Report

A Simplified Inclination Platform as a Stabilization Base for Sun Tracker using the PIC Microcontroller

Reza Shisheie

Department of Electrical and Computer Engineering

Cleveland State University

May 2, 2018

r.shisheie@vikes.csuohio.edu

Contents

Abstract

Solar energy is one of the primary sources of clean and renewable energy. One of the key elements to maximize the power intake of solar panels is adjusting the solar panel angle properly such that the panel is always perpendicular to the sun. In this project a stabilization platform is designed for solar panels, which can track the sun based on online data or offline data. To control this system and handle input and outputs, a PIC16 microcontroller is selected. The angle of the solar panel at any moment is measured by a precision digital inclinometer and sent back to microcontroller as an analog input. A simple PID (Proportional – Integral – Derivative) control scheme with just a proportional gain is designed to take the input angle from inclinometer and send a PWM control signal to the motor driver hardware to drive the panel to the desired position. For testing and validation, the desired angle of the panel is set to zero. Observations proved that the control scheme with reasonable gains has good performance and disturbance rejection. It can go unstable if input sampling rate is not adequate or the gain is set too high. These two effects are shown in presented videos.

1. Introduction

With the rapid increase in population, increase energy cost, and consequences of global warming, utilizing renewable energy is becoming a key solution. Solar energy is one of the primary sources of clean energy. There is much ongoing research in the solar energy field, ranging from increasing the energy intake to optimally distributing energy on the grid, and increasing the energy capacity of fuel cells to store more energy. One of the fields of research is maximizing the energy gain by making improvements on the material of solar cells and optimally driving the cells to be always in direct contact with light [1] [2].

The angle of the solar panel with respect to the sun can play a significant role in performance of the device and it can increase up to 25% in performance [3]. In many cases for simplicity and ease of maintenance, the base angle, which aims at the motion of the sun in the sky, is tuned for each season and the solar panel only tracks the motion of the sun in the sky. In other words, there are two platforms, one is adapting to the inclination angle of the sun in each season, and the other platform tracks the sun in the sky during daylight. This solution, however, is not a sustainable solution for solar farms and it is not only nearly impossible to hand-tune all solar

panels but also very expensive. The solution would be fully automating the panel to adjust to both the inclination of sun and the trajectory of sun in sky.

In section 2, system architecture is explained and the method that was employed in this project is discussed. In section 3, hardware selection and reasoning is discussed. In section 4, time table and milestones are discussed. In this section all difficulties in different stages of project are discussed as well. Results are shown and analyzed in section 5. Finally, a project summary with future work is presented in section 6.

2. System Architecture and Design Process

The goal of this project is designing a simple embedded sun tracker stabilization platform as a proof of concept. Like any IO system, input and outputs must be determined first.

The input of this system is the current angle of the platform and the output is a control signal to the hardware to drive the platform to the desired angle. Once the input angle from inclinometer is measured, it will be compared to the desired angle and the difference is fed as an input to the control box. The desired angle can be a constant angle or an angle that changes with respect to time. It is assumed that the location of sun is known for each time of the day and thus the desired angle is known as a function of time for tracker without the need to actively locate the sun in sky during day light. For further simplicity and ease of demonstration, this angle is preset to zero, which means that the sun is located at its highest point in sky which is at noon.

The next step is designing a control algorithm. PID controllers are widely used in both academia and industry. The architecture of PID controllers is shown below:

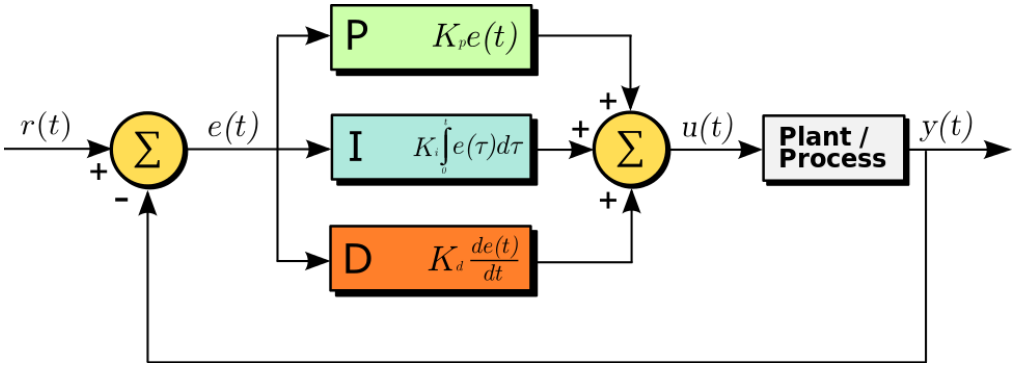


Figure 1. PID controller architecture *[4]*

It consists of three gains: proportional k_p, integral k_i, and k_d derivative and the output signal would be the result of the following equation in time:

$$u(t) = K_\mathrm{p} e(t) + K_\mathrm{i} \int_0^t e(t')\,dt' + K_\mathrm{d} \frac{de(t)}{dt}$$

For simplicity, the proportional gain is tuned first and, other two gains of integral and derivatives are tuned afterwards, if needed. The controller signal *u(t)* is the outcome of the controller, which is used to drive the actuator. This signal is sent to the actuator driver which amplifies the signal and drives the actuator with the use of external power. As the actuator moves, new angles are measured by inclinometer and sent back to the microcontroller, as feedback, to be compared with the desired angle, which adjusts the control signal ultimately. Here is the flow chart of the design:

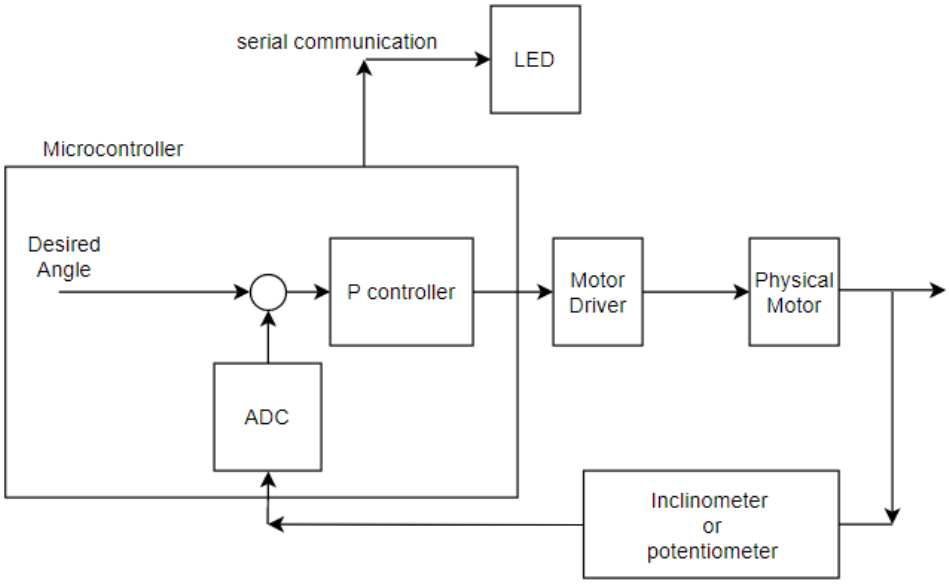


Figure 2. Flowchart of the design

The current angle that is measured by inclinometer is analog. In order to make this understandable for the microcontroller, it should go through ADC section of the microcontroller. If the input data was already in digital format, it could be sent to microcontroller through a serial port like UAR or RS232.

Once the analog input is converted to digital, microcontroller will show it on a display online.

