Remote Control Robot

Dario Canelon canel011@umn.edu
Jiannan Zhang zhan2001@umn.edu

Abstract

We designed and implemented a remote controlled tank in the lab. Several modules were applied to give the car have some basic functionality: movement in four directions, and remote control via a computer. Additionally the car has 3 speed levels that can be changed via the controller GUI.

Introduction

As a peripheral controller, the PIC has multiple modules that can support serial communications and motor control via an H-bridge. X-Bee is a common used wireless communication chip that employs the Zigbee protocol and can act as a serial line replacement. A pair of XBee chips were utilized to as the communication interface between the PIC microcontroller and the computer. By adding some other mechanical structures and creating a controller GUI using C#, we can implement a remote control vehicle.

Component List

Component	Quantity
PIC 18f4550 microcontroller	1
Microchip DC motor	2
X-Bee	2
H Bridge	2 (one chip)
Serial data line	1
Chassis and treads	Several
1kΩ Resistor	1
220 Resistor	2
Transistor	2
Crystal	1
22 pF Capacitor	2
0.1 F Capacitor	2
9V battery and holder	1

System Structure and Data flow

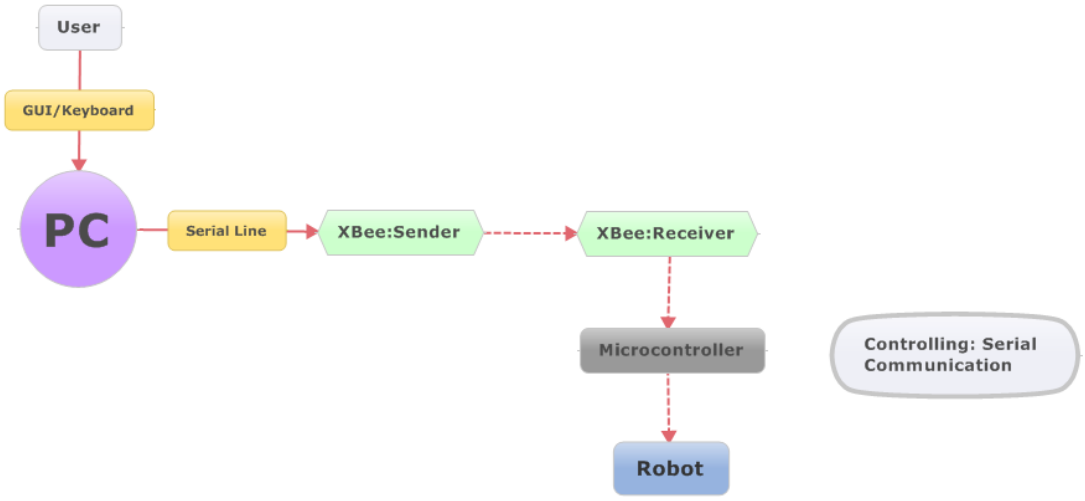

Figure 1. System Structure

Communication Protocol

Data Transmitted (byte)	Command
0x00	Stop
0x01	Forward
0x02	Backward
0x03	Left
0x04	Right
0x05	Super Slow Mode
0x06	Normal Speed Mode
0x07	Rocket Mode

Technical Details

1) User side: GUI implementation

C# is a fast and easy tool to build graphical user interface. The controller interface contains the following elements: direction control buttons (F/B/L/R/Stop) to make the car go to different directions and speed control component (TextBox/Confirm button) to change the speed of the car.

Once the user sends a command from the GUI, the data will go through XBee wireless network and is received by the XBee chip on the vehicle. The XBee passes the data to the PIC microcontroller for further processing.

2) Controlling the robot

The microcontroller installed on the robot continuously receives serial data from the receiver XBee using EUSART module. Once a byte arrives, the PIC processes the command as Table 2 shows. PORTD is connected to the H-Bridge in order to change the direction of the car (by changing current direction through the DC motor), and the PWM module (CCP1/CCP2) was designed to generate square waves of particular duty cycle to change the speed.

