University of Florida
Department of Electrical Engineering

EEL 5666
Intelligent Machines Design Lab

NavBot (Final Report)

by

Kenton Newby

Instructor: Dr. Antonio A. Arroyo

Table of Contents

Abstract

NavBot is a mobile, autonomous agent controlled by the Motorola 68HC11 microcontroller. Its primary function is navigation from place to place by triangulating off of three beacons. To accomplish this, the robot is equipped with multiple IR emitters to communicate with the beacons and several sonar transducers to receive sonar pulses from the beacons. NavBot operates by sending a certain number of IR pulses to request a beacon to pulse its sonar. The beacons each only respond to a specific number of IR pulses. Once the robot sends its IR pulses, it starts a timer. Once the beacon receives the IR pulses, it sends a sonar signal. The time it takes for the sonar to get back to the robot is how the distance is calculated.

Executive Summary

NavBot consists of a Motorola MC68HC11 EVBU board and the ME11 expansion boards. Along with these boards, a special board was constructed which contains the components for the IR cannon circuit. NavBot's purpose is to navigate from place to place within a room by triangulating off of three beacons. Two sensors let NavBot respond to obstacles-IR and bump. The IR sensors allow the robot to detect objects at a distance and allow it to avoid bumping into an object altogether. The bump switches were incorporated as a back up system to the IR in case the IR happens to fail for some reason. The beacons consist of a Mekatronix MSCC11 single-chip board, an IR receiver, and a sonar transmitter circuit. The IR receiver allows the beacons to recognize when the robot wants their sonar pulse to be sent.

Once the robot is ready to receive a pulse of sonar from a particular beacon, it turns on its 32kHz IR cannon and pulses them a specific number of times. The number of times the IR cannon is pulsed depends on the beacon that is being requested. The IR cannon's emitters are arranged in such a manner such that the IR signal is transmitted in a complete circle. Once the robot sends the IR pulses, it starts its timer.

The beacons quietly (sonar off) wait for an IR pulse to be received. Once the first IR pulse is received, the beacon begins to count all subsequent pulses for a given length of time. Once that time has elapsed, the beacon compares the number of pulses counted to the its beacon number. If the two are not equal, it resets its counter and continues to wait for another IR pulse to restart the entire process. If that comparison does show the two values to be the same, then the beacon turns on its sonar. It then turns the sonar off, resets the count, and awaits another IR pulse to start the process again.

Introduction

NavBot is an experiment in one of the most challenging aspects of robotics-robot navigation. It explores the feasibility of using beacons with both sonar and IR to let a robot calculate its position within a room. A system such as this has many potential uses. For instance, in an industrial environment, robots could be used to load and unload supplies. The robot could be preprogrammed with the positions of its destinations and could avoid any objects that it would encounter along the way. This is but one example where a system like this would be useful. However, almost any situation where a robot would be used for extended periods of time and where it would remain in the same general settings could benefit from this type of system.

The main objectives of the project were to build a robot that could function in its environment by using the basic collision avoidance and bump detection algorithms. Then I wished to expand its functionality by giving it the ability to move from a specific point to any other point within a room. Three beacons mounted in the corners of the room would define the room. This would allow for a user to preprogram the path that the robot was to take. Using its collision avoidance and bump detection, combined with its knowledge of where it is at any given instant, the robot would be able to make it safely from point “A” to point “B”.

Integrated System

NavBot is made using the Talrik Platform as its main body. This platform supports both the MC68HC11 and the ME11 expansion boards. Eight Nickel-Cadmium batteries power the robot and two hacked servos provide the robot motion. A wide array of sensors were planned for NavBot including the following:

Sensor	**Primary Function**	**#**
40kHz IR Emitters	Collision Avoidance	4
40kHz Hacked IR Detectors	Collision Avoidance	3
Micro Switches	Bump Detection	7
Sonar Transducers	Sonar Reception	6
32kHz IR Cannon	Beacon Communication	8
	Total Number of Sensors	**28**

The beacon also consists of several parts. The main part of the beacon, and that which provides all the control, is the MSCC11 single-chip board. Connected to this board are a sonar transmitter circuit and an IR receiver.