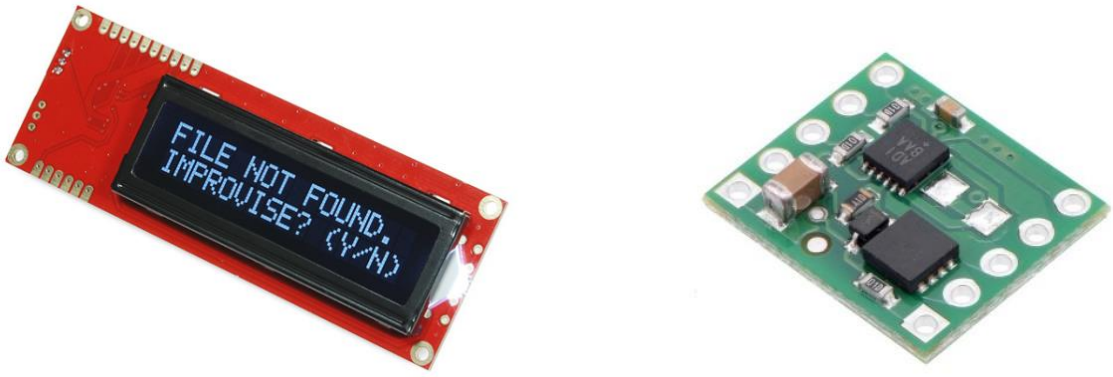
3. Hardware Selection and CAD Design

As far as microcontroller a Microchip PIC16F87X [5] is used and libraries for communication and were developed to support the hardware.

To obtain the current angle of the rotating panel, an analog inclinometer [6] is used. This sensor has two outputs, one for X axis and one for Y axis but only one of them was used for this project. The range of output is [0,5] v where 2.5v represents level angle. Data is provided by 100 samples per second rate, which is more than enough for a slow-response mechanical system like ours.

The output signal is analog ranging from 0 to 5v. This signal is sent to a motor driver, which maps the input signal range proportionally to the power range. The input power ranges [0,12] v with a maximum current of 800 mA.

The current angle at any time is not only used as feedback to the controller but also shown on LCD. A 2x16 segment LCD is used for this purpose [7]. This LCD has a separate communication module, which converts the parallel data format, which is intrinsic for such LCDs to a serial format. The selected serial format is UART, which means that both LCD and microcontroller have to be set to the same baud-rate. 9600 baud is used as the default baud-rate. LCD and motor driver are shown in Figure 3.



(a) (b)

Figure 3. (a) Sparkfun LCD with a UART communication port *[7]* (b) Single channel motor driver *[8]*

All the coding in this project is done in C dues to ease of use and simplicity.

To emulate perturbation in this setup, a gimbal-like frame is placed around the platform, which can be rotated by user to see how the controller compensates for the change. The mechanical configuration of the setup is shown below:

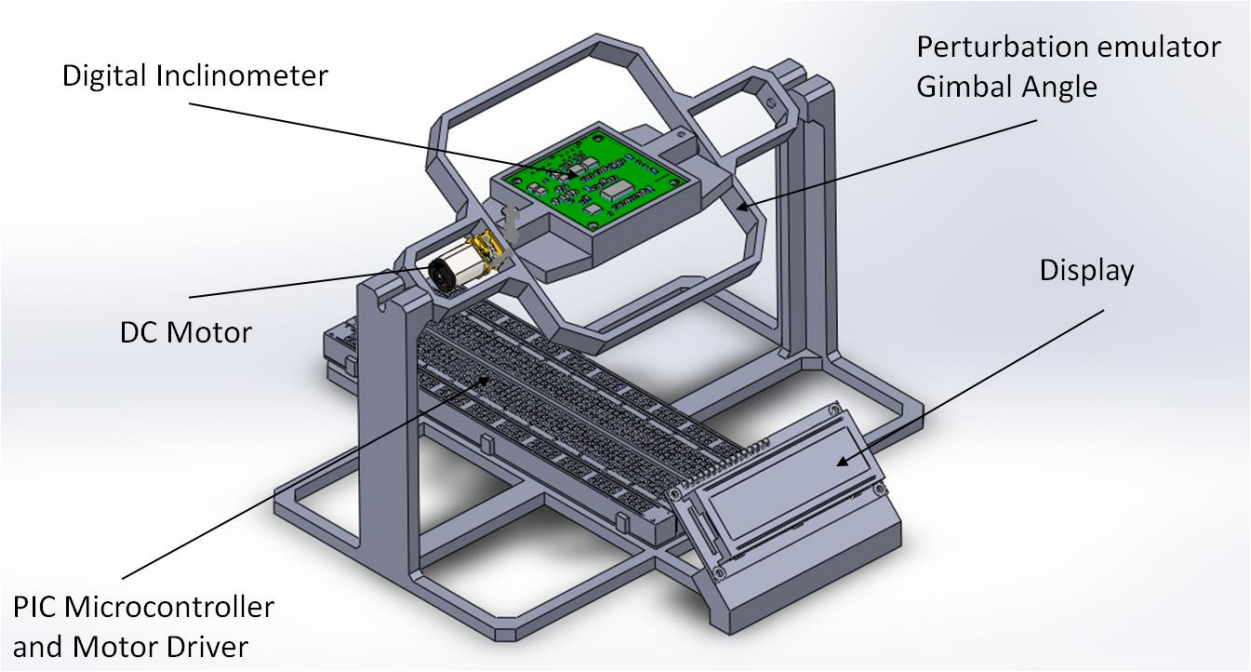


Figure 4. Physical look of the platform designed in SolidWorks

4. Time Table & Milestone.

For implementation phase, this project was broken down into 4 stages.

The first stage is writing random texts on the LCD using the serial port and making sure communication between PIC and LCD is working properly. For this stage various routines must be developed which initializes the LCD to the proper baud-rate, converts any decimal number to a sequence of ASCII characters, and adding + and – for positive and negative angles values. This stage was one of the most difficult parts of the project. For many days the characters shown on display were completely gibberish. To figure out what was causing that, I used Tera Term software on windows and printed results directly from PIC onto serial port with a UART-to-USB driver. All result after a few tweaks became meaningful on windows serial port. It eventually

turned out that I accidentally sent a special command from PIC to LCD and changes the baud rate on LCD side. To program the LCD back to its original firmware I boot-loaded the chip on the LCD to its default firmware.

The second stage is running the inclinometer, testing outputs of inclinometer using multi-meter to see if they make sense or not, receiving inputs using the ADC, and finally showing the results in the debugger to see if the input voltage is changing.

The third stage is getting PWM module to work by initializing it and sending arbitrary voltage commands to the PWM port and see if multi-meter captures the correct average voltage. Direction and speed control of the DC motor can be tested in this stage too. For motor drive control two sets of pins from PORTC were used, one for PWM and one for direction.

The fourth and last stage is designing a simple control algorithm and hooking up all components and testing the setup.

Table 1.Time table

March 16	Submit proposal document. Finish the preliminary Solidworks design
March 31	Order all the necessary parts from Digikey or McMaster and become proficient with C programming in Microchip
April 10	Complete implementation of display driver using serial port.
April 17	Complete implementation of the controller.
April 20	Finalize application and testing the controller.
April 22	Complete project, prepare for presentation.
April 29	Finalize report.
May 1	Presentation
May 6	Final report

5. Results and Analysis

Upon completion of this project the PIC microcontroller takes the input voltage from the inclination sensor as the current state ranging from [0,5] v. This value will be used by a proportional controller to generate a PWM control signal. This signal is sent to the motor driver

to drive the actuator. The motor driver receives the PWM signal from microcontroller from one channel, and the driving voltage (power) from the other channel and amplifies the PWM signal proportionally to generate the output driving signal to the DC motor. At the same time, PIC is printing the current angle on the LCD UART port. Disturbance is modeled as a free ring around the sensor plate which can be turned by the user. Upon motion, the DC motor should correct the inclination of the plate to the desired angle.

