of LSM303

Motor driver board: V44-A3967 is an easy to use stepper motor drive. It able to drive the stepper motor up to 30V. It also has an on-board voltage regulator for the digital interface that can be set to 5V or 3.3V. Therefore, it can gain the goal which integrate the ADC input with PWM motor drive program to realize the stepper motor speed control. As the potential meter increases its output. That is the reason this project chooses V44-A3967 as a motor driver board.



Item	Electrical characteristics
Power consumption	850 mA
Input power range	7V to 30V
SC Maximum STEP Frequency	500 K Hz

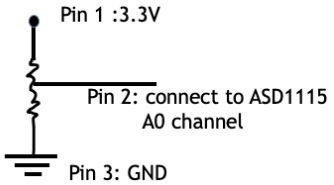
Fig.4. The electrical characteristics of V44-A3967

Stepper motor: In order to be able to play full functions with the motor driver board, it also needs to buy a stepper motor. choosing Nema 17 stepper motor can satisfy it needed. This stepper motor supports each phase draws current 0.4A at 12V, allowing for a holding torque of 26Ncm(36.8oz.in). It totally meets the requirement.

Item	Electrical characteristics
Power consumption	400 mA
Input power range	12 V

Fig.5. The electrical characteristics of Nema 17 stepper motor

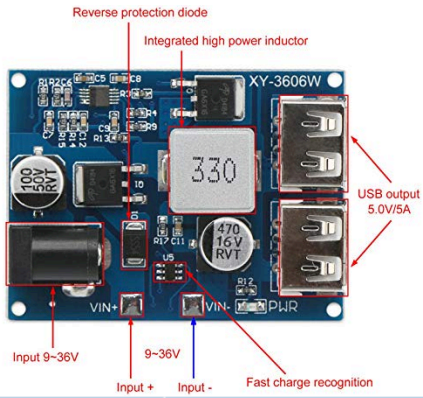
Potentiometer: To build ADC input circuit to produce an output voltage from 0.0 VDC to 3.3 VDC, it needs a potentiometer to get the output voltage.



Item	Electrical characteristics
Resistor	10 K ohm

Fig.6. 10 K ohm potentiometer

Power unit: Based on the above electrical characteristics, at least, its design needs the 5V/2.5A for raspberry pi and 12V/850mA for the motor function. In order to meet this requirement. The power unit is necessary. The supply module volt step down transformer board can fix this problem. It supports input voltage range is 9-36V and the output voltage is 5V. If the DC in between 9-24V, it could provide 6A current for output current. Thus, this power board could cover the whole system's power consumption.



Item	Electrical characteristics
Power consumption	6000 mA
Input power range	9-36V
5V power output	5.0V/5A

Fig.7. The electrical characteristics of power unit board

The major components were list on the above. For a deeper understanding of this hardware architecture, the System block diagram provides a simple concept. The position of the component can be provided immediately to understand the corresponding components and their functions when people see the actual drawing in the following chapters. Moreover, we put the LSM303 sensor on the stepper motor. Therefore, when the motor rotates it, the LSM303 sensor will feedback the accelerometer and magnetometer data. Through this way, he can get the relative relationship between ADC data and motor moving data.

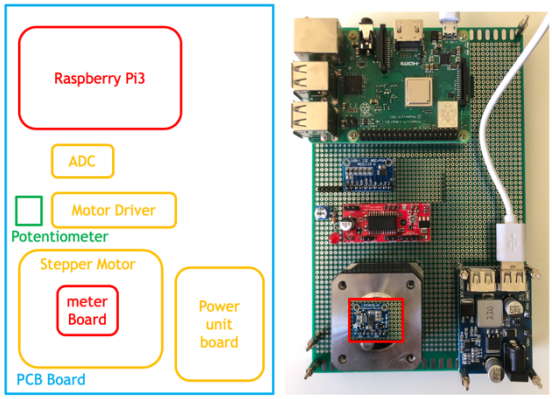


Fig.8. System block diagram

The next figure will show this project's schematic design. It will provide a connection between the different board. In addition, people could know where the signal or power comes from. Based on this schematic design, people could understand the whole hardware design much easier. The use of arrows makes the relative relationship between components more explicit. For example, the SDA of I2C is the two ways direction and SCL is the one-way direction. Using the arrows make it more clearly. Furthermore, for GPIO pin, it uses pin22 GPIO25 as out to send the PWM signal to step pin on the motor driver board. On the other hand, pin40 GPIO 21 is an output pin for control the DIR pin on motor driver board. Direction Input (DIR). The state of the DIRECTION input will determine the direction of rotation of the motor. When DIR pin is low, the stepper motor rotates clockwise. When DIR pin is high, the stepper motor rotates counterclockwise.

