Linux based stepper motor driver control system with angular feedback

Sarika Satish Natu, 014529138
Electrical Engineering Department, College of Engineering
San Jose State University, San Jose, CA 94303
E-mail: sarikasatish.natu@sjsu.edu

Abstract

The aim of this exercise is to generate compensation function. ADC output is examined over complete dynamic range of 3.3V to 0V from analog signal from potentiometer and using Fast Fourier transform to validate the ADC data for any aliasing to rectify its sampling rate. This validated data is used to create a target angle for the stepper motor along with the PID control with feedback received from LMS303 magnetometer. The noise from the magnetometer is reduced by the technique of Gaussian filter with convolution. The system is achieved by interfacing MCP3008 ADC, LMS303 magnetometer and Easy driver -Nema17 stepper motor with Raspberry Pi 4 board.

1. Introduction

Raspberry Pi 4 is Quad core 64-bit ARM-Cortex A72 running at 1.5GHz with Broadcom BCM2835 SoC. Analog to digital conversion is accomplished by MCP3008 which has 10-bit resolution. MCP3008 communicates using SPI communication protocol. A trim-pot potentiometer ranging from 10 ohm – 2M ohm is used as resistive sensor to produce analog signal for the simulation. LMS303 is a magnetometer and it also has an acceleration sensor and temperature sensor on it. LMS303 interacts with Raspberry Pi using I2C communication protocol. The feedback provided by LSM303 sensor is used in computing the target angle using ADC data. The noise from the feedback is reduced by applying Gaussian filter technique to convolution. The analog signal from potentiometer is provided as input to ADC MPC3008. The digital output of MPC3008 ranging from 0-1023 provided using SPI to Raspberry Pi4 to plot characteristic linear curve. Linear interpolation function is used to achieve compensation function for MCP3008 ADC. This target signal is then provided as a step to the NEMA 17 stepper motor through the Easy driver. These steps are the PWM pulses provided to drive the motor and achieve the desired angle.

2. Methodology

The following sections elaborate on the technical challenges faced while attaining the project goal. It also states the procedures used for deriving compensation function, calculating power spectrum, using Gaussian filter along with convolution and PID control which a feedback loop mechanism.

2.1. Objectives and Technical Challenges

The objectives of the project are:
1. Interfacing MCP3008 ADC IC with Raspberry Pi use SPI communication protocol The ADC get analog signal from a potentiometer and converts it to digital output which is transmitted through SPI to Raspberry Pi.
2. Along with compensation function, Fast Fourier transform (FFT) is used to validate ADC digital values then validated output is used to compute target angle when potentiometer is rotated.
3. The other input to compute the target angle is obtained by using Gaussian filter to eliminate the noise from the LSM303 feedback. This works as an PID control loop for Raspberry Pi. Further depending on the error and target angle computation, the Raspberry Pi will rotate the motor according to the error through Easy driver.

The challenges faced during the development:
1. Setting up the LSM303 to sense the stepper motor motion.
2. Calculating the PID constants and using gaussian filtering with convolution.
3. Achieving a power spectrum and changing the sampling frequency which does not result in aliasing.

2.2. Problem Formulation and Design

This section provided the system design of the hardware layout of the components on the circuit board and problem formulations.

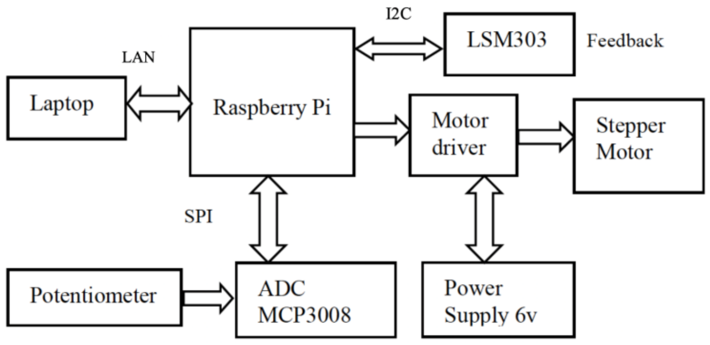


Figure 1 System Design

Mathematical formulation is provided below:

Compensation function:

error $\Delta = D_{actual} - D_{linear}$ … (1)

Accumulative error

$E = \Sigma(D_{actual}(i) - D_{linear}(i))^2$ where i = 1 to 10 … (2)

$(y_2 - y_1) / (y - y_1) = (x_2 - x_1) / (x - x_1)$ … (3)

$h(v) = a'v + b'$ … (4)

Power Spectrum using FFT:

By making use of FFT program the digital output is taken from ADC MCP3008 is validated. The magnitude is calculated using this data and the power spectrum is plotted.

