```
#include <16F877A.h>
#device ADC=16
#FUSES NOWDT                   // No Watch Dog Timer
#FUSES NOBROWNOUT              // No brownout reset
#FUSES NOLVP                   // No low voltage prgming, B3(PIC16)
or B5(PIC18) used for I/O
#use delay(crystal=20000000)
#define LCD_RS_PIN      PIN_D1
#define LCD_RW_PIN      PIN_D2
#define LCD_ENABLE_PIN  PIN_D3
#define LCD_DATA4       PIN_D4
#define LCD_DATA5       PIN_D5
#define LCD_DATA6       PIN_D6
#define LCD_DATA7       PIN_D7
#include <lcd.c>
#include <stdlib.h>

int countchar=0;
char UART_Buffer;
char str[25];
int pos_d=0,pos_e=0;
char ax[6],ay[6],pwmval[4],angleval[4];
int16 angleint;
int32 ovrflow=0;
int16 overflowcount;
int allowovf;

#byte TRISC=0x87
#byte TRISD=0x88

#byte PORTC=0x07
#bit C0=PORTC.0
#bit C1=PORTC.1

#byte TXREG=0x19

#byte TXSTA=0x98
#bit TRMT=TXSTA.1
#bit BRGH=TXSTA.2
```

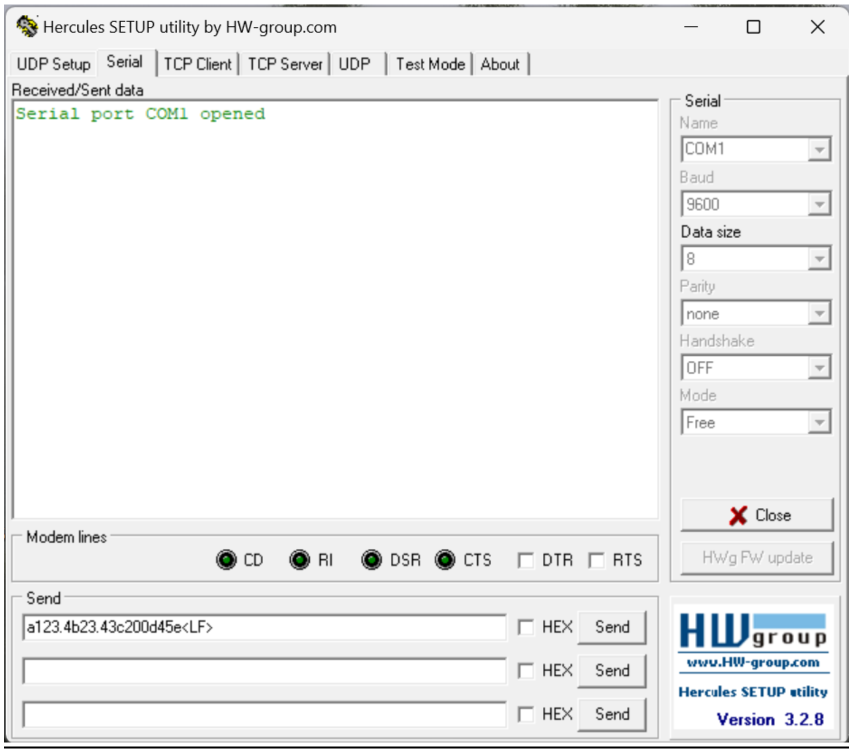
```
#bit SYNC=TXSTA.4
#bit TXEN=TXSTA.5

#byte RCSTA=0x18
#bit CREN=RCSTA.4
#bit SPEN=RCSTA.7

#byte SPBRG=0x99
#byte RCREG=0x1A

#byte PIE1=0x8C
#bit RCIE=PIE1.5

#byte INTCON=0x0B
#bit PEIE=INTCON.6
#bit GIE=INTCON.7

#byte INTCON=0x0B

#byte T1CON=0x10
#bit TMR1ON=T1CON.0

#byte TMR1L=0x0E
#byte TMR1H=0x0F

#byte T2CON=0x12
#byte TMR2=0x11
#byte PR2=0x92
#byte PIR1=0x0C
#byte CCPR1L=0x15
#byte CCPR1H=0x16
#byte CCP1CON=0x17

#byte TMRO=0x01
#byte OPREG=0x81

#int_timer0
void ngatt0()
{
   if(allowovf==1)
   {
      overflowcount+=1; // Counting overflow time (if allow)
   }
   TMRO=0;
}

#int_timer1
void ngatt1()
{
   ovrflow+=1;
   TMR1L=0;
   TMR1H=0;
}

#int_rda
void uart_rcv()
{
   UART_buffer=RCREG;
   if(UART_buffer=='d') pos_d=countchar; // Get position of 'd' and
'e' character
   else if(UART_buffer=='e') pos_e=countchar;
   if(UART_buffer>=32) // Sort for character with decimal value >=32
   {
      str[countchar]=UART_buffer;
      countchar+=1;
   }
   if(pos_e>0) // Extract coordinates, pwm value and desired speed
string
   {
      for(int i=1;i<6;i++) ax[i-1]=str[i];
      for(i=1;i<6;i++) ay[i-1]=str[i+6];
      for(i=13;i<pos_d;i++) pwmval[i-13]=str[i];
      for(i=pos_d+1;i<pos_e;i++) angleval[i-pos_d-1]=str[i];
   }
}

int16 round(float number)
{
    return (int16)(number+0.5); // Round number to the closest
integer
}


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

void uart_init()
{
   BRGH=1; // High speed, asynchronous mode
   SPBRG=129; // Baudrate 9600 bps
   SYNC=0;
   SPEN=1;
   RCIE=1;
   PEIE=1;
   GIE=1;
   CREN=1; // Enable data reception
   TXEN=1; // Enable UART transmission
}

void pwm_init()
{
   PR2=0xff;
   CCP1CON=0b00001100; // Set up CCP1 as PWM mode
   T2CON=0b00000111; // Set up timer 2 with prescaler is 16, 1:1
postcale
   TMR2=0;
}

void timer1_init()
{
   INTCON=0b11100000; // Enable global, peripheral and TMR0 overflow
interrupts
   PIE1=0b00000001;
   T1CON=0;
```

```
}

void timer0_init()
{
   OPREG=0b00000110; // Prescaler 1:128
}

void main()
{
   TRISC=0b11000001;