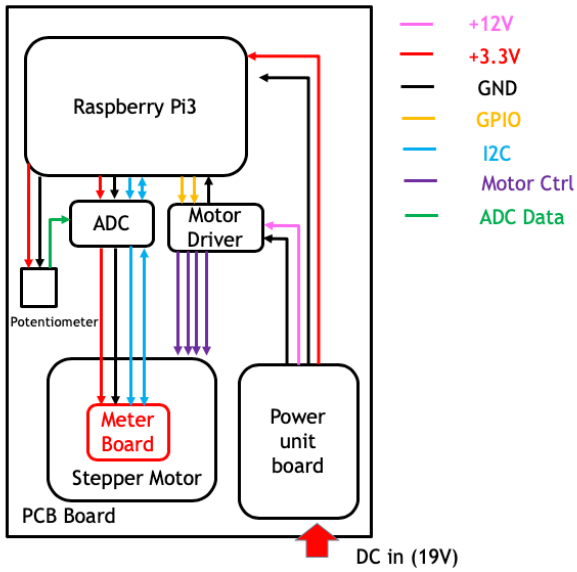


Fig.9. Schematic design

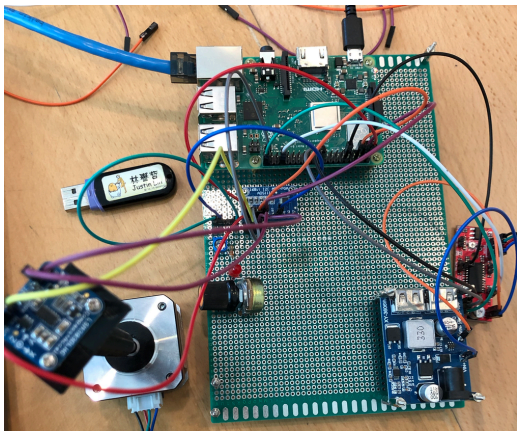


Fig.10. Actual picture

3.2. Software Design

The programming language will base on python. Python is a general-purpose language, which means it can be used to build just about anything, which will be made easy with the right tools/libraries. This project uses a lot of libraries to realize different function.

Professionally, Python is great for backend web development, data analysis, artificial intelligence, and scientific computing. Many developers have also used Python to build productivity tools, games, and desktop apps, so there are plenty of resources to help you learn how to do those as well.

To give basic process of software design. The Fig 11 shows top level flow chart. It presents whole process of this software design. This paragraph will explain each flow step by step.

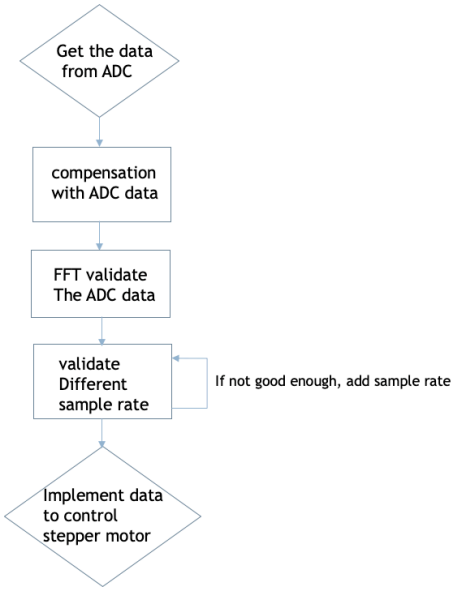


Fig.11. Top level flow chart

First of all, it discusses with how to set python library environment to reach out goal. It will mention how to use python code to get the ADC sample data. As mentioned in the hardware section, the sample data got through I2C. CPU should communicate with ADS1115 with I2C to find the A0 channel register in order to get the sample data. In this part, it should install the ADS1115 library package to build the python environment. To install from the Python package index, connect to a terminal on the Raspberry Pi and execute the following commands:

```
sudo apt-get update
sudo apt-get install build-essential python-dev python-smbus python-pip
sudo pip3 install adafruit-ads1x15
```

Code 1. ADS1115 library

In addition, to plot the figure with python it should install Matplotlib. Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. As a result, On the terminal, we should execute the following code.

```
sudo apt-get install python3-matplotlib
```

Code 2. matplotlib library

Last but not least, for implementing FFT function, we also the math formula from SciPy library. The SciPy library is one of the core packages that make up the SciPy stack. It provides many user-friendly and efficient numerical routines such as routines for numerical integration and optimization. The install commands as code 3.

```
sudo apt update
apt-cache show python3-scipy
sudo apt install -y python3-scipy
```

Code 3. SciPy library

After building library environment, this paragraph will prove the flow charts for detailed lower level implementations, algorithm, and pseudo code.

