Final Report

PEST

by
Marcin Owczarz

4/21/99

Intelligent Machine Design Laboratory
EEL5666

Table of Contents:

1. Abstract

This paper covers an autonomous robot, named Pest, which is designed to follow a walking person. The person has attached a 40kHz infrared beacon. The robot is using an array of infrared sensor to determine location and distance to the beacon. It uses a set of sonar transmitter/receiver pairs to perform collision avoidance. It also has a set of bump switches to detect collisions. The implemented behaviors are: calibration, beacon location and following, collision avoidance, and collision resolution. Currently, the robot has a limited range of the beacon detection. There are also problems with overheating sonar transmitter circuit.

2. Executive Summary

The main objective for the project was to develop a robot that is able to follow autonomously a walking person.

To achieve this objective, 5 hacked infrared sensors spaced 20 degrees in front of the robot were used. The array allowed the robot to sense how off robot's center the beacon's location was. Analog hack allowed to relate intensity of the beacon's infrared signal to a distance to the beacon. Through this, robot is not only able to locate position of the infrared source, but also how far behind it is.

The autonomy of the robot is sustained by 40kHz sonar. Transmitter is sending a 40kHz sound wave that is reflected by solid objects. The sonar receiver acquires reflected waves. A receiving circuit is set to relay the received wave to the processor only when certain strength is reached. In such a way the robot is able to sense obstacles. There are 3 transmitter/receiver pairs on Pest. A collision avoidance procedure uses signal received by sonar to avoid running into things while following the beacon. If an obstacle is not detected or is not detected in time to avoid collision, bump switches engage a collision resolution procedure.

3. Introduction

The idea for a person following robot originated with an example of a mule robot in the syllabus for this class [1]. As it is presented there, the purpose for the robot can be from carrying books and supplies to serving as a pack animal. Although, the behavior also can be used as a quick and dirty way to teach a robot. It can be a “Follow me and I will show you where you need to go” equivalent of human behavior. Since I am a mechanical engineer, my purpose for taking the course was to learn more about sensor implementation and behavior programming. Hence, because of that and the limited time for the project, I tried to keep the mechanical part of the project to a bare minimum.

This paper describes the robot’s construction. It introduces a platform and a sensor design. It explains how sensors were constructed and modified. It describes mechanical components as well as the programming used to achieve the goal of following a person. Also, this paper suggests what further improvements might be feasible to refine robot’s performance.

4. Integrated System

The whole system consists of a robot unit and an infrared emitting beacon. The beacon is attached to a moving person who is to be followed and it allows the robot to locate the person and determine how far he or she is. The robot unit is propelled by a pair of high rpm motors driving two wheels. A coasting wheel supplies a third point of stability along with a capability to turn around without a motion forward. The robot has a set of infrared (IR) sensors handling the beacon location and evaluating a distance to the beacon. It also has a set of sonar transducer/receiver pairs handling collision avoidance tasks. All processing is handled by an M68HC11EVBU board (EVBU board) made by Motorola. The motors are handled by an EVBU I/O and memory expansion board - ME11 by Mekatronics. The sonar sensing circuit is based on Maxim's MAX266 filter and sending circuit relies on TIP120 transistor. Both circuits are connected to the EVBU. The beacon's IR circuit is designed around 555 CMOS chip. The robot is also equipped with a piezzo speaker. The speaker is mostly used to generate beeps informing of an error or a change in behavior.

Pest incorporates several basic behaviors indicated on the flow chart in Figure 1. Its basic behavior is looking for a beacon and trying to center it in the middle of the IR array. One of the primitive behaviors is calibration, during which the robot spins around sensing an infrared noise in the environment. Another primitive behavior is a collision sensed by bump switches. When collision is sensed, Pest is going to back up and then return to its basic behavior. Collision avoidance stems from looking for the beacon behavior.

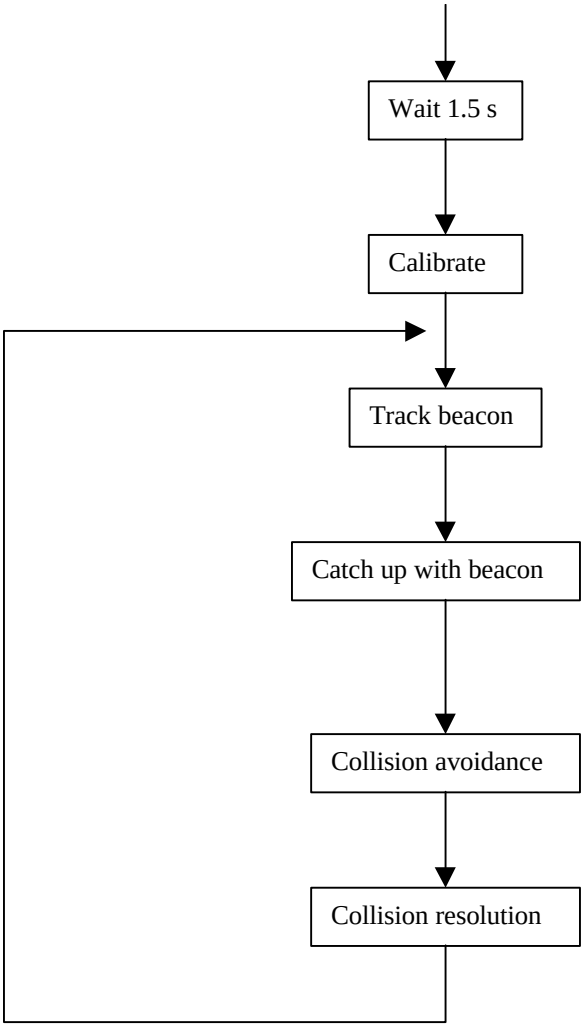


Figure 1. General flowchart of behavior

When sonar senses obstacle on the robot's way, the program jumps into procedure that modifies motor driving values trying to avoid contact with the obstacle. A procedure controlling distance to the beacon works similarly. When sensed maximum IR value goes below a certain threshold, but is still higher than registered during calibration noise, the procedure modifies motor driving values, so the robot not only centers on the beacon, but also approaches it.

