Mr. Roboto: The Infrared Finder Embedded Microcontroller VI

Final Report

Submitted to

The Faculty of Operation Catapult XCIV

Rose-Hulman Institute of Technology

Terre Haute, Indiana

By
Group 36

Sarah Bednar	North Penn High School Lansdale, Pennsylvania
Steven O'Brien	Batavia High School Batavia, New York
Jade Pierce	Atholton High School Laurel, Maryland

July 25th, 2013

Background

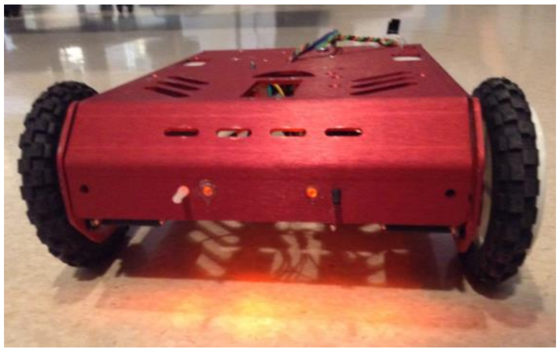
Infrared (IR) is a type of light on the electromagnetic spectrum. Out of the different types of radiation, infrared is distinct in its wavelength and has a longer wave than visible light. The longer wavelength of infrared – closer to microwaves than visible light – is not seen but felt as heat. The heat that is felt from the sun is infrared; over half of the energy from the sun is infrared. In contrast, the shorter wavelength of infrared is not thermal and is commonly used in remotes. Similar to most types of light, infrared bounces off of opaque objects rendering it useful as a distance sensor. Infrared also goes through clear objects, comparable to how white light will go through a window.

Both snakes and insects can see infrared, but mammals including humans cannot. Moreover, infrared is commonly used in everyday appliances. The advantages of using infrared can range from the low power requirements and circuitry costs to its high noise immunity as to not interfere with signals from other devices such as television remotes.

Goal

The goal of this project was initially to have an autonomous robot that drove to an ultrasonic emitter. It was planned that if there was time to have the robot be able to navigate around objects. Unfortunately halfway through the program the objective had to change because it was discovered that the building of a device that receives the ultrasonic signal would be too complex to make. So after that discovery, it was decided to have the robot sense and drive to an infrared (IR) emitter that would be embedded into a ball. In order to stop at the IR emitter, it was necessary to use a premade ultrasonic sensor in conjunction to the infrared receiver to have the robot know that it was at the ball containing the IR LEDs. If there was more time, it would have been nice to have the robot be able to avoid objects too.

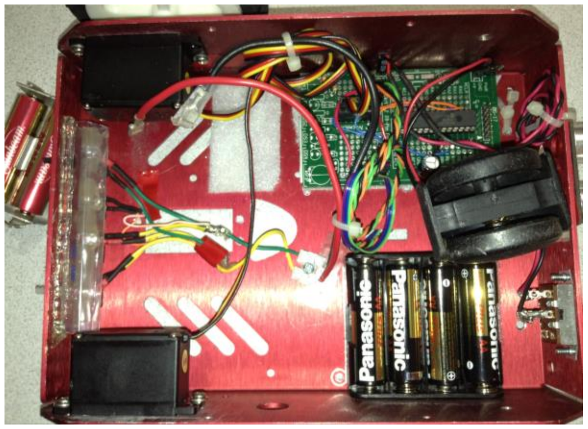
Hardware


Orange underglow to compliment the red, premade frame

The body of the robot was premade. It was composed from aluminum which makes the body light yet sturdy. Encased in red paint making it not only practical but also appealing, the body was embellished with orange under glow by adding a line of LEDs to the base. The primary reason the base was chosen was because it has premade holes which could be repurposed for attaching the infrared detector and other components. Cable ties were used to hold the ultrasonic sensor and the infrared receiver onto the robot, making attachment quick and simple. It was necessary to add a piece of Velcro® between the frame and the circuit board to replacement. If the board did short circuit then everything would stop working and all the electronics would have to be replaced.