For getting the ADC data, the concept will follow the pseudo code 1. The first line will import time function to control the time of getting data. The second line will import ADS1115 library to gain the sample data. Next will set “data” to use this library and set N = 10 for the N numbers of sample data from ADC. It will loop N times and pause 1 sec set sampling frequency to 1K SPS (samples per second).

```
import time
import Adafruit_ADS1x15

data = Adafruit_ADS1x15.ADS1115()
N = 10 # Take the N numbers of sample data from ADC

for j in range(N):
    values = [0]

    time.sleep(1) # Time of getting data

```

Pseudo Code 1. Get 10 ADC data

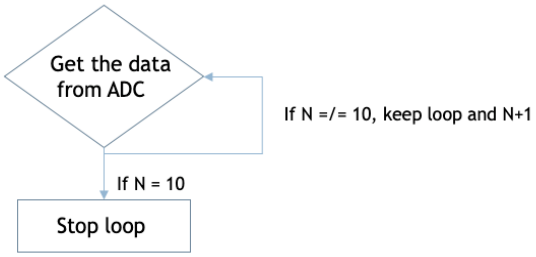


Fig.12. Flow chart for ADC

Next, it will show how to do the FFT. Since it uses SciPy library, it makes the pseudo code very simply. The pseudo code will present in pseudo Code 2.

```
import numpy as np
from scipy.fftpack import fft,ifft

l = [] # l assume l is a list which data is getting ADC
lf = fft(l) # FFT l list
lf1 = abs(lf) # Power spectrum lf

```

Pseudo Code 2. FFT

Pseudo Code 3 presents how to use raspberry pi's GPIO pin to produce PWM signal. The first five lines show on the Pseudo Code 3 will import all library it needed. The line 11 set the GPIO 25 as an GPIO OUT, and the line 12 set this GPIO OUT to become PWM signal. Line 15 creates an infinite loop. This loop will never stop until type ctrl + c command. In this loop, it will base on ADC data modify the data for duty-cycle percentage and frequency. As the potentialmeter increases its output voltage the stepper motor will increase its speed, as the input voltage reaches 3.3V limit, the motor reaches the highest speed, as the input voltage reaches 0.0V, the motor should stop.

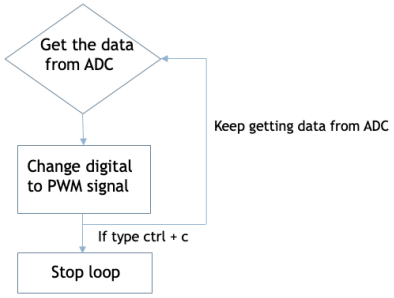


Fig.13. Flow chart for PWM signals

```
import pigpio
import time
import matplotlib.pyplot as plt
import numpy as np
import Adafruit_ADS1x15
import RPi.GPIO as GPIO

LED_PIN = 25 # choose GPIO 23 as our GPIO pin
PWM_FREQ = 200 # PWM setting on 200
GPIO.setmode(GPIO.BCM)
GPIO.setup(LED_PIN, GPIO.OUT) # set the GPIO 23 as GPIO OUT
pwm = GPIO.PWM(LED_PIN, PWM_FREQ) # import the PWM function
pwm.start(0)

data = Adafruit_ADS1x15.ADS1115()

while True: # keeping get data until ctrl + C

    values = [0] # Read A0 channel values in a list

    for i in range(1):

        values[i] = data.read_adc(i, gain=GAIN)

    l.append(values[i]) #add the values[i] into list
    print('| {0:>6}| {1:>6}| {2:>6}|'.format(*values, accel_x, accel_y))
    pwm.ChangeDutyCycle(int((values[i]/264))) # 26400 is the biggest data at 3.3V
    pwm.ChangeFrequency(int((values[i]/264))*2+1)


```

Pseudo Code 3. produce PWM signals

Finally, this project mentions how to control the rotation of the stepper motor. The goal is when potentiometer increases its output voltage the stepper motor rotates the corresponding angle according to the voltage value. For example, when the voltage of potentiometer is 0V, the motor will not move. On the other hand, when the voltage of potentiometer is 3.3V, the stepper motor will rotate 360 degrees. To achieve this goal, we need to more understand our stepper motor.

