MuleBot

by
Steve Vander Ploeg

Intelligent Machines Design Laboratory
Spring 2003

Dr. A.A. Arroyo
Dr. Schwartz
TA's Jason Plew &
Uriel Rodriguez

Table Of Contents

Abstract

The Mulebot is an autonomous robot that carries things like my backpack, supplies, and other items and follows me around.

Executive Summary

The Mulebot is an autonomous robot that carries things like my backpack, supplies, and other items. This robot is able to mimic my movement, such as speeding up, slowing down, and stopping. Mulebot will also reverse and turn in the proper direction when I get to close. These behaviors are accomplished by using Ultrasonic Ears, and Ultrasonic Proximity detector, IR proximity detectors, an RF Transmitter and Receiver, and using the Progressive ATMega 163 board.

Introduction

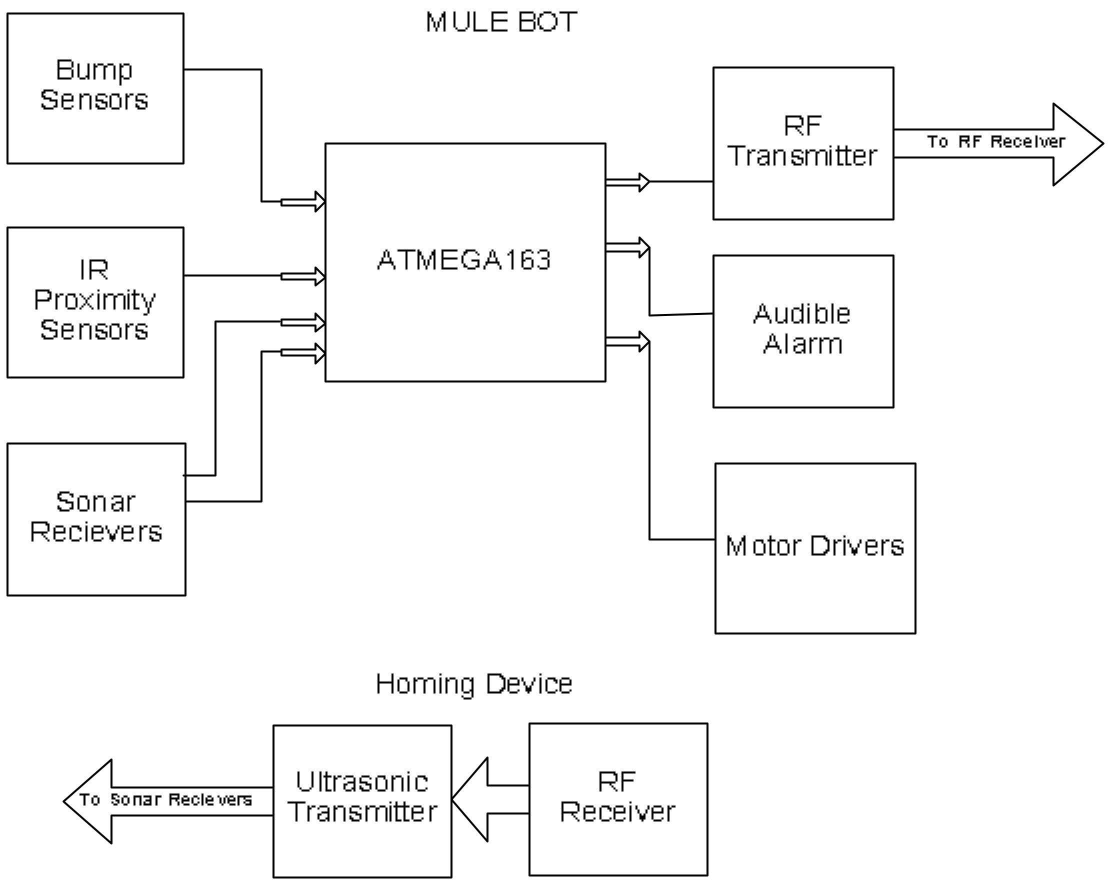
There are many situations where it is inconvenient to carry around heavy items. One examples of this could be at an airport where a person has to carry their luggage. Another example could be at a golf course where a person has to lug around their golf clubs. Another situation that I thought of could be at a grocery store it is inconvenient to have to push around the grocery carts throughout the store. Because of these situations and many others, I decided to create an autonomous robot that would be able to carry heavy objects. Because of price constraints, I decided to design Mulebot to have the capability of carrying my book bag for me around campus.

MULEBOT accomplishes these tasks by obtaining sensory data from a homing device. The homing device uses a RF receiver and Ultrasonic signals to communicate with the MULEBOT. Mulebot contains two ultrasonic ears, an ultrasonic proximity detector, and an RF transmitter. The Atmel Mega163 chip on Mulebot interprets the sensory data, and determines the behavior of the robot based on this information.

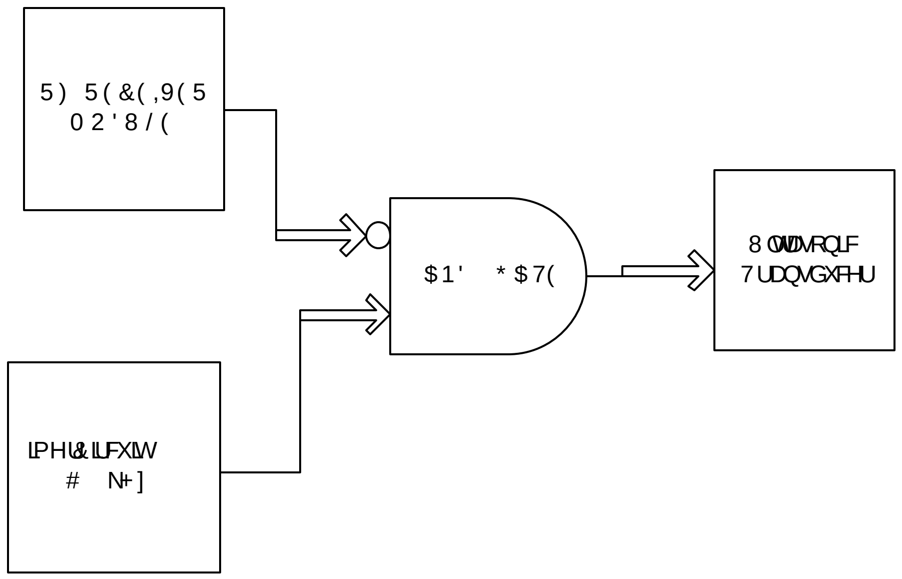
This report contains sections concerning MULEBOT's Integrated System, Mobile Platform, Actuation, Sensors, and Behavior. It also contains lessons learned, test code, a list of sources for parts, and links to websites that I found useful in designing Mulebot.