The Discrete Fourier Transform can be expressed as

$$\mathbf{F}(n) = \sum_{k=0}^{N-1} f(k)\, e^{-j2\pi nk/N} \qquad (n = 0..1\ :N-1)$$

The relevant inverse Fourier Transform can be expressed as

$$f(k) = \frac{1}{N}\sum_{n=0}^{N-1} \mathbf{F}(n)\, e^{j2\pi nk/N} \qquad (k = 0, 1, 2..., N-1)$$

PID control using Gaussian low pass filter:

PID (Proportional, Integral, Derivative) control is implemented to compute the error received from LMS303 magnetometer sensor.

U(t)= output,

E(t)= error value

PID makes use of three constant parameters to decrement the error magnitude as fast as possible.

K_p= proportional constant that counts for present error.

K_i= Integral error that accounts for historical error values.

K_d= Derivative constant that accounts for future error values.

Note: This project uses square of e(t) in integral term and for derivative, project does central differentiation.

Central difference:

The central difference convolution with Gaussian Filter is used to reduce the noise. The formula is as follows:

CD=½ *[e(t) - e(t-2)]

Gaussian Low Pass Filter:

$$g(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}.$$

Here, the sigma is user defined and y = 0 as its 1-D function.

3. Implementation

The hardware and software design to interface Raspberry Pi and ADC MCP3008, Easy driver and magnetometer sensor LMS303.

3.1. Hardware Design

Provide the following description and discussion:

1. System block diagram.

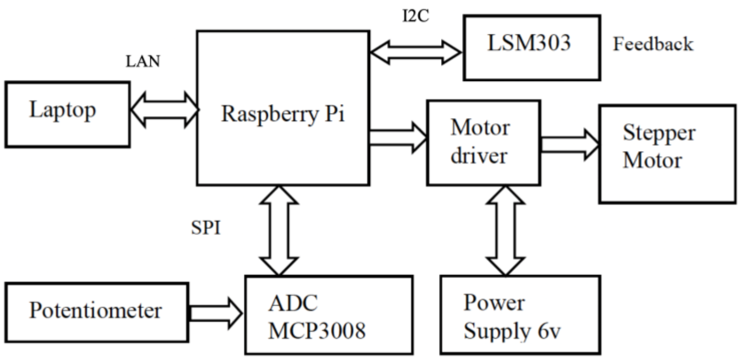


Figure 2 Block Diagram

2. Schematic design.

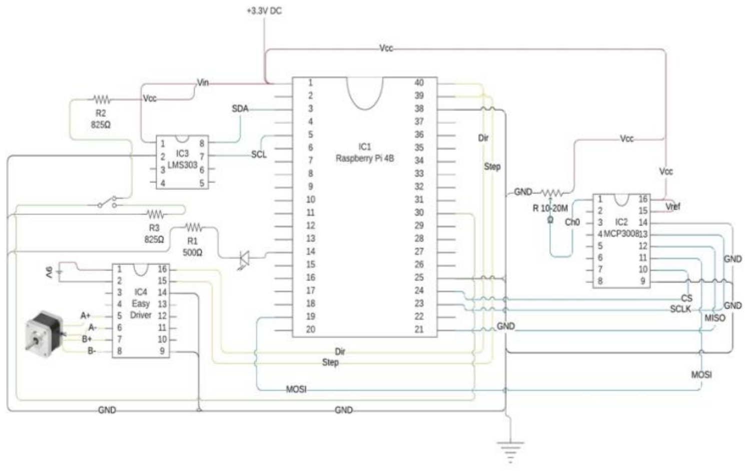


Figure 3 Schematic Design

3. Pictorial form

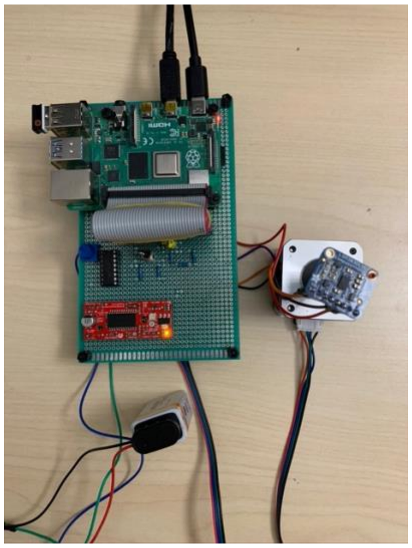


Figure 4 Actual Hardware Interfacing

The pin connections are as following:

1. For ADC – Hardware SPI

Pin	Raspberry Pi	Pin	MCP3008
1	3V3	16	VDD
1	3V3	15	VREF
6	GND	14	AGND
6	GND	9	DGND
23	SPI0_SCLK	13	SCLK
21	SPI0_MISO	12	MISO
19	SPI0_MOSI	11	MOSI
24	SPI0_CE0_N	10	CE
Pot2	Potentiometer	1	CH0

Table 1 Pin connections for MCP3008

2. For Easy motor driver

Pin	Raspberry Pi	Easy Driver
	M+(Battery)	M+
	GND(Battery)	GND
40	GPIO 21	Dir
38	GPIO 20	Step
6	GND	GND(2)
	Stepper Motor	A+ A- B+ B-

Table 2 Easy motor driver connections

3. For LMS303 – I2C

Pin	Raspberry Pi	LMS303
1	3V3	Vin
6	GND	GND
3	SDAI2C	SDA

Table 3 LMS303 Pin connections

4. For other GPIO

Pin	Raspberry Pi
16	LED
14	GND(Led)
22	GPIO 25
30	GND (switch)

Table 4 GPIO circuit

3.1.2 Bill of material (list of components).

Table 5 Bill of material

Item no	Description
1	Wire wrapping board(10x10″)
2	Raspberry Pi 4
3	MCP3008
4	NEMA 17 stepper motor
5	Easy Driver
6	LMS303 Magnetometer
7	9V Battery
8	2*8 16 pin IC socket
9	Potentiometer
10	Header strips (Male and Female)
11	LAN cable
12	Breadboard
13	Soldering gun
14	Solder
15	De-soldering Pump
16	LEDs
17	Resistors

3.2. Software Design

This section describes the flow chart, algorithm and the pseudo code for the software implementation

1. Flow chart

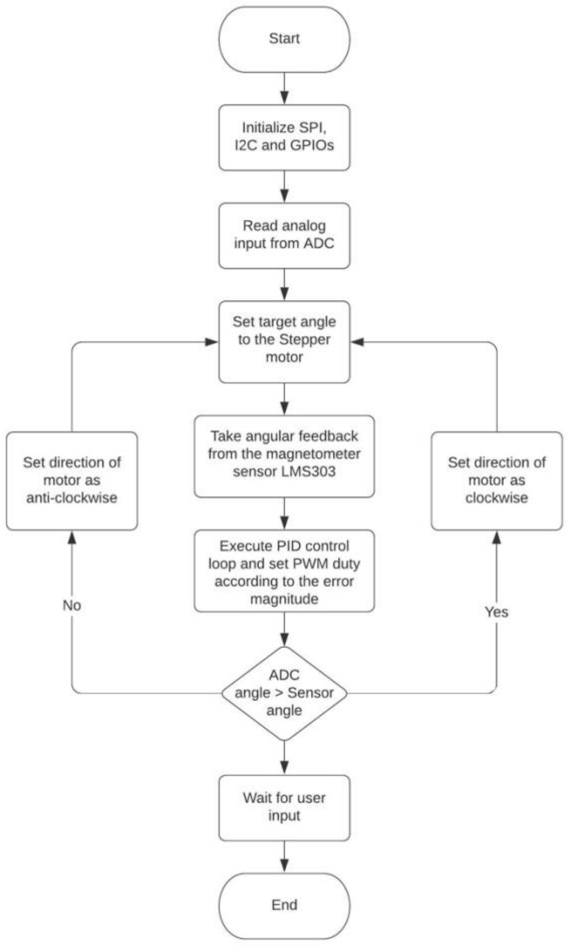

Start
Initialize SPI, I2C and GPIOs
Read analog input from ADC
Set target angle to the Stepper motor
Take angular feedback from the magnetometer sensor LMS303
Execute PID control loop and set PWM duty according to the error magnitude
Set direction of motor as anti-clockwise
Set direction of motor as clockwise
No
Yes
ADC angle > Sensor angle
Wait for user input
End