Mobile Platform

NavBot's body is constructed from the Talrik pattern and provides plenty of support for all the parts required for the robot. This platform also has plenty of room for expansion of the basic design in case extra sensors or other items need to be added later on. The robot still remains light enough to not wear out the motors too quickly as well, which is a nice feature that wasn't exactly planned but proved beneficial. The circular shape also makes it easy for the robot to negotiate its way into tight spots and not get stuck.

Actuation

Two hacked servos provide the propulsion for the robot. These are the only actuation mechanisms. A stop on the servos had to be removed to allow it to spin in a complete circle, thereby providing continuous rotation. This basically created a gearhead DC motor. A non-powered wheel provides support in the rear of the robot. The ME11 board uses an H-bridge to control the motors. This dual motor system yields a robot that can turn about its own axis, which is a great feature if you need that maneuverability. It also decreases the complexity of the design and isn't too expensive to implement.

A motor control algorithm was used to smooth the operation of the motors and prevent jerking as the direction changed. The algorithms two main functions are an increment and a decrement function. These functions increment and decrement the speed of the motor by a small amount with the final result of making a smoother operating curve than the almost square curve found without such changes. These two functions are shown below:

Increment: $new\_speed = 1 / (1-k)^{n+1} * old\_speed$

Decrement: $new\_speed = (1-k)^{n+1} * old\_speed$

Caution has to be taken when using the decrement function however because this equation never gets to zero. Therefore, if you want the robot to stop, you have to wait until you get "close enough to zero", with that term being defined by the programmer.

Sensors

NavBot's primary sensors are the three IR sensors mounted on the front of the robot. The sensors were altered from their original design so that they would provide an analog signal to the MC68HC11 instead of the original analog signal. The sensors are facing outward from the robot with one in the center and the other two about 45 degrees left and right of the centerline.

All three sensors are fed into an individual analog port on the MC68HC11. These values can be directly interpreted by the software and provide most of the collision avoidance data for the robot.

NavBot also has bump switches mounted along its outside edge. There are five mounted in the front of the robot and the others are mounted in the rear. The switches were mounted using hot glue. The switches are connected to a resistor network, which is shown below:

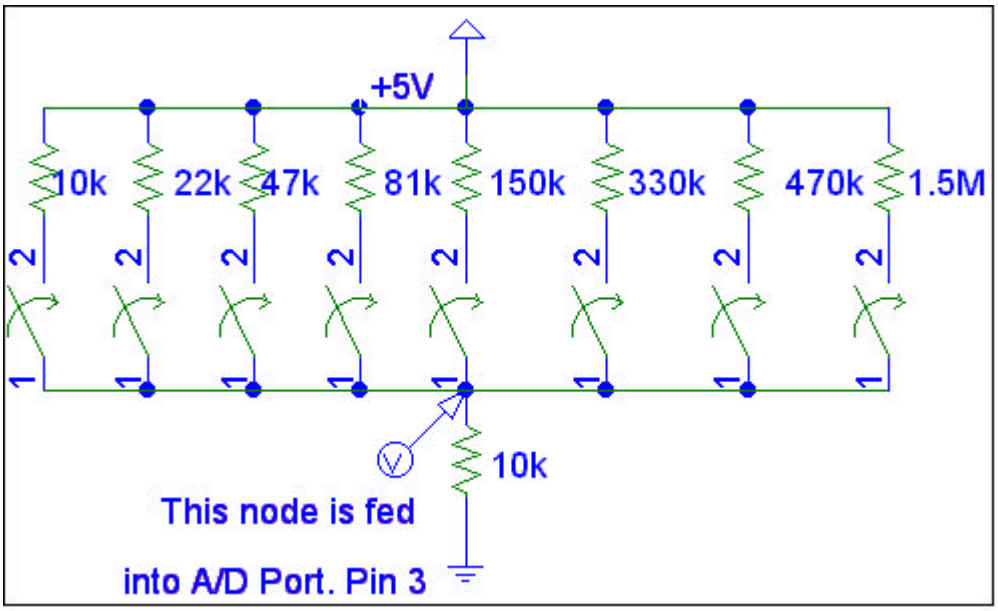

Figure 1. Bump Switch Resistor Network

Each switch, or combination of switches, produces a different value on the A/D port. This, in turn, lets the robot know which switch was pressed so it can respond accordingly. The robot is able to use this design to determine if has accidentally run into some object that the IR detectors did not pick up. This could happen, for instance, if the object is dark in color or if the IR receiver gets knocked out of position somehow.