Integrated System

Below is a block diagram of both MULEBOT and the homing device:



This is a block diagram of my homing device:

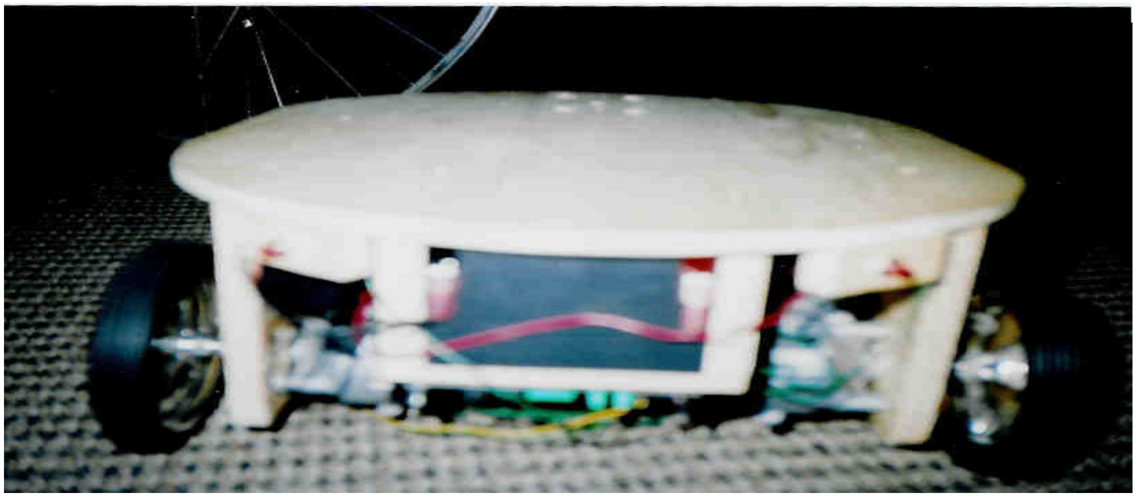


Mulebot is able to follow the homing device by using the systems in the previous diagrams. First, Mulebot sends an RF pulse to the homing device. Two eight bit timers are started on my ATMega 163 board as soon as the RF pulse is initiated. Due to the circuitry in the homing device, an Ultrasonic Pulse train at 40 KHz is sent back to Mulebot as soon as the RF pulse is received by the homing device. The length of this pulse train is determined by the length of the RF pulse. When the left ear on Mulebot detects the ultrasonic pulse train, one timer is stopped. When the right ear receives the ultrasonic pulse train, the other timer is stopped. Mulebot uses this timing information to determine my orientation. For example, if the timing value for the right ear is less than the timing value for the left ear, then Mulebot knows that I am located on his right side. If the timing values are close, then Mulebot knows that I am straight ahead.

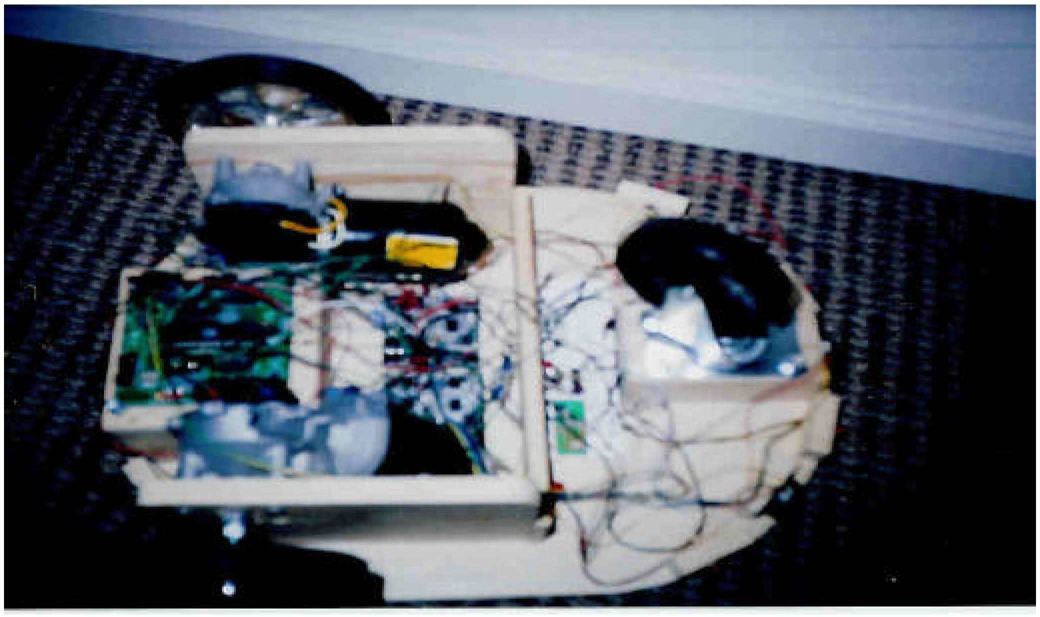
An ultrasonic proximity detector is used in order to determine my separation distance from Mulebot. I am using the same circuit for the Ultrasonic Proximity detector as I am for the Ultrasonic Ears. I will describe the details of how I use this proximity detector in the Sensors section of this report.

Mobile Platform

The following pictures best describe the design of Mulebot's platform:



This is the rear view of Mulebot. As you can see from this picture, the entire platform was created from wood. I placed my battery in the rear of the robot, and made a housing for it so that it was easy to remove the battery for charging. I attached a Velcro strip to the back of the battery in order to keep the battery in the housing. The ATMega 163 board was attached to the bottom of the battery housing. This progressive ATMega 163 board came without screw mounting holes, so I mounted this board by drilling holes into the battery housing at the edges of the progressive board. I used screw nuts to hold the progressive board into place. One of the screws and screw nuts can be seen in the above picture on the left side of my battery housing.



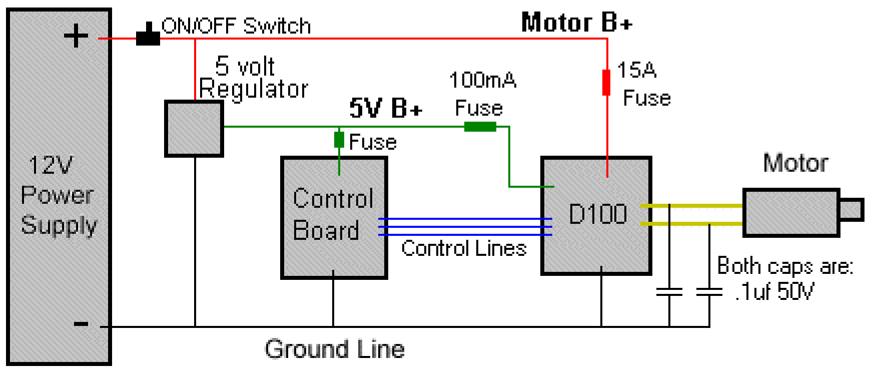
In the above picture, you can see a better view of how everything was mounted. As you can see in this picture, I used a wheel caster for the front wheel. I chose this design because I eventually would like Mulebot to operate outdoors. Therefore, a ball caster is not the best choice because dust and dirt would seriously affect the performance of the ball caster. I found that this wheel caster worked very well.