Thus, with such an array of sensors and behaviors, Pest is able to follow a person with the beacon while trying to avoid any possible obstacles or resolving collisions.

5. Mobile Platform

Since the electronic part of the robot was more of an interest in this project, the platform is kept simple. Its main purpose is to support the processing boards, sensors and motors. It consists of a .25 inch thick plywood sheet. As the robot should be able to rotate in place, the supporting platform is of a round shape to eliminate any corners that might snag on objects. The size of the motors dictates a 16 inch diameter of the platform. There are two cutouts in the platform accommodating 2.75 inch diameter wheels and allowing motors to be mounted right to the platform, keeping the center of gravity low. The motors are mounted using two 1.5 inch maximum diameter hose clamps and 2 self-drilling screws each.

The IR sensors are attached to the platform by hot glue. Square brackets are used to mount sonar transducers/receivers pairs under the platform. The EVBU and ME11 boards are mounted on top of the platform through a set of screws and risers. 5 bump sensors are installed on the perimeter of the platform with the hot glue to detect collisions that will not be avoided using other sensors. A bumper made out of vinyl coated wire is suspended from two square brackets. The wire covers 160 degree of the platform's perimeter and hangs right in front of each bump sensor.

6. Actuation

The only actuators of the robot are two Maxon 137579 motors with 110364 4.8:1 ratio gearheads propelling and turning it. They have 2.75 inch diameter wheels mounted right on the shafts. The motors are capable to output enough rotations per minute (rpm) for the robot to keep up with a fast walking person (about 3 ft/s). For 2.75 in. diameter wheels, the motor has to run at about 400 rpm to achieve this. The motors used are capable of outputting 550 rpm at 12V, while still keeping a continuos torque of 17 oz-in and requiring less then 1A current each.

Turning of the robot is controlled through the speed difference between the wheels. For example, if the left motor is running faster, the robot turns right, and vice versa. Thus, if the left motor is running at the same speed as the right one but in a different direction, the robot is able to turn without advancing. This capability allows installing all the sensors in front of the platform, since the robot is capable of turning around blindly without bumping into objects. The motors are controlled by SN754410NE dual-control DC motor chip integrated into ME11 circuitry.

Actuation is performed by a motor routines incorporated into IC's RugWarrior p-code. A basic motor controlling command looks like:

```
motor( 0, 20.0);
```

Where the first number indicates for which motor the command is issued (in the case of Pest 0 or 1), and the second one is a percentage value of the full capacity of the driver (here 20%). Apart from primitive behaviors, the main motor driving instructions look like:

```
motor(0, 15.0 + motorMod + motorAvoidMod);
```

motor(1, 15.0 - motorMod + motorAvoidMod);

The second value in the command is dependent on how off center the beacon is sensed and is controlled by the main routine. In the case above, the beacon has been sensed all the way to the left, thus the motors were given values to turn in place sharply to the right.

MotorMod is a modifying value that controls how fast Pest advances toward the beacon and is also controlled by the main routine. Since one motor is running in reverse, the modifier for one motor is added and subtracted for the other. If the IR values go over a threshold the motorMod is zeroed, thus not influencing centering behavior, but stopping advancing. The values used for motorMod in the program are 20.0 and 0.0 .

MotorAvoidMod is a modifying value that controls how much sway the robot has into one direction or the other. It is set by the collision avoidance procedure in attempt to "sway" Pest away from detected obstacles. For both motors, the value is negative for putting more importance on turning right and positive for turning left. In the program a values of 10.0, 0.0 or –10.0 are used to control swaying.

This kind of approach allowed me to successfully control several behaviors of the robot in a few lines of code. The only problem seems to be quite a bit of an overshoot when the beacon is sensed extremely to the right or extremely to the left. I checked several different values trying to optimize the system with some success. Since it is a closed loop system there is a way of modeling it through discreet control equations. Such an optimization would be very desirable in this case, but because of the other difficulties I did not have time to perform the analysis.

7. Sensors

7.1 IR Sensors

5 IR receivers were chosen for the purpose of locating and following the beacon. Each sensor covers an angle of 20 degrees. The 100 degree coverage with 5 inputs gives enough resolution to adjust robot turns properly preventing loosing the beacon out of sight.

3 IR LEDs are connected to a basic astable circuit presented in Figure 2 run by

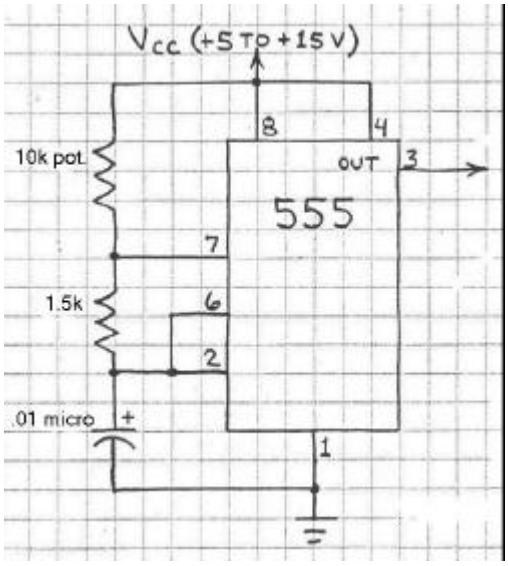


Figure 2 Basic astable circuit

555 CMOS [2]. 10 kOhm potentiometer, 1.5 kOhm resistor, and .01 microFarad capacitor are used in the circuit. The potentiometer is adjusted to achieve 40 kHz signal. The LED are connected in series. The purpose of the beacon is to spread 40 kHz IR signal to make beacon visible to the system from a variety of angles. The beacon is run on a separate pack of batteries.

Hacked SHARP GP1U58Y IR detectors are connected to PE0-PE4 analog ports of a 68HC11 EVBU board. The receivers were modified according to the instructions found on Mekatronics web page [3].