Nema 17 stepper motor is a standard size, 200-steps-per-revolution, NEMA 17 (1.7 in. square footprint, 5 mm shaft diameter), 12 V motor. This motor, like most stepper motors is a permanent magnet motor. The Mosaic stepper is typical of common high-resolution motors – a full revolution requires 200 steps, while each step turns the shaft only 1.8° for a full step, or 0.9° in half-stepping mode. This sized motor is commonly used in household appliances, medical equipment, stage lighting devices, and in various industrial control applications.

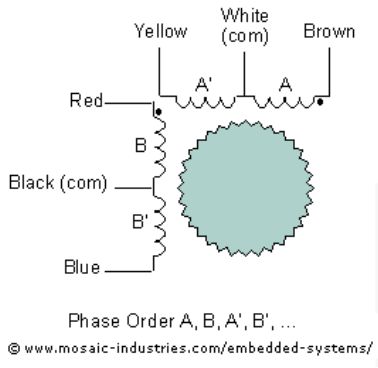


Fig.14. Stepper motor winding diagram showing phases and wire colors.

In use, the center taps of the windings are typically wired to the positive supply, and the two ends of each

winding are alternately grounded through a drive circuit to reverse the direction of the field provided by that winding. The Motor Wiring Diagram also illustrates the order of the stator poles in the motor: A, B, A', B'. This is the order in which they must be energized to cause the motor to step in a specified direction (clockwise or counterclockwise).

As mentioned above, stepper motor has full revolution requires 200 steps, while each step turns the shaft only 1.8° for a full step. Therefore, every single pulse will move 1.8°. Using 200 pulses achieve 360 °. We need to set motor driver board to full step mode [3]. Thus, we put down the MS1 pin and MS2 pin to set the full step mode. The detail presents on Fig15.

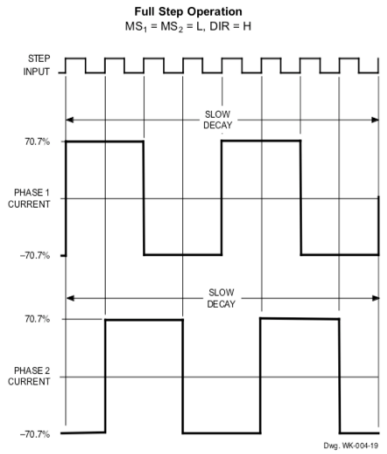


Fig.15. The vector addition of the output currents at any step is 100%.

After setting the environment, we will focus on our code structure. For implementing LSM303 function, we need to import busio, board and Adafruit_LSM303 library. Those libraries make Raspberry pie can get the accelerometer and magnetometer data through the I2C by using simple command code. The install commands as code 4.

```
sudo pip3 install board
sudo pip3 install adafruit-blinka
sudo pip3 install Adafruit-LSM303
```

Code 4. LSM303 library

Since this ASD1115 is setting at GAIN = 1, that means voltage between +/- 4.096V, and ADS1115 provides 16-bit precision. If the ideal voltage is 3.3V, it can calculate the highest data will be 2^16*3.3/ (4.096*2). The highest number in this design will be 26400. Those 26400 data are divided equally into 200 steps. After calculated, every 132 ADC date need the one step move. The Pseudo Code 4 shows the example how to realize our design. The parameter q means how many steps step motor needs to move. If q =1, that means motor will move one step.

```
q = ((data.read_adc(0, gain=GAIN))//132)
if (q) >0:
        for j in range(q):
            GPIO.output(25, 1)
            time.sleep(0.0025)
            GPIO.output(25, 0)
            time.sleep(0.0025)

while True:  # keeping get data until ctrl + C

    accel, mag= sensor.read()
    mag_x, mag_y, mag_x= mag
    angle = (math.atan2(mag_y, mag_x)*180)/math.pi
    if angle < 0:
        angle += 360

    GPIO.setmode(GPIO.BCM)
    GPIO.setup(25, GPIO.OUT,initial=GPIO.LOW)
    GPIO.setup(21, GPIO.OUT)
```

Pseudo Code 4. produce rotation control

By using angle formula 2, the angle can be calculated from the obtained x-axis and y-axis data. In order to Normalize to 0-360°, if angle < 0, the angle have to add 360°. We set the GPIO 25 and GPIO21 as an GPIO OUT. GPIO 25 mainly focus on control how many steps the stepper motor should move and GPIO 21 duty is control the direction of stepper motor. Figure 15 shows the flow chart of rotation control. Figure 16 presents the result when we implement the python code. The detail code can refer Appendix.5.