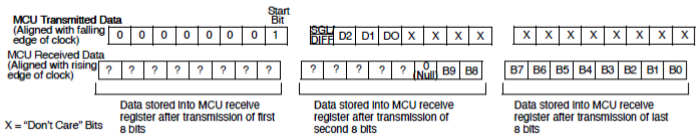
Figure 5 Flow Chart

2. Algorithm
    1. Initialize SPI and set SPI speed
    2. Initialize I2C bus
    3. Configure GPIOs
    4. Send SPI request frame to read ADC channel 1
    5. Extract raw ADC data from the received SPI response
    6. Convert to ADC data to voltage
    7. Convery angle into voltage
    8. Read LSM303 magnetometer data to receive feedback
    9. Convert micro- Teslas into angle(degrees)
    10. Take difference between ADC angle data and magnetometer angle data
    11. Use this error to set the PWM duty cycle
    12. Provide this to stepper motor
    13. Repeat steps 5 to 12 until error is zero
        y-axis = ADC data

3. Pseudo code

ADC:

```
1.  Send SPI request frame [0x01, 0x80, 0x00]
2. Extract raw ADC data:
   10bit ADC => last 2 bits of 2nd byte + 8 bit 3rd byte
```


MCU Transmitted Data (Aligned with falling edge of clock)
Start Bit
0 0 0 0 0 0 0 1
SGL/DIFF D2 D1 DO X X X X
X X X X X X X X
MCU Received Data (Aligned with rising edge of clock)
? ? ? ? ? ? ? ?
? ? ? ? ? 0 (Null) B9 B8
B7 B6 B5 B4 B3 B2 B1 B0
X = "Don't Care" Bits
Data stored into MCU receive register after transmission of first 8 bits
Data stored into MCU receive register after transmission of second 8 bits
Data stored into MCU receive register after transmission of last 8 bits

```
3. Convert to voltage = 10bit data * 3.3V /1023bits
4. Convert to angle = ADC voltage / 3.3V * 360degrees
```

LMS303:

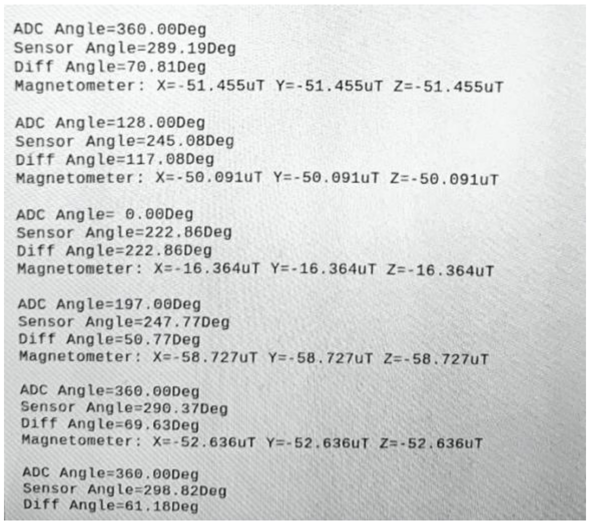
```
1. Setup I2C channel
2. Read magnetometer data
3. Convert micro Teslas to degrees =
   math. atan2(mag_x, mag_y)
       if heading < 0:
           heading += 2*math.pi
       if heading > 2*math.pi:
           heading -= 2*math.pi
   headingDegrees=round(math.degrees(heading),2)
```

Stepper motor:

```
1. Convert angle into duty cycle
2. If ADC angle > LMS angle = Forward motion
3. If ADC angle < LMS angle = Backward motion
```