7.2 Sonar Sensors

Sonar array is used for collision avoidance. Sonar array consists of 3 emitters and 3 receivers. Each transmitter is mounted under the platform 1.5 inch apart from its receiver. The pairs are mounted at -45, 0, and 45 degrees to the main axis of the platform. They all use the same 40 kHz transducer obtained from Electronic Goldmine. Figures 3 and 4 show circuits used which were obtained from James O'Connor's report on

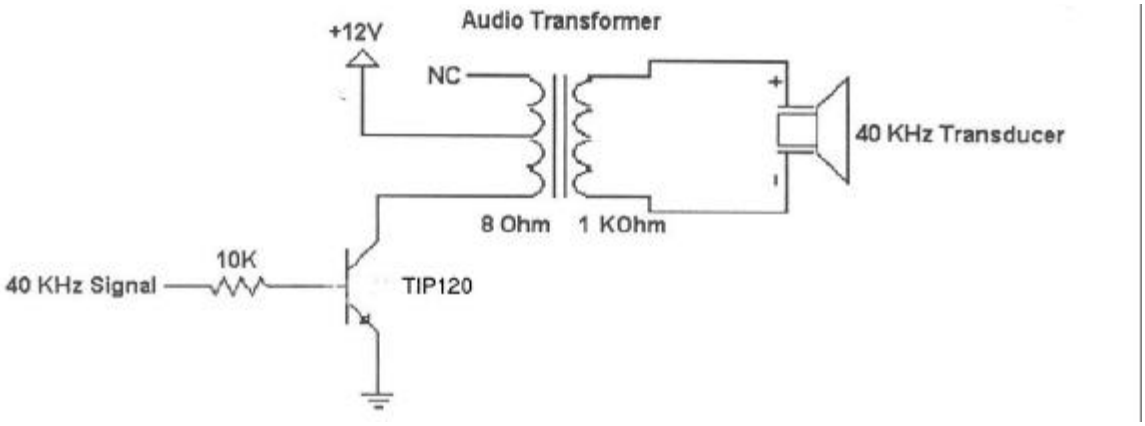


Figure 3. Sonar transmitter circuit.

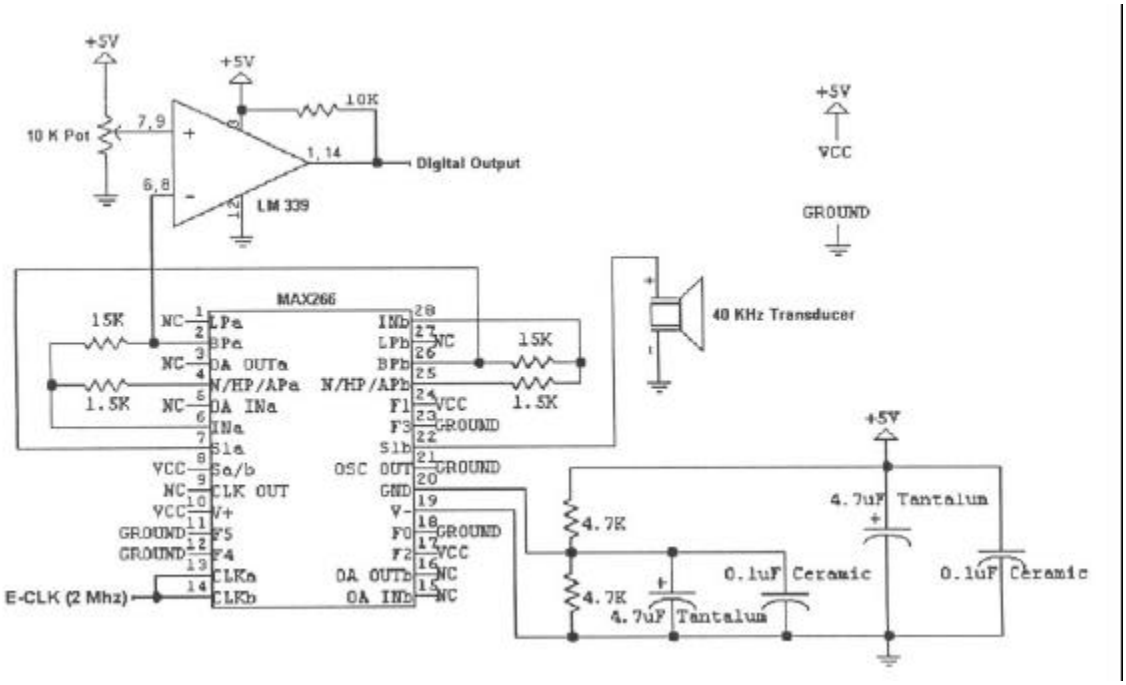


Figure 4. Sonar receiver circuit.

Ranos [4]. The transmitting transducers run of 40kHz generated by ME11. Because a transmitting transistor was overheating, a TIP120 transistor replaced used by James 2N222. It still overheats, but it is able to withstand more punishment that the other one.

Maxim MAX266 filter was used along with LM339 comparator for a receiver circuit. It was connected to PE6 analog port of EVBU board. Only one receiving circuit was used for all three receivers. They were connected to the circuit through a set of three fast-acting reed relays operated by multiplexer digital output ports 2, 3, and 4. The multiplexer is a modified version of a circuit posted on IMDL web-page [5]. The posted diagram has wrong multiplexer/latch data connections. For the circuit to operate correctly, the order of bits have to be reversed. The correct diagram of the circuit is presented in Figure 5. Thus, before every use of the receiver, the latch has to be set to engage a single relay and then the reading can be taken from PE7, which is the port used by multiplexer.

Using a receiver potentiometer, the sensitivity of the receiver was set. Since the sonar is used for collision avoidance, there is no need to use a time of flight algorithm. The sensitivity of the receiver was set, so the oscillations of the output signal started at a distance of 10 inches from an obstacle. When there was no output, the port readings were 0. When the circuit was sending signal, the values varied from 0 to 249 depending on when in a period sampling was done. The closer to obstacle the sonar was, the more square the output wave become. To recognize correctly the signal from the noise and compensate for not being able to read peeks every time, a sum of 5 readings is used. Essentially, the closer the sonar was to an object, the higher the sum was. A threshold value used in the program was determined through experimentation.

