ADC-PWM-PID interface with LPC Node and ARM11

Saurabh Deshmukh (011435060), Nikhil Namjoshi (011432213),
Samiksha Ambekar (010974587), Rimjhim Ghosh (011416002)
Computer Engineering Department, Charles W. Davidson College of Engineering San Jose State University, San Jose, CA 95112
Email: saurabhravindra.deshmukh@sjsu.edu, nikhil.namjoshi@sjsu.edu, samiksha.ambekar@sjsu.edu, rimjhim.ghosh@sjsu.edu

Abstract

This paper describes the design and development details of hardware and software components requirement and implementation of the interface of LSM303 sensor, SG90 and ADC with Samsung S3C6410 ARM-11 board and LPC1758 using I2C and UART protocol. LPC1758 and ARM11 is communication via UART protocol and the sensor is connected to nod using I2C protocol and ADC is connected to LPC175 and validate the sampling frequency on ARM11.

Keywords: PID, UART, I2C, PWM, ADC.

1. Introduction

The Agenda of this project is Interface LSM303 with LPC 1758 and make connection with ARM11 which will work as server using UART protocol and use Servo Motor to adjust the PWM in LPC1758 side so that the servo will always point to the first direction which it was pointing before. And in second part of Project we are going to Interface Potentiometer with LPC1758 and getting ADC readings and sending it to server ARM11 to validate the sampling frequency.

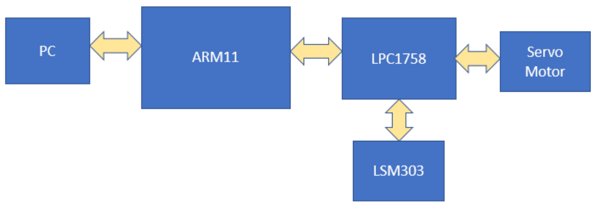


Fig 1. Block diagram of sensor and motor interface

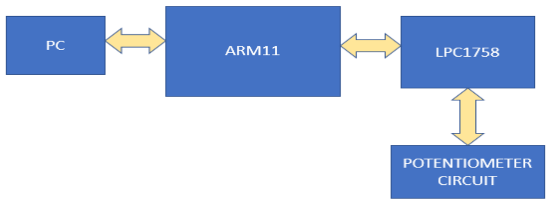


Fig 2. Block diagram potentiometer interface

2. Methodology

We have used Proportional Integral Derivative(PID) closed loop feedback system, to stable the servo motor and point it to fix direction and used power spectrum plot to validate the sampling frequency chosen by us.

2.1. Objectives and Technical Challenges

The Objective of this project is Learning implementation of PID controller and how to Validate Sampling Frequency using Power Spectrum Plotting. And then change it so that we can Regenerate Signal without losing much information.

Technical challenges in this project is complete the individual modules and perform integration by using communication protocol.

2.2. Problem Formulation and Design

The Problem Statement of our project is mostly divided into two parts -

- Interface Potentiometer with ADC in LPC1758 and Establish communication of LPC1758 with ARM11 to validate the Sampling frequency by plotting Power Spectrum on the ARM11 Server side.
- Interface LSM303 sensor to get values from Magnetometer and send it to ARM11 and Implement PID on ARM11 side and send the Value of PWM to LPC1758 again which will drive servo motor to point it to the first direction so that even after movement of board the Sensor and the servo will point to that direction only.

3. Implementation

The requirements are divided into 2 parts – hardware and software side.

Hardware requirements:

1. ARM11 Board
2. LSM303
3. ADC
4. LPC1758

5. SG90 servo motor

Software requirements:
1. Linux (Ubuntu Host)
2. ARM Embedded Linux (2.6.38)
3. ARM Linux Toolchain
4. Minicom

3.0.1 ADC interfacing and validation

A potentiometer, informally a pot, is a three-terminal resistor with a sliding or rotating contact that forms an adjustable voltage divider. If only two terminals are used, one end and the wiper, it acts as a variable resistor or rheostat.