   TRISD=0x00;
   lcd_init(); // Initialize
   lcd_putc('\f');
   timer0_init();
   timer1_init();
   uart_init();
   pwm_init();

   int32 time_count,sec_count;
   float encoder_read;
   int done1st=0,sent=0;

   float kp=0.02,ki=0.1,kd=0; // Define PID parameters
   double setpoint=0,volt=0,in_speed=0,tsamp=0.01,tcal; //Sampling
time value for the first iteration only!
   double integral=0,last_error=0,error=0,derivative=0;
   int16 pwmvalue;
   CCPR1L=0;
   C1=0;
   lcd_gotoxy(1,1);
   printf(lcd_putc,"Desired:");
   lcd_gotoxy(1,2);
   printf(lcd_putc,"Feedback:");
   while(TRUE)
   {
      if(angleint==0) angleint=atol(angleval); // Convert from string
to long integer value
      else // Desired speed received!
      {
         if(done1st!=0) tsamp=tcal; // Sampling time for next 
iterations
         allowovf=1;
         TMRO=0; // Start counting sampling time
         overflowcount=0;
         setpoint=(float)angleint;
         error=setpoint-in_speed; // PID calculation
         derivative=(error-last_error)/tsamp;
         integral+=error*tsamp;
         last_error=error;
         volt=kp*error+ki*integral+kd*derivative;
         if (volt<0) C1=1; // If the voltage<0, the motor direction 
is changed
         else C1=0;
         if (volt>12) volt=12; // Create upper and lower limit
         if (volt<-12) volt=-12;
         pwmvalue = round(255*abs(volt)/12.0); //Scale to PWM value 
(0->255)
         if(pwmvalue>255) pwmvalue=255;
         CCPR1L=(unsigned int)pwmvalue;
         TMR1L=0;
         TMR1H=0; // Calculate pulse frequency from the encoder input
         while(C0==1);
         while(C0==0);
         TMR1ON=1;
         while(C0==1);
         while(C0==0);
         TMR1ON=0;
         time_count=make16(TMR1H,TMR1L)+ ovrflow*65536;
         encoder_read=(5000000.0/time_count)/24*60; // Pulse per 
Rev=24
         in_speed=round(encoder_read); // Speed from 
encoder=Freq/Pulse per Rev*60(s)
         done1st=1; // Done 1st iteration
         ovrflow=0;
         allowovf=0;
         sec_count=TMRO+256*overflowcount; // Stop counting sampling 
time
         tcal=sec_count*128*1.0/5000000; // Change to second
```

```
         lcd_gotoxy(9,1);
         printf(lcd_putc,"%ld     ",angleint);
         lcd_gotoxy(10,2);
         printf(lcd_putc,"%ld     ",round(encoder_read));
         if(sent==0) // Send the received value to the terminal 1
time only!
         {
            for(int i=0;i<5;i++) uart_send(ax[i]);
            uart_send('\n'); // LF & CR to separate sent values
            uart_send('\r');
            for(i=0;i<5;i++) uart_send(ay[i]);
            uart_send('\n');
            uart_send('\r');
            for(i=0;i<pos_d-13;i++) uart_send(pwmval[i]);
            uart_send('\n');
            uart_send('\r');
            for(i=0;i<pos_e-pos_d-1;i++) uart_send(angleval[i]);
            uart_send('\n');
            uart_send('\r');
            sent=1;
         }
         delay_ms(10);
      }
   }
}
```