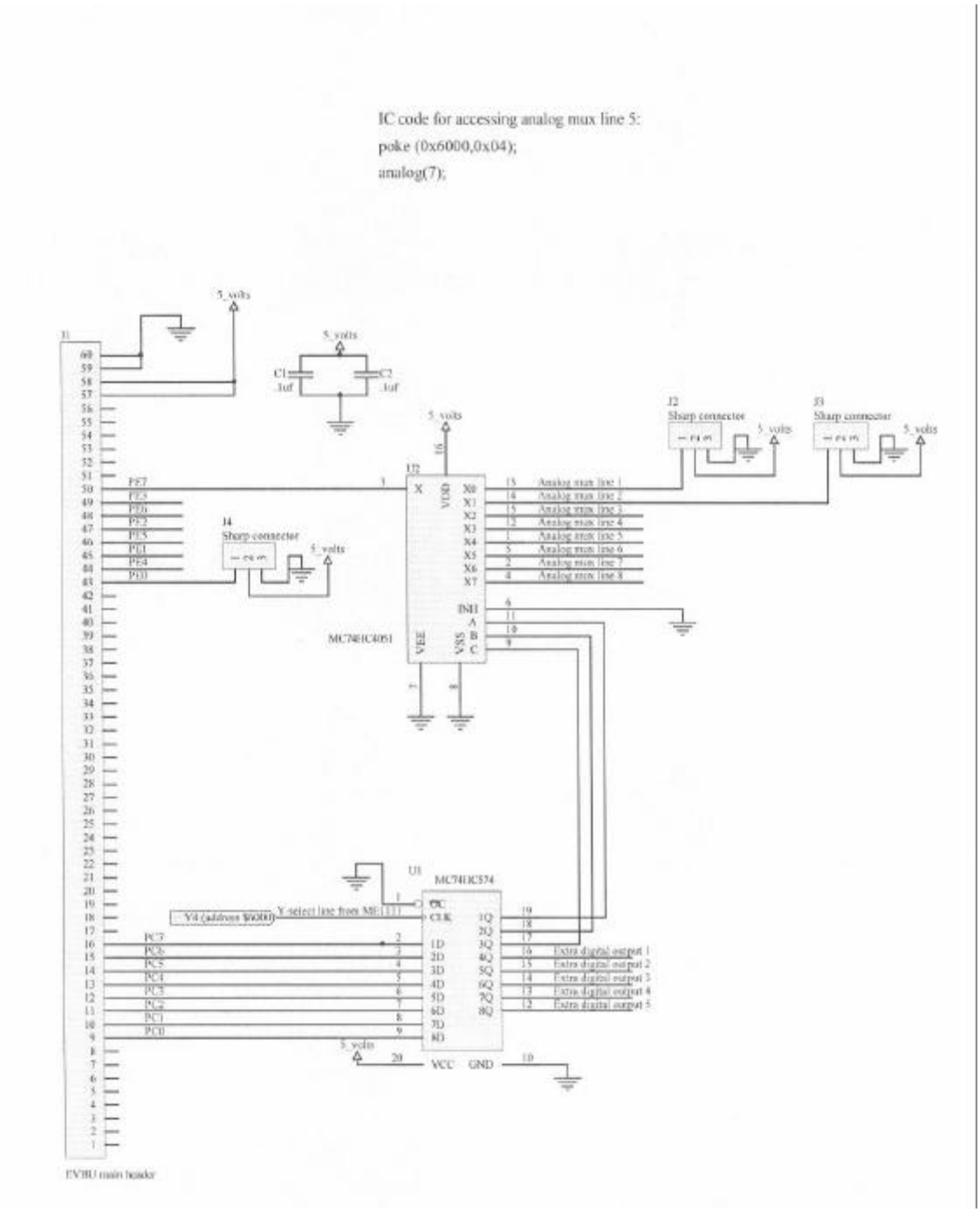


Figure 5. Correct multiplexer circuit.

The sonar circuit was for me the biggest hurdle in this class. There were a lot of things going wrong right from the begining. Since I am not an electrical engineer, a lot of times it took me a couple good days to figure out some, probably, basic things. The

transmitter circuit was giving me a lot of problems. For example, I discovered (after about 6 hrs and 3 transistors) that the 2N222 transistors were coming in two series, and, in spite of looking identical, their pin-out was completely different. Then, I thought that the circuit was simple enough, so I soldered it right a way. Took me a couple of desolderings to realize the value of sockets and breadboard. Another big problem, encountered by everyone in the class using sonar, was overheating of the transistor in the transmitting circuit. Replacing 2N222 with TIP120 does not solve the problem, but it extends the life of the transistor. I tried putting a resistor at the collector lead and also a couple of current direction controlling diodes, but the resulting signal was much weaker and had extra spikes. Because of lack of time, I used the original James' circuit only replacing the transistor with TIP120. I limited a use of the transmitter by switching the 40kHz signal from ME11 on and off. It seemed to work well and the transistor was not overheating too much. The problem was that the circuit, when left unatended for long periods of time, starts melting the audio transformer. I solved this problem by putting a separate Sonar On/Off switch. A much nicer solution would be to put a normally open relay on the transmitter circuit power supply. Only thanks to patience of Scot and Aamir I was able to get the transmitter to work.

7.3 Bump Sensors

A set of 5 bump sensors was installed in parallel on the perimeter of the platform. A 10k resistor was used as a load on the closed circuit. The array of switches was connected to the port 1 of the multiplexer. The readings are 0 for pressed and 255 for depressed. No intermittent values were detected.

The switches are polled once every pass through the main loop. Once the program detects the switches being pressed, it jumps into a collision resolving procedure. Sometimes, because of the poll implementation of sensing, there was some interaction between the collision avoidance routine and the collision resolution routine. Once sonar detected an obstacle and was engaged trying to negotiate it, it happened occasionally that it took a while for the program to detect a collision. The solution for this might be an interrupt implementation of collision sensing. In IC it would have to be programmed using Assembly language, but it can be done also using IC11. Unfortunately, I did not have enough time to implement this.

7.4 Speaker

A piezzo speaker was connected to port PE3 on EVBU board. While it can produce some simple music (using music.c) it was an invaluable tool for debugging a running robot. It is easy to set flags to be posted to the screen while the robot is connected to the serial port, but once running around a series of beeps seems to nicely take over the task.

In the program, the speaker is used to signal with a single beep the end of the calibration and a collision avoidance procedure enable.