The main function of NavBot is to perform the navigation function mentioned earlier. To perform that, the robot uses a sonar receiver circuit borrowed from Michael Apodaca's final IMDL report. The circuit shown below was the one found in that report and used on NavBot.

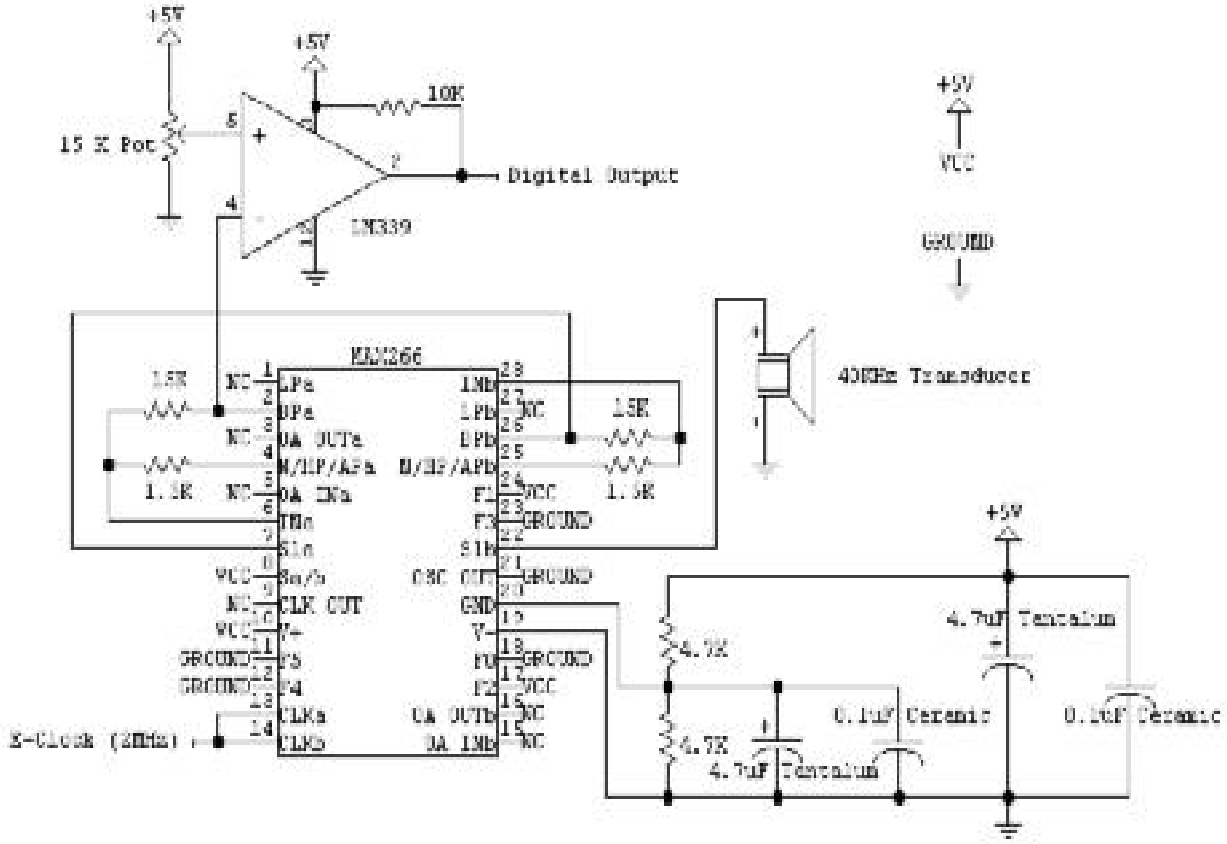

Figure 2. Sonar Receiver Circuit

The circuit consists of a Maxim MAX266 IC, which acts as a resistor controlled filter for the signal, which is input by the sonar transducer. The output of the MAX chip is run through a comparator whose output is a digital output roughly between 0 and 5 volts. This circuit produced extremely good results under somewhat surprising circumstances. For instance, during a test in the lab, when the receiver was mounted at one end of the table and the transmitter was at the other (about 15-20 ft away), a student did not realize I was testing and proceeded to use the phone. The phone was placed directly in front of my sonar transmitter and when I started to ask him to move it, I noticed that the signal on the oscilloscope had not changed. Apparently the sonar receiver would receive even if something were directly in front of the transmitter. The only time this was not the case was when the transmitter was completely covered, say by your hand, so that there was no gap inbetween it and the object covering it. This makes sense since the sonar transmitter is using sound and sound can bend around objects. This is one of the main reasons for using sonar for beacon navigation.

Six of these sonar transducers were mounted around the robot so that it could *listen* in a complete circle. It turns out that the sonar transducers are rather directional, with a useable arc of only 60 degrees or so.

The other circuit used on the robot is the IR cannon circuit. This is used to provide communication between the robot and the three beacons. The IR cannon was used so that the IR signal could travel farther, otherwise, the size of the room would be limited. The IR cannon requires a 32kHz signal to modulate the LED's. This ensures that their signal does not interfere with the collision avoidance, which operates at 40kHz. However, the signal is actually set at about 28kHz because the 40kHz IR receivers have a wide enough

bandwidth to still pick up 32kHz. The 28kHz center frequency is too low for them to pick up, but is still within the bandwidth of the IR receivers used on the beacon. This is how the two systems operate without interfering with one another. The 32kHz signal was generated using a 555 timing IC and various other components. It was powered by one of the leftover MC68HC11 pins so that the robot could control when it turned on the IR cannon and could therefore control the pulsing of the cannon. Both the 32kHz generator and the IR cannon circuit itself are shown in the following figures:

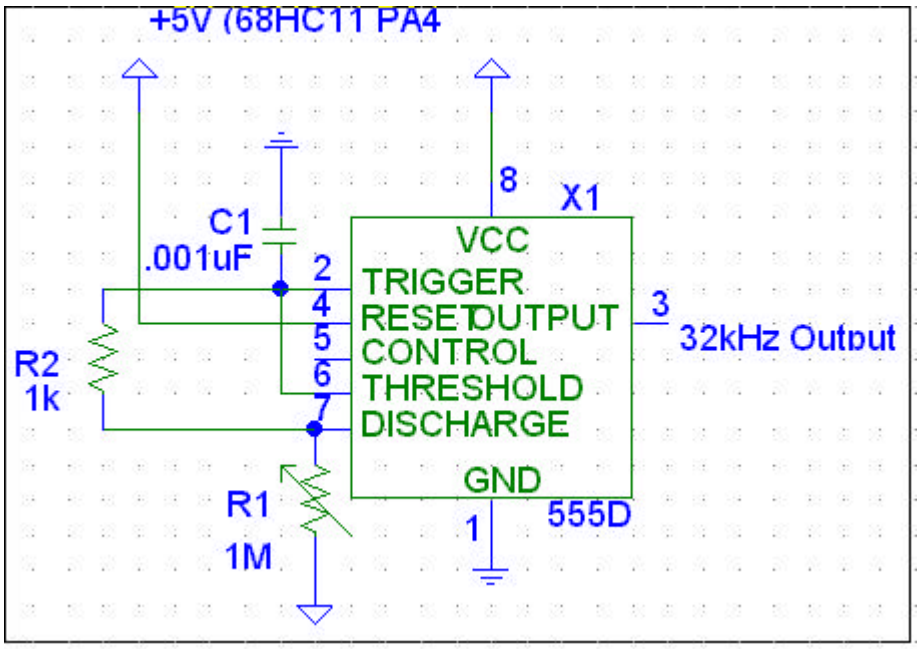

Figure 3. 32kHz Signal-Generating Circuit

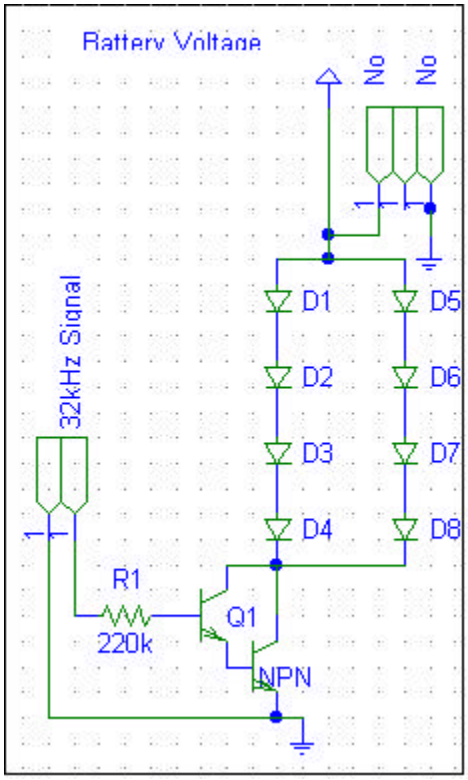

Figure 4. IR Cannon Circuit

Discussion of the system's sensors cannot be considered complete without mentioning the beacons themselves. Even though they only have one sensor (by the strict definition), they play an integral part in the overall operation of the system. The beacon has a 32kHz IR detector that is kept in its original digital form. The beacon also has a sonar transmitter circuit borrowed from the same final report as the sonar receiver. The sonar transmitter is shown on the following page.

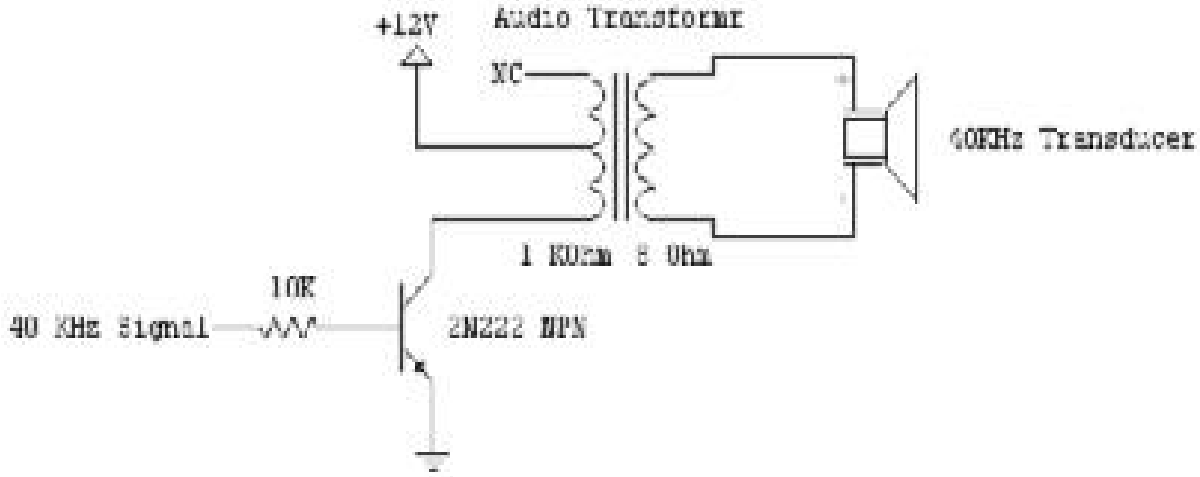

Figure 5. Sonar Transmitter Circuit

It should be noted, however, that the transformer in the above diagram is connected backwards. It may not seem so by looking at the circuit diagram, but as many of us found out in lab this semester, the above wiring will not work correctly. Once switched, however, this circuit worked very well. The 40kHz signal needed for this circuit was produced by using the same method used to get 40kHz IR signals on the TJ mobile agent. However, instead of the IR LED, a sonar transducer was connected to the PORTB pin 2. The sonar transmitter and receiver worked extremely well by themselves. However, getting multiple receivers to work at the same time proved very challenging. I was unable to get the six receivers on the robot to work so I was unable to have the robot listen in the 360 degrees required. In hindsight, it would have probably been easier to simply build six sonar receiver circuits.