1.1. Program and simulation procedure explanation

- Software used: In order to send a string with RS232 communication to the PIC16F877 microcontroller, our group used two softwares, which are Virtual Serial Port Driver and Hercules SETUP utility.
- Hercules SETUP utility software is used to send and receive data through RS232 communication. To send a string, type that string into the box and click the "Send" button. The received string is displayed on the "Received/Sent data" window.
- Virtual Serial Port Driver software is used to create two virtual COM ports and pair them.


Hercules SETUP utility by HW-group.com

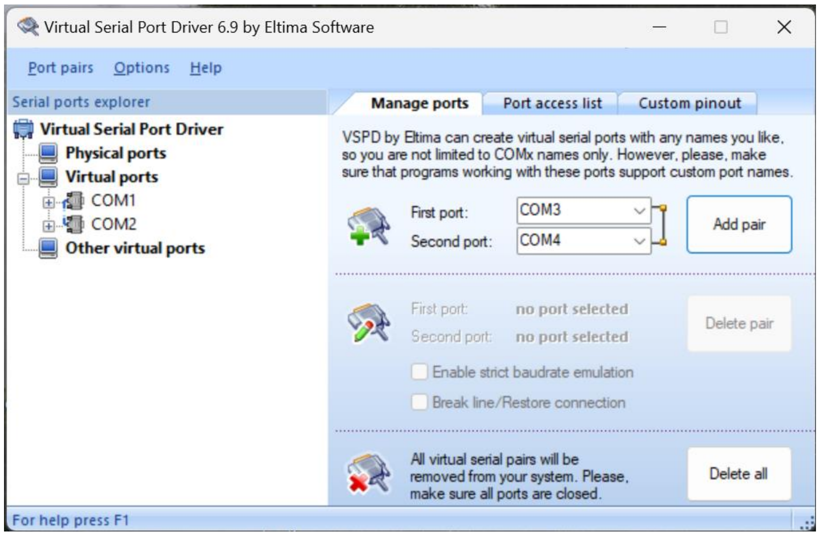
```
Serial port COM1 opened
```

Serial:
- Name: COM1
- Baud: 9600
- Data size: 8
- Parity: none
- Handshake: OFF
- Mode: Free

Send: a123.4b23.43c200d45e<LF>

Hercules SETUP utility by HW-group.com

- The string that our group sent contained a "<LF>" (Line Feed) string, which indicates that every time a new string is sent, it will be displayed in a new line, instead of staying on the same line sticking to the old string, causing confusion.