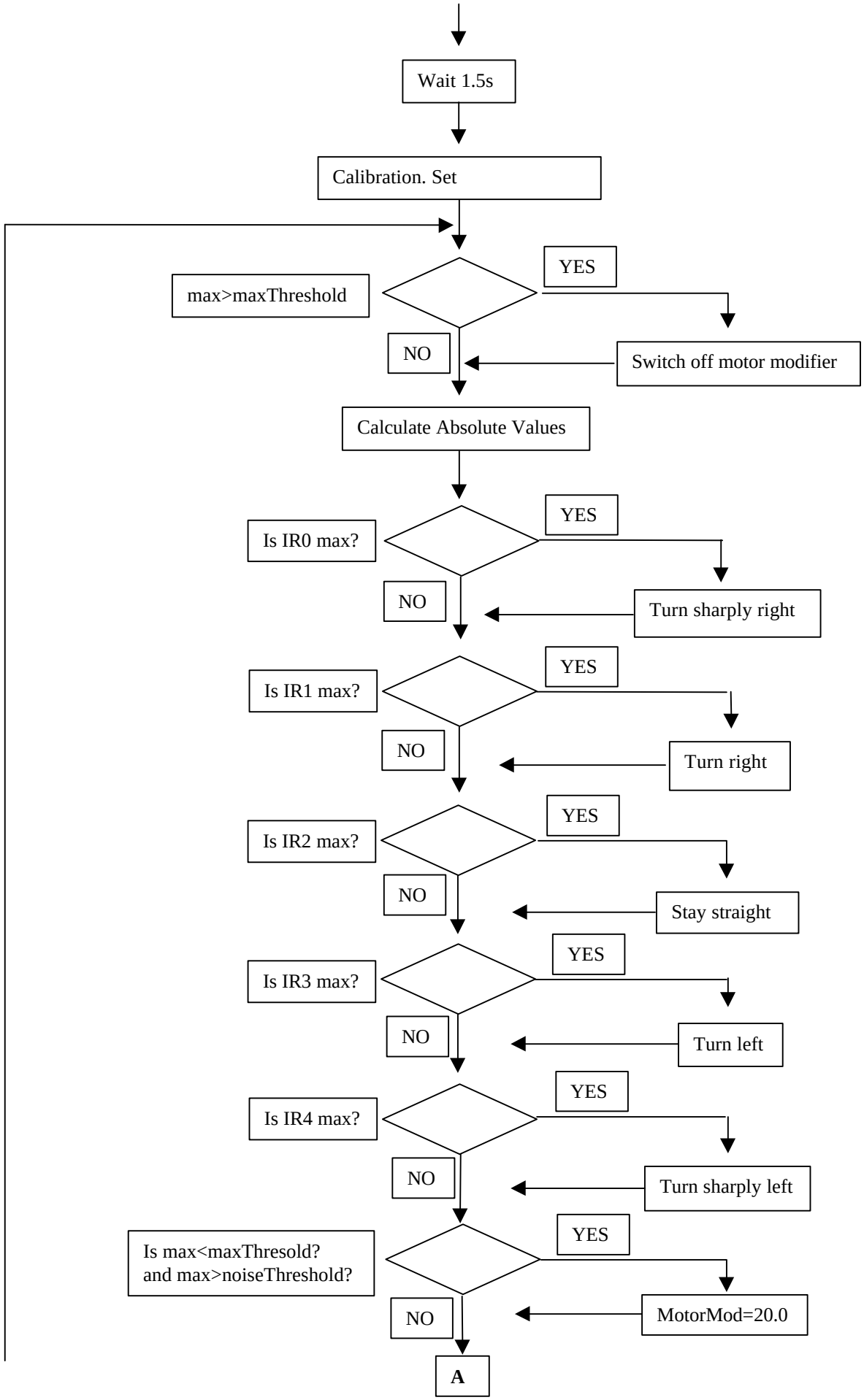
8. Behaviors

Figure 6 shows the detailed flowchart of the whole program running Pest. Pest begins its program by 1.5 s wait, to allow for an easy pickup after a reset button is pressed. Then it jumps to the calibration procedure. To calibrate itself, Pest is rotating steadily while comparing initially scanned values with current values of SHARP sensors. The highest difference at the end of the scan is registered as a noise value - noiseThreshold.

The main behavior of Pest is beacon searching. This is the main procedure which sets the motor values. The procedure scans current values of received through sensors infrared signal. Through a series of if statements, a sensor with maximum value is determined. Depending on the location of the sensor with maximum value, a basic value for the motors is determined. For example, if the maximum value is received by the left most sensor, the basic motor values are set to 15.0 for the both motors (since one motor is running in reverse to other, the robot will turn sharply right). For the left, center, right, and extremely right sensors 10.0, 0.0, -10.0, and -15.0 values are used respectively.

If the absolute maximum value falls in the range between minThreshold and noiseThreshold that means that the beacon has moved away too far and Pest should catch up. The value of motorMod modifier is set to 20.0.

Next, sonar procedure is called. It calculates sums of 5 readings each for left, center and right pair of transducers. If any of the values is bigger than 200 (it relates to an obstacle being 6-8 inches away), the program executes collision avoidance procedures. For value of sLeft and sRight bigger that 200, motorAvoidMod modifier is set to 10.0 and


Wait 1.5s
Calibration. Set
max>maxThreshold
YES
NO
Switch off motor modifier
Calculate Absolute Values
Is IR0 max?
YES
NO
Turn sharply right
Is IR1 max?
YES
NO
Turn right
Is IR2 max?
YES
NO
Stay straight
Is IR3 max?
YES
NO
Turn left
Is IR4 max?
YES
NO
Turn sharply left
Is max<maxThresold?
and max>noiseThreshold?
YES
NO
MotorMod=20.0
A

