University of Florida
Department of Electrical and Computer Engineering
EEL 5666
Intelligent Machines Design Laboratory

GatorVac
Written Report 2

By
M. Gabriel Jiva
July 8, 2003

Table of Contents

Abstract

The GatorVac is an autonomous vacuum. It is turned on, and proceeds to vacuum an entire room in a random manner, following walls, and avoiding obstacles. It does this using a combination of IR and bump sensors, IR beacons, and high-torque servos. The body is a platform made to accommodate a Dustbuster,. Once finished the GatorVac finds its way to its base.

Executive Summary

The GatorVac is an autonomous vacuum. It randomly vacuums a room and follows walls and other large obstacles as it moves. IR beacons help it home in on its base, once finished.

The microcontroller is an ATMega323, using a Progressive Resources board. PORT A is used for the ADC, for the IR detectors to interface to the board. PORTB is used as an input port to interface the IR beacon. PORTC is connected to an LED bank, and is used to provide information about the status of the robot, such as when it detects an obstacle, and what direction it detects the IR beacon. PORTD is used for its external interrupt, to which the bump sensor is connected, and also for the 2-channel PWM timer, which drives the actuation servos.

The combination IR detectors and bump sensor works very well to achieve obstacle avoidance. The detectors were calibrated so that the robot slows down once the detectors see something, and once the object is too close, it takes action, either by backing up or just turning, in order to not collide. There is a small blind spot between the two IR detectors, and the bump sensor is placed there.

The IR beacon, and its brother at the home base can detect each others' IR emissions, and the direction from which they are coming. GatorVac then uses this information to navigate back towards the other beacon, at the home base. This works rather coarsely, as the inaccurate nature of the wheels do not allow the robot to travel in a perfectly straight line, and adjustments have to be made on the go. Another impediment is that the robot still has to avoid obstacles on the way to its home base, which sometimes causes it to veer even further off-course.

Therefore, while the homing-in function needs some tweaking, the object avoidance, vacuuming, and wall-following functions have been fully achieved.

Introduction

Vacuum cleaners have come a long way since their invention at the turn of the century. They have become electrical, and then more and more powerful, so that they can suck up more and more dirt. However, for the past decade, there have been virtually no advancements in vacuuming technology. Some manufacturers have produced self-propelled vacuum cleaners, but clearly the next major advancement in the vacuuming industry is going to be automation. The technology is there, and it is ripe for using.

The automated vacuum cleaner is something most of us would like to have in our home. Few people like vacuuming, and most people consider it a chore. Therefore, the GatorVac is something that most people would like to see built.

This paper is report on the construction of GatorVac, which is a body designed to carry a Dustbuster, which actually does the vacuuming. The robot follows walls, couches, and other such large obstacles, and avoids making contact with most objects. After it is done, it finds its home base, indicated by a directional IR beacon.

Integrated System

The integrated system consists of 4 sensors for input, a microcontroller for data interpretation, and motors for output.

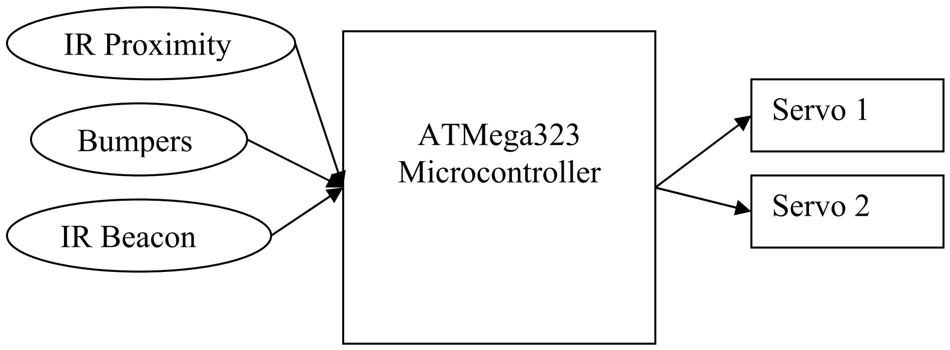

Figure 1

Figure 1 shows a general view of the integrated systems for GatorVac. The IR Proximity and bumper switches comprise an object avoidance system, while the IR beacon provides home-base locating. The microcontroller uses the information provided by the sensors to control the motors, and therefore, the direction in which it travels.