Virtual Serial Port Driver 6.9 by Eltima Software

VSPD by Eltima can create virtual serial ports with any names you like, so you are not limited to COMx names only. However, please, make sure that programs working with these ports support custom port names.

First port: COM3
Second port: COM4

First port: no port selected
Second port: no port selected

Enable strict baudrate emulation
Break line/Restore connection

All virtual serial pairs will be removed from your system. Please, make sure all ports are closed.

Virtual Serial Port Driver by Eltima Software

- Our group used COM1 for Hercules SETUP utility and COM2 for the COM port connected to the PIC16F877 TX/RX pins.

```
#include <16F877A.h>
#device ADC=16
#FUSES NOWDT                    // No Watch Dog Timer
#FUSES NOBROWNOUT               // No brownout reset
#FUSES NOLVP                    // No low voltage prgming, B3(PIC16)
or B5(PIC18) used for I/O
#use delay(crystal=20000000)
```

#include <16F877A.h>: Use header file 16F877A.h to determine the constants, the register, and other fundamental functions for MCU PIC16F877A.

#FUSES NOBROWNOUT: No brown out reset.

#FUSES NOLVP: No low voltage programming, B3(PIC16) or B5(PIC18) used for I/O.

#use delay(crystal=20000000): Set up the 20 MHz crystal frequency that we use for the MCU.

```
#define LCD_RS_PIN      PIN_D1
#define LCD_RW_PIN      PIN_D2
#define LCD_ENABLE_PIN  PIN_D3
#define LCD_DATA4       PIN_D4
#define LCD_DATA5       PIN_D5
#define LCD_DATA6       PIN_D6
#define LCD_DATA7       PIN_D7
#include <lcd.c>
```

Define the connection pins between the MCU and LCD. In this exercise, we just need to determine 4 pins RS, RW, ENABLE of the LCD as we are using the 4-bit interface.

#include <lcd.c>: This line includes the lcd.c file, which contains the fundamental functions and definitions to communicate with the LCD.

```
int countchar=0;
char UART_Buffer;
char str[25];
int pos_d=0,pos_e=0;
char ax[6],ay[6],pwmval[4],angleval[4];
int16 angleint;
int32 ovrflow=0;
int16 overflowcount;
```

```
int allowovf;
```

Variables declaration.

```
#byte TRISC=0x87
#byte TRISD=0x88

#byte PORTC=0x07
#bit C0=PORTC.0
#bit C1=PORTC.1

#byte TXREG=0x19

#byte TXSTA=0x98
#bit TRMT=TXSTA.1
#bit BRGH=TXSTA.2
#bit SYNC=TXSTA.4
#bit TXEN=TXSTA.5

#byte RCSTA=0x18
#bit CREN=RCSTA.4
#bit SPEN=RCSTA.7

#byte SPBRG=0x99
#byte RCREG=0x1A

#byte PIE1=0x8C
#bit RCIE=PIE1.5

#byte INTCON=0x0B
#bit PEIE=INTCON.6
#bit GIE=INTCON.7

#byte INTCON=0x0B

#byte T1CON=0x10
#bit TMR1ON=T1CON.0

#byte TMR1L=0x0E
#byte TMR1H=0x0F
```

```
#byte T2CON=0x12
#byte TMR2=0x11
#byte PR2=0x92
#byte PIR1=0x0C
#byte CCPR1L=0x15
#byte CCPR1H=0x16
#byte CCP1CON=0x17

#byte TMRO=0x01
#byte OPREG=0x81
```

#byte: Defines a byte variable at a specific memory address.

#bit: Defines a bit variable at a specific bit within a byte.