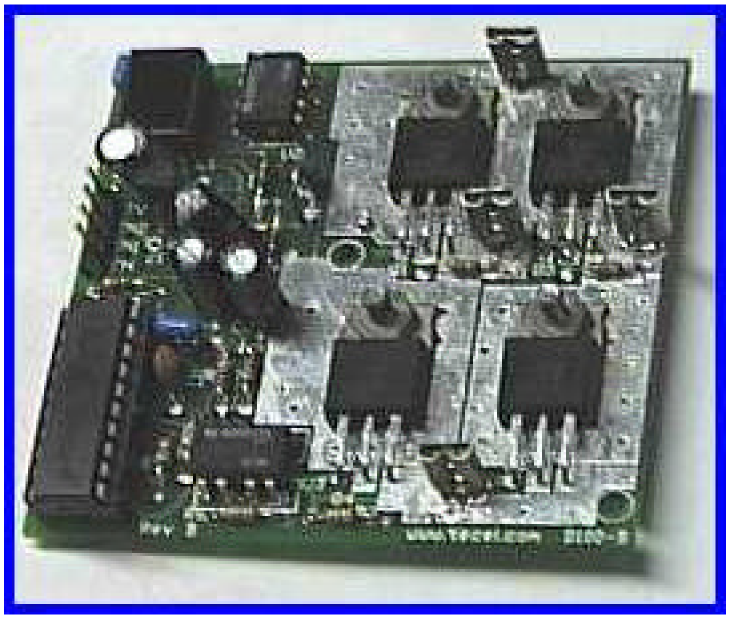
You can also see from the above picture the mounting of my DC motors and wheels. The wheels that I used for Mulebot were training wheels for a bicycle. The shaft on the motor contained metric threads since these motors were designed to be windshield wiper motors. To mount these wheels, I found a metric screw with the same diameter as the shaft of the motor. I then used lock washers to hold the screw to the shaft of the motor. I attached the metric screw to the training wheel by using a silicon rubber glue. I also used a dremel to etch out one side of the wheel so that I could embed the head of the screw into the wheel.

Actuation

In order to actuate Mulebot, I used two 12V DC gearhead windshield wiper motors. I control the steering of Mulebot by sending separate PWM values to each motor. The maximum speed of these motors is about 110 RPM. At this speed, the motors consume about 4 Amps of current at 12V. I do not know the exact torque values of these motors, however they are extremely powerful. These motors were purchased from All Electronics Corp., which is a surplus retailer. I decided to purchase these motors because I found that similar motors with the same performance cost well over $100. I purchased these motors for $20 each, and would recommend them to anyone building a robot that needs powerful motors.

In order to control these motors, I used two TECEL D100-B Motor Driver Boards. These boards are capable of powering each motor with 25 Amps of current. These boards use three control lines which make these motor driver boards easy to use. The IN1 and IN2 pins control the direction of the motor, and the EN pin accepts PWM. The following diagrams show how I connected these motor drivers to my motors. These diagrams are provided courtesy of TECEL:





This is a picture of the D100-B motor driver board.

Enable	IN1	IN2	Action
H	L	L	Fast Motor Stop
H	L	H	Forward
H	H	L	Reverse
H	H	H	Fast Motor Stop
L	X	X	Free Running Motor Stop

This truth table demonstrates the functions provided by the TECEL D-100b board.

In addition to the wiring diagram above, I added a 2000uF rated at 16V between the 12V and Ground on my battery. This was necessary for my robot because the noise from driving these motors will travel across the common ground. This noise greatly affected the performance of my sonar receiver circuit. The capacitor acts to absorb some of the spikes caused by the motor. Note that this capacitor is not necessary for the operation of these motors, but it will definitely help to eliminate the noise in your system. For my robot, this capacitor was essential.

Sensors

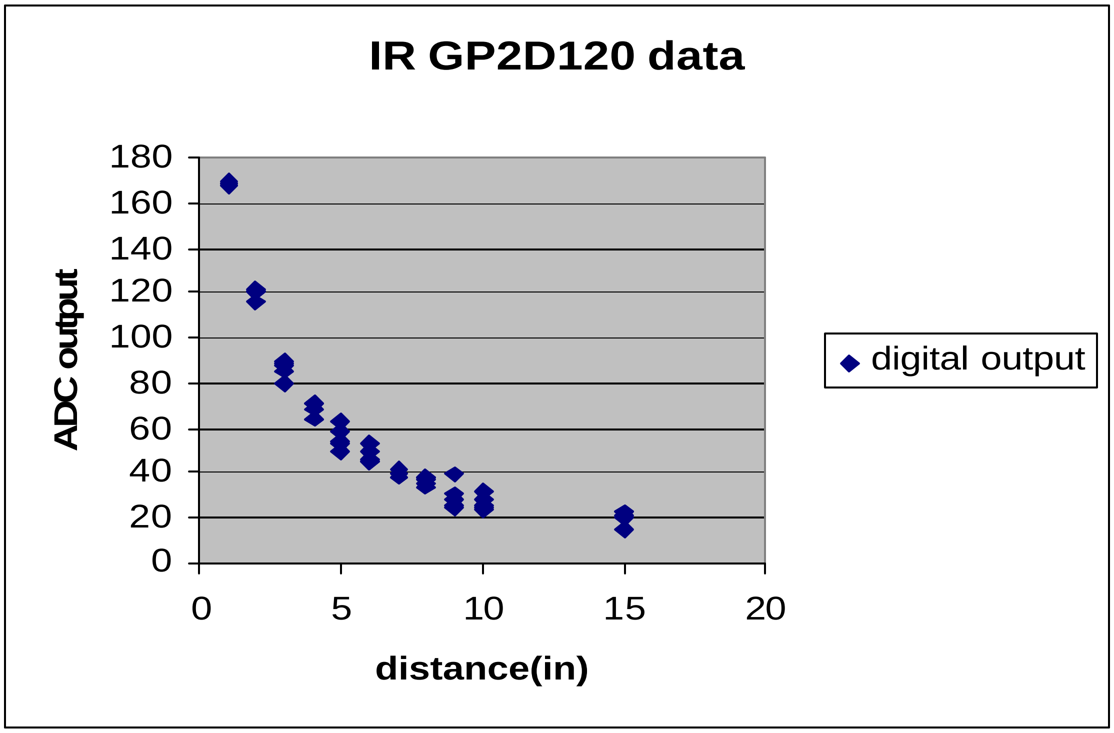
For Mulebot, I will use bump sensors, IR proximity detectors, an ultrasonic transmitter and 2 receivers, an ultrasonic proximity detector, and a RF transmitter and receiver.

Bump Sensors:

Because there are many available I/0 pins on the Progressive ATMega 163 board, I connected each of the 6 bump switches directly to Port B. This method is much easier than using a resistor network and taking the A/D converter value to find out which switch is pressed. I connected one end of the switch to ground and the other end of the switch went directly to my Progressive board. I also activated an internal pull-up resistor on the ATMega 163 so that the pin would only go low when the switch was pressed. Based on the location of the switch, I decided which direction I would have Mulebot move when the switch was pressed. These switches were mainly used for obstacle avoidance.

IR Proximity Detectors

The IR detectors that I am using for the Mulebot are the Sharp GP2D120. These sensors have the capability of locating obstacles between the distances of 4 to 30 centimeters. The output of this device is analog, and is updated approximately every 32ns. On Mulebot, I used three of these IR proximity detectors. I mounted all three in the front of the platform, since Mulebot will primarily be traveling forward.