After several tests, a proportional gain of 2 was found reasonable. With this gain value, the rotating platform gets stabilized to zero with and error of +/- 2 degrees. Any value less than 2 results in slow response of the platform in response to input perturbation. On the other hand, any proportional gain value larger than 2 results in overshoots and undershoots before convergence. Any proportional gain over 6 cause several undershoots and overshoots and in some case instability. To investigate the effect of slow sample rates on the performance and stability of platform, a 500 ms delay was placed in code, which successfully destabilize the platform.

Here is an actual photo of the assembled setup with all structures made by 3D printer.

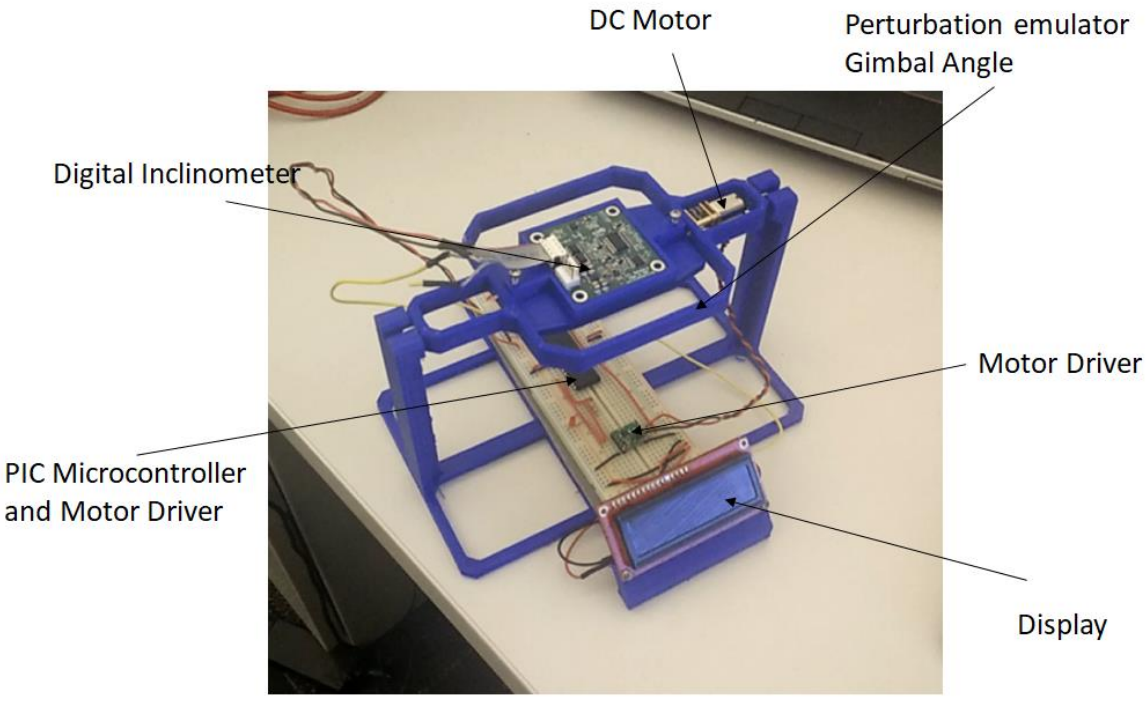


Figure 5. Actual setup

6. Conclusion and Future Work

In this project a simple inclination platform was designed which can successfully converge to goal with a +/- 2 degrees error. This error is caused by many factors such as dc motor dead zone. Once the driving voltage is in this range, dc motor does not rotate and it needs special attention to be compensated. Dead zone is illustrated in Figure 6.

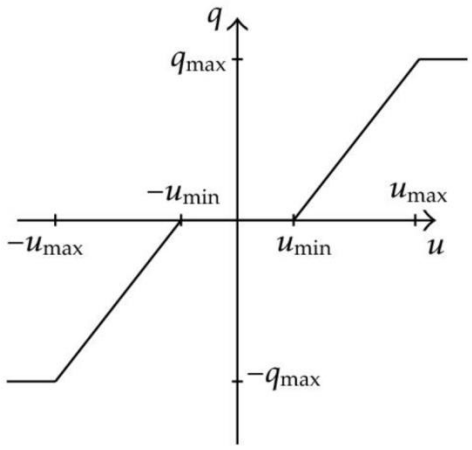


Figure 6. Dead zone illustration

A PID controller with an additional integral part can be a solution too. Integral part of PID integrates error over time and as the value of integral increases, the input control *u(t)* also gets increased until it overcomes the dead zone. Adding integral to PID can be a remedy but not a permanent and reliable solution. A good solution would be a built-in nonlinear compensator which compensates for the voltage loss once *u(t)* is in the zone.