This ADC is connected to LPC1758 that sends 10 values to ARM11 board which validates the data using power spectrum.

Power Spectrum Equation:

$$P(n) = \sqrt{Real(X(n))^2 + Img(X(n))^2}$$

Fig 3. ADC

3.0.2 Sensor and Motor interfacing

We have used LSM303 sensor and SG90 motor connected to LPC1758. Data is collected from sensor and send to ARM11 via LPC1758 which sends back PWM after the evaluation of PID.

PID: A proportional–integral–derivative controller (PID controller) is a control loop feedback mechanism (controller) commonly used in industrial control systems. A PID controller continuously calculates an error value as the difference between a desired set point and a measured process variable and applies a correction based on proportional, integral, and derivative terms (sometimes denoted P, I, and D respectively) which give their name to the controller type.

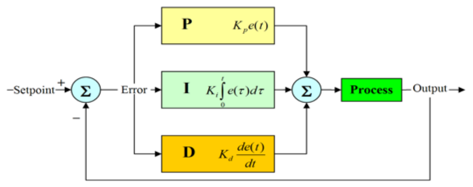
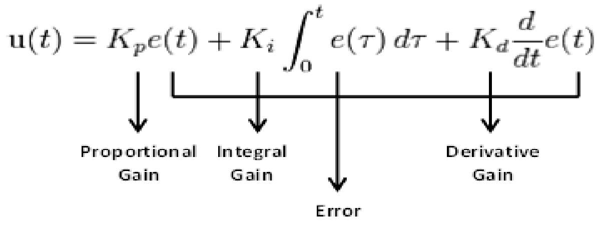


Fig 4. PID Block diagram

PID Equation:



$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\,d\tau + K_d \frac{d}{dt} e(t)$
Proportional Gain
Integral Gain
Error
Derivative Gain

3.1. Hardware Design

The entire hardware design is divided to two parts i.e ADC and LSM303 and SG90 interface with ARM 11 and LPC1758 respectively.

3.1.1 LSM 303 Sensor

The following configuration to be followed for setting the control register of the LSM303 for enabling Linear Acceleration.

Fig 5. LSM 303 Sensor

The LSM303DLHC has linear acceleration full scales of ±2g / ±4g / ±8g / ±16g and a magnetic field full scale of ±1.3 / ±1.9 / ±2.5 / ±4.0 / ±4.7 / ±5.6 / ±8.1 gauss. The LSM303DLHC includes an I2C serial bus interface that supports standard and fast mode 100 kHz and 400 kHz.

3.1.2 SG90 servo motor

It comes with a 3 horns (arms) and hardware. The shaft moves through 180 degrees on change in PWM.

Fig 6. SG90 servo motor

3.1.3 ADC interfacing and validation

We built a board that interfaced ADC with LPC1758 and ARM11 board. LPC node and ARM11 communicated via UART. The ADC is connected to LPC1758 that sends value to ARM11 which further evaluated and validates the data using Power Spectrum.

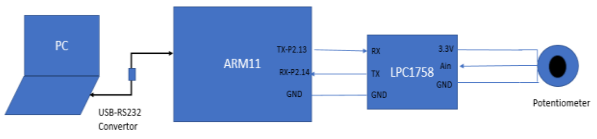


Fig 7. Schematic of ADC Interface

3.1.4 Sensor and Motor interfacing

We built a board that interfaced LSM303 and SG90 with LPC1758 and ARM11 board. LSM303 and SG90 is connected to LPC1758 which is further connected to ARM11 board. The sensor takes the values and sends to LPC1758 which further send to ARM11 for the evaluation of PID and sending the appropriate PWM to motor.