The frame did have to be modified to accommodate the servos. The body was originally premade for two DC motors, so it was necessary to cut holes in the side to add the servos. Since both servos were of the same model, they turn in the same direction. Because one was flipped to have a motor on each side, it was necessary for the motors to be driven opposite directions in the code to correct the physical difference. Moreover, the servos had to be modified so that they would move over 360 degrees. The servos were opened up in order for a nub on an internal gear to be ground down. Also, a tab on the axle of the servo had to be cut off so that the servo could be used to power the robot's wheels. Despite the adjustments necessary for the servos, using them was simpler than using DC motors.

Circuitry


The bottom view of Mr. Roboto

The circuitry for the robot was relatively simple. At the heart of it was the PIC16F883 microcontroller. A microcontroller is a low-level simple computer that has 4 kilobytes of program memory and 256 bytes of electrically erasable programmable read-only memory (EEPROM). The PIC has an 8 level stack and has an internal system clock. The PIC16F883 is a 28 port microcontroller which has ports for both digital and analog inputs and outputs.

The components each were attached to ports on the microcontroller. The servos, infrared receiver and the ultrasonic, had three pins. One pin was for ground, another power and the third pin was for signal and was connected to the PIC. The servos were plugged into digital outputs which send out the high and low pulses to the servos at different speeds that tell the servos which way to turn. The ultrasonic was connected to an analog input so that it would take in the information, which conveys how far an object is from the robot. A digital input port was used for the IR so that if it saw infrared it would send a low signal to the microcontroller. The only difficult thing with making the circuit was soldering the wires to the board and trying not to accidently shorts something out.

The infrared ball contained a separate circuit. The orb held an additional PIC16F883 which sent signals to two series of LEDs. There were a total of four infrared LEDs and one blue LED which was used to tell if the circuit was on or off. The LEDs required a resistor connected transistor. Transistor From either of the transistors there were LEDs leading to 100ohm resistors. All the circuitry was necessary to have the LEDs pulsing and to prevent them from burning up. The leads to the LEDs had to be wrapped up to prevent them from touching and shorting out. The circuit and battery pack fit into the orb which had holes made in it so that the light emitted would be stronger.

Software

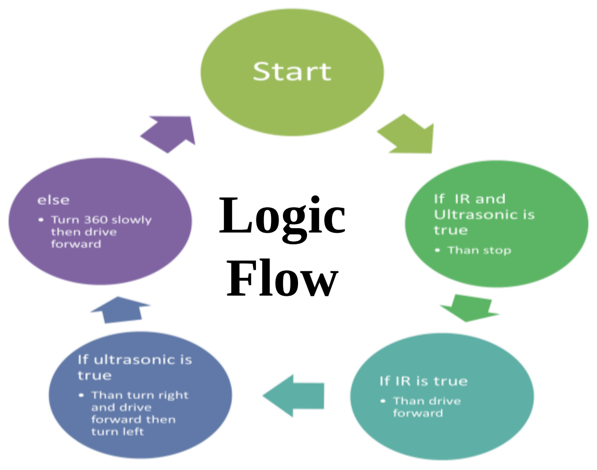
To add to the complexity of the project was that no one on the team knew the programing language, C. One person knew both C++ and java, but those languages are very different than C. The intricacy of programing this specific microcontroller increased the problems tenfold because things such as programing a pin to be an output were difficult and required analysis of sample code given. Much time was saved by simply asking Dr. Song for help instead of spending hours on a single problem. Over time the programing improved, but still what would take experienced C programmers a few minutes, would take the primary programmer an hour or two.

The first step in programming began with getting the robot to perform basic movements like moving forward. In order to do this, the servos had to be programmed to rotate in C using mplab with the PIC16F883. An example servo code was available for editing to manipulate specifically for Mr. Roboto.

Once the robot could move around, the ultrasonic sensor also had to be programmed so that the robot could sense the distance it would be from the target. It could have been used possibly to sense obstacles in the way if time allowed it too. Originally, the sample code was causing numerous headaches and problems until it was realized the code was not appropriate to the ultrasonic sensor in use so a switch to the correct code was implemented promptly after the realization. Then, the sensor was programmed to transmits a pulse of sound and measure the time required for the echo return to determine the distance to the target. The code was modified lastly as a subroutine in the main code of the servos to be incorporated with the logic of the motors and IR.