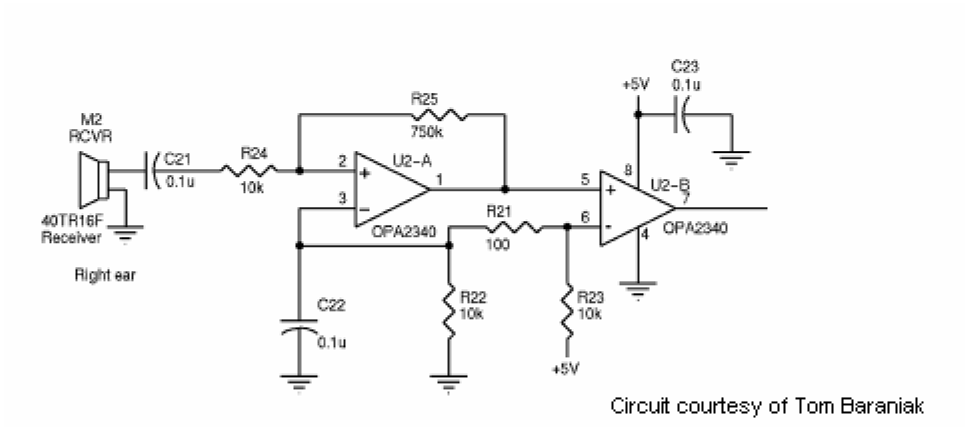


I tested one of my IR proximity detectors and obtained the above graph of my ADC output versus distance. For each distance, I took five values so that I could get a measure of the inaccuracy of the proximity detectors. As you can see from the above graph, the ADC output remains fairly constant at a given distance.

I would not recommend these IR proximity detectors for obstacle avoidance. The reason for this is that because they are short range detectors, and many times my robot would run in to things because it would not detect an object until it was really close. Therefore, I had to run my robot at a fairly slow speed so that it would not run in to things. Many other people in the IMDL course used the Sharp GP2D12 sensors, and those seemed to work much better for obstacle avoidance.

The other thing I learned about these sensors is that they are extremely noisy. I eventually discovered that each IR detector generated about twice as much noise between my power and ground as my motor circuit. You can easily eliminate this noise by placing an electrolytic capacitor of about 50uF between the +5V and Ground going to the IR detector.

Ultrasonic Receiver Circuit/ Also used as Proximity Detector



I used the above circuit in order to detect ultrasonic pulses from my homing device. The output of the ultrasonic transducer is a small signal on the order of a few milli-volts. Because it is a small signal, it is amplified by the first op amp. The second op amp acts as a comparator, and will compare the amplified signal to the threshold voltage set by the voltage divider network created by R23, R21, and R22 in the above diagram. The output of this circuit will be a 40KHz square wave with an amplitude proportional to separation distance between the transmitter and the receiver. The highest amplitude I was able to obtain was about 1.5V. Therefore, I connected the output of this circuit to the base of a transistor. I connected the collector to one end of a resistor and the other end of the resistor to +5V. I connected the emitter of my transistor to ground. I took the output between the collector and the resistor. In doing this, I was able to convert this square wave to logic levels. I also found it necessary to increase the value of R21 due to the noise in my system. This is because the op amp will amplify noise in the system as well as the 40KHz ultrasonic signal. If I did not increase this resistor value, my robot would think that it was receiving the ultrasonic signal, when in reality it was receiving the amplified noise in my system. The lowest resistor value that I was able to use was about 700 ohms for R21. I was only able to lower it to this value after I put large capacitors between power and ground by the motors and IR proximity detectors. Note that this resistor value needs to be as low as possible. The higher the value for this resistor, the worse the reception of the ultrasonic signal. With the resistor value of 700 ohms, I am able to transmit the ultrasonic signal about 6 feet. If I were able to eliminate the noise from my IR proximity detectors and motors, I would probably be able to transmit this ultrasonic signal about 15-20 feet by lowering the resistor value of R21. I did not opto-isolate the motors that I am using. If I were to build this robot again, I would definitely do this because it would greatly improve the performance of my sonar receivers.

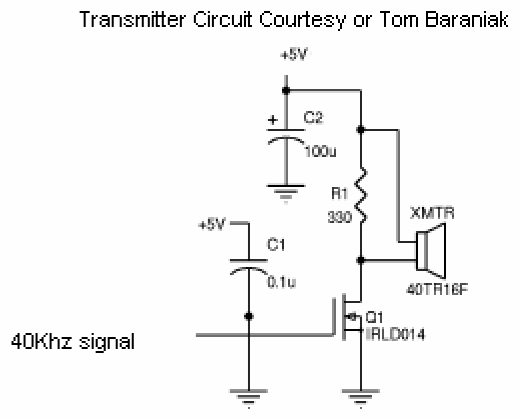
I am also using this above circuit as a proximity detector since the amplitude of the output wave is proportional to separation distance between the transmitter and receiver. Because the output of this is a 40KHz wave, I had to take an average of A/D converter values to get a value for the amplitude of the wave. I found that taking the average of about 10 -15 A/D values would give a pretty good value for separation distance.

I would highly recommend using this circuit as a proximity detector, especially for robots that will be operated outdoors. The reason for this is because IR detectors can not be used outdoors because sunlight contains IR. Ultrasonic proximity detectors by

Devantech can be purchased, however they are fairly expensive(about $30). This ultrasonic proximity detector can be made for about $10. Note that the Ultrasonic Transmitter Circuit (see next section) operating at 40KHz needs to be added to this circuit to obtain a proximity detector similar to the IR detectors.

Ultrasonic Transmitter

Below is the circuit that I used for the Ultrasonic Transmitter:

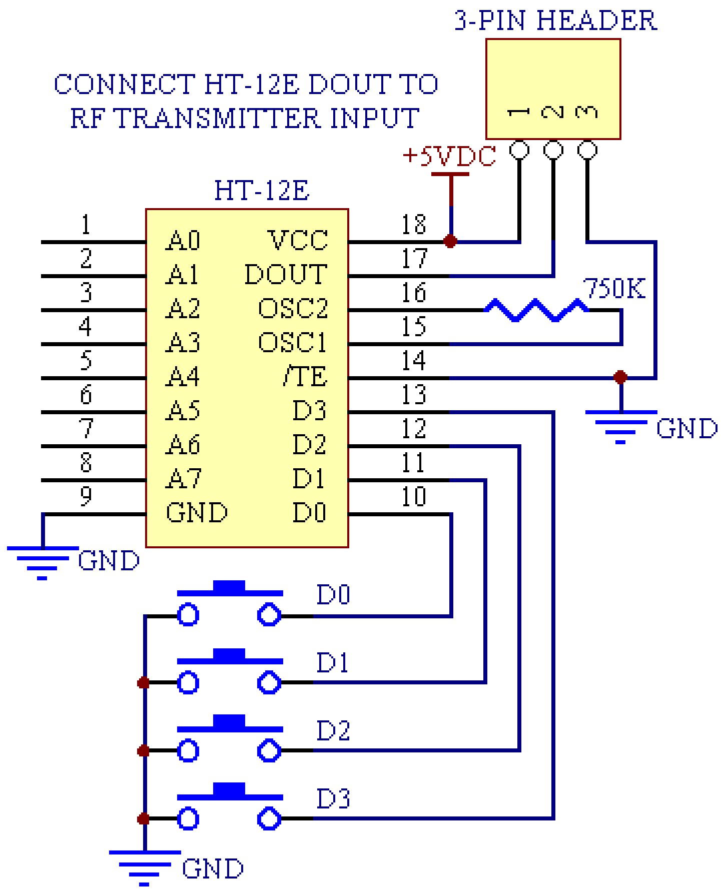


I generated the 40KHz signal for this circuit by using a 555 timer. The only other difference that I made to this circuit is that instead of using +5V(connected to R1), I used about +20V. The reason that I did this was because I needed to raise the amplitude of the 40KHz signal in order for my robot to receive the signal at a decent separation distance. It is safe to raise this voltage value up to about 24V. I did not have any problems operating this circuit at 20V.