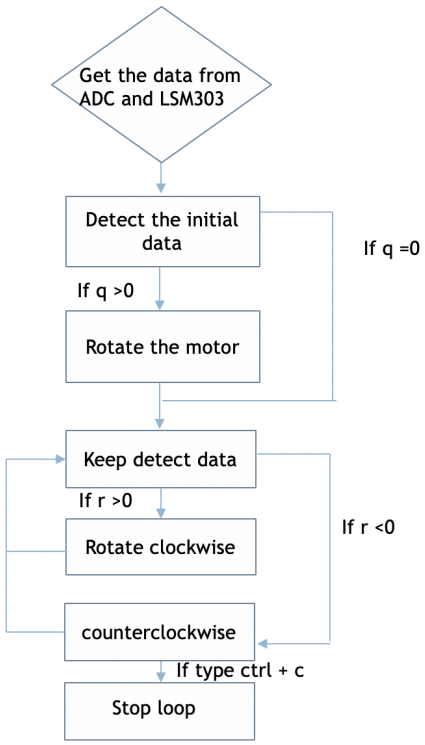


Fig.15. Flow chart for rotation control

```
pi@raspberrypi:~/Desktop $ python3 motor_rotate.py
Get the sampling data From ADC
ADC       x       y       heading angle
-------------------------------------
 1019|    -66|    -145|  245.52628118095532|
 1010|    -92|    -146|  237.78345394142434|
 1211|    -90|    -141|  237.4499965078866|
 3989|    -94|    -144|  236.8643904512811|
 6702|   -126|    -152|  230.34305575857516|
 9096|    -75|    -157|  244.4658670003524|
10695|     16|    -149|  276.129080764968|
12487|     91|    -148|  301.5858975038667|
15083|    160|    -148|  317.23117460803127|
16141|    254|    -136|  331.83393082067914|
11852|    279|    -133|  334.5127451692368|
 8977|    125|    -151|  309.6184529478681|
 5143|    -10|    -160|  266.4236656250026|
 3233|   -119|    -149|  231.3871170257645|
 3227|   -116|    -138|  229.95027223429176|
```

Fig.16. Result of feedback data

4. Testing and Verification

For calculating compensation, it uses electric meter, it measures 10 different voltage. These ten points are 0.33V, 0.66V, 0.99V, 1.32V, 1.65V, 1.98V, 2.31V, 2.64V, 2.97V and 3.27V. The last voltage is not 3.3V because of this voltage only can reach 3.27V. The measurement environment as Fig 13. Then, using the ADC.py code at Appendix2 at this voltage point it will get 10 ADC data as below list: [2750, 5379, 7955, 10580, 13140, 15934, 18606, 21121, 23989, 26332]. The result figure as Fig 14. It gets the ADC value at the corresponding voltage. Therefore, it can use those points to find the f(x). f(x) means actual measured equation. It gains as formula 3.

$$\frac{y - 2750}{x - 0.33} = \frac{5379 - 2750}{0.66 - 0.33}$$

$$\Rightarrow \quad f(x) = 7967x - 121$$

Formula 3. f(x) for ADC linear equation

Since this ASD1115 is setting at GAIN = 1, that means voltage between +/- 4.096V, and ADS1115 provides 16-bit precision. If the ideal voltage is 3.3V, it can calculate the highest data will be 2^16*3.3/ (4.096*2) The highest number in this design will be 26400. The ideal a(x) will be a(x) = 26400/3.3 *x. Then, f(x) +g(x) = a(x). It can get g(x) = a(x) – f(x) = 32x – 121. Now, the compensation function g(x) = 32x – 121.