Component	Description	Count
Tiny 6410 ARM	ARM11 board	1
Power Adapter	110V Ac to 5V DC	1
LSM303	Compass	1
Servo Motor	SG90	1
LPC1758	SJOne	1
FTDI cable	USB to serial converter	1

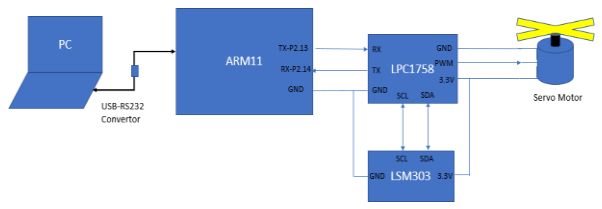


Fig 8. Schematic of Sensor and Motor interface

3.2. Software Design

Software part for ADC and PWM are independent of each other. Implementation of a program to get magnetic sensor value is done on the LPC node side that then sends value to ARM11 for the evaluation of PID. Similarly, for ADC data validation potentiometer is connected to LPC1758 that send value to ARM11 for validating the sampling frequency.

Component	Description	Count
Tiny 6410 ARM	ARM11 board	1
Power Adapter	110V Ac to 5V DC	1
ADC	Potentiometer	1
LPC1758	SJOne	1

3.2.1 ADC interfacing and validation

We have connected the ADC on the LPC node side wherein we are at first setting up the pin on LPC1758 to set it as the AIN input pin.

```
LPC_PINCON->PINSEL1 |= (1 << 20);
```

Then we are continuously capturing the values and sending it to ARMSVR for the validation of sampling frequency via UART.

```
while(1)
{
   int adc3 = int(floor(adc0_get_reading(3))/4) ;

   if(adc3 > 0 && adc3 < 1023)
      adc3 = adc3 - 2;
      sprintf(str,"%d",adc3);
      int fd = atoi(str);
      printf("adc3 = %d\n ", fd);
      for(int i = 0; i < 20; i++)
        Uart2::getInstance().putChar(str[i],20);
       }
```

Compensation Function: adc3 = adc3 - 2;

Taking forward we capture values and send to ARM11 via UART which then takes 10 values and compute the power spectrum by just calling the FFT function.

We open the file and start the UART communication.

```
fd = open("/dev/ttySAC3", O_RDWR);
```

Continuously we read the values from the uart pin coming from the ADC convert them to integer but atlast work with only 10 values and apply the FFT in them.

```
while(1)
{
        write_bytes = read(fd,buf1,sizeof(buf1));
        value = atoi(buf1);
        arr[i] = (float) value;

        i++;
 if(i == 10 )
{
                for (j = 0; j < 10; j++)
                {
                X[j].a = arr[j];
                X[j].b = 0.0;
                }
FFT();
i = 0;
}
}
```

3.2.2 Sensor and Motor interfacing

For sensor and motor interfacing we have connected sensor and motor on the LPC1758 side that collects the data from the sensor then it sends the value to the ARM11 board that then calculates the PWM based on PID evaluation and sends back to LPC node for the servo motor to rotate.

Reading value of sensor send by LPC node via UART

```
write_bytes = read(fd,buf1,sizeof(buf1));
compass_reading = atof(buf1);
```

Then PWM is calculated using PID

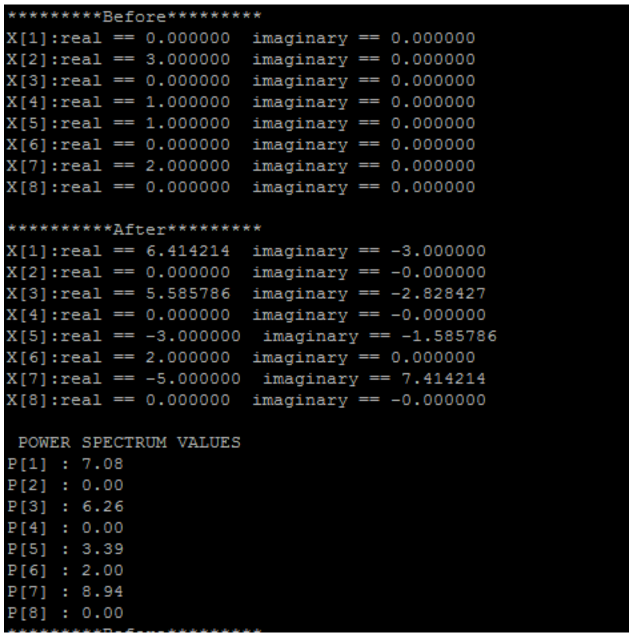
```
float error;
float P = error;
float D = (error - lasterror)/0.01;
float I = 0;
      for(i = 0; i < 5; i++)
        {
                I_error[i] = I_error[i + 1];
                }
                I_error[5] = error;
                for(i = 0; i < 6 ; i++)
                {
                        I +=  I_error[i] * I_error[i];
                }
                I = I * 0.01;
                error = 0 - compass_reading;
                sumError = kp * P + kd * D + ki * I;
                value1 = 0.053 * error;
```