Mobile Platform

The platform used is made to accommodate a Dustbuster vacuum cleaner. The auxiliary circuitry is mounted on the inside (bottom surface), and the ATMega board will is mounted on the top surface. The Dustbuster will be mounted on the incline. The two servos are mounted on either side of the platform, and a ping-pong ball is mounted towards the front for extra support.

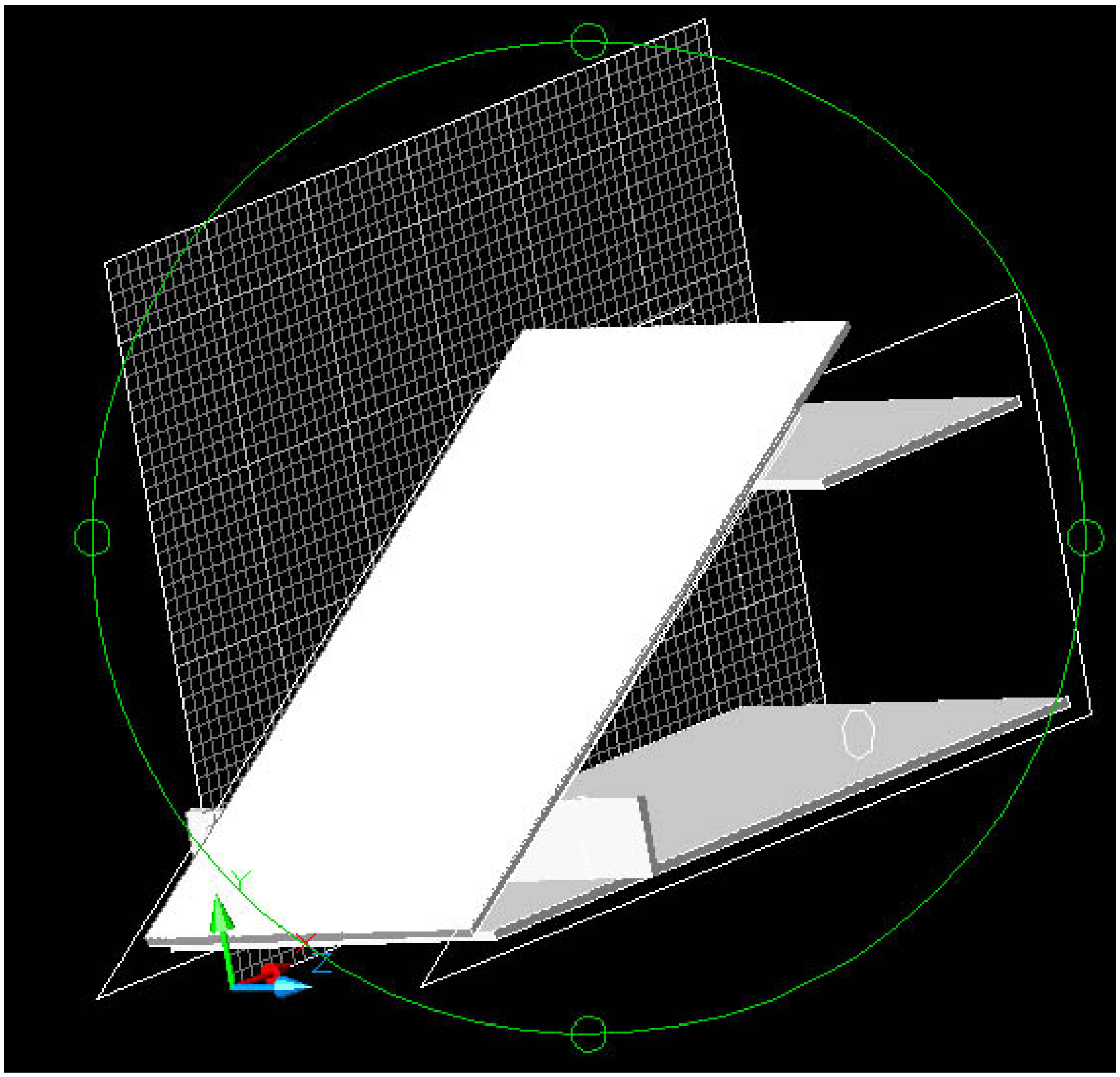

Figure 2

Since the Dustbuster hangs off of the platform, I added a U-shaped attachment, which goes around the Dustbuster and connects to the rest of the platform, in order to be able to put a bump-sensor on the front of the robot. The IR detectors had to be then recalibrated, so that a lower number would indicate a collision, since the robot's front protruded much farther.