3) Motors and mechanical structures

To drive the DC motor, two H-Bridges are applied, the inputs of these H-Bridges are linked to 4 PORTD pins, and their output drive the DC motors. The enable pins of the two H-Bridges are connected to CCP1/CCP2 pins in PWM mode for speed control.

We used a toy vehicle chassis as the body of the car and to hold the circuit board and batteries. 10 wheels and tracks make the vehicle a "tank". The motor drives the front-most sprockets.

4) Other Modules

DC-DC convertor: converts a 9V DC power supply from a battery to a steady 5V DC so that we can correctly drive PIC and other electronics.

XBee driver: To drive XBees (especially sender) to make sure the XBees work well.

Code Utilized

1) Controller GUI

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO.Ports;

namespace robot_DarioJiannan
{
    public partial class Form1 : Form
    {
        SerialPort p;
        byte[] data_protocol;

        public Form1()
        {
            InitializeComponent();
        }
```

```
private void Form1_Load(object sender, EventArgs e)
{
        try
        {
                data_protocol = new byte[1];
                p = new SerialPort();
                p.PortName = "com2";
                p.BaudRate = 9600;
                p.Open();
        }
        catch (Exception ex)
        {
                MessageBox.Show("Failed to create a serial port instant;");
        }
}

private void button2_Click(object sender, EventArgs e)
{
        int speed=0;
        switch (textBox1.Text)
        {
                case "1":
                        data_protocol[0] =5;
                        p.Write(data_protocol, 0, 1);
                        break;
                case "2":
                        data_protocol[0] = 6;
                        p.Write(data_protocol, 0, 1);
                        break;
                case "3":
                        data_protocol[0] = 7;
                        p.Write(data_protocol, 0, 1);
                        break;
                default:
                        break;
        }
}

private void button1_Click(object sender, EventArgs e)          //F
{
        data_protocol[0] = 0x01;
        p.Write(data_protocol, 0, 1);
}
```

```
        private void button5_Click(object sender, EventArgs e)          //B
        {
            data_protocol[0] = 0x02;
            p.Write(data_protocol, 0, 1);
        }

        private void button3_Click(object sender, EventArgs e)          //L
        {
            data_protocol[0] = 0x03;
            p.Write(data_protocol, 0, 1);
        }

        private void button4_Click(object sender, EventArgs e)          //R
        {
            data_protocol[0] = 0x04;
            p.Write(data_protocol, 0, 1);
        }

        private void button6_Click(object sender, EventArgs e)          //Stop
        {
            data_protocol[0] = 0x00;
            p.Write(data_protocol, 0, 1);
        }
    }
}
```

2) PIC on robot

```
#include <p18f4550.h>
#include <delays.h>

#pragma config PLLDIV=2, CPUDIV=OSC1_PLL2, USBDIV=2, IESO=ON, WDT=OFF
#pragma config BOR=OFF, PWRT=ON, LVP=OFF, FOSC=HSPLL_HS, FCMEN=OFF, VREGEN=OFF
#pragma config MCLRE=ON, STVREN=ON, LPT1OSC=ON, PBADEN=OFF

char get_command();
void process_command(char c);
void direction(int direction);
void send_command(char x);
void change_speed(int speed_level);
void delay_nms(int sn);

void high_isr(void);
void low_isr(void);
```

```
#pragma code high_isr_entry=8
void high_isr_entry(void)
{
	_asm goto high_isr _endasm
}

#pragma code low_isr_entry=0x18
void low_isr_entry(void)
{
	_asm goto low_isr _endasm
}
#pragma code

#pragma interrupt high_isr
void high_isr(void)
{
}

#pragma interruptlow low_isr
void low_isr(void)
{
}

char get_command()
{
	while (!PIR1bits.RCIF);
	return RCREG; // automatically take care of RCIF
}

void process_command(char c)
{
	char speed;
	if((c>=0) && (c<=4))
	{
		send_command(c);
	}
	else
	{
		switch(c)
		{
			case 5:change_speed(1);break;//Slow mode 0.1
			case 6:change_speed(2);break;//Middle mode 0.5
			case 7:change_speed(3);break;//Super mode 0.8
			default:break;
```