The IR receiver was much more simple in programming by defining the pin and making it into a digital input. The receiver would let the robot know if the infrared ball was ahead and, if not, to move around.

The final stage in the programming was employing the logic by combining the servos, ultrasonic, and IR so that the robot could autonomously drive to the ball. For example, if there was ultrasonic and IR, then the robot would be at the ball and should stop. If there was only IR, the robot would be facing the right direction and should move forward; if there was only ultrasonic, the robot should turn right, drive forward, and turn left. The other possibility would be that the robot could not sense ultrasonic or IR; therefore, the robot was programed to rotate 360 degrees till it found the IR or ultrasonic to be prompted to the next command.


Start
If IR and Ultrasonic is true
• Than stop
If IR is true
• Than drive forward
If ultrasonic is true
• Than turn right and drive forward then turn left
else
• Turn 360 slowly then drive forward
Logic Flow

Problems

There were numerous unexpected issues with the drive. Initially, DC motors were tried for the drive. The problem with motors was that they could only rotate backwards with a motor controller, and to be able to turn a specific angle, the motors have to be timed to use encoders. It was attempted to program the motors with the motor controller, but it was overly complex, and with the inclusion of encoders, the difficulty would have been even more extreme. Instead of using motors, it was decided that it would be faster and more efficient to use servos that could be modified to turn over 360 degrees. With an example servo code provided, modifying the programming was significantly simpler. After the switch to servos, they were plugged in for testing, and one started smoking. Another servo was tested in the same spot with no negative effects, leading to no final conclusion on the reason the servo stopped running and started smoking. Overall, the drive was very perplexing and took a while to figure it out.

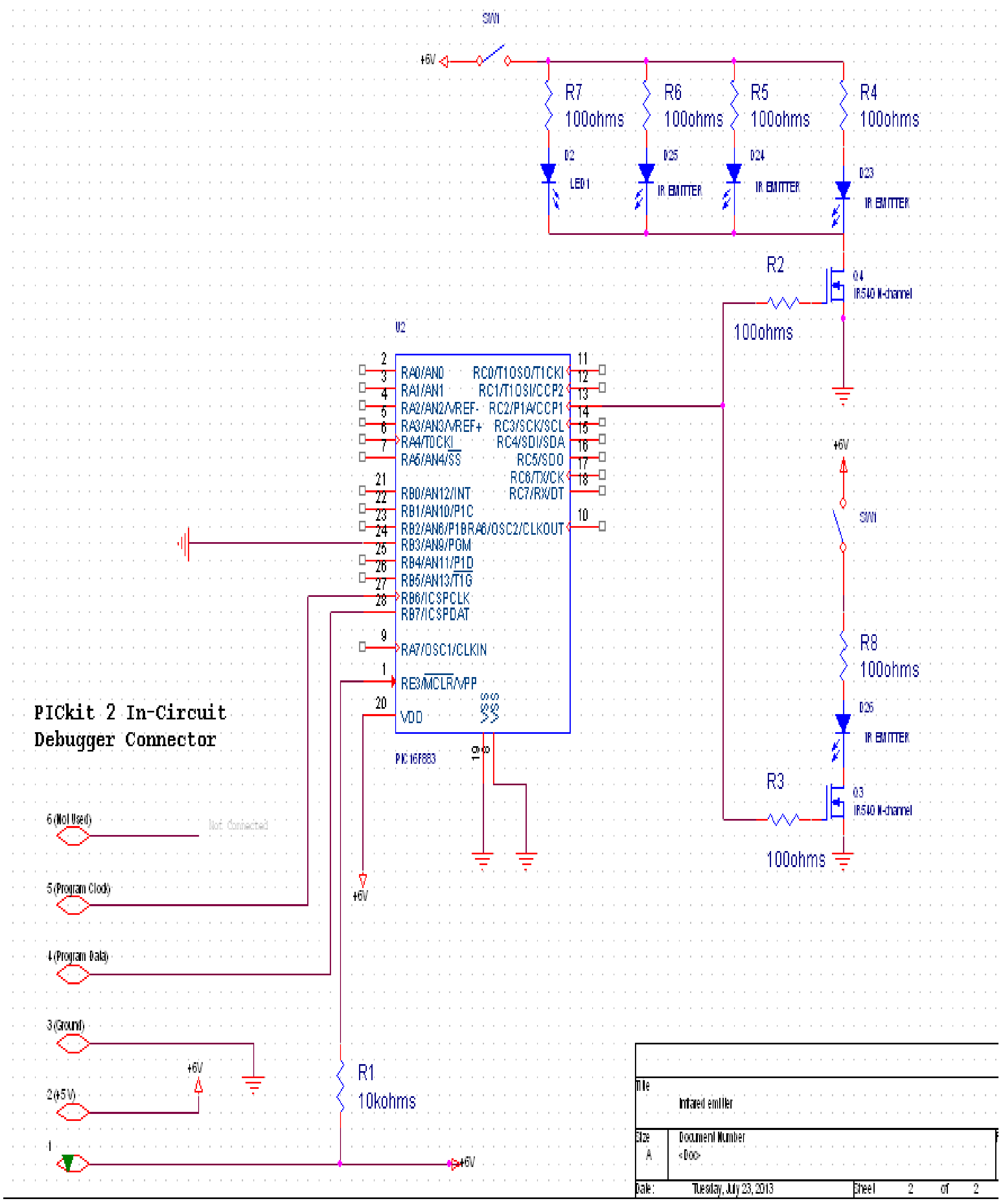
The outside of the IR ball when it is on