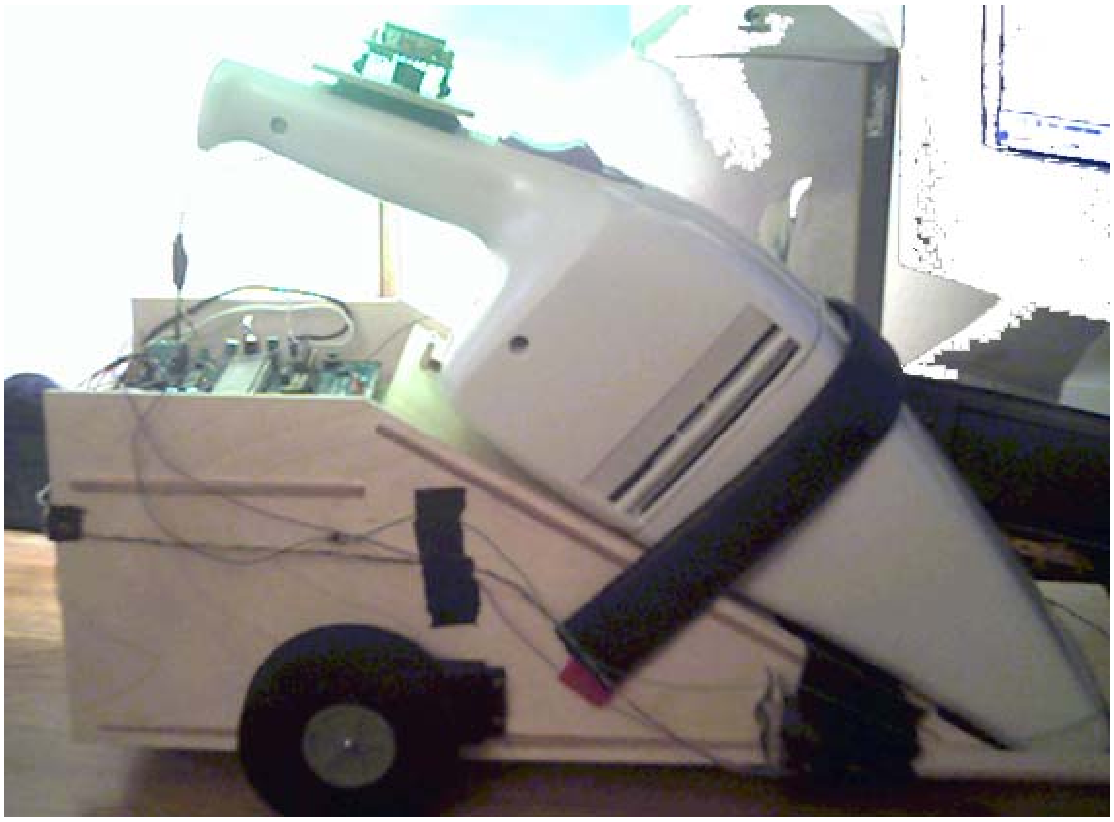

Figure 3

Figure 3 shows the final robot. The Dustbuster's tail holds the IR beacon at nearly the highest point on GatorVac, and the auxiliary board is unseen inside the platform.

Actuation

The robot is controlled by two heavy-duty servos, which are needed because of the relatively heavy body of the Dustbuster. A third support mechanism is a hemisphere ping-pong ball, at the front of the platform.

The servos used are Hitachi brand, with the following specifications.

HS-700BB Large Scale Servo

- Top Ball Bearing
- Water tight
- Large Size: 2.3" x 1.1" x 2.0"
- Torque: 174 oz.-in.
- Weight: 3.68 oz.
- Speed: 0.17s / 60 degrees

Figure 4.

A large, 6”x1.5” foam wheel was attached to the servo in order to provide adequate traction and support. The servos and wheels were obtained from LynxMotion. I hacked the servos in order to obtain full range of motion from them. In order to do that, I had to cut a mechanical stop off of a gear, two mechanical stops from the gear housing, and one from the potentiometer encasement. I also had to remove the potentiometer and replace it with a voltage divider circuit, in order to fool the circuitry into thinking the potentiometer was still there. Also, on one of the motors, I reversed the polarity of power, so that the same instruction would give the servos different direction (forward is clockwise for the right servo, and counterclockwise for the left one), in order to make the programming a little less confusing.