Behaviors

The main behavior for NavBot is navigation using information from three beacons and triangulating off of those beacons. This function would work in the following manner: The robot would send a series of IR pulses out in a circle, which would be received by all three beacons. The number of IR pulses sent would be (n+1), where n is the number of the beacon to be requested. The IR would be pulsed on for 1ms, off for 2ms. This means that the longest time the IR would be sent for (assuming beacon 3, so four pulses) would be 12ms. Once the robot sent the IR pulses, it would start a timer. The beacons would start off by looking for an IR pulse. They would have a counter that starts at -1 and increments each time a pulse is received. After the first pulse, the beacon would start a timer and would listen for 12ms. After that time had elapsed, the beacon would compare the number of pulses counted to its own individual beacon number. If the two were equal, it would send a sonar pulse back to the robot. If not, it would reset the counter and start the process all over again. The robot would wait for the sonar pulse to be received and would then stop the timer and calculate the distance based on the time it took. Both reception systems (on the robot and the beacons) would use the input capture subsystem and would look for the rising edges of those signals.

I believe this algorithm is feasible, but I also think there may be a better way to accomplish this same goal. This is simply because this design is a bit limited in the size of the room that can be used and the conditions under which the robot must operate (for instance, it wouldn't work outside as well as it would indoors)

Conclusion

NavBot's main purpose was to navigate using the three beacons placed in the corners of a room. However, the fact that I could not get the beacon and the robot to work together very well wound up being a major stumbling block for my project. For this reason, my robot as of right now cannot perform the navigation function. The motor control portion of the final design works very well and was one of my main goals for the robot. Afterall, smoother motion would yield better operation overall and longer battery life. However, this hardly compensates for the fact that I could not get the robot and beacons to work together. I think if anyone decides to do this type of project, they should have a crystal clear design in mind. One of my problems was that I kept changing my mind on how exactly I wanted to do it. With all the tools at an engineer's disposal, it is up to *them* to decide which ones are best for a given job. I suggest that this decision be made before the semester starts! This is one thing I would do differently if I had it to do all over again. I would also like to incorporate some sort of interactivity into the design as well. For instance, maybe a remote control system where each key pressed on the remote makes the robot go to a different place in the room. Imagine the possibilities! I might also change the way in which the robot triangulates off of the beacons, if I found a better way than IR/sonar that would work in various locations fairly well.