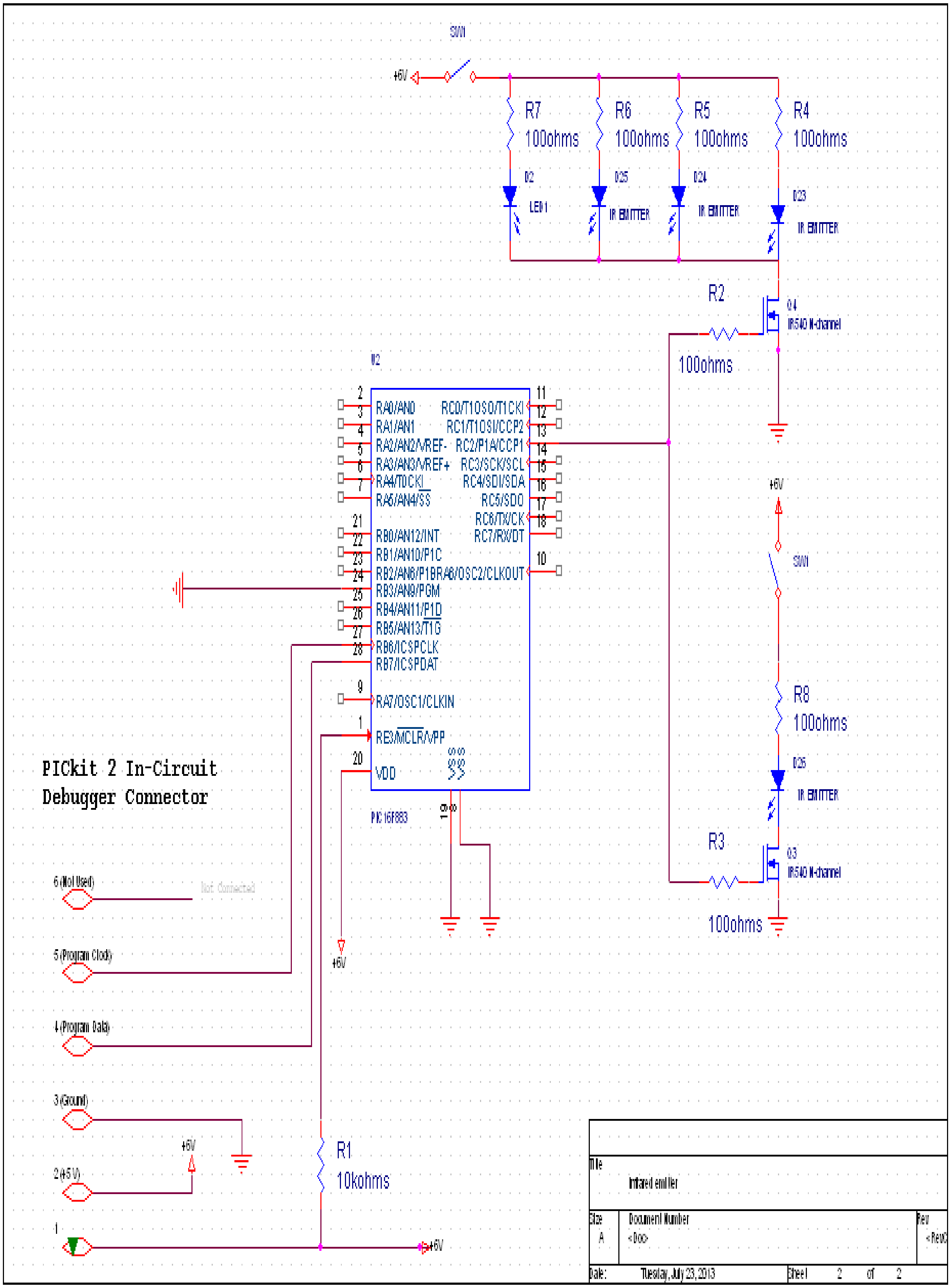
Furthermore, it was difficult figuring out how to have the robot find the object. In the beginning, the plan was to have the robot find an ultrasonic emitter. Then after a few days into the project it was discovered that building an ultrasonic receiver would be too complex in both skill level and time. After that the projects goals had to be adjusted. It was decided that creating an infrared radiator and receiver would be feasible to make and use. The robot may have been able to sense the object but it was necessary to add another sensor to identify the location object. To sense the object a premade ultrasonic sensor was added to the circuit, and was used for determining the exact distance in millimeters. A way of making the project interesting the IR lights were put in a ball that could be rolled around and the robot would drive to it.