Sensors

The sensors used in the creation of GatorVac are as follows:

IR Sensors

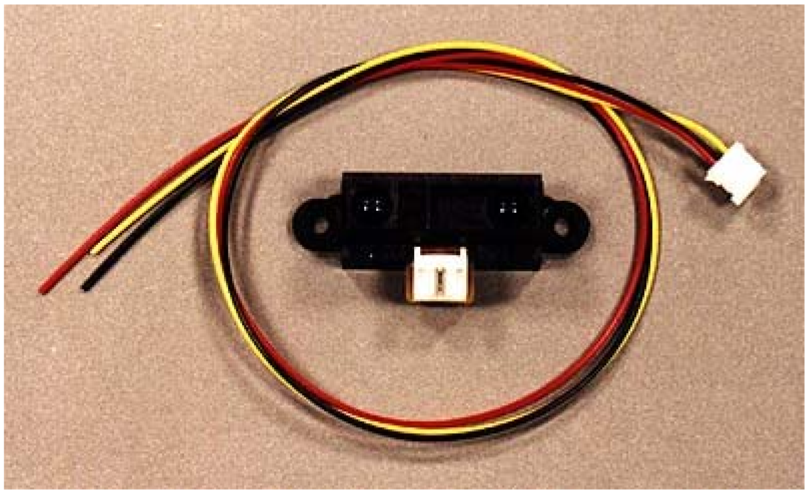

Figure 5.

Two Sharp GP2D12 Infrared sensors are used for GatorVac. They are mounted on either side of the platform, on specially designed "ears". I was going to use a third sensor placed on the platform itself in order to compensate for a small blind spot, but it would have been very difficult to place a sensor there, and experimentation showed that the blind spot was very thin. The sensors features are:

- Less influence on the color of reflected objects, reflectivity
- Analog voltage corresponding to distance
- Detecting distance of 10 to 80 cm
- External control circuit unnecessary
- Low cost

The microcontroller's built-in ADC converts the output of the sensor into a number, which it then uses to figure out the distance to the object it is sensing. I did testing, and 52 was the lowest number that was detected, while 130 was the highest. However, if the 130 proximity was reached, the collision will have already occurred. Therefore, I set the "too close" value to be at 85, which was about 1" in front of the robot. If it senses a value higher than 85, it automatically backs up. At values of less than 70, but more than 52, which corresponds to circa a foot in front of the robot, GatorVac slows down and approaches with caution.

The GP2D12 sensors were obtained from Acroname Robotics.

Bump switches

These are simple button devices that, if pushed, indicate a collision with another object. I designed a frontal attachment on which the bump switch was placed.
The switch was connected to an external IRQ on the microcontroller board, so that readings will only be taken if the bumper is triggered. This is an unlikely event, as the shape of the platform makes it much more likely for a corner to bump instead of the center, since the robot will almost never collide exactly head-on. The function of the bump sensor is the take care of the blind spot left by the IR proximity detectors, in that unlikely circumstance.

IR Beacons

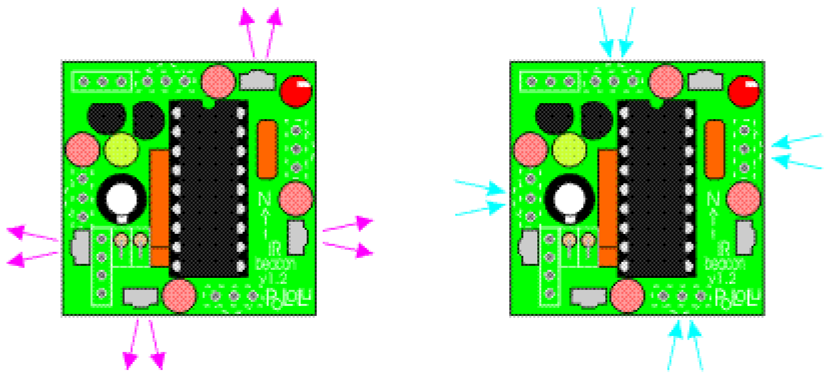

Figure 6

IR Beacons are used to locate the direction of the home base. The product purchased consists of 2 identical devices. These were actually kits, with 25 parts that I then had to solder together. The beacon is actually a series of visible light LEDs, infrared LEDs, and Infrared detectors. They are all connected to a PIC microcontroller that controls their emission and detection rates.