The only other thing I will say about this circuit is that it is very important to make sure that the input to the ultrasonic transducer is very close to 40KHz. The receiver circuit for this has a fairly narrow bandwidth of about 40KHz plus or minus about 1KHz.

RF Transmitter

I am using a RF Transmitter to send a pulse from Mulebot to my homing device. In order to do this, I decided to use the Hotek 12E encoder chip with the Ming TX-99 transmitter. The below diagrams show how these devices were interfaced with each other:



Here I am using a transistor instead of a switch so that my robot can initiate a signal. I connected the collector to ground, the base to my I/O port on my ATMega 163 board, and the emitter to the D0 input of the Hotek encoder chip. Since I am only sending one pulse, I left inputs D1 – D3 open.

Ming TX-99 V3.0 300 MHz AM, RF Transmitter

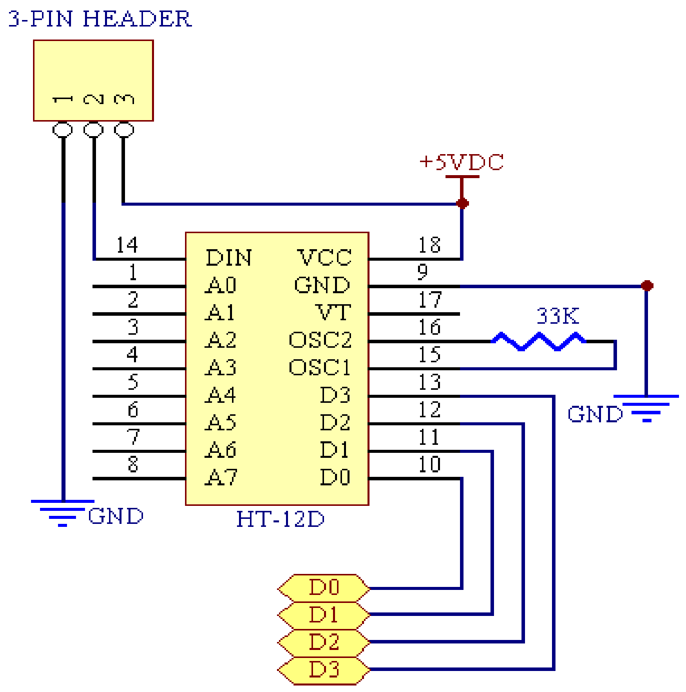
I found this device very easy to use, and it seemed to transmit at a fairly long distance. I did not connect an external antenna to my receiver, and it would receive the RF signal about 20-30 feet away.

Initially, I tried to send a signal through the data channel of the RF transmitter without using the Encoder chip. I discovered that it would transmit a pulse by using this method, however the RF pulse received would have a maximum amplitude of only 1.5V. Looking back, I probably could have used a transistor to obtain logic levels, however the encoder chip does that as well. I decided to use the encoder chip because it will allow me to send up to sixteen different signals to my robot. This may be useful if I ever add any additional functions to Mulebot in the future.

RF Receiver Module

I used the Ming RE-99 RF Receiver Module with the Hotek 12D decoder chip in order receive the RF pulse to the homing device from Mulebot. The below diagrams show how these devices were interfaced with each other:

RF Receiver Module



Ming RE-99 V3.0A 300 MHz AM, RF Receiver Module

Although these RF transmitter and RF Receiver modules are very easy to use, I would not recommend them in a system where timing is critical. Initially, I planned on using my timer values from when Mulebot sent the RF signal to when Mulebot received the ultrasonic pulse train as a measurement of my separation distance from Mulebot.

When I ran some tests, I discovered that the timing values would vary drastically when the separation distance between Mulebot and the homing device were held constant. After testing every part of my system(including my code), I discovered that something must be happening in the RF unit which would affect my timing values. After viewing my timing values closely, I discovered that the timing value at a constant separation distance would decrease during each reading and then eventually jump up to a higher timing value. It would then start to decrease again and the process continued in this manner. I believe the problem is that somehow the charge on the electrolytic capacitors in the receiver module affect the time it takes to for the RF receiver to initiate the sonar pulse train. At any rate, the timing values would vary drastically, and I decided that it was impossible to use these RF devices to obtain timing values as a measurement of separation distance. This is why I am using an ultrasonic proximity detector as a measurement of my separation distance. One more thing about this point is that it did work to use the timing values as a measurement of my direction or orientation from Mulebot. The reason for this is because I am subtracting the two timer values from each other. Therefore, whatever inconsistency that happens in one timing value, also happens in the other.

Experimental Layout and Results

One thing I found necessary is to test every part of my system in order to get my robot to operate. I also found it necessary to calibrate each of my sensors to discover at what level they worked the best.

To calibrate my IR proximity detectors, I converted the analog output of the sensor to a digital value and displayed it on PORTC which contains an array of LED's. I used a Wait subroutine to display the digital voltage value on PORTC long enough for me to write down the values. I then had to convert the binary number into decimal so that I could understand how these sensors operated. The graph from this experiment is displayed in the Sensors section of this report. I also used the same method to calibrate my ultrasonic proximity detectors. The only difference for the Ultrasonic proximity detector was that I also had to determine how many readings I needed so that the average of the readings would provide a consistent distance value.

I also found it necessary to write test code separately for each of Mulebot's subsystems. Most of this test code is located in the appendix of this report.

Conclusion

When everything was completed on Mulebot, it was able to follow me at about a separation distance of about 5 or 6 feet with me pointing the homing device in the direction of Mulebot. I hoped that I would be able to clip my homing device to my belt, and have Mulebot follow me without holding the homing device towards it. I think that it would be possible to design a similar Robot to Mulebot with much better performance. This could be accomplished by eliminating the noise(or isolating it from the sonar receivers) caused by the motors, the IR proximity detectors, the RF transmitting unit, and other sources. One thing that I would change is that I would not share common ground between my motors and the other subsystems. The motors generate a lot of noise, and this limits the reception of my sonar receivers.

Documentation

I got the idea to use the ultrasonic receivers as ears for this robot from the following website: http://www.circuitcellar.com/library/print/1102/baraniak148/2211014.pdf

Source of Parts

Ultrasonic transducers	www.jameco.com
All other electronics components	www.digikey.com
RF Transmitter and Receiver	www.rentron.com
12V DC Windshield Wiper Motors	www.allelectronics.com
ATMega163 Board	www.prllc.com
12V DC SLA Battery	www.batterycountry.com
D100-B Motor Driver Boards	www.tecel.com

I would to thank T.A.'s Uriel Rodriguez and Jason Plew for their help and advice on this project.