4. Testing and Verification

The results of successful compilation of the experiment are below:



```
ADC Angle=360.00Deg
Sensor Angle=289.19Deg
Diff Angle=70.81Deg
Magnetometer: X=-51.455uT Y=-51.455uT Z=-51.455uT

ADC Angle=128.00Deg
Sensor Angle=245.08Deg
Diff Angle=117.08Deg
Magnetometer: X=-50.091uT Y=-50.091uT Z=-50.091uT

ADC Angle= 0.00Deg
Sensor Angle=222.86Deg
Diff Angle=222.86Deg
Magnetometer: X=-16.364uT Y=-16.364uT Z=-16.364uT

ADC Angle=197.00Deg
Sensor Angle=247.77Deg
Diff Angle=50.77Deg
Magnetometer: X=-58.727uT Y=-58.727uT Z=-58.727uT

ADC Angle=360.00Deg
Sensor Angle=290.37Deg
Diff Angle=69.63Deg
Magnetometer: X=-52.636uT Y=-52.636uT Z=-52.636uT

ADC Angle=360.00Deg
Sensor Angle=298.82Deg
Diff Angle=61.18Deg
```

Figure 6 Stepper motor control using feedback mechanism python file

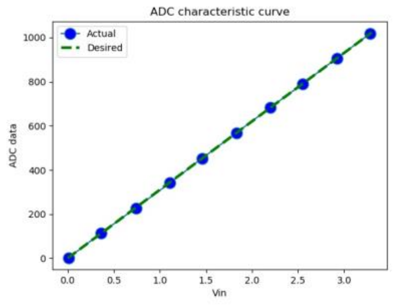


Figure 7 ADC Characteristic curve plot

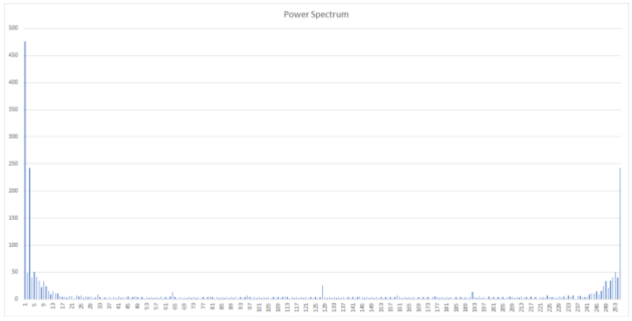


Figure 8 Power Spectrum

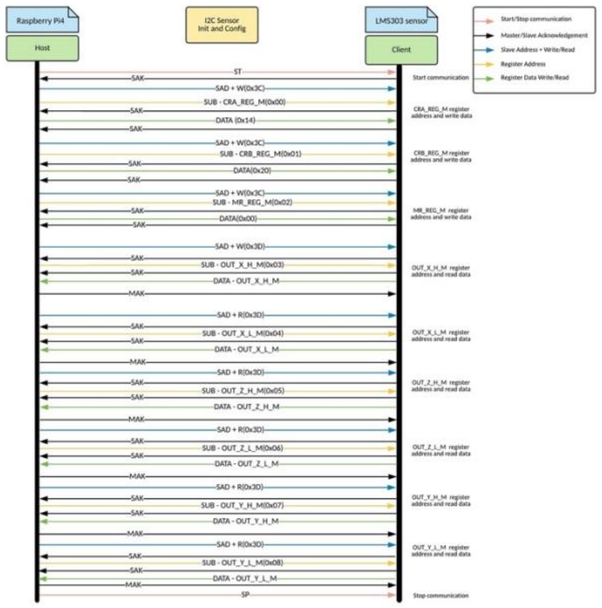


Figure 9 LMS303-I2C interfacing

5. Conclusion

The objective of the project is achieved successfully. From using compensation function to rectify errors in ADC data to validate ADC data using FFT and plotting the power spectrum was accomplished. Later using PID and Gaussian filtering function was used to reduce noise in error for LMS303 sensor data and this error for then fed to the stepper motor as target angle to control it.

6. Acknowledgement

I would like to thank Dr. Harry Li for teaching the concepts of linear interpolation and FFT, DFT, compensation function, Gaussian filtering, and PID control. I could see applying these concepts taught while implementing the project. Also, his guidance and support throughout the project was of great help.