Address	Name	Description
01h	TMR0	Timer0 Module Register, use Timer 0
07h	PORTC	PORT C
19h	TXREG	USART Transmit Data Register
1Ah	RCREG	USART Receive Data Register
81h	OPREG	The OPTION_REG Register is a readable and writable register, which contains various control bits to configure the TMR0 prescaler
0Bh	INTCON	The INTCON register is a readable and writable register, which contains various enable and flag bits for the TMR0 register overflow
8Ch	PIE1	The PIE1 register contains the individual enable bits for the peripheral interrupts.
10h	T1CON	Timer1 Operation in Timer Mode Timer mode is selected by clearing the TMR1CS (T1CON<1>) bit. In this mode, the input clock to the timer is *the crystal clock's frequency/4.*
0Eh	TMR1L	The Timer1 module is a 16-bit timer/counter consisting of two 8-bit registers (TMR1H and TMR1L) which are readable and writable.
0Fh	TMR1H	

87h	TRISC	PORTC Data Direction Register
88h	TRISD	PORTD Data Direction Register
98h	TXSTA	Transmit Status And Control Register
18h	RCSTA	Receive Status And Control Register
99h	SPBRG	Baud Rate Generator Register
12h	T2CON	Timer2 Control Register
11h	TMR2	Timer2 Module Register
92h	PR2	Timer2 Period Register
0Ch	PIR1	Contains the individual flag bits for the peripheral interrupts
15h	CCPR1L	Capture/Compare/PWM Register 1 (LSB)
16h	CCPR1H	Capture/Compare/PWM Register 1 (MSB)
17h	CCP1CON	Controls the operation of CCP1

```
#int_timer0
void ngatt0()
{
   if(allowovf==1)
   {
      overflowcount+=1; // Counting overflow time (if allow)
   }
   TMRO=0;
}
```

#int_timer0: Defines an interrupt function called when there's an interrupt from Timer0.

In the ngatt0() function, the overflowcount variable is incremented to 1 as Timer1 overflows if overflow is allowed, then Timer0 is reset to 0.

```
#int_timer1
void ngatt1()
{
   ovrflow+=1;
   TMR1L=0;
   TMR1H=0;
}
```

#int_timer1: Defines an interrupt function called when there's an interrupt from Timer1. In the ngatt1() function, the ovrflow variable is incremented to 1 as Timer1 overflows, then Timer1 is reset to 0.

```
#int_rda
void uart_rcv()
{
   UART_buffer=RCREG;
   if(UART_buffer=='d') pos_d=countchar; // Get position of 'd' and
'e' character
   else if(UART_buffer=='e') pos_e=countchar;
   if(UART_buffer>=32) // Sort for character with decimal value >=32
   {
      str[countchar]=UART_buffer;
      countchar+=1;
   }
   if(pos_e>0) // Extract coordinates, pwm value and desired speed
string
   {
      for(int i=1;i<6;i++) ax[i-1]=str[i];
      for(i=1;i<6;i++) ay[i-1]=str[i+6];
      for(i=13;i<pos_d;i++) pwmval[i-13]=str[i];
      for(i=pos_d+1;i<pos_e;i++) angleval[i-pos_d-1]=str[i];
   }
}
```

#int_rda: Defines an interrupt function called when there's a UART signal sent to the PIC16F877 microcontroller via RX pin. The received character is loaded from the RCREG register. The positions of ‘d’ and ‘e’ text is kept track so that they can be used for further processing. If the character has text form, it will be assigned to the str string.

When the pos_e variable receives a value, ax, ay, pwmval and angleval string will get their values from the str string. Ax is the x coordinate of A, Ay is the y coordinate of A, pwmval is the PWM value and angleval is the desired speed.

```
int16 round(float number)
{
    return (int16)(number+0.5); // Round number to the closest
integer
}
```

Create a function that rounds the float number to the closest integer number.

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

Create a function that sends a char through RS232 communication. First, the bit TRMT is checked. If it is not empty, the data is loaded to the TXREG to send to the computer through RS232 communication.

```
void uart_init()
{
   BRGH=1; // High speed, asynchronous mode
   SPBRG=129; // Baudrate 9600 bps
   SYNC=0;
   SPEN=1;
   RCIE=1;
   PEIE=1;
   GIE=1;
   CREN=1; // Enable data reception
   TXEN=1; // Enable UART transmission
}

void pwm_init()
{
   PR2=0xff;
   CCP1CON=0b00001100; // Set up CCP1 as PWM mode
   T2CON=0b00000111; // Set up timer 2 with prescaler is 16, 1:1
postcale
   TMR2=0;
}

void timer1_init()
{
   INTCON=0b11100000; // Enable global, peripheral and TMR0 overflow
interrupts
   PIE1=0b00000001;
   T1CON=0;
}

void timer0_init()
{
   OPREG=0b00000110; // Prescaler 1:128
}
```

Initialize uart, pwm, timer1 and timer0 parameters.