Conclusion

Overall, the project was largely a success in creating a robot that could find an infrared. The robot can autonomously drive around and locate a glowing ball that emits infrared with the amalgamation of servos, infrared, and an ultrasonic sensor. This project taught a lot in relation to circuitry, programming and teamwork. The overall project may have been frustrating to make, but once it started to work, it became more enjoyable. Once the ultrasonic started working, everything began to flow smoothly. The project did not end up being what was planned in the beginning, but it was successfully completed and transformed into a new project.

If there was more time it would have been nice to improve upon the robot and to add more capabilities. Obstacle avoidance in the robot was unreliable and did not always work well; so if there was more time, it would have been possible to improve upon its process for avoiding objects. There was an idea of having LEDs on top of the robot which would light up based on how far away it was away from the infrared ball, but there was not enough time to include that feature. If the infrared ball was thought through more carefully and in more detail, there also would have been more IR LEDs added so that the robot can find the ball easier. It would have been fun to add some more which emit light in the visible to make the ball prettier and making the product more appealing. If there was double the amount of time, it would be interesting to try to have the robot be able to pick up the ball and attempt to shoot it back in the direction it came from. To add that feature would be extremely complex, but it would be an interesting challenge.


SW1
+6V
R7 100ohms
R6 100ohms
R5 100ohms
R4 100ohms
D2 LED1
D25 IR EMITTER
D24 IR EMITTER
D23 IR EMITTER
R2 100ohms
Q4 IRF540 N-channel
U2
RA0/AN0
RA1/AN1
RA2/AN2/VREF-
RA3/AN3/VREF+
RA4/T0CKI
RA5/AN4/SS
RB0/AN12/INT
RB1/AN10/P1C
RB2/AN8/P1BRA6/OSC2/CLKOUT
RB3/AN9/PGM
RB4/AN11/P1D
RB5/AN13/T1G
RB6/ICSPCLK
RB7/ICSPDAT
RA7/OSC1/CLKIN
RE3/MCLR/VPP
VDD
VSS
RC0/T1OSO/T1CKI
RC1/T1OSI/CCP2
RC2/P1A/CCP1
RC3/SCK/SCL
RC4/SDI/SDA
RC5/SDO
RC6/TX/CK
RC7/RX/DT
PIC16F883
+6V
SW1
R8 100ohms
D26 IR EMITTER
R3 100ohms
Q3 IRF540 N-channel
PICkit 2 In-Circuit Debugger Connector
6 (Not Used)
Not Connected
5 (Program Clock)
4 (Program Data)
3 (Ground)
2 (+5 V)
1
+6V
R1 10kohms
Title Infrared emitter
Size A
Document Number <Doc>
Date: Tuesday, July 23, 2013
Sheet 2 of 2



SW1
+6V
R7 100ohms
R6 100ohms
R5 100ohms
R4 100ohms
D2 LED1
D25 IR EMITTER
D24 IR EMITTER
D23 IR EMITTER
R2 100ohms
Q4 IR540 N-channel
U2
RA0/AN0
RA1/AN1
RA2/AN2/VREF-
RA3/AN3/VREF+
RA4/T0CKI
RA5/AN4/SS
RB0/AN12/INT
RB1/AN10/P1C
RB2/AN8/P1BRA6/OSC2/CLKOUT
RB3/AN9/PGM
RB4/AN11/P1D
RB5/AN13/T1G
RB6/ICSPCLK
RB7/ICSPDAT
RA7/OSC1/CLKIN
RE3/MCLR/VPP
VDD
RC0/T1OSO/T1CKI
RC1/T1OSI/CCP2
RC2/P1A/CCP1
RC3/SCK/SCL
RC4/SDI/SDA
RC5/SDO
RC6/TX/CK
RC7/RX/DT
PIC16F883
+6V
SW1
R8 100ohms
D26 IR EMITTER
R3 100ohms
Q3 IR540 N-channel
PICkit 2 In-Circuit Debugger Connector
6 (Not Used)
Not Connected
5 (Program Clock)
4 (Program Data)
3 (Ground)
2 (+5 V)
1
+5V
R1 10kohms
+6V
Title
Infrared emitter
Size A
Document Number <Doc>
Rev <RevC
Date: Tuesday, July 23, 2013
Sheet 2 of 2