Appendices

Here is my some of my test code, and final code for Mulebot:

Originally, I used a Stepper Motor from Jameco to control the steering of Mulebot. I found that this does not work very well because of mechanical reasons. However, I will include the code that I wrote for that:

```
/* I will use this program to test my 4 phase unipolar stepper motor.
The stepper motor driver(Allegro 5804)has two inputs, the step input and the direction input.
The stepper motor steps when a transition occurs on the step input.  The
stepper motor will rotate one direction for a high on the direction pin, and
will rotate the other direction if the input is zero.  This program controls
the speed and direction of the motor.*/

#include <io.h>
#include <interrupt.h>
#include <sig-avr.h>
```

```
#define AVR_MEGA 0

/*define constants*/

int Hard =       30;
int Medium =     15;
int Soft =       5;

/*subroutine causes a delay, used in this program to control speed of motor*/
void Wait(int time)
{
volatile int a, b, c, d;
for (a = 0; a < time; ++a) {
for (b = 0; b < 10; ++b) {
for (c = 0; c < 66; ++c) {
d = a + 1;
}
}
}
return;
}


/* This routine will cause the stepper motor to step the specified value to the right*/

void SMotorRight(int steps)
{
    /*in this routine, the direction pin is 1*/
    int i;
    for(i=0; i<steps; ++i){
      sbi(PORTA, 1); /*sets pins A1(dir)to 1*/
      sbi(PORTA, 0); /* and A0(speed) to 1*/
      Wait(5);
      cbi(PORTA, 0); /*sets A0 (speed) to 0*/
      Wait(5);
    }
    return;
}

/* This routine will cause the stepper motor to step the specified value to the left*/

void SMotorLeft(int steps)
{

    /*in this routine, the direction pin is 0*/
    int i;
    for(i=0; i<steps; ++i){
    cbi(PORTA, 1); /*sets A1(dir) to 0*/
    sbi(PORTA, 0); /*sets A0(speed) to 1*/
    Wait(5);
    cbi(PORTA, 0); /*sets pin A0 speed to 0*/
    Wait(5);
    }
    return;
}
```

```
void Straight(int time)
{
      cbi(PORTA, 0);
     cbi(PORTA, 1); /*sets pins A0(dir) to 0 and A1(speed) to 0*/
     Wait(time);
}

int main(void)
{
  outp(0x00, DDRA);  /*configures DDA0 as input pin*/
  int j;
  for(j=0; j<5; j++){
            SMotorLeft(Hard);
            Straight(200);
            SMotorRight(Hard);
            Straight(400);
            SMotorRight(Medium);
            Straight(150);
            SMotorLeft(Medium);
            Straight(400);

  }
}

//********************************************************

/*this program will perform obstacle avoidance using 6 bump switches.
This program will work using two D100 motor driver boards from TECEL, and using
the progressive AVRMEGA163 board.  For more sensitivity on the three IR detectors,
lower the threshold value.  The 3 IR detectors used for this code was the GPD120.*/

#include <io.h>
#include <interrupt.h>
#include <sig-avr.h>
#define AVR_MEGA 0
#define error 0x96


int threshold = 0x2B; /*this is the value obtained for the IR being 5 inches away*/
int cthreshold = 0x30;
int rmotor = 0x30;
int lmotor = 0x30;
int speedmotors = 0x30;
typedef unsigned char u08;
u08 IR_Right = 0;
u08 IR_Center = 0;
u08 IR_Left = 0;

int main(void)
{
configureMotorDrivers();
setUpADC();
setUpBumpSwitches();
stop();
outp(0xFF,DDRC);
```

```
Wait(200);

for(;;){

  sbi(PORTB, PB7); //generates rf signal
   register u08 LED = inp(PINB);
   register u08 bump = inp(PINB)&0x3F;
   register u08 f_left = bump&0x01;
   register u08 f_c_left = bump&0x02;
   register u08 f_c_right = bump&0x04;
   register u08 f_right = bump&0x08;
   register u08 b_right = bump&0x10;
   register u08 b_left = bump&0x20;
   outp(LED,PORTC);

  /*IR detection*/
  IR_Right  =  analog(0x20);/*get value from PA0*/
  IR_Center =  analog(0x22);/*get value from PA2*/
  IR_Left   =  analog(0x24);/*get value from PA4*/

  /*write if statements to cover the 8 possible situations arising from
  the three infared signals*/
  Wait(150);
  cbi(PORTB, PB7); //turns of rf signal, creates pulse.

  if (IR_Left>error || IR_Right>error || IR_Center>error) {
     Wait(10);
  }

  else if (IR_Right<threshold && IR_Center<threshold && IR_Left>threshold){
     forward();
     right();
  }
  else if (IR_Right<threshold && IR_Center>threshold && IR_Left<threshold){
     reverse();
     randomDir();
  }
  else if (IR_Right<threshold && IR_Center>threshold && IR_Left>threshold){
     reverse();
     randomDir();
  }
  else if (IR_Right>threshold && IR_Center<threshold && IR_Left<threshold){
     forward();
     left();
  }
  else if (IR_Right>threshold && IR_Center<threshold && IR_Left>threshold){
     forward();
     straight(0x25);
     /*in this situation, the robot will go slow because it might be about to crash*/
  }
  else if (IR_Right>threshold && IR_Center>threshold && IR_Left<threshold){
     reverse();
     randomDir();
  }
  else if (IR_Right>threshold && IR_Center>threshold && IR_Left>threshold){
     reverse();
```

```
    randomDir();
}

/*bump detection*/
/*I have 4 switches up front and 2 in the back*/
/*PB0 is front left
  PB1 is front center left
  PB2 is front center right
  PB3 is front right
  PB4 is back right
  PB5 is back left*/

else if (f_left == 0)
{
  reverse();
  left();
  Wait(150);
}
else if (f_c_left == 0)
{
  reverse();
  hardleft();
  Wait(150);
}
else if (f_c_right == 0)
{
  reverse();
  hardright();
  Wait(150);
}
else if (f_right == 0)
{
  reverse();
  right();
  Wait(150);
}
else if (b_right == 0)
{
  forward();
  right();
  Wait(150);
}
else if (b_left == 0)
{
  forward();
  left();
  Wait(150);
}
else
{
forward();
straight(speedmotors);
}