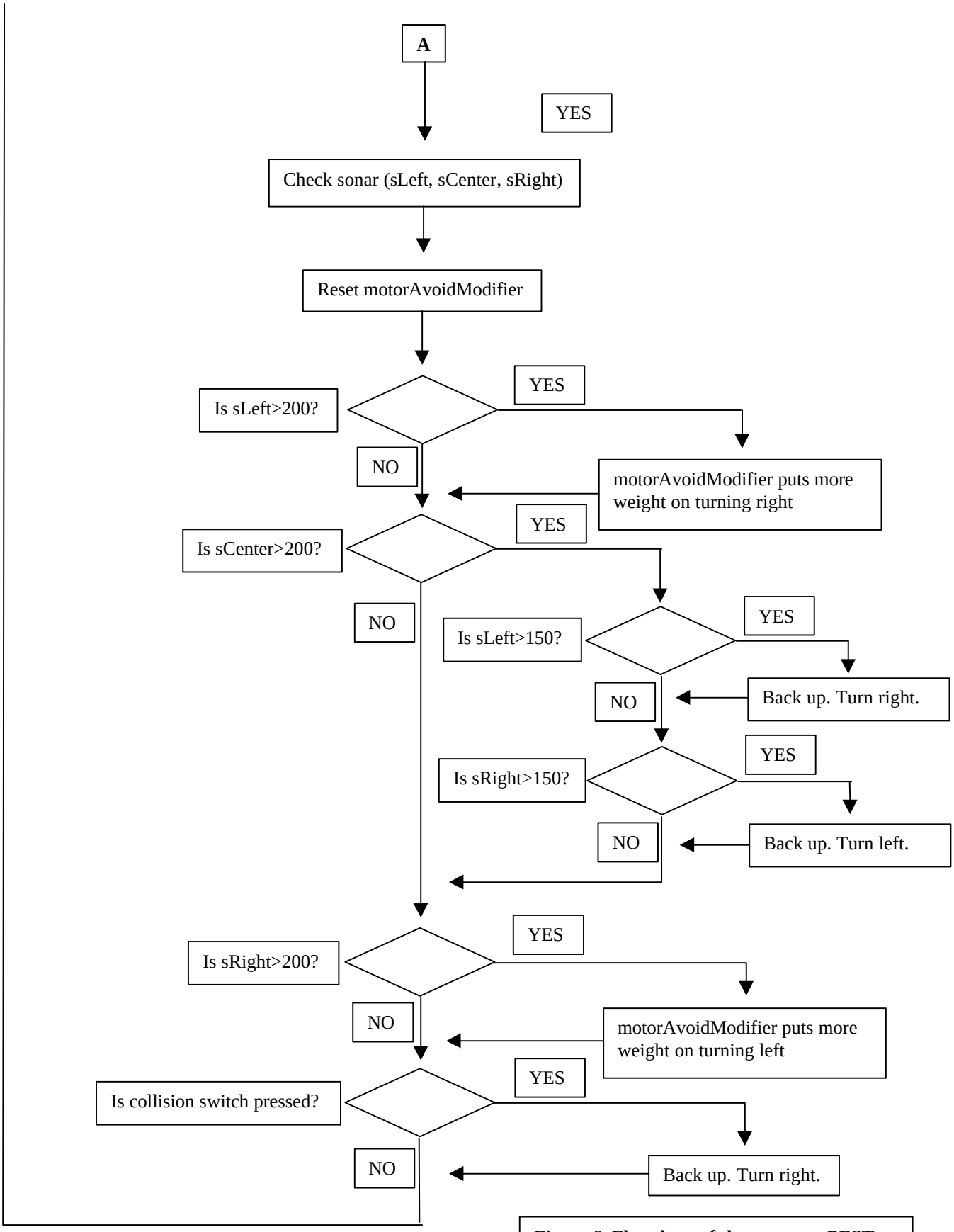


Figure 6. Flowchart of the program PEST.c

-10.0 respectively, making Pest sway more towards direction away from obstacle. If sCenter is bigger than 200, robot will back up and turn sharply in place for .75s in the direction with lower sonar sum reading.

Then, the bump switches are polled. If any of the switches is engaged, Pest will stop, back up and turn right in place for 1.5s.

The infinite loop reads infrared sensor values again and calculates absolute values. If the beacon is sensed closer that a maxThreshold, the motorMod is zeroed making Pest stop advancing towards the beacon. The polling of the sensors continues in the similar fashion.

9. Experimental Layout and Results

The beacon with 1 IR LED hooked up and board with 1 receiver operational was tested. The purpose of the experiment was to test the useful range of IR system as well as its distance reading capabilities. Table 1 presents results of the experiment.

Measurement #	Distance to beacon in in.	Reading
1	No light/Control	85
2	2	129
3	4	128
4	6	128
5	8	128
6	10	128
7	12	127
8	14	124
9	16	121
10	18	118
11	20	115
12	22	113
13	24	111
14	26	110
15	28	109
16	30	107
17	32	105
18	34	103
19	36	100
20	38	98
21	40	96
22	42	95
23	44	94
24	46	93
25	48	92
26	50	91
27	52	91
28	54	91
29	56	90

Table 1: IR sensor reading at various distances to the IR beacon.

Figure 7 presents the results graphed.