UART

- BRGH=1: High speed, asynchronous mode
- SPBRG=129: Used to generate baudrate 9600 bps
- CREN=1: Enable data reception
- TXEN=1: Enable UART transmission

PWM

- CCP1CON=0b00001100: Set up CCP1 as PWM mode
- T2CON=0b00000111: Set up timer 2 with prescaler is 16, 1:1 postcale

Timer1 and turn it on.

- INTCON=0b11100000: Enable global, peripheral and TMR0 overflow interrupts

Timer0

- OPREG=0b00000110: Select the prescaler 1:128

```
void main()
```

```
{
   TRISC=0b11000001;
   TRISD=0x00;
   lcd_init(); // Initialize
   lcd_putc('\f');
   timer0_init();
   timer1_init();
   uart_init();
   pwm_init();

   int32 time_count,sec_count;
   float encoder_read;
   int done1st=0,sent=0;

   float kp=0.02,ki=0.1,kd=0; // Define PID parameters
   double setpoint=0,volt=0,in_speed=0,tsamp=0.01,tcal; //Sampling
time value for the first iteration only!
   double integral=0,last_error=0,error=0,derivative=0;
   int16 pwmvalue;
   CCPR1L=0;
   C1=0;
   lcd_gotoxy(1,1);
   printf(lcd_putc,"Desired:");
   lcd_gotoxy(1,2);
   printf(lcd_putc,"Feedback:");
```

The TRISC is set to 0b11000001, which enables RS232 communication with TX and RX pin, receives input pulse from the motor encoder and controls the motor direction. The TRISD is set to 0x00 as output pins because they are all connected to the LCD.

Modules such as LCD, Timer0, Timer1, UART and PWM are then initialized.

PID parameters are set with Kp=0.02, Ki=0.1, Kd=0. The sampling time here equals 0.01. A thing that must be noticed here is that this is only the initial value for sampling time, which means that this value is only used for the first iteration. On the next iterations, the sampling time is measured and calculated using Timer0, like the request. Initially, the duty cycle is set to 0, which means that the motor will not rotate.