outp(lmotor,OCR1AL);
outp(rmotor,OCR1BL);
```

```
}
}


/*note: one input of each switch is grounded, and the other goes to port B*/

void setUpBumpSwitches(void){

   cbi(DDRB, PB0);  /*the following lines configure port b as inputs*/
   cbi(DDRB, PB1);
   cbi(DDRB, PB2);
   cbi(DDRB, PB3);
   cbi(DDRB, PB4);
   cbi(DDRB, PB5);
   sbi(PORTB, PB0); /*the pullup resistor on these pins are activated*/
   sbi(PORTB, PB1);
   sbi(PORTB, PB2);
   sbi(PORTB, PB3);
   sbi(PORTB, PB4);
   sbi(PORTB, PB5);
}

void setUpADC(void){
   outp(0xCE,ADCSR);
   outp(0x20,ADMUX);
}

void configureMotorDrivers(void){
/*configure port D as outputs and set up pwm*/
   sbi(DDRD, PD4); /*pwm motor1*/
   sbi(DDRD, PD5); /*pwm motor2*/
   sbi(DDRD, PD2); /* IN2 motor1*/
   sbi(DDRD, PD3); /* IN1 motor1*/
   sbi(DDRD, PD6); /* IN1 motor2*/
   sbi(DDRD, PD7); /* IN2 motor2*/
   outp(0xA1,TCCR1A);
   outp(0x04,TCCR1B);
   outp(0x00,TCNT1H);
   outp(0x00,TCNT1L);
   outp(0x00,OCR1AH);
   outp(0x00,OCR1BH);
}

void reverse(void)
{
   cbi(PORTD, PD7); /*this is the IN1 pin*/
   sbi(PORTD, PD6); /*this is the IN2 pin*/
   cbi(PORTD, PD3); /*this is the IN1 pin*/
   sbi(PORTD, PD2); /*this is the IN2 pin*/
}

void forward(void)
{
   sbi(PORTD, PD7); /*this is the IN1 pin*/
   cbi(PORTD, PD6); /*this is  the IN2 pin*/
```

```
  sbi(PORTD, PD3); /*this is the IN1 pin*/
  cbi(PORTD, PD2); /*this is the IN2 pin*/
}

void stop(void)
{
  cbi(PORTD, PD7); /*this is the IN1 pin*/
  cbi(PORTD, PD6); /*this is the IN2 pin*/
  cbi(PORTD, PD3); /*this is the IN1 pin*/
  cbi(PORTD, PD2); /*this is the IN2 pin*/
}
void straight(int speed)
{
  rmotor = speed;
  lmotor = speed;
}

void left(void)
{
  rmotor = 0x30;
  lmotor = 0x15;
  Wait(150);
}

void hardleft(void)
{
  rmotor = 0x30;
  lmotor = 0x10;
  Wait(150);
}

void right(void)
{
  rmotor = 0x15;
  lmotor = 0x30;
  Wait(150);
}

void hardright(void)
{
  rmotor = 0x10;
  lmotor = 0x30;
  Wait(150);
}
/*subroutine causes a delay,*/
void Wait(int time)
{
volatile int a, b, c, d;
for (a = 0; a < time; ++a) {
for (b = 0; b < 10; ++b) {
for (c = 0; c < 66; ++c) {
d = a + 1;
}
}
}
return;
```

```
}

u08 analog(u08 channel)
{
u08 sample_L, sample_H;
outp(channel, ADMUX);
sbi(ADCSR, ADSC); /* begin conversion */
loop_until_bit_is_set(ADCSR, ADIF);
sample_L = inp(ADCL); /* get low 8 bits */
sample_H = inp(ADCH);/* get high bits */
sbi(ADCSR, ADIF); /* clear ADC interrupt flag */
return sample_H;
}


int randomDir(void){
/*generates random direction and speed*/
if (TCNT1L > 0x80){
  randomTurnSpeedRight();
  }
else{
  randomTurnSpeedLeft();
  }
}
//I will use this to generate random speeds.
// I will specifically use this when backing up.
int randomTurnSpeedLeft(void){
if (TCNT1L < 0x55){
  rmotor = 0x30;
  lmotor = 0x09;
  Wait(150);
  }
else if (0x55 < TCNT1L < 0xa0){
  rmotor = 0x40;
  lmotor = 0x10;
  Wait(150);
  }
else{
  rmotor = 0x35;
  lmotor = 0x10;
  Wait(150);
  }
}

int randomTurnSpeedRight(void){
if (TCNT1L < 0x55){
  lmotor = 0x30;
  rmotor = 0x09;
  Wait(150);
  }
else if (0x55 < TCNT1L < 0xa0){
  lmotor = 0x40;
  rmotor = 0x10;
  Wait(150);
  }
else{
   lmotor = 0x35;
   rmotor = 0x10;
   Wait(150);
   }
}
//*******************************************************************
//final code
#include <io.h>
#include <interrupt.h>
#include <sig-avr.h>
#include <math.h>
#define AVR_MEGA 0

typedef unsigned char u08;
int speedupdist = 0x15;
int slowdowndistanceh = 0x13;

volatile int speedup = 0;
volatile int slowdown = 0;

volatile char inc = 0;
volatile char timerval0 = 0;
volatile char timerval2 = 0;
volatile char difference = 0;
volatile char goright = 0;
volatile char goleft = 0;

volatile int rmotor = 0x30;
volatile int lmotor = 0x30;
volatile int speedmotors = 0x30;
int threshold = 0x30; /*this is the value obtained for the IR being 5 inches away*/

u08 IR_Right = 0;

int main(void)
{
outp(0xCF,ADCSR);
outp(0x26,ADMUX);

configureMotorDrivers();

cbi(DDRD,PD2);
cbi(DDRD,PD3);
outp(0xFF, PORTD);
outp(0xFF,DDRC);
outp(0xff, DDRB);

cbi(MCUCR, ISC11);  //these four commands initiate initiate int1 and int2 when
cbi(MCUCR, ISC10);  //PD2 and PD3 are logic low
cbi(MCUCR, ISC01);
cbi(MCUCR, ISC00);
outp(0x01, TIMSK);   //enable timer0 overflow interrupt

for(;;){

  cli();
```

```
/*IR detection*/

u08 IR_Righta  =  analog(0x20);/*get value from PA0*/
u08 IR_Rightb  =  analog(0x20);/*get value from PA0*/
u08 IR_Rightc  =  analog(0x20);/*get value from PA0*/


IR_Right = (IR_Righta + IR_Rightb + IR_Rightc)/3;

sbi(PORTB, PB7);
Wait(40);
//need to get average since output of sonar is square wave
//with amplitude proportional to distance
register u08 sonara = analog(0x26);
register u08 sonarb = analog(0x26);
register u08 sonarc = analog(0x26);
register u08 sonard = analog(0x26);
register u08 sonare = analog(0x26);
register u08 sonarf = analog(0x26);
register u08 sonarg = analog(0x26);
register u08 sonarh = analog(0x26);
register u08 sonari = analog(0x26);
register u08 sonarj = analog(0x26);
register u08 sonark = analog(0x26);
register u08 sonarl = analog(0x26);
register u08 sonarm = analog(0x26);
register u08 sonarn = analog(0x26);
register u08 sonaro = analog(0x26);
register u08 sonarp = analog(0x26);
register u08 sonarq = analog(0x26);
register u08 sonarr = analog(0x26);
register u08 sonars = analog(0x26);
register u08 sonart = analog(0x26);
register u08 sonaru = analog(0x26);
register u08 sonarv = analog(0x26);
register u08 sonarw = analog(0x26);
register u08 sonarx = analog(0x26);
register u08 sonary = analog(0x26);
cbi(PORTB, PB7);
Wait(30);