7. References

[1] https://github.com/hualili/CMPE242-Embedded-Systems-/tree/master/2019S
[2] https://learn.adafruit.com/reading-a-analog-in-and-controlling-audio-volume-with-the-raspberry-pi/connecting-the-cobbler-to-a-mcp3008
[3] https://pimylifeup.com/raspberry-pi-adc/
[4] http://abyz.me.uk/rpi/pigpio/
[5] https://learn.adafruit.com/lsm303-accelerometer-slash-compass-breakout/coding

8. Appendix

Source code: Steppermotor_Fbk_ctrl.py

```
import pigpio
import time, sys
import spidev
from time import sleep
import os
import board
import busio
import adafruit_lsm303_accel
import adafruit_lsm303dlh_mag
import math

stp = 20                              #GPIO_20
dir = 21                              #GPIO_21
Boardpin = 23                         #GPIO_23
pot_channel = 0                       #ADC channel
sleepTime = 1

spi = spidev.SpiDev()                 #setup SPI bus for
communicating with MCP3008 ADC
spi.open(0,0)
spi.max_speed_hz=1000000

i2c = busio.I2C(board.SCL, board.SDA)      #setup I2C
configuration
mag = adafruit_lsm303dlh_mag.LSM303DLH_Mag(i2c)
#read LMS303 magnetomoter data


pi = pigpio.pi()                      #setup dir and step as
output  for Easy driver
pi.set_mode(dir, pigpio.OUTPUT)
pi.set_mode(stp, pigpio.OUTPUT)

def move(duty, direction):            #stepping motor
motion function
   pi.write(dir, direction)
   pi.set_PWM_dutycycle(stp, duty)
   pi.set_PWM_frequency(stp, 500)
   sleep(.05)

def readADCch(channel):               #Read ADC data
from MCP008
   adc = spi.xfer2([1,(8+channel)<<4,0])
   data = ((adc[1]&3) << 8) + adc[2]
   return data

def convertDuty(angle):               #Convert target angle
into duty cycle
   duty = (angle) / float(360)
   duty = duty * 255 #pwm full cycle
   duty = int(duty)
   return duty

def convertVoltage(bitValue):         #Convert ADC
data into voltage
   voltage = (bitValue * 3.3) / float(1023)
   voltage = round(voltage, 2)
   return voltage

def convertAngle(vtg):                #Convert ADC
voltage into angular form(degrees)
   angle = (vtg) / float(3.3)
   angle = angle * 360
   angle = int(angle)
   return angle

def getHeading(mag_x, mag_y):         #Convert
LMS303 sensor output micro-Teslas into degrees
      heading = math.atan2(mag_x, mag_y)
      if heading < 0:
         heading += 2*math.pi
      if heading > 2*math.pi:
         heading -= 2*math.pi
      headingDegrees = round(math.degrees(heading),2)
      return headingDegrees

while True:
   bitData = readADCch(pot_channel)       #get ADC data
   vtgData = convertVoltage(bitData)      #convert ADC
data into voltage
   angleDataAdc = convertAngle(vtgData)    #convert
ADC voltage into degrees

   mag_x, mag_y, mag_z = mag.magnetic      #read data
from LMS303 magnetic sensor
   angleDataSen = getHeading(mag_x, mag_y) #convert
sensor data into degrees

   angleDiff = abs(angleDataAdc - angleDataSen)

   print("ADC
Angle={0:5.2f}Deg".format(angleDataAdc))
   print("Sensor
Angle={0:5.2f}Deg".format(angleDataSen))
   print("Diff Angle={0:5.2f}Deg".format(angleDiff))

   dutyData = convertDuty(angleDiff)       #convert angle
into duty cycle to generate PWM signal
   if (angleDataAdc > angleDataSen):
         sleep(sleepTime)
         move(dutyData, 1)            #move stepper motor
in clockwise direction
   elif (angleDataAdc < angleDataSen):
         sleep(sleepTime)
         move(dutyData, 0)            #move stepper motor
in anti-clockwise direction
   else: print("else block")
```

```
    print("Magnetometer: X={0:7.3f}uT Y={0:7.3f}uT
Z={0:7.3f}uT".format(mag_x, mag_y, mag_z))
    print("")
```