References

[1] J.-M. a. C.-L. L. Wang, "Design and implementation of a sun tracker with a dual-axis single motor for an optical sensor-based photovoltaic system," *Sensors*, vol. 13, no. 3, pp. 3157-3168., 2013.

[2] H. a. S.-M. S. Vendig, "Seasonally adjustable mounting system for solar panels having dual

motor assembly". U.S. Patent Patent 7,476,832, 13 1 2009.

[3] C. R. Landau. [Online]. Available: http://www.solarpaneltilt.com/. [Accessed 13 3 2018].

[4] Wikipedia, "Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/PID_controller. [Accessed 21 3 2018].

[5] Microchip Inc., [Online]. Available: http://ww1.microchip.com/downloads/en/devicedoc/33023a.pdf. [Accessed 13 03 2018].

[6] CTI Sensors, [Online]. Available: http://ctisensors.com. [Accessed 14 03 2018].

[7] Sparkfun, "SparkFun Serial Enabled LCD Kit," [Online]. Available: https://www.sparkfun.com/products/10097. [Accessed 13 3 2018].

[8] Pololu, "MAX14870 Single Brushed DC Motor Driver Carrier," [Online]. Available: https://www.pololu.com/product/2961. [Accessed 13 3 2018].

Appendix

```
// CONFIG
#pragma config FOSC = HS        // Oscillator Selection bits (HS oscillator)
#pragma config WDTE = OFF       // Watchdog Timer Enable bit (WDT disabled)
#pragma config PWRTE = ON       // Power-up Timer Enable bit (PWRT enabled)
#pragma config BOREN = OFF      // Brown-out Reset Enable bit (BOR disabled)
#pragma config LVP = ON         // Low-Voltage (Single-Supply) In-Circuit
Serial Programming Enable bit (RB3/PGM pin has PGM function; low-voltage
programming enabled)
#pragma config CPD = OFF        // Data EEPROM Memory Code Protection bit
(Data EEPROM code protection off)
#pragma config WRT = OFF        // Flash Program Memory Write Enable bits
(Write protection off; all program memory may be written to by EECON control)
#pragma config CP = OFF         // Flash Program Memory Code Protection bit
(Code protection off)


#include <xc.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define _XTAL_FREQ 3686400
#define TMR2PRESCALE 4
long PWM_freq = 5000;

PWM_Initialize()
{
    PR2 = (_XTAL_FREQ/(PWM_freq*4*TMR2PRESCALE)) - 1; //Setting the PR2
formulae using Datasheet // Makes the PWM work in 5KHZ
    CCP1M3 = 1; CCP1M2 = 1;  //Configure the CCP1 module
    T2CKPS0 = 1;T2CKPS1 = 0; TMR2ON = 1; //Configure the Timer module
    TRISC2 = 0; // make port pin on C as output
}

PWM_Duty(unsigned int duty)
{
    if(duty<1023)
    {
        duty = ((float)duty/1023)*(_XTAL_FREQ/(PWM_freq*TMR2PRESCALE)); // On
reducing //duty = (((float)duty/1023)*(1/PWM_freq)) / ((1/_XTAL_FREQ) *
TMR2PRESCALE);
        CCP1X = duty & 1; //Store the 1st bit
        CCP1Y = duty & 2; //Store the 0th bit
        CCPR1L = duty>>2;// Store the remining 8 bit
    }
}

void ADC_Initialize()
{
    ADCON0 = 0b01000001; //ADC ON and Fosc/16 is selected
    ADCON1 = 0b11000000; // Internal reference voltage is selected
}

unsigned int ADC_Read(unsigned char channel)
{
```

```
    ADCON0 &= 0x11000101; //Clearing the Channel Selection Bits
    ADCON0 |= channel<<3; //Setting the required Bits
    __delay_ms(2); //Acquisition time to charge hold capacitor
    GO_nDONE = 1; //Initializes A/D Conversion
    while(GO_nDONE); //Wait for A/D Conversion to complete