Sending PWM to LPC1758

```
bytes_w = write(fd,&pwm[0],sizeof(pwm[0]));
```

4. Testing and Verification

We have interfaced potentiometer with 10-bit ADC on LPC1758 and used it at 10 Hz Sampling Frequency which results to 10 points ADC conversion per second and calculating FFT and power spectrum using FFT.c provided.



```
*********Before*********
X[1]:real == 0.000000  imaginary == 0.000000
X[2]:real == 3.000000  imaginary == 0.000000
X[3]:real == 0.000000  imaginary == 0.000000
X[4]:real == 1.000000  imaginary == 0.000000
X[5]:real == 1.000000  imaginary == 0.000000
X[6]:real == 0.000000  imaginary == 0.000000
X[7]:real == 2.000000  imaginary == 0.000000
X[8]:real == 0.000000  imaginary == 0.000000

*********After*********
X[1]:real == 6.414214  imaginary == -3.000000
X[2]:real == 0.000000  imaginary == -0.000000
X[3]:real == 5.585786  imaginary == -2.828427
X[4]:real == 0.000000  imaginary == -0.000000
X[5]:real == -3.000000  imaginary == -1.585786
X[6]:real == 2.000000  imaginary == 0.000000
X[7]:real == -5.000000  imaginary == 7.414214
X[8]:real == 0.000000  imaginary == -0.000000

 POWER SPECTRUM VALUES
P[1] : 7.08
P[2] : 0.00
P[3] : 6.26
P[4] : 0.00
P[5] : 3.39
P[6] : 2.00
P[7] : 8.94
P[8] : 0.00
```