 u08 sonar = (sonara + sonarb + sonarc + sonard + sonare + sonarf + sonarg +
 sonarh + sonari + sonarj + sonark + sonarl + sonarm + sonarn + sonaro + sonarp + sonarq +sonarr
 + sonars + sonart + sonaru + sonarv + sonarw + sonarx +sonary)/25 ;

 if (sonar > speedupdist){
   speedup =  1;
   slowdown = 0;
 }

 if (sonar < slowdowndistanceh){
   speedup =  0;
   slowdown = 1;
 }
```

```
   outp(0x00, TCNT0); //reset all timer values and variables.
   outp(0x00, TCNT2);
   inc = 0;
   timerval0 = 0;
   timerval2 = 0;
   sei();  //enable i bit for interrupts
   sbi(GIMSK, INT0);  //this will enable the external interrupt 0
   sbi(GIMSK, INT1);  //enable external interrupt 1.

   sbi(PORTC, PC7);
   sbi(PORTC, PC6);
   sbi(PORTC, PC5);
   sbi(PORTC, PC4);
   sbi(PORTC, PC3);
   sbi(PORTC, PC2);

   outp(0x05, TCCR0);  //select CK/1024 and start timer 0
   outp(0x07, TCCR2);  //selects CK/1024 and starts Timer 2
   sbi(PORTB, PB7);
   Wait(50);
   cbi(PORTB, PB7);
   Wait(70);

   if(timerval0 != 0 && timerval2 != 0 && speedup == 1  ){ //in this case both interrupts occured, so get
values
     difference = fabs(timerval0 - timerval2); //speedup and go straight

     speedmotors = speedmotors + 6;

     if(difference > 1 && goright == 1){ //turn right
       cbi(PORTC, PC6);
       forward();
       right(speedmotors);
     }
     else if(difference > 1 && goleft == 1){ //turn left
       cbi(PORTC,PC5);
       forward();
       left(speedmotors);
     }
     else{  //go straight
       cbi(PORTC, PC4);
       forward();
       straight(speedmotors);
     }
       cbi(PORTC, PC2);
       Wait(30);

   }

   else if(timerval0 != 0 && timerval2 != 0 && slowdown == 1 ){ //in this case both interrupts occured, so
get values
     difference = fabs(timerval0 - timerval2); // slow down and go straight

     if (speedmotors > 0x18){
       speedmotors = speedmotors - 10;
```

```
    }

    if(difference > 1 && goright == 1){
      cbi(PORTC, PC6);
      forward();
      right(speedmotors);
    }
    else if(difference > 1 && goleft == 1){
      cbi(PORTC,PC5);
      forward();
      left(speedmotors);
    }
    else{
      cbi(PORTC, PC4);
      forward();
      straight(speedmotors);
    }

    cbi(PORTC, PC3);
    Wait(30);
  }

  else if(IR_Right > threshold){
     reverse();
     leftIR();
     cbi(PORTC, PC4);
     Wait(50);
  }

  else if (timerval0 != 0 && timerval2 == 0){
  cbi(PORTC, PC6); //cooresponds to turn right
  forward();
  right(speedmotors);
  Wait(50);
  }

  else if (timerval0 == 0 && timerval2 != 0){
  cbi(PORTC, PC5); //cooresponds to turn left
  forward();
  left(speedmotors);
  Wait(50);
  }


  else if (timerval0 == 0 && timerval2 == 0){ //cooresponds to no signal
  cbi(PORTC, PC7);
  stop();
  speedmotors = 0x30;
  Wait(50);
  }

  outp(rmotor,OCR1AL);
  outp(lmotor,OCR1BL);

}
}
```

```
/*subroutine causes a delay,*/
void Wait(int time)
{
volatile int a, b, c, d;
for (a = 0; a < time; ++a) {
for (b = 0; b < 10; ++b) {
for (c = 0; c < 66; ++c) {
d = a + 1;
}
}
}
return;
}

 SIGNAL(SIG_INTERRUPT1) //PD3
  {
    cbi(GIMSK, INT1);  //this will disable interrupt 1
    outp(0x00, TCCR0); //stop timer 0
    timerval0 = TCNT0;
    if (timerval2 == 0){  //in this case, this ear received first
    goright = 1;
    }
    else{
    goright = 0;
    }
  }

SIGNAL(SIG_INTERRUPT0) //PD2
{
   cbi(GIMSK, INT0);  //disable the interrupt 0.
   outp(0x00, TCCR2);  //this will stop Timer 2
   timerval2 = TCNT2;
   if (timerval0 == 0){  //in this case, this ear received first
   goleft = 1;
   }
   else{
   goleft = 0;
   }
}

SIGNAL(SIG_OVERFLOW0)
{
   inc = inc + 1;

}

u08 analog(u08 channel)
{
u08 sample_L, sample_H;
outp(channel, ADMUX);
sbi(ADCSR, ADSC); /* begin conversion */
loop_until_bit_is_set(ADCSR, ADIF);
sample_L = inp(ADCL); /* get low 8 bits */
sample_H = inp(ADCH);/* get high bits */
sbi(ADCSR, ADIF); /* clear ADC interrupt flag */
return sample_H;
}
void configureMotorDrivers(void){
/*configure port D as outputs and set up pwm*/
   sbi(DDRD, PD4); /*pwm motor1*/
   sbi(DDRD, PD5); /*pwm motor2*/
   sbi(DDRC, PC0); /* IN2 motor1*/
   sbi(DDRC, PC1); /* IN1 motor1*/
   sbi(DDRD, PD6); /* IN1 motor2*/
   sbi(DDRD, PD7); /* IN2 motor2*/
   outp(0xA1,TCCR1A);
   outp(0x04,TCCR1B);
   outp(0x00,TCNT1H);
   outp(0x00,TCNT1L);
   outp(0x00,OCR1AH);
   outp(0x00,OCR1BH);
}
void reverse(void)
{
   cbi(PORTD, PD7); /*this is the IN1 pin*/
   sbi(PORTD, PD6); /*this is the IN2 pin*/
   cbi(PORTC, PC0); /*this is the IN1 pin*/
   sbi(PORTC, PC1); /*this is the IN2 pin*/
}

void forward(void)
{
   sbi(PORTD, PD7); /*this is the IN1 pin*/
   cbi(PORTD, PD6); /*this is the IN2 pin*/
   sbi(PORTC, PC0); /*this is the IN1 pin*/
   cbi(PORT C, PC1); /*this is the IN2 pin*/
}

void stop(void)
{
   cbi(PORTD, PD7); /*this is the IN1 pin*/
   cbi(PORTD, PD6); /*this is the IN2 pin*/
   cbi(PORTC, PC0); /*this is the IN1 pin*/
   cbi(PORTC, PC1); /*this is the IN2 pin*/
}

void straight(int speed)
{
   if (speed > 0x20){
   rmotor = speed;
   lmotor = speed;
   }
   else{
   rmotor = 0x20;
   lmotor = 0x20;
   cbi(PORTC, PC7);

   }
}

void left(int speed)
```

```
{
  if (speed > 0x20){
   rmotor = 0x45;
   lmotor = 0x30;
  }
  else {
   rmotor = 0x23;
   lmotor = 0x08;
   cbi(PORTC, PC7);

  }
}

void right(int speed)
{
   if (speed > 0x20){
   lmotor = 0x45;
   rmotor = 0x30;
   }
   else{
   rmotor = 0x08;
   lmotor = 0x23;
   cbi(PORTC, PC7);

   }
}

void rightIR(void)
{
   lmotor = 0x08;
   rmotor = 0x40;
}

void leftIR(void)
{
   lmotor = 0x40;
   rmotor = 0x08;
}
```