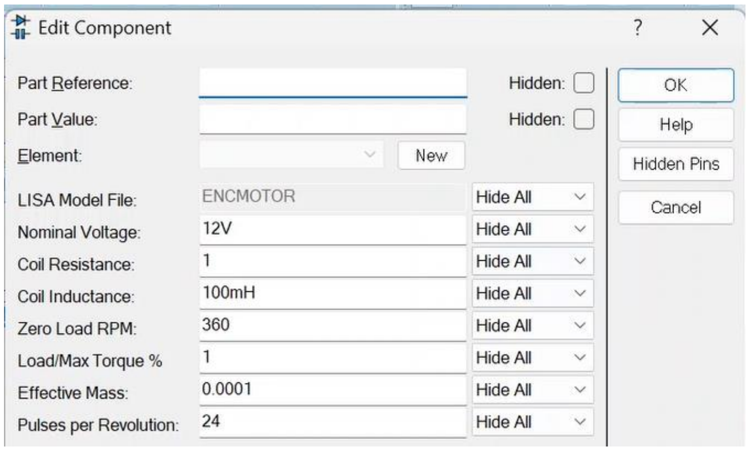
```
   while(TRUE)
   {
      if(angleint==0) angleint=atol(angleval); // Convert from string
long integer value
      else // Desired speed received!
      {
         if(done1st!=0) tsamp=tcal; // Sampling time for next
iterations
         allowovf=1;
         TMRO=0; // Start counting sampling time
         overflowcount=0;
         setpoint=(float)angleint;
         error=setpoint-in_speed; // PID calculation
         derivative=(error-last_error)/tsamp;
         integral+=error*tsamp;
         last_error=error;
         volt=kp*error+ki*integral+kd*derivative;
         if (volt<0) C1=1; // If the voltage<0, the motor direction
is changed
         else C1=0;
         if (volt>12) volt=12; // Create upper and lower limit
         if (volt<-12) volt=-12;
         pwmvalue = round(255*abs(volt)/12.0); //Scale to PWM value
(0->255)
         if(pwmvalue>255) pwmvalue=255;
         CCPR1L=(unsigned int)pwmvalue;
         TMR1L=0;
         TMR1H=0; // Calculate pulse frequency from the encoder input
         while(C0==1);
         while(C0==0);
         TMR1ON=1;
         while(C0==1);
         while(C0==0);
         TMR1ON=0;
         time_count=make16(TMR1H,TMR1L)+ ovrflow*65536;
         encoder_read=(5000000.0/time_count)/24*60; // Pulse per
Rev=24
         in_speed=round(encoder_read); // Speed from
encoder=Freq/Pulse per Rev*60(s)
         done1st=1; // Done 1st iteration
         ovrflow=0;
         allowovf=0;
         sec_count=TMRO+256*overflowcount; // Stop counting sampling
time
         tcal=sec_count*128*1.0/5000000; // Change to second
         lcd_gotoxy(9,1);
         printf(lcd_putc,"%ld     ",angleint);
         lcd_gotoxy(10,2);
         printf(lcd_putc,"%ld     ",round(encoder_read));
         if(sent==0) // Send the received value to the terminal 1
time only!
         {
            for(int i=0;i<5;i++) uart_send(ax[i]);
            uart_send('\n'); // LF & CR to separate sent values
            uart_send('\r');
            for(i=0;i<5;i++) uart_send(ay[i]);
            uart_send('\n');
            uart_send('\r');
            for(i=0;i<pos_d-13;i++) uart_send(pwmval[i]);
            uart_send('\n');
            uart_send('\r');
            for(i=0;i<pos_e-pos_d-1;i++) uart_send(angleval[i]);
            uart_send('\n');
            uart_send('\r');
            sent=1;
         }
         delay_ms(10);
      }
   }
}
```

The angleint variable contains the integer value of the desired speed. Initially, when the string is not sent to the microcontroller, that variable remains 0. But when the microcontroller receives the desired speed value, it will move to the motor controlling part. Atof, Atol function is used to convert strings to float and long integer values.

In the motor controlling part, first, the program checks if this is the first iteration or not. If so, it will use the sampling time value from above. If not, it will use the value which is measured from the previous iteration.
allowovf=0: This means that we start counting the sampling time with Timer0. After that the setpoint is get from angleint variable. PID calculation process then takes place and the volt variable will contain the pid value, which is the voltage used to control the motor.
The C1 pin is used to control the direction of the motor. If the voltage is less than 0, it will rotate opposite to the case where the voltage is greater than 0. The voltage will not exceed upper and lower limit voltage, which is the nominal voltage of the motor (12V for this motor).
pwmvalue = round(255*abs(volt)/12.0): The input voltage is then scaled to PWM value, with 255 equals to 12V, as the L298N motor drives use PWM input to control the motor speed. Again, an upper limit is used for the PWM value to ensure no error occurs.
The frequency of the pulse from the encoder is measured using timer 1. From the pulse frequency, the motor speed can be calculated from the formula:

Motor speed = Encoder pulse frequency / Motor pulse per revolution*60(s)

Motor pulse per revolution equals 24 as we set in Proteus. We can change to any value by editing the component in Proteus.



Part Reference:		Hidden: ☐
Part Value:		Hidden: ☐
Element:		
LISA Model File:	ENCMOTOR	Hide All
Nominal Voltage:	12V	Hide All
Coil Resistance:	1	Hide All
Coil Inductance:	100mH	Hide All
Zero Load RPM:	360	Hide All
Load/Max Torque %	1	Hide All
Effective Mass:	0.0001	Hide All
Pulses per Revolution:	24	Hide All

Motor parameters

After calculating the values, the sampling time is recorded and ready to be used for the next iteration. The desired and feedback values from the encoder are then displayed on the LCD so that they can be kept track easier.

The x coordinate of A,y coordinate of A, the PWM value and desired speed is then sent again back to the terminal, but in separated lines. The LF ‘\n’ and CR ‘\r’ chars are sent with them in order to separate them.