Fig 9. Output of ADC

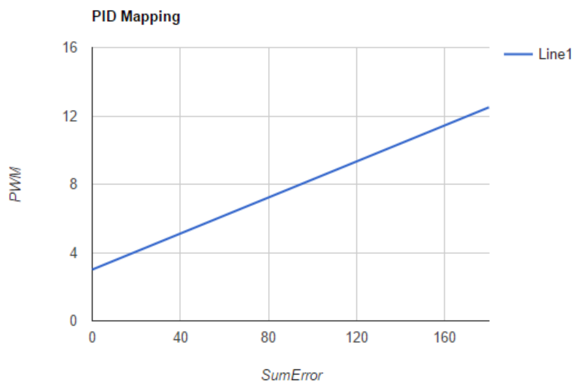


Fig 10. PID Mapping Graph

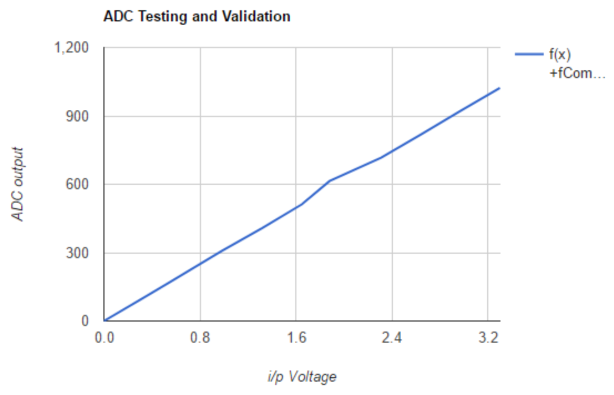


Fig 11. Testing and Validation Graph

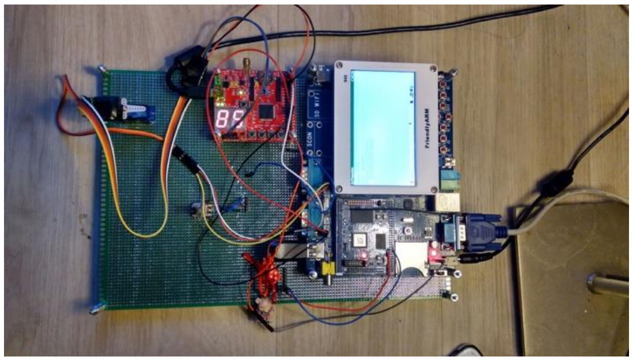


Fig 12. Board Setup

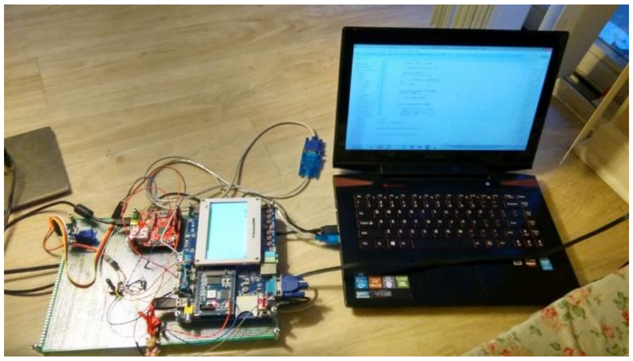


Fig 13. Entire Setup for the project

5.Conclusion

We successfully collected data from ADC and validated data using power spectrum. Then again, we collected sensor data from LSM303 and send to ARMSVR via LPC1758 and respectively send the PWM from ARM11 to LPC1758 that was given to SG90 after calculating PID in ARM11.

6. Acknowledgement

I would like to thank Dr Harry Li for explaining the various steps in the project in detail.

7. Individual Contribution

Nikhil – PID tuning & ADC on LPC Node
Saurabh- SG90 servo motor and ADC Data Validation
Rimjhim- LSM303 & SG90 servo motor
Samiksha- PID tuning & UART

8. References

1. H.Li, Guidelines for CMPE242 project and report, Computer Engineering Department, San Jose State University, San Jose 95112
2. H.Li, CMPE242 Lecture Notes, Computer Engineering Department, San Jose State University, San Jose 95112
3. S3C6410 User manual
4. LSM 303 DHL datasheet
5. SG 90 servo motor datasheet
6. ADC Datasheet

9. Appendix

[A] Source Code – ADC Interface

LPC1758:

```
char str[20];

 int main(void)
 {

              Uart2::getInstance().init(115200);
             LPC_PINCON->PINSEL1 |= (1 << 20);
             while(1)
             {
                          int adc3 = 1023 -
int(floor(adc0_get_reading(3))/4) ;
                          if(adc3 > 0 && adc3 < 1023)
                                       adc3 = adc3 - 2;
                          sprintf(str,"%d",adc3);
                          int fd = atoi(str);
                          printf("adc3 = %d\n ", fd);
                          for(int i = 0; i < 20; i++)
                           Uart2::getInstance().putChar(str[i],20);
             }
}
```

ARM11:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<unistd.h>
#include<linux/fs.h>
#include<string.h>
#include <fcntl.h>
#include<sys/stat.h>
```

```
#include <math.h>

struct Complex
{           double a;        //Real Part
            double b;        //Imaginary Part
}         X[5], U, W, T, Tmp;

void FFT(void)
{
            int M = 3;
            int N = pow(2, M);

            int i = 1, j = 1, k = 1;
            int LE = 0, LE1 = 0;
            int IP = 0;


            for (k = 1; k <= M; k++)
            {
                        LE = pow(2, M + 1 - k);
                        LE1 = LE / 2;

                        U.a = 1.0;
                        U.b = 0.0;

                        W.a = cos(M_PI / (double)LE1);
                        W.b = -sin(M_PI/ (double)LE1);