Code

```
//File Name: PingUltrasonicSubroutine.c
//Date: July 2013
//Purpose: Ultrasonic Subroutine
//Modified by: Group 36 Operation Catapult
//Pin RC2/CCP1, set as input
//capture 16-bit Timer1 count in CCPR1H:CCPR1L
//Control register CCP1CON
//Four modes
//CCP1M3-0==0B1000: every falling edge
//CCP1M3-0==0B1001: every rising edge
//CCP1M3-0==0B1010: every 4th rising edge
//CCP1M3-0==0B1011: every 16th rising edge
//Flag CCP1IF of pin 0 in PIR1. It must be cleared in software.


#include <pic.h>


//define all hardware interface pins
#define TriggerAndEchoPin RC2    //input caputure CCP1
#define PinDirection  TRISC2     //needs to be switched from input to output, etc.
#define WarningLight RB2
#define WarningLightDirection TRISB2=0; ANS8=0;


#define TenMicroSecond 20//312 usecond
#define TenMilliSecond 200 //
#define Frequency 8 //MHz
#define SpeedOfSound 350 //millimeters per milliseccond at 25 degrees Celsius
#define Timer1ClockPeriod 0.5       //usec 4/Frequency
DelayTime(unsigned int count) {
        unsigned int i, j;
        for(i=0;i<count;i++) for(j=0;j<count;j++) {};
}


const unsigned char ASCII_digits[] = {'0','1','2','3','4','5','6','7','8','9'};
#define Rows 3
#define Columns 16
const unsigned char Title[]={"PING ))) Example"};


unsigned long int PingUltrasonicSubroutine(void)
{
        unsigned int FirstEdge, SecondEdge;
        unsigned long int distance, Count;  //counts of two rising edges
        unsigned char Ones, Tens, Hundreds, Thousands;
        unsigned int j,i;
        TMR1ON = 1;//turn on Timer 1
        TMR1CS = 0; //internal clock (Fosc/4)
        WarningLightDirection
        TRISE=0;
        WarningLight = 1;     //turn it off
        TRISC=0b11111011;
        GIE=0;        //turn off interrupts to make accurate measurement
        PinDirection = 0;     //output to start sending ultrasonic sound
        TRISC=0b11111011;
        TriggerAndEchoPin = 0;
        DelayTime(TenMicroSecond);
        TriggerAndEchoPin = 1;     //generate a pulse
        DelayTime(TenMicroSecond);      //minimum pulse width
        TriggerAndEchoPin = 0;     //turn off the pulse to start sending sound
        PinDirection = 1;     //switch to input to receive echo
        TRISC=0b11111111;
//      DelayTime(TenMicroSecond);
        CCP1IF = 0;
        CCP1CON = 0b00000101;  //input capture on every rising edge
//wait for rising edge
        while(CCP1IF==0){}; //waiting for rising edge
        FirstEdge = CCPR1H;
        FirstEdge = FirstEdge<<8;
        FirstEdge = FirstEdge+CCPR1L;
        CCP1IF = 0;
        CCP1CON = 0b00000100;  //input capture on every falling edge
        DelayTime(TenMicroSecond);      //minimum pulse width
        while(CCP1IF==0){}; //waiting for falling edge
        SecondEdge = CCPR1H;
        SecondEdge = SecondEdge<<8;
        SecondEdge = SecondEdge+CCPR1L;
        CCP1IF = 0;
        GIE=1;        //turn on interrupts
        if(FirstEdge < SecondEdge)
                Count = SecondEdge-FirstEdge;
        else {
                Count = -FirstEdge;