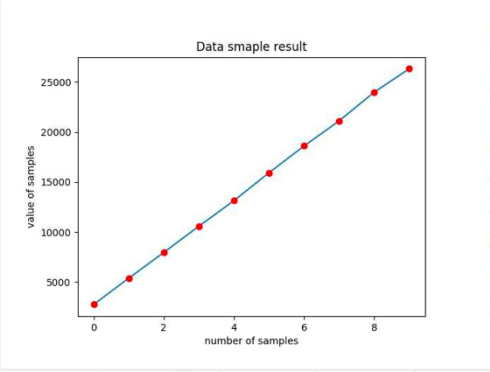


Fig.17. The result figure for ADC

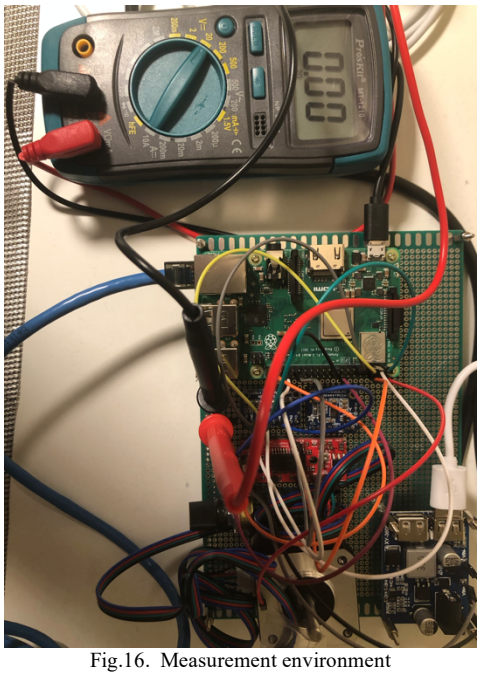


Fig.16. Measurement environment

After the compensation, it will use FFT program provided in the class to validate the ADC data by computing its power spectrum, based on your result to identify if there is any aliasing, and to make sampling rate change accordingly.

In the beginning, it still takes 10 points/ sec to FFT and computing its power spectrum. The result shows on Fig-18. As it can see, at point (N/2- 1) point, the result was far away from 0. Therefore, it keeps adding the sample rate. It adds sample rate to 20 points/ sec to FFT and computing its power spectrum. The result shows as Fig-16. It is obviously much close to 0 at the point (N/2- 1). Last, it adds sample rate to 500 points/ sec. The power spectrum result is nearly perfect. At the point (N/2- 1), the value very close to 0. It means adding the sample rate will help the digital data closer to the real world.

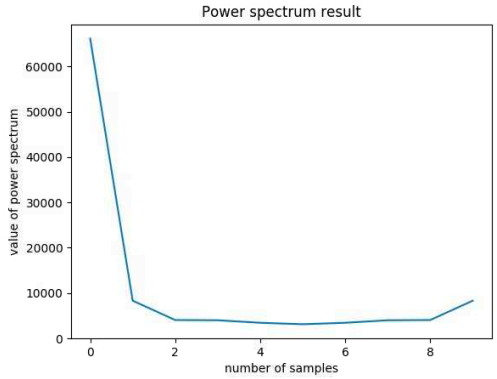


Fig.18. Power spectrum of 10 ADC DATA per sec

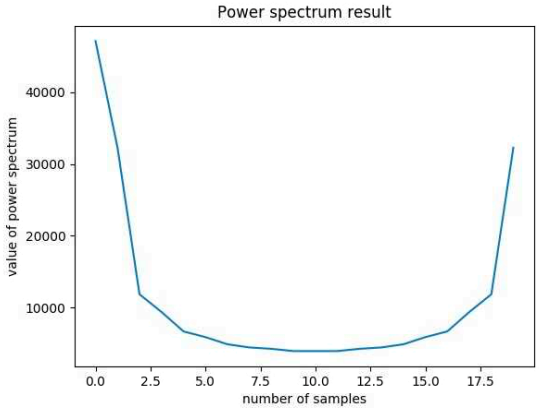


Fig.19. Power spectrum of 20 ADC DATA per sec

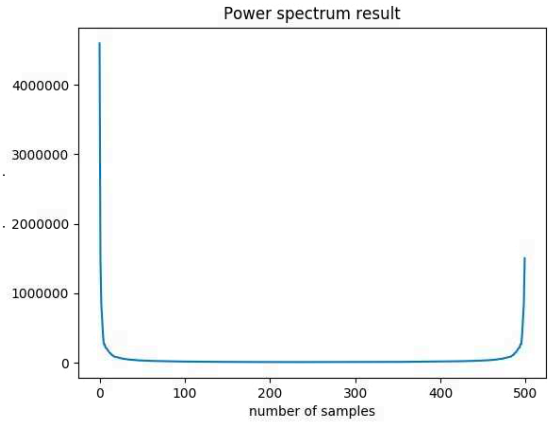


Fig.20. Power spectrum of 500 ADC DATA per sec

In addition, it also uses comps formula 4 to validate the result. The trend of this formula shows that when it adds the N (sample number), the comps result will closer to 0. That means if it keeps adding the sample rate, it is helpful for analysis the signal data. Make people react to more realistic data. This will make the PWM signal generation timelier and more realistic.

$$\eta = \frac{\Delta 1}{\Delta \Sigma} = \frac{\sum_{m=0}^{m_0} P(m)}{\sum_{m=0}^{N/2-1} P(m)}$$

Formula 4. f(x) for ADC linear equation

5. Conclusion

This project helps people learn how to get the data from ADC and compensate this data to driver motor stepper controlling. In this process, it shows how to build the ARM hardware environment and realize the whole by using python. Integrating different types of libraries to achieve the desired goals is also very important for the Python language. Furthermore, this project also needs to apply some mathematical formulas to the programming language. Even though these applications are basic, they are quite useful as a first step in using software to control hardware. This project allows people to experience the complete process of the real-world embedded system. It is a very important step for those who are just entering the field.