Appendix 1: Program Code for NavBot

```
int too_close=115;
int six_inches=110;
int twelve_inches=100;

int diff=10;
int irmin=90;

int left;
int right;
int center;
int bump;

int r_motor=0;
int l_motor=1;

float init_speed;
float final_speed;
float n=0.0;
float old_speed;
float new_speed;
float x;
float k=0.05;
float stopped=1.0;
float speed;

float incr_speed(float old){
  if (old == 0.0){
    old=1.0;
  }
  x=(1.0/(1.0-k)^(n+1.0));
  new_speed=x * old;
  n++;
  old_speed=new_speed;
  return new_speed;
}
```

```
void go_fwd (float old_speed, float final_speed)
{
  speed=incr_speed(old_speed);
  motor(r_motor,speed);
  motor(l_motor,speed);

  while(speed < final_speed){

    speed=incr_speed(old_speed);
      if (speed > final_speed){
        speed=final_speed;
      }
    motor(r_motor,speed);
    motor(l_motor,speed);
  }
  n=0.0;
}

float decr_speed(float old){
  x=((1.0-k)^(n+1.0));
  new_speed=x * old;
  n++;
  if (new_speed < 1.0)
    new_speed=0.0;
  old_speed=new_speed;
  return new_speed;
}
```

```
void go_back(float final_speed, int delay_time){
  if (old_speed == 0.0){
    old_speed = 1.0;
  }

  if (old_speed < 0.0){
    old_speed= -old_speed;
  }

  if (final_speed < 0.0){
    final_speed= -final_speed;
  }

  speed=incr_speed(old_speed);
  motor(r_motor,-speed);
  motor(l_motor,-speed);

  while(speed < final_speed){
    speed=incr_speed(old_speed);

      if (speed > final_speed){
        speed=final_speed;
      }

    motor(r_motor,-speed);
    motor(l_motor,-speed);
  }
  n=0.0;
  wait(delay_time);
}

void wait (int milli_seconds)
{
  long timer_a;

  timer_a = mseconds() + (long) milli_seconds;
  while (timer_a > mseconds()) {
    defer();
  }
}
```

```
void read_sensors()
{
  right=analog(0);
  center=analog(1);
  left=analog(2);
  bump=analog(3);
}

void turn_right(float final_speed, int delay_time)
{
  speed=incr_speed(old_speed);
  motor(r_motor,-speed);
  motor(l_motor,speed);

  while(speed < final_speed){
    speed=incr_speed(old_speed);

      if (speed > final_speed){
        speed=final_speed;
      }

    motor(r_motor,-speed);
    motor(l_motor,speed);
  }
  old_speed=speed;
  new_speed=speed;
  n=0.0;
  wait(delay_time);
}
```

```
void turn_left(float speed, int delay_time)
{
  speed=incr_speed(old_speed);
  motor(r_motor,speed);
  motor(l_motor,-speed);

  while(speed < final_speed){
    speed=incr_speed(old_speed);

      if (speed > final_speed){
        speed=final_speed;
      }

    motor(r_motor,speed);
    motor(l_motor,-speed);
  }
  old_speed=speed;
  new_speed=speed;
  n=0.0;
  wait(delay_time);
}

void halt (int delay_time)
{
  if (old_speed < 0.0){
    old_speed= -old_speed;