                        for (j = 1; j <= LE1; j++)
                        {
                                    for (i = j; i <= N; i = i + LE)
                                    {
                                                IP = i + LE1;
                                                T.a = X[i].a +
X[IP].a;
                                                T.b = X[i].b +
X[IP].b;
                                                Tmp.a = X[i].a -
X[IP].a;
                                                Tmp.b = X[i].b -
X[IP].b;
                                                X[IP].a = (Tmp.a *
U.a) - (Tmp.b * U.b);
                                                X[IP].b = (Tmp.a *
U.b) + (Tmp.b * U.a);
                                                X[i].a = T.a;
                                                X[i].b = T.b;
                                    }
                                    Tmp.a = (U.a * W.a) - (U.b *
W.b);
                                    Tmp.b = (U.a * W.b) + (U.b *
W.a);
                                    U.a = Tmp.a;
                                    U.b = Tmp.b;
                        }
            }

            int NV2 = N / 2;
            int NM1 = N - 1;
            int K = 0;

            j = 1;
            for (i = 1; i <= NM1; i++)
            {
                        if (i >= j) goto TAG25;
                        T.a = X[j].a;
                        T.b = X[j].b;

                        X[j].a = X[i].a;
                        X[j].b = X[i].b;
                        X[i].a = T.a;
                        X[i].b = T.b;
TAG25: K = NV2;
TAG26: if (K >= j) goto TAG30;
                        j = j - K;
                        K = K / 2;
                        goto TAG26;
TAG30: j = j + K;
            }
}

int main(void)
{

            int fd;
            int bytes_recieved;
            int write_bytes;
            int i = 0 ;
            int value;
            int count = 0;
            float arr[10];
            int j = 0;

            char buf1[20] ;//= { '0' , '0' ,'0' , '0'};

             fd = open("/dev/ttySAC3", O_RDWR);
            printf("fd value id %d\n",fd);

            if(fd < 0)
            {
                         perror("open device serial uart");
              exit(1);
            }
             while(1)
            {
                        write_bytes =
read(fd,buf1,sizeof(buf1));

                        value = atoi(buf1);
                        arr[i] = (float) value;
                        //printf("va;ue[%d] = %0.2f \n",i,arr[i]);
                        i++;
                        if(i == 10 )
                        {
                                    for (j = 0; j < 10; j++)
                                    {
                                                X[j].a = arr[j];
```

```
                                        //printf("arr[%d] =
%0.2f \n",j,arr[j]);
                                        //printf ("X[%d]:real
== %f  imaginary == %f\n", j, X[j].a, X[j].b);
                                        X[j].b = 0.0;
                                }

                                /*
                                for( j = 0; j < 10; j++ )
                                {
                                        printf("array is %0.2f
and value of j is %d HI \n",arr[j],j);

                                }*/


                                printf
("*********Before*********\n");
                                        for (j = 1; j <= 8; j++)
                                                printf
("X[%d]:real == %f  imaginary == %f\n", j, X[j].a,
X[j].b);
                                FFT();
                                printf
("\n\n**********After*********\n");
                                        for (j = 1; j <= 8; j++)
                                                printf
("X[%d]:real == %f  imaginary == %f\n", j, X[j].a,
X[j].b);
                                i = 0;
                        }



        //      printf("data recieved is %d\n",value);
        }



        close(fd);
        return 0;
}
```

[B] Source Code – Sensor and Motor Interface and PID

LPC1758:

```
double compass_reading;
int compass_last;
double final_Reading  = 0;
uint8_t data;
uint8_t compass[2] = {0};
char reg_address ; //command register
char dev_address= 0xC0 ; //device address
uint16_t compass1;
uint16_t compass2;
double turn_angle1;
double turn_angle2;
int count1 = 0;
char chDat[6] = "";
char buff[20] = "";
char ch;

static PWM motor(PWM::pwm1,50);
                float dataSend = 0;
                reg_address = 0x02;