6. Acknowledgement

The completion of this project relies on Dr. Hua Harry Li step by step guidance. Professor has provided great help from the purchase of materials and the derivation of formulas. Moreover, group teammates also provide their advice and assistance. So, when the project encounters difficulties, some people can provide other ideas. This project could not do it without them.

7. References

[1] D. A. H. Samuelsen and O. H. Graven, "Low-cost multi-channel analog sampler and signal generator for remote **laboratories,"** *Proceedings of 2015 12th International Conference on Remote Engineering and Virtual Instrumentation (REV)*, Bangkok, 2015, pp. 100-104.

[2] https://en.wikipedia.org/wiki/Fast_Fourier_transform

[3] https://learn.adafruit.com/lsm303-accelerometer-slash-compass-breakout?view=all#calibration

8. Appendix

Item	Price
Raspberry pi B +	$ 38.1
ADS1115 ADC Converter	$ 6.99
Meter Board LSM303	$ 17.01
Motor driver board V44-A3967	$ 3.40
NEMA 17 stepper motor	$ 11.69
10 K Potentiometer	$ 6.54
Empty PCB board	$ 4.22
Stand off	$ 7.49
MicroSD card	$ 7.99
Cable	$ 10.99
Total price	$ 114.31

Appendix.1. Bill of material

```python
import time
import matplotlib.pyplot as plt
import numpy as np
import Adafruit_ADS1x15

data = Adafruit_ADS1x15.ADS1115()

GAIN = 1 # GAIN = 1 means voltage between +/-4.096V
l =[] # create empty list
print('Get the sampling data from ADC')
print('ADC channel A0'.format(*range(1)))
print('-' * 15)
N = 10 # Take the N numbers of sample data from ADC
for j in range(N):  # Main loop.

    values = [0] # Read A0 channel values in a list
    for i in range(1):

        values[i] = data.read_adc(i, gain=GAIN)
    # Read the specified ADC channel using the previously set gai


    l.append(values[i]) #add the values[i] into list

    print('| {0:>6}|'.format(*values))
    time.sleep(15) # Time of getting data


print(l)
l2=range(N)

plt.title('Data smaple result')
plt.xlabel('number of samples')
plt.ylabel('value of samples')
plt.plot(l)
plt.plot(l2, l,'ro')
plt.draw()
plt.pause(10)
plt.savefig("Data-figure.jpg")  #save picture
plt.close()   #close picture
```

Appendix.2. Code for getting ADC data

```python
import time
import numpy as np
import matplotlib.pyplot as plt
import Adafruit_ADS1x15
from scipy.fftpack import fft,ifft

adc = Adafruit_ADS1x15.ADS1115()

GAIN = 1 # GAIN = 1 means voltage between +/-4.096V
l =[] # create empty list
print('Get the sampling data From ADC')
print('| {0:>6}|'.format(*range(1)))
print('-' * 15)

for j in range(500): # Main loop.

    values = [0] # Read A0 channel values in a list
    for i in range(1):

        values[i] = adc.read_adc(i, gain=GAIN)

    l.append(values[i]) #add the values[i] into lis

    print('| {0:>6}|'.format(*values))
    time.sleep(0.002) # Time of getting data

lf = fft(l) # FFT l list
lf1 = abs(lf) # Power spectrum lf
plt.title('Power spectrum result')
plt.xlabel('number of samples')
plt.ylabel('value of power spectrum')
plt.plot(lf1)
plt.draw()
plt.pause(10)
plt.savefig("FFT-figure.jpg")
plt.close()
```