    speed=decr_speed(old_speed);
    motor(r_motor,-speed);
    motor(l_motor,-speed);

    while(speed != 0.0){

      speed=decr_speed(old_speed);
      motor(r_motor,-speed);
      motor(l_motor,-speed);
    }
  }/*end if speed < 0.0*/

  if (old_speed > 0.0){
    speed=decr_speed(old_speed);
    motor(r_motor,speed);
    motor(l_motor,speed);
```

```
    while(speed != 0.0){

      speed=decr_speed(old_speed);
      motor(r_motor,speed);
      motor(l_motor,speed);
    }/*end while*/
  }/*end if speed > 0.0*/

n=0.0;
wait(delay_time);
}

void avoid()
{
  while(1){
    poke(0x7000,0xff);
    wait(1);
    read_sensors();

  if (bump!=0){
/*    if ((bump==7) || (bump==8) || (bump==1) || (bump==2)){
      halt(1);
      go_fwd(old_speed,100.0);
    }*/

    if (bump>=15){
      halt(1);
      go_back(-25.0,500);
      halt(1);
      turn_right(25.0,1000);
      go_fwd(old_speed,50.0);
    }
  }/*end if bump!=0*/

  if (bump==0){
    if (center < twelve_inches){

      if ( (left > irmin) && (left > right+diff) ){
        turn_right(100.0,0);
        go_fwd(old_speed, 50.0);
      }
      else
```

```
  if ( (right > irmin) && (right > left+diff) ){
    turn_left(100.0,0);
    go_fwd(old_speed, 50.0);
  }
  else

  if ( (left < right+diff) && (left > right) && (right < left+diff) &&
   (right > left) ) {
      go_fwd(old_speed, 100.0);
  }
} /* end if center greater than 12in */

if ( (center >= twelve_inches) && (center < six_inches) ){

  if ( (left > irmin) && (left > right+diff) ){
    turn_right(50.0,0);
    go_fwd(old_speed,25.0);
  }
  else

  if ( (right > irmin) && (right > left+diff) ){
    turn_left(50.0,0);
    go_fwd(old_speed,25.0);
  }
  else

  if ( (left < right+diff) && (left > right) && (right < left+diff) &&
   (right > left) ) {
      go_fwd(old_speed,50.0);
  }
} /* end if center btwn 6in and 12in */

if (center >= six_inches && center < too_close){

  if ( (left > irmin) && (left > right+diff) ){
    turn_right(25.0,0);
    go_fwd(old_speed,15.0);
  }
  else

  if ( (right > irmin) && (right > left+diff) ){
    turn_left(25.0,0);
    go_fwd(old_speed,15.0);
  }
```

```
      else

      if ( (left < right+diff) && (left > right) && (right < left+diff) &&
       (right > left) ) {
          go_fwd(old_speed,25.0);
      }
    } /* end if center between too_close and 6in */


    if (center >= too_close){
      halt(50);
      go_back(-25.0,500);
      halt(50);
      turn_right(15.0,1000);
      go_fwd(old_speed,50.0);
    }/* end if center is too_close */

  }/*end if bump==0*/

  }/*end while(1)*/

}/*end avoid()*/

void main()
{
  go_fwd(stopped,100.0);
  wait(250);
  start_process(avoid());
}
```