The beacons have power, ground and enable inputs, and 4 outputs: north, south, east and west. The outputs are low-true, and they are asserted whenever the beacon senses infrared from another beacon, from that particular direction.

Testing on the beacons shows that they have a 10-12' range, and that direction does not matter, as it detects the signal from all directions. Since the frequency is 56kHz, certain household remote controls can also fool the sensor into thinking it is detecting an IR beacon.

Device specifications

PCB size:	1.3" x 1.3"
IR modulation frequency:	56 kHz
Output Refresh rate	20 Hz
Detection range:	6 inches to 20 feet
Supply voltage:	5.1-10 V
Data voltage:	0 and 5 V
Number of IR detectors:	4
Components in kit:	25

The beacons have 4 digital outputs which toggle when IR is detected by the corresponding detector. There are 4 detectors, one on each side of the square PCB, and next to each detector is also a visible LED which is turned on when the IR detector detects IR. Since the IR detectors are positioned at right angles, it is possible to extrapolate the direction from which the infrared is emitted (e.g., NW relative to the PCB). Then, it is simply a matter of directing the servos to propel the GatorVac in the direction.

Behaviors

The GatorVac will exhibit four behaviors: vacuuming, collision avoidance, wall-following, and base locating. The following flow chart indicates the different states the robot takes:

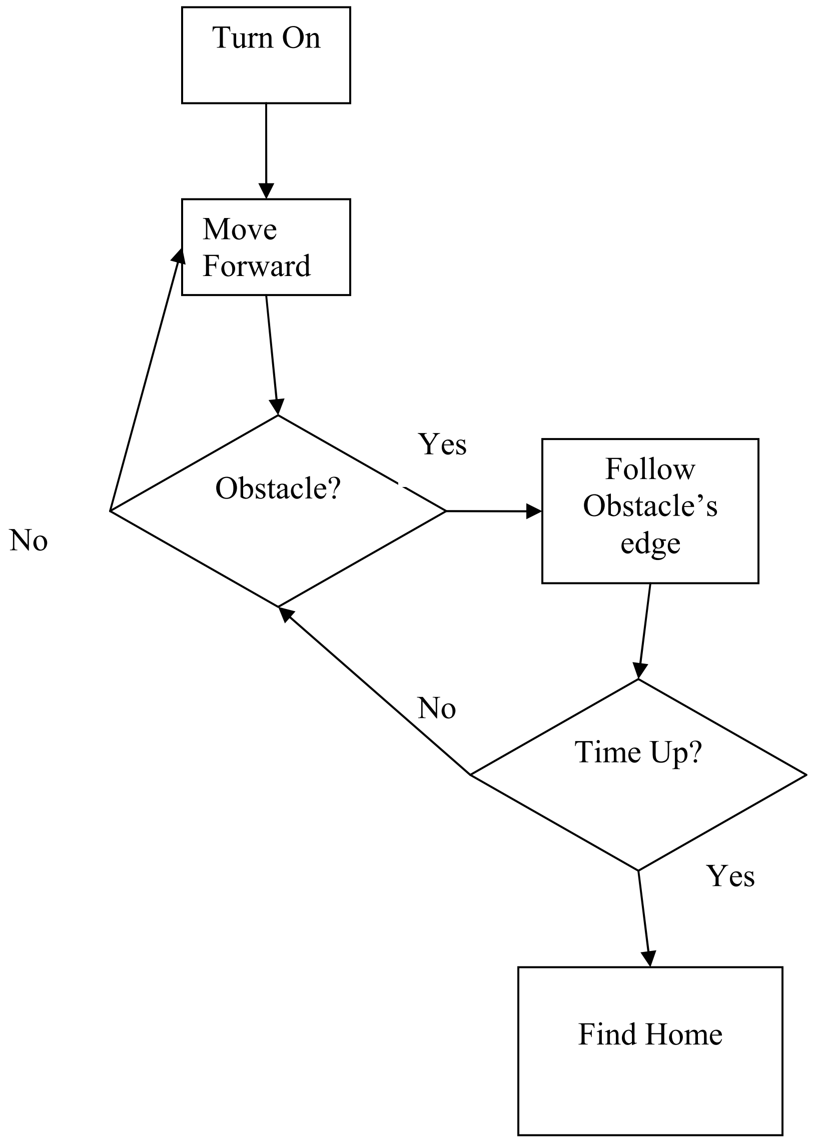

Figure 7

Object avoidance

The robot uses the IR proximity detectors in conjunction with the bump sensor to detect and avoid collision with objects. Since it has 2 IR detectors, it has binocular vision, and acts differently if it sees an object closer in the right detector, than if it saw them at the same distance in both detectors. There is one spot where the IR detectors fail to observe anything, because it is a blind spot, and the bump sensor is there to take care of that eventuality.

Wall-following

Whenever the robot detects an obstacle it turns slightly away from it. This enables it to run alongside the obstacle. If it happens to veer towards it, it turns away again. This behavior is important in a vacuum cleaner so that it vacuums around couches, tables, etc, where most of the dirt is, instead of just vacuuming open spaces.

Vacuuming

This behavior is a combination of the previous two behaviors, which results in a random path, which turns out to be the best way to cover more than 90% of a room. Studies show that if a random path is followed in a medium sized room, more than 90% of the floor surface will be covered in 15 minutes.

Home-base return

After a certain period of time, which is programmed into it, GatorVac stops vacuuming and tries to return to its home base, which is indicated by an IR beacon. It uses another IR beacon to accomplish this task by extrapolating the direction of the beam, and then propelling itself in that direction. This is by far the hardest behavior, as the robot is not accurate, and therefore has trouble returning to its base.

Conclusion

The robot, GatorVac, was successful in its endeavors. It avoids obstacles splendidly, follows walls very well, and makes a great autonomous vacuum cleaner. The only behavior that did not work as well as expected is the return-to-base function, which seems to require higher quality servos and directional beacons in order to work as well as anticipated.

This project, while very time-consuming, was also extremely rewarding, as I overcame a lot of technical difficulties in putting the robot together. It also gave me experience with the ATMega family of processors, which is the third that I’m familiar with, besides the PIC and Motorola 68HC11.