Appendix.3. Code for calculation of power spectrum

```python
#import numpy as np
import Adafruit_ADS1x15
import RPi.GPIO as GPIO
import board
import busio
import Adafruit_LSM303
import time

i2c = busio.I2C(board.SCL, board.SDA) # Get the data from LSM303
sensor = Adafruit_LSM303.LSM303(i2c)

Motor_PIN = 25 # choose GPIO 25 as our GPIO pin
PWM_FREQ = 200 # PWM setting on 200
GPIO.setmode(GPIO.BCM)
GPIO.setup(Motor_PIN, GPIO.OUT) # set the GPIO 25 as GPIO OUT
pwm = GPIO.PWM(Motor_PIN, PWM_FREQ) # import the PWM function
pwm.start(0) # PWM start

data = Adafruit_ADS1x15.ADS1115()

GAIN = 1 # GAIN = 1 means voltage between +/-4.096V
l =[] # create empty list
print('et the sampling data From ADC')
print('ADC    x-accel   y-accel'.format(*range(1)))
print('-' * 15)
while True: # keeping get data until ctrl + C
    accel, mag= sensor.read() # Get the mag and accel data
    accel_x, accel_y, accel_z= accel

    values = [0] # Read A0 channel values in a list
    for i in range(1):
        values[i] = data.read_adc(i, gain=GAIN)

    l.append(values[i]) #add the values[i] into list
    print('| {0:>6}| {1:>6}| {2:>6}|'.format(*values, accel_x, accel_y))
    #pwm.ChangeDutyCycle(50) # test code
    pwm.ChangeDutyCycle(int((values[1]/264))) # 26400 is the biggest data at 3.3v, 100 is max value for dutycycle
    pwm.ChangeFrequency(int((values[1]/264))*2+1) # +1 is because of frequency can't be 0
    #r = int(((values[1]/264))) # debug code
    #print(r) # debug code
    time.sleep(0.5) # Time of getting data
GPIO.cleanup()
```

Appendix.4. Code for motor speed control

```python
#import pigpio
#import numpy as np
import Adafruit_ADS1x15
import RPi.GPIO as GPIO
import time
import board
import busio
import Adafruit_LSM303
import math

i2c = busio.I2C(board.SCL, board.SDA) # Get the data from LSM303
sensor = Adafruit_LSM303.LSM303(i2c)

STEP_PIN = 25 # choose GPIO 25 as our STEP GPIO pin
DIR_PIN = 21 # choose GPIO 21 as our DIR GPIO pin
GPIO.setmode(GPIO.BCM)
GPIO.setup(STEP_PIN, GPIO.OUT,initial=GPIO.LOW)  # set the GPIO 25 as GPIO OUT and initial to low
data = Adafruit_ADS1x15.ADS1115()

GAIN = 1  # GAIN = 1 means voltage between +/-4.096V
l = []  # create empty list
print('Get the sampling data From ADC')
print('ADC      x        y        heading angle'.format(*range(1)))
print('-' * 40)

q = ((data.read_adc(0, gain=GAIN))//132) # q is for the first position revise.
if (q) >1:                               # The reason using 132 is full step, 26400/132 =200= 360 degree
        for j in range(q):
            GPIO.output(STEP_PIN, 1)  # duty cycle is 50%, since High and low are half half.
            time.sleep(0.0025)
            GPIO.output(STEP_PIN, 0)
            time.sleep(0.0025)

while True:  # keeping get data until ctrl + C

    accel, mag= sensor.read()
    mag_x, mag_y, mag_z= mag
    heading = (math.atan2(mag_y, mag_x)*180)/math.pi # Calculate the angle of the vector y,x
    if heading < 0:
        heading += 360 # Normalize to 0-360
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(STEP_PIN, GPIO.OUT,initial=GPIO.LOW)
    GPIO.setup(DIR_PIN, GPIO.OUT)
    #GPIO.output(24,dirm)

    values = [0]  # Read A0 channel values in a list

    for i in range(1):
        values[i] = data.read_adc(i, gain=GAIN)

    l.append(values[i])  # add the values[i] into list
    #print(values[i], angel, end='') # debug code
    print('| {0:>6}| {1:>6}| {2:>6}|  {3:>6}|'.format(*values, mag_x, mag_y, heading))


    if len(l) >1: # r is for the currently position detect.
        r = (((len(l)-1) - l[len(l)-2))//132)
        #print(r)
        if r > 0:
            for n in range(r):
                GPIO.output(DIR_PIN, 0)
                GPIO.output(STEP_PIN, 1)
                time.sleep(0.0025)
                GPIO.output(STEP_PIN, 0)
                time.sleep(0.0025)
        else:
            for n in range(abs(r)):
                GPIO.output(DIR_PIN, 1)
                GPIO.output(STEP_PIN, 1)
                time.sleep(0.0025)
                GPIO.output(STEP_PIN, 0)
                time.sleep(0.0025)
    GPIO.cleanup()
    time.sleep(1)  # Time of getting data
```

Appendix.5. Code for motor speed control