Appendix 2: Program Code for Beacons
(uses slightly altered TJ code)

```
#include <hc11.h>
#include <mil.h>
#include <analog.h>
#include <irtj.h>
#include <int32.h>
#include <vectors.h>


#define PERIOD 40000   /*Period = 20ms*/

#pragma interrupt_handler irread2;
void irread2();

int beacon_num=3;              /*BEACON NUMBER*/
int counter=0;
int pulse_count=-1;
int analog_value2[8];
int i;
long t;
int BIT2=0x04;

extern INT32 time2;

void
msleep(t)                  /*Created 7/21/94 Chuck McManis */
int       t;               /* number of milleseconds to sleep */
{
          unsigned int    i;

          asm(" sei");
          i = LO_WORD(time2) + t;
          asm(" cli");
          while (i > LO_WORD(time2)) ;
          return;
}
```

```
/* void init_ir2()
        Will initialize the interrupt-driven ir readings for TJ.  OC3 will be used
        for timing control
*/

void init_ir2()
{
        int i;

        INTR_OFF();

        for (i = 0; i < 9; i++)
           analog_value2[i] = 0;

        TOC3 = 10000;
        CLEAR_BIT(TCTL1, 0x30);  /* Interrupt will not affect OC3 pin */
        SET_BIT(TMSK1, 0x20);    /* Turn on OC3 interrupt */
        INTR_ON();

}

/* sonar_pulse() modulates PORTB, bit 0 sonar transducer at 40kHz */
void sonar_pulse()
{
        asm("ldaa    #1\n"
                        "ldy    #255\n"         /* 2*(# of IR pulses) */
                        "staa    4100\n");      /* 4100 = Port B */
        asm("loop2 : ldaa 4100\n"               /* 4 cycles - necessary */
                        "eora #1\n"             /* 2 cycles - necessary */
                        "staa 4100\n"           /* 4 cycles - necessary */
                        "nop\n"
                        "nop\n"
                        "nop\n"                 /* 8 cycles */
                        "next2 : nop");
        asm("dey\n"                             /* 4 cycles - necessary */
                        "bne loop2");           /* 3 cycles - necessary */
                                                /* total = 25 cycles = 40 kHz*/
        asm("ldaa 4100");                       /* Clear Port B - Turn LEDs off */
        asm("anda #254");
        asm("staa 4100");
}

/*Calls the modulator and then reads the analog ports */
/* Care should be taken since it also kills interrupts
       for that period of time2*/
```

```
/* irread2() is the interrupt handler */


void irread2()
{   pulse_count=-1;
          TCTL2=0x10;                          /*Set IC1 to capture rising edges only*/
          if (TFLG1 & BIT2){
            t=TIC1;
            CLEAR_FLAG(TFLG1, 0x40);      /*Clears IC1 flag*/
            pulse_count++;
          }/*end if*/

            while (TCNT < t+0x5DC0){           /*Look for pulses for 24000 cycles(12ms)*/
              if (TFLG1 & BIT2){
                CLEAR_FLAG(TFLG1, 0x40);   /*Clears IC1 flag*/
                pulse_count++;
              }/*end if*/
            }/*end while
          }/*end if*/


}
void main()
{
  irread2();
  if (pulse_count==beacon_num)
             sonar_pulse();
}
```