Appendix

Special thanks to David Winkler, whose drive function I adapted, along with some other snippets of code.

```
/*********************************************
This program was produced by the
CodeWizardAVR V1.23.8 Evaluation
Automatic Program Generator
© Copyright 1998-2003 HP InfoTech s.r.l.
http://www.hpinfotech.ro
e-mail:office@hpinfotech.ro

Project :
Version :
Date    : 8/7/2003
Author  : M. Gabriel Jiva
Company :
Comments:


Chip type           : ATmega323
Program type        : Application
Clock frequency     : 6.000000 MHz
Memory model        : Small
External SRAM size  : 0
Data Stack size     : 512
*********************************************/

#include <mega323.h>
#include <delay.h>
#include <stdio.h>

#define GOHOMETIME 2000

// Declare your global variables here
long int gohome=0;
short int speed_val_l=0;
short int speed_val_r=0;
int ir,rand,rand1;
bit N,E,S,W,bumper=0,north,signal,home=0;

void disable_pwm()
{TCCR1A=0x01;}
void enable_pwm()
{TCCR1A=0xA1;}
```

```
int random(unsigned char range){
//generates random numbers between 0-254
range++;
return TCNT1L % range;
}

void drive(int,int);
int check_ir();
void obstacle_avoidance();

// External Interrupt 0 service routine
interrupt [EXT_INT0] void ext_int0_isr(void)
{
bumper=1;

}

// Timer 1 overflow interrupt service routine
interrupt [TIM1_OVF] void timer1_ovf_isr(void)
{
gohome++;

}

#define FIRST_ADC_INPUT 0
#define LAST_ADC_INPUT 2
unsigned char adc_data[LAST_ADC_INPUT-FIRST_ADC_INPUT+1];
#define ADC_VREF_TYPE 0x20
// ADC interrupt service routine
// with auto input scanning
#pragma savereg-
interrupt [ADC_INT] void adc_isr(void)
{
#asm
   push r26
   push r27
   push r30
   push r31
   in   r30,sreg
   push r30
#endasm
register static unsigned char input_index=0;
// Read the 8 most significant bits
// of the AD conversion result
adc_data[input_index]=ADCH;
// Select next ADC input
```

```
if (++input_index > (LAST_ADC_INPUT-FIRST_ADC_INPUT))
  input_index=0;
ADMUX=(FIRST_ADC_INPUT|ADC_VREF_TYPE)+input_index;
// Start the AD conversion
ADCSR|=0x40;
#asm
   pop  r30
   out  sreg,r30
   pop  r31
   pop  r30
   pop  r27
   pop  r26
#endasm
}
#pragma savereg+

/**********************************************
Function: drive
Inputs: Two integers between -2 and 2 representing
the speed of the motors. A speed > 0 is forward,
while a speed < 0 is backward. The greater the
magnitude of the inputs the faster the motors will
turn.
Outputs: none
Notes: A 100 ms delay is implemented when changing
motor speeds to protect the motors from burnout.
-left/right as seen from the back of robot
**********************************************/
void drive(short int speedl, short int speedr){

        while(speed_val_l != speedl || speed_val_r != speedr){
                if(speed_val_l > speedl){
                        OCR1AL = OCR1AL +1;
                        speed_val_l--;
                }
                if(speed_val_l < speedl){
                        OCR1AL = OCR1AL -1;
                        speed_val_l++;
                }
                if(speed_val_r > speedr){
                        OCR1BL = OCR1BL +1;
                        speed_val_r--;
                }
                if(speed_val_r < speedr){
                        OCR1BL = OCR1BL -1;
                        speed_val_r++;
                }
                delay_ms(25); //delay on each increment to protect motors
        }
}   //end drive()

/****************************************
Function: check_ir()
checks both ir detectors, and outputs a
number (1-9) based on both of their states.
This number is then used by the main function
to determine what direction to take.
adc[0]=right, [2]=left
****************************************/
int check_ir(void){

    if(adc_data[2]>=52 && adc_data[2]<85) {   //left something
        PORTC.4=1;
        if(adc_data[0]>=52 && adc_data[0]<85) { //right something
                PORTC.5=1;
                return 5;}
        else if(adc_data[0]>=85 && adc_data[0]<130) { //right too close
                PORTC.5=0;
                return 6;}
        else{   //right nothing
                PORTC.5=1;
                return 4;}
                }
    else if(adc_data[2]>=85 && adc_data[2]<130) {//left too close
        PORTC.4=0;
        if(adc_data[0]>=52 && adc_data[0]<85) {//right something
                PORTC.5=1;
                return 8;}
        else if(adc_data[0]>=85 && adc_data[0]<130) {//right too close
                PORTC.5=0;
                return 9;}
        else{   //right nothing
                PORTC.5=1;
                return 7;}
        }
    else{   //left nothing
        PORTC.4=1;
        if(adc_data[0]>=52 && adc_data[0]<85) {  //right something
                PORTC.5=1;
                return 2;}
        else if(adc_data[0]>=85 && adc_data[0]<130) { //right too close
                PORTC.5=0;
                return 3;}
        else{   //right nothing
                PORTC.5=1;
                return 1;}
        }

}//end check_ir()

/*****************************************
function:  obstacle_avoidance()
*****************************************/
void obstacle_avoidance(void){
  rand=TCNT1L;