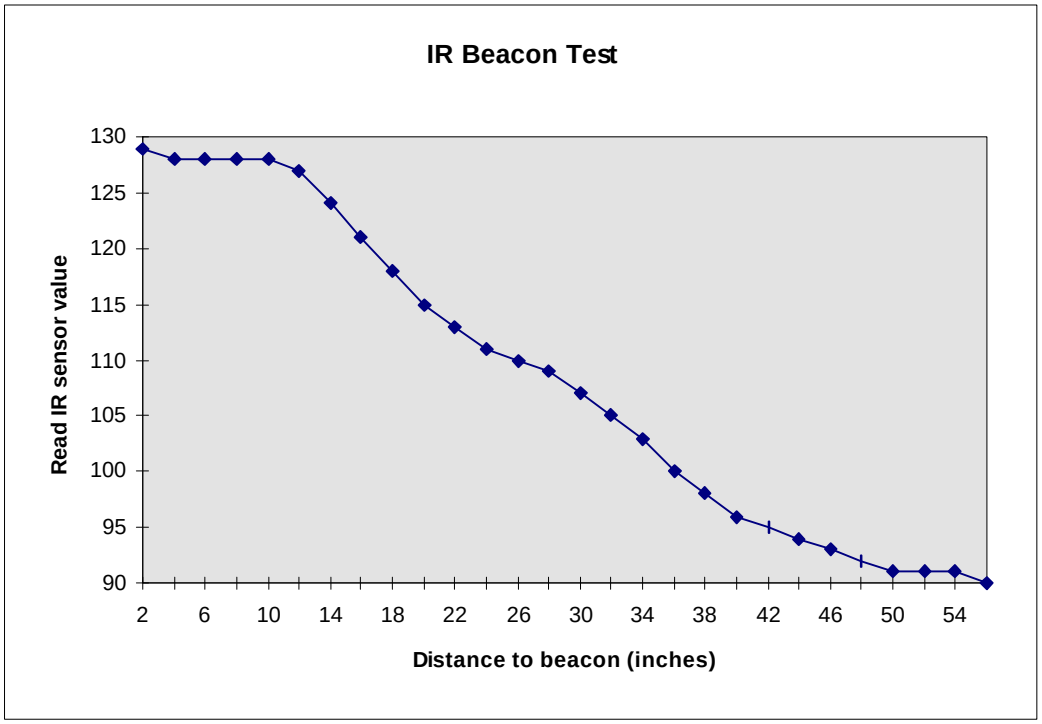


Figure 7: IR Reading vs. Distance to Beacon.

The results are linear over a range of 14-42 inches distances to the beacon. This is a range where relatively accurate distances can be measured. The system does not perform well for the distances below 12 inches and above 50 inches. Although, the beacon can be still detected at 56 inches away while the accurate distance reading is impossible. When several IR transmitters are bundled together the usable range for distance measurement and beacon location might be extended.

Sonar system capabilities were tested. Sonar was fully installed on the platform for the test. All transmitters were connected in parallel and running of a single transmitting circuit. All receivers were hooked up through reed relays into the receiving circuit. A laboratory notebook served as an obstacle. An oscilloscope was connected in parallel with the receiver's ground and signal out. Pest was connected to a computer through a serial

cable and had a program sonar.c (Appendix B) running. Hyperterminal was used to gather outputs from Pest over serial port. The notebook was moved towards and away from each of the receivers. The wave outputted by the receiver has sharp peaks when the object is at the limit of the sonar detection (set by a receiver's potentiometer to about 12 inches) and they become more square when the object is moving closer. Thus, the closer the object is the better probability of getting higher values when sampling the wave. A sum of five reading is used to average the reading. The testing was performed to find out a good threshold value to use in the collision avoidance procedure. The sonar should detect an obstacle at 6-8 inches away for the robot to still have time to react. The readings for 7 inch distance varied from 160-270. Thus, a 200 value of five readings is used as the threshold for collision avoidance.

Values for driving the motors had to be assigned. In the limited time at the end of the semester only few were tested. The value of 15.0 was picked for a sharp turn, 10.0 for a normal turn, 20.0 for the advance modifier and 10.0 for the collision avoidance modifier. These values give Pest agility, while trying to reduce the oscillations. A more thorough analysis of the driving algorithm is recommended.

10. Conclusion

The robot is following the beacon. Since the infrared sensor range is about 40 inches, the distance Pest if following behind at leaves much to be desired. I tried to install beacon on my leg, but the movement of the leg shades IR LEDs. Either the spread of the beacon's range has to be widened or the sensitivity of the receivers improved. At the beginning of the semester I considered implementing sonar into the beacon, but because of problems with the sonar system I did not have time to toy with the idea. It is worth considering, since sonar has a much greater range. Although, some sort of IR-sound synchronization would have to be designed to allow for time of flight readings. Once the following distance is expanded, the beacon can be attached to a person's waist, which would stabilize IR signal.

The robot seems to perform satisfactory at a distance of up to 30-40 inches to the beacon (depending on the noise). The following is prompt. It stops about 10-12 inches in front of the beacon when the beacon is stationary. There is still an overshoot, when the extreme positions are sensed while going in the opposite direction. The beacon has to stop and allow the robot to approach it closer to stabilize the advance.

The sonar works. It is quite hard to distinguish extra collision avoidance procedure sway from the usual beacon following. Although, you can see the difference when you turn the sonar off. The collision avoidance works. Sometimes, the collision avoidance procedure seems to execute still when a collision occurs and the collision resolution is delayed. If I had time, I would like to implement the collision resolution as interrupt driven. It would require rewriting the code into ICC11. The collision avoidance could use

some improvement too. I was really rushing to get everything ready for the demonstration and I was more interested in making the collision avoidance visible, than robust.

It is necessary to rework the sonar transmitting circuit. At the least, a relay should be installed on the transmitter power supply. Although, it would be nice to resolve the overheating problem altogether.

I tested motors to about 45% maximum (with all modifiers added). The robot is swift and certainly has a capability of following a fast paced person. However, for the faster movement, either new driving algorithm is required or an optimization of the current one. Because of the lack of time, I came up with the values for the motors and the motor modifiers only after several tries. It would not be too hard to model the system with discrete equations and optimize Pest's performance.

In conclusion, I have to admit that the sonar gave me the most trouble during this class. I estimate, I spent about 50% of the time setting up and getting my sonar to work (and bugging Scot and Aamir). Nevertheless, I also learned a lot from my mistakes and I certainly learned a lot about sensing and sensor implementation.

11. References

[1] A. Antonio Arroyo, *Syllabus for EEL5666 - Intelligent Machines Design Laboratory*, UF Dept. of Electrical and Computer Engineering, Gainesville, FL, 1/99, page 6.

[2] Forrest M. Mims III, *Engineer's Mini-Notebook - 555 Timer IC Circuits*, Siliconcepts, Fort Worth, TX, 84, page 7.