        I2C2::getInstance().readRegisters(dev_address,
reg_address, compass, 16);
                compass1 = ((compass[0]<<8) |
(compass[1] & 0xFF));
                compass_last = compass1%10;
                compass2 = (compass1/10) ;
                final_Reading = (int)compass2 + 0.1 *
compass_last;
                //printf("Read %0.2f\n",final_Reading);


                if(final_Reading > 180)
                {
                        dataSend = 360 -
final_Reading;
                }
                else
                {
                        dataSend = - final_Reading;
                }

        sprintf(buff,"%0.2f",dataSend);

                delay_ms(20);
                printf("Final Reading: %s \n", buff);

                Uart2::getInstance().putline(buff,20);


        if(Uart2::getInstance().getChar(&ch,10))
                {
                        chDat[count1] = ch;
                        //printf("Char %c \n",ch);

                        count1++;
                        if(count1 == 6)
                        {

                                pwm_value =
atof(chDat);

        motor.set(pwm_value);
                                printf("PWM %0.2f
\n\n\",pwm_value);
                                for(int i = 0; i < 6;
i++)
                                        chDat[i]= ' ';
                                count1 = 0;
                        }
                }
```

ARM11:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<unistd.h>
#include<linux/fs.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <math.h>
#include <stdio.h>

void delay_ms(unsigned int count)
{

        count = count + 4000000;
        while(count--);

}
int main(void)
{

        int fd;
        int bytes_recieved;
        int write_bytes;
        int bytes_w;
        int i = 0 ;
        int value;
        float compass_reading;
        int count = 0;
        float arr[10];
        int j = 0;
        float error;
        float value1;
        int setpoint = 0;
        char pwm[6] = "";
        float I_error[6] = {};
        float lasterror = 0;
        float kp = 0.5;
        float kd = 0.1;
        float ki = 0.09;
        float sumError = 0;

        char buf1[20] ;//= { '0' , '0' ,'0' , '0'};

         fd = open("/dev/ttySAC3", O_RDWR);
        printf("fd value id %d\n",fd);

        if(fd < 0)
        {
                perror("open device serial uart");
          exit(1);
        }
         while(1)
        {
                sleep(0.2);
                setpoint = 0;
                        write_bytes =
read(fd,buf1,sizeof(buf1));
                        compass_reading = atof(buf1);


                        printf("value read is %0.2f\n
",compass_reading);

                        float error;
        float P = error;
                        float D = (error - lasterror)/0.01;
                        float I = 0;
                        for(i = 0; i < 5; i++)
                        {
                                I_error[i] = I_error[i + 1];
                        }
                        I_error[5] = error;
                        for(i = 0; i < 6 ; i++)
                        {
                                I +=  I_error[i] * I_error[i];
                        }
                        I = I * 0.01;

                        error = final_Reading - firstRead;
                        if(error < -180  )
                        {
                                error = 360 + error;
                        }
                        if(error >=0 && error <= 180)
        {
            printf("Error %0.2f \n",error);
                                sumError = p * P + d * D + i *
I;
                                pwm_value = 0.053 * error +
3;
                                printf("PWM %0.2f
\n",pwm_value);
                                motor.set(pwm_value);
        }


                        sumError = kp * P + kd * D + ki * I;
                        value1 = 0.053 * error;
                        printf("PWM %0.2f \n",value1);

                        sprintf(pwm,"%0.2f",value1);



                        sleep(0.07);
                        bytes_w =
write(fd,&pwm[0],sizeof(pwm[0]));
                        sleep(0.07);
                        bytes_w =
write(fd,&pwm[1],sizeof(pwm[1]));
                        sleep(0.07);
```

```
                            bytes_w =
write(fd,&pwm[2],sizeof(pwm[2]));
                            sleep(0.07);
                            bytes_w =
write(fd,&pwm[3],sizeof(pwm[3]));
                            sleep(0.07);
                            bytes_w =
write(fd,&pwm[4],sizeof(pwm[4]));
                            sleep(0.07);
                            bytes_w =
write(fd,&pwm[5],sizeof(pwm[5]));
                            sleep(0.07);

                            printf("value written is %s
\n\n",value1);

              }

               close(fd);
              return 0;}
```