                Count = Count+SecondEdge;
        }
        distance = Count*Timer1ClockPeriod;
        distance = distance >>1;
        distance = distance*SpeedOfSound;
        distance = distance>>10;    //in millimeters


        Count = 0;
        Count = 1;
        return distance;
//      DelayTime(TenMilliSecond);
}




// PIC16F84 file: MrRoboto.c
// Written by: Jianjian Song, Rose-Hulman Institute of Technology
// Modified by: Group 36 Operation Catapult
// Date: July 2013
// Main code
#include <pic.h>
// crystal frequency: 20MHz


#if defined(_16F887) || defined(_16F883)  //defined by Configure -> Select Device
        __CONFIG(CP_OFF & DEBUG_ON &  FOSC_INTRC_CLKOUT  & WDTE_OFF &
PWRTE_ON & LVP_OFF);
//using internal oscillator, diabling watchdog timer, turning on power-up timer, diabling low-
voltage program
//FOSC_INTRC_CLKOUT
#endif


void InterruptInitialization(void);
void interrupt ServoDriver(void);


#define LeftServo RA0
#define LeftServoControlPinConfiguration TRISA0=0; ANS0=0;
#define RightServo RA1
#define RightServoControlPinConfiguration TRISA1=0; ANS1=0;
#define StopLeft TRISA0=1; ANS0=0;
#define StopRight TRISA1=1; ANS1=0;
#define IR RB1
#define IRpinConfig TRISB1; ANS10=0;




// interrupt duration
#define INITIAL_COUNT 200                       // 128 microseconds per interrupt at 8MHz
// pulse width to control rotation of a motor
#define other       30
#define ZeroDegree     15                       //1.1ms or less
#define Degree180            4                          // 2.3 ms, 1.75ms or more
#define Neutral      15                         // 1.5ms
```

```
// pulse period
#define PULSE_PERIOD 61          //11.5ms period


unsigned int period, Leftpulsewidth, Rightpulsewidth,LeftPULSELENGTH,
RightPULSELENGTH, count;
unsigned long int PingUltrasonicSubroutine(void);
unsigned long int distance;


void main(void)
{
unsigned int i;
        IRCF2=1; IRCF1=1; IRCF0=1;
        IRpinConfig
        LeftServoControlPinConfiguration;  //Servo Control Pin is output
        RightServoControlPinConfiguration;
        InterruptInitialization();
        period = PULSE_PERIOD;
        LeftServo=1;
        RightServo=1;
        LeftPULSELENGTH = Neutral;
        RightPULSELENGTH = Neutral;
        distance=150;


        while(1)
        {
                distance= PingUltrasonicSubroutine();

                if((distance < 100) && (IR == 0))     //(IR == 0)&&
                {
                        //stop
                        StopLeft;
                        StopRight;

                }
                else if(IR == 0)
                {
                        for(i=0;i<300;i++);
                        //drive foward
                        LeftServoControlPinConfiguration;
                        RightServoControlPinConfiguration;
                        LeftPULSELENGTH=other;
                        RightPULSELENGTH=Degree180;
                        for(i=0;i<60000;i++);
                }
                else if(distance < 100)
                {