```
            }
        }
        return;
}

void send_command(char x)
{
        switch(x)
        {
            case 0x01:PORTD=0x05;break;      //0000 0101 F
            case 0x02:PORTD=0x0A;break;      //0000 1010 B
            case 0x03:PORTD=0x04;break;      //0000 0100 L
            case 0x04:PORTD=0x01;break;      //0000 0001 R
            case 0x00:PORTD=0x00;break;                              //0000 0000 Stop
            //PWM is needed for left and right
        }
}

void setupUSART(void)
{
        TRISCbits.TRISC6 = 0;      // TX output
        TRISCbits.TRISC7 = 1;      // RX input

        RCSTA = 0b10010000;    // SPEN=1, 8 bit xmission, SREN=x, CREN=1 enable receiver
                                    // ADDEN=0 disable addr detec, FERR=x, OERR=x, RX9D=0=x
                                    //     TXSTA = 0b10100000;
        BAUDCONbits.BRG16 = 1;
        SPBRGH = 312 >> 8;
        SPBRG = 312 & 0x00FF;
}

void setupPWM(void)           //Fastest mode at beginning
{
        //CCP2
        TRISBbits.TRISB3=0;
        CCPR2L=0b101011;                  //MSB 8 bits
        CCP2CON&=0b11101111;              //Duty cycle=0.6Period=57577ns CCPR2L:5:4=43.2

        //CCP1
        TRISCbits.TRISC2=0;
        CCPR1L=0b101011;
        CCP1CON&=0b11101111;

        //TMR2
```

```
    PR2=18;                    //PWM Period=18*4*83.3*16=95961.6ns PR2=18 PRE=16 F=9.6KHz
    T2CON=0x0f;
}

void change_speed(int speed_level)
{
    switch(speed_level)
    {
        case 1:CCPR1L=0b1110;CCPR2L=0b1110;break;          //20 duty 0.2Period 14
        case 2:CCPR1L=0b101011;CCPR2L=0b101011;break;      //60 duty
        case 3:CCPR1L=0b1001000;CCPR2L=0b1001000;break;        //100              duty
    100Period 72
    }
    return;
}

void direction(int direction)
{
    switch(direction)
    {
        case 1:                    //F
        {
            CCP1CON&=0b11111100;CCP1CON|=0b00001100;
            CCP2CON&=0b11111100;CCP2CON|=0b00001100;
            break;
        }
        case 2:                    //B
        {
            CCP1CON&=0b11111100;CCP1CON|=0b00001100;
            CCP2CON&=0b11111100;CCP2CON|=0b00001100;
            break;
        }
        case 3:                    //L
        {
            CCP1CON&=0b11110000;
            CCP2CON&=0b11111100;CCP2CON|=0b00001100;
            break;
        }
        case 4:                    //R
        {
            CCP1CON&=0b11111100;CCP1CON|=0b00001100;
            CCP2CON&=0b11110000;
            break;
        }
```

```
            case 0:
            {
                CCP1CON&=0b11110000;
                CCP2CON&=0b00000000;
            }
        }
        return;
}

void delay_nms(int sn)
{
        int i;
        for(i=0;i<sn;i++)
            Delay1KTCYx(12);          //120000 instructions,1ms
}

void main(void)
{
        char command;
        setupUSART();
        setupPWM();
        TRISD=0x00;
        PORTD=0x00;
        delay_nms(1000);
        while(1) {
            command = get_command();
            process_command(command);
        }
}
```

Conclusion

We designed and implemented a remote control car using hardware and software modules. The system finally works stably and well by modular design and development. In additional, this product has strong scalability. Three directions are clear: 1. For more practical concern, a digital camera could be installed on this car for real-time monitoring. 2. For more intelligent controlling methods, we would use Kinnect, so that we can move the car simply by moving our hands or arms. 3. For industrial requirements, distributed robot groups are often needed, they work together by coordination and communication to handle some complex jobs. Hence, this simple version of remote control robot could be a basic frame for future extension.