    return ((ADRESH<<8)+ADRESL); //Returns Result
}

char UART_Init(const long int baudrate)
{
    unsigned int x;
    x = (_XTAL_FREQ - baudrate*64)/(baudrate*64);   //SPBRG for Low Baud Rate
    if(x>255)                                       //If High Baud Rage
Required
    {
        x = (_XTAL_FREQ - baudrate*16)/(baudrate*16); //SPBRG for High Baud
Rate
        BRGH = 1;                                     //Setting High Baud
Rate
    }
    if(x<256)
    {
        SPBRG = x;                                    //Writing SPBRG
Register
        SYNC = 0;                                     //Setting Asynchronous
Mode, ie UART
        SPEN = 1;                                     //Enables Serial Port
        TRISC7 = 1;                                   //As Prescribed in
Datasheet
        TRISC6 = 1;                                   //As Prescribed in
Datasheet
        CREN = 1;                                     //Enables Continuous
Reception
        TXEN = 1;                                     //Enables Transmission
        return 1;                                     //Returns 1 to indicate
Successful Completion
    }
    return 0;                                       //Returns 0 to indicate
UART initialization failed
}

void UART_Write(char data)
{
    while(!TRMT);
    TXREG = data;
}

char UART_TX_Empty()
{
    return TRMT;
}

void UART_Write_Text(char *text)
{
    int i;
    for(i=0;text[i]!='\0';i++)
    {
```

```
        UART_Write(text[i]);
    }
}



void main()
{
  int adc_value;

  TRISC = 0x00; //PORTC as output
  TRISA = 0xFF; //PORTA as input
  TRISD = 0x00;
  ADC_Initialize();  //Initializes ADC Module
  PWM_Initialize();  //This sets the PWM frequency of PWM1

  PORTCbits.RC1 = 0;

  UART_Init(9600);

  UART_Write(0x80);
  UART_Write(0x00);
  __delay_ms(500);

  UART_Write(0x80);
  UART_Write(0xFF);
  __delay_ms(500);

  int ms_val = 1;
  int PID_K = 2;

  while(1)
  {
      UART_Write(0xFE);
      UART_Write(0x01);
      //__delay_ms(ms_val);

      char buffer[20];

      //__delay_ms(ms_val);

      adc_value = ADC_Read(4); //Reading Analog Channel 0
      int newADC = 0;
      if (adc_value>512)
      {
         newADC = PID_K*(adc_value - 512);
         PORTCbits.RC1 = 1;

      }
      else
      {
         newADC = PID_K*(512 - adc_value) ;
         PORTCbits.RC1 = 0;

      }

      volatile int aa = 0;
```

```
        aa = newADC/12;
        int flag = 0;
        char buffer [20];
        int i = 0;
        while (flag==0)
        {
            int bb = aa/10;
            bb = bb*10;
            int cc = aa-bb;
            cc = cc + 0x30;
            buffer[i] = (char)cc;
            if (aa<10)
            {
                flag = 1;
            }
            aa=aa/10;
            i=i+1;
        }

        i = i-1;
        int max = i;
        char buffer2 [20] ;

        int ii = 0;
        while(ii<=max)
        {
            buffer2[ii] = buffer[i];
            i = i-1;
            ii = ii+1;
        }
        int iii = 0;
        for (iii=0;iii<ii;iii = iii+1)
        {
            int dd = buffer2[iii];
            dd = dd;
            UART_Write(dd);
        }

        //__delay_ms(10);

        PWM_Duty(newADC);

        //__delay_ms(ms_val);
    }
}
```