                        //turn
                        for(i=0;i<300;i++);
                        LeftServoControlPinConfiguration;
                        RightServoControlPinConfiguration;
                        LeftPULSELENGTH=Degree180;
                        StopRight;
                        for(i=0;i<6000;i++);


                        //drive foward
                        LeftServoControlPinConfiguration;
                        RightServoControlPinConfiguration;
                        LeftPULSELENGTH=other;
                        RightPULSELENGTH=Degree180;

                        //turn
                        for(i=0;i<300;i++);
                        LeftServoControlPinConfiguration;
                        RightServoControlPinConfiguration;
                        RightPULSELENGTH=Degree180;
                        StopLeft;
                        for(i=0;i<6000;i++);
                }
                else
                {
                        for(i=0;i<300;i++);
                        //turn
                        LeftServoControlPinConfiguration;
                        RightServoControlPinConfiguration;
                        LeftPULSELENGTH=Degree180;
                        StopRight;

                }
        }
}
```

```
/*
*	Delay functions
*	See delay.h for details
*
*/
#include	"delay.h"

void
DelayMs(unsigned char cnt)
{
#if	XTAL_FREQ <= 2MHZ
	do {
		DelayUs(996);
	} while(--cnt);
#endif


#if    XTAL_FREQ > 2MHZ
	unsigned char	i;
	do {
		i = 4;
		do {
			DelayUs(250);
		} while(--i);
	} while(--cnt);
#endif
}


/*
*	Delay functions for HI-TECH C on the PIC
*
*	Functions available:
*		DelayUs(x)	Delay specified number of microseconds
*		DelayMs(x)	Delay specified number of milliseconds
*
*	Note that there are range limits: x must not exceed 255 - for xtal
*	frequencies > 12MHz the range for DelayUs is even smaller.
*	To use DelayUs it is only necessary to include this file; to use
*	DelayMs you must include delay.c in your project.
*
*/


/*	Set the crystal frequency in the CPP predefined symbols list in
	HPDPIC, or on the PICC commmand line, e.g.
	picc -DXTAL_FREQ=4MHZ
```

```
        or
        picc -DXTAL_FREQ=100KHZ

        Note that this is the crystal frequency, the CPU clock is
        divided by 4.


*       MAKE SURE this code is compiled with full optimization!!!

*/


#ifndef XTAL_FREQ
#define     XTAL_FREQ  20MHZ            /* Crystal frequency in MHz */
#endif


#define     MHZ    *1000L               /* number of kHz in a MHz */
#define     KHZ    *1               /* number of kHz in a kHz */


#if     XTAL_FREQ >= 12MHZ


#define     DelayUs(x)     { unsigned char _dcnt; \
                    _dcnt = (x)*((XTAL_FREQ)/(12MHZ)); \
                    while(--_dcnt != 0) \
                        continue; }
#else


#define     DelayUs(x)     { unsigned char _dcnt; \
                    _dcnt = (x)/((12MHZ)/(XTAL_FREQ))|1; \
                    while(--_dcnt != 0) \
                        continue; }
#endif


extern void DelayMs(unsigned char);
```