rand1=rand+300;
ir=check_ir();
switch(ir){
        case 1: //left nothing, right nothing
                drive(2,2);  //drive straight
                delay_ms(rand1);
                break;
        case 2: //left nothing, right something
                drive(0,1);  //turn left
                delay_ms(rand1);
                break;
        case 3: //left nothing, right too close
              drive(-1,1); //turn hard left
              delay_ms(rand1);
              break;
        case 4: //left something, right nothing
                drive(1,0);  //turn right
                delay_ms(rand1);
                break;
        case 5: //left something, right something
                drive(1,1);  //forward slowly
                delay_ms(rand1);
                break;
        case 6: //left something, right too close
                drive(-1,-1); //backup and turn left
                delay_ms(rand1);
                drive(0,1);
                delay_ms(rand1);
                break;
        case 7: //left too close, right nothing
                drive(1,-1); //turn hard right
                delay_ms(rand1);
                break;
        case 8: //left too close, right something
                drive(-1,-1); //backup and turn right
                delay_ms(rand1);
                drive(1,0);
                delay_ms(rand1);
                break;
        default: //left and right too close
                drive (-2,-2); //backup fast and turn
                delay_ms(rand1+300);
                drive(random(2),random(2));
                delay_ms(rand1);
                break;
        }

 /*if(bumper){    //bumper switch
        drive(-2,-2); //backup fast and turn
        delay_ms(rand1);
        drive(random(2),random(2));
        delay_ms(rand1);
        bumper=0;
        }  */
 }
//end obstacle_avoidance()




void main(void)
{
// Declare your local variables here

// Input/Output Ports initialization
// Port A initialization
// Func0=In Func1=In Func2=In Func3=In Func4=In Func5=In Func6=In Func7=In
// State0=T State1=T State2=T State3=T State4=T State5=T State6=T State7=T
PORTA=0x00;
DDRA=0x00;

// Port B initialization
// Func0=In Func1=In Func2=In Func3=In Func4=In Func5=In Func6=In Func7=In
// State0=T State1=T State2=T State3=T State4=T State5=T State6=T State7=T
PORTB=0x00;
DDRB=0x00;

// Port C initialization
// Func0=Out Func1=Out Func2=Out Func3=Out Func4=Out Func5=Out Func6=Out
Func7=Out
// State0=1 State1=1 State2=1 State3=1 State4=1 State5=1 State6=1 State7=1
PORTC=0xFF;
DDRC=0xFF;

// Port D initialization
// Func0=In Func1=In Func2=In Func3=In Func4=Out Func5=Out Func6=In Func7=In
// State0=T State1=T State2=T State3=T State4=0 State5=0 State6=T State7=T
PORTD=0x00;
DDRD=0x30;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: Timer 0 Stopped
// Mode: Normal top=FFh
// OC0 output: Disconnected
TCCR0=0x00;
TCNT0=0x00;
OCR0=0x00;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: 23.438 kHz
// Mode: Ph. correct PWM top=00FFh
// OC1A output: Non-Inv.
// OC1B output: Non-Inv.
// Noise Canceler: Off
// Input Capture on Falling Edge
TCCR1A=0xA1;
TCCR1B=0x04;
TCNT1H=0x00;
TCNT1L=0x00;
OCR1AH=0x00;
OCR1AL=0x11;
OCR1BH=0x00;
OCR1BL=0x15;

// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer 2 Stopped
// Mode: Normal top=FFh
// OC2 output: Disconnected
ASSR=0x00;
TCCR2=0x00;
TCNT2=0x00;
```

```
OCR2=0x00;

// External Interrupt(s) initialization
// INT0: On
// INT0 Mode: Falling Edge
// INT1: Off
// INT2: Off
GICR=0x40;
MCUCR=0x02;
MCUCSR=0x00;
GIFR=0x40;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x04;

// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
// Analog Comparator Output: Off
ACSR=0x80;
SFIOR=0x00;

// ADC initialization
// ADC Clock frequency: 187.500 kHz
// ADC Voltage Reference: AREF pin
// Only the 8 most significant bits of
// the AD conversion result are used
ADMUX=FIRST_ADC_INPUT|ADC_VREF_TYPE;
ADCSR=0xCD;

// Global enable interrupts
#asm("sei")

disable_pwm();
//delay_ms(1500);
enable_pwm();
drive(-1, 1); //begin moving
while (gohome<GOHOMETIME) //obstacle avoidance until gohome time
      {
obstacle_avoidance();
      };

 //time to go home
 drive(0,0);   //stop
 disable_pwm();
 delay_ms(2000);
```

```
enable_pwm();
while (!home){
        drive(0,0); //stop
        delay_ms(500);
        N=PINB.0;   //assign directions to port B inputs
        E=PINB.1;
        S=PINB.2;
        W=PINB.3;
        PORTC.0=N;  //represent them on LEDs
        PORTC.1=E;
        PORTC.2=S;
        PORTC.3=W;
        north=!N&E&S&W;
        signal=N|S|E|W;
        if(north){//if it acquired north
                drive(2,2);
                delay_ms(500);
                obstacle_avoidance();
                }
        else{
                        if(!N) drive(2,2);//if beacon is north, go to it
                        if(!E) drive(1,-1);//if beacon is east, turn hard right
                        if(!S) drive(-2,2);//if beacon is south, turn hard left
                        if(!W) drive(-1,1);//if beacon is west, turn hard left
                        delay_ms(200);
                        obstacle_avoidance();
                }
 }
}
```