[3] Keith L. Doty, Erik de la Iglesia, *Sharp Sensor Hack for Analog Distance Measurement*, UF Dept. of Electrical and Computer Engineering, Gainesville, FL, 9/96 http://www.mil.ufl.edu/imdl/handouts/sharphack.pdf

[4] James O'Connor, *Ranos Report*, UF Dept. of Electrical and Computer Engineering, Gainesville, FL, Fall 1998, page 22, http://www.mil.ufl.edu//imdl/papers/IMDL_Report_Fall_98/Bill_OConnor/Ranos.pdf

[5] Machine Intelligence Lab, *Multiplexed Output Ports*, UF Dept. of Electrical and Computer Engineering, Gainesville, FL, 1/99, http://www.mil.ufl.edu/imdl/handouts/mux1.pdf

Appendix A:

Vendor Information

IR emitters/detectors

* LED emitters:
  Mekatronics
  316 NW 17th St., Suite A
  Gainesville, Fl 32603
  (407)672-6780
  http://www.mekatronics.com
  (purchased from Scott Jantz)
  $0.75 each

* Detectors:
  Sharp GP1U58Y via Mekatronics
  (purchased from Scott Jantz)
  $3.00 each

Bump Switches:
Mechatronics
(purchased from Scott Jantz)
$0.75 each

Sonar:

* Transducers:
  Electronic Goldmine
  P.O. Box 5408
  Scottsdale, AZ 85261
  (602)451-7454
  http://www.goldmine-elec.com
  part #G2528
  $1.50 each

* Audio output transformer: Radio Shack (local store)
  part # 273-1380
  $1.99

* Fast-acting reed relays:
  Radio Shack(local store)
  part # 275-233
  $2.49 each

* Max 266 chip:	Maxim Integrated Products 120 San Gabriel Dr. Sunnydale, Ca 94086 (408)737-7600 http://www.maxim-ic.com Part # MAX266 Free sample of two

Actuators:

* Gearhead motors:	Cynatix Inc. 800 Airport Boulevard Suite 304 Burlingame, Ca 94010 (604)344-1193 Part # 137579 motor with 110364 gearhead about $100 each

Sound:

* Piezzo speaker:	Radio Shack (local store) Part # 273-091 $2.49

Appendix B:

Programs

```
/*********************************************************
 *Program:      PEST.c                                    *
 *Programmer:   Marcin Owczarz                            *
 *Description:  Control of EVBU/ME11 based robot to       *
 *              follow IR beacon and avoid obstacles      *
 *                                                        *
 *Date:         4/15/1999                                 *
 *********************************************************/

void main(void)

{
int i0 = 0, i1 = 0, i2 = 0, i3 = 0, i4 = 0;
int ia0, ia1, ia2, ia3, ia4, max = 0, i, j = 0;
int sLeft, sCenter, sRight;
float motorMod = 0.0, valueMod=20.0;
float motorAvoidMod = 0.0;
int noiseThreshold, minThreshold=10, maxThreshold=15;

/*initialise IRs*/
i0=analog(0);
i1=analog(1);
i2=analog(2);
i3=analog(3);
i4=analog(4);

/*Wait 1.5 sec so the operator can pick it up
  right after pressing reset button*/
sleep(1.5);

/*Calibrate IRs*/
noiseThreshold=calibrateIR(i0, i1, i2, i3, i4);
beep();

/*Main loop*/
while(1)
   {

   if (max>maxThreshold) motorMod=0.0;
```

```
/*calculate absolute values*/
ia0=analog(0)-i0;
ia1=analog(1)-i1;
ia2=analog(2)-i2;
ia3=analog(3)-i3;
ia4=analog(4)-i4;

/*Go towards maximum IR signal*/
if((ia0>ia1)&&(ia0>ia2)&&(ia0>ia3)&&(ia0>ia4))
{
  max=analog(0)-i0;
  if ((max<minThreshold)||(max>noiseThreshold)) motorMod=valueMod;
  if (max>maxThreshold) motorMod=0.0;
  motor(0,15.0+motorMod+motorAvoidMod);
  motor(1,15.0-motorMod+motorAvoidMod);
}

if((ia1>ia0)&&(ia1>ia2)&&(ia1>ia3)&&(ia1>ia4))
{
  max=analog(1)-i1;
  if ((max<minThreshold)||(max>noiseThreshold)) motorMod=valueMod;
  if (max>maxThreshold) motorMod=0.0;
  motor(0,10.0+motorMod+motorAvoidMod);
  motor(1,10.0-motorMod+motorAvoidMod);
}

if((ia2>ia1)&&(ia2>ia0)&&(ia2>ia3)&&(ia2>ia4))
{
  max=analog(2)-i2;
  if ((max<minThreshold)||(max>noiseThreshold)) motorMod=valueMod;
  if (max>maxThreshold) motorMod=0.0;
  motor(0,0.0+motorMod+motorAvoidMod);
  motor(1,0.0-motorMod+motorAvoidMod);
}

if((ia3>ia1)&&(ia3>ia2)&&(ia3>ia0)&&(ia0>ia4))
{
  max=analog(3)-i3;
  if ((max<minThreshold)||(max>noiseThreshold)) motorMod=valueMod;
  if (max>maxThreshold) motorMod=0.0;
  motor(0,-10.0+motorMod+motorAvoidMod);
  motor(1,-10.0-motorMod+motorAvoidMod);
}

if((ia4>ia1)&&(ia4>ia2)&&(ia4>ia3)&&(ia4>ia0))
```

```
{
  max=analog(4)-i4;
  if ((max<minThreshold)||(max>noiseThreshold)) motorMod=valueMod;
  if (max>maxThreshold) motorMod=0.0;
  motor(0,-15.0+motorMod+motorAvoidMod);
  motor(1,-15.0-motorMod+motorAvoidMod);
}

/*Collision avoidance*/
checkSonar(&sLeft, &sCenter, &sRight);

motorAvoidMod=0.0;

if (sLeft>200)
{
  beep();
  beep();
  motorAvoidMod = -10.0;
}
if (sCenter>200)
{
  beep();
  beep();
  if (sLeft>150)
  {
    motor(0,0.0);
    motor(1,0.0);
    motor(0,-20.0);
    motor(1,20.0);
    for(i=0;i<200;i++);
    motor(0,0.0);
    motor(1,0.0);
    motor(0,20.0);
    motor(1,20.0);
    for(i=i;i<300;i++);
  }
  else if (sRight>150)
  {
    motor(0,0.0);
    motor(1,0.0);
    motor(0,-20.0);
    motor(1,20.0);
    for(i=0;i<200;i++);
    motor(0,0.0);
    motor(1,0.0);
```

```
        motor(0,-20.0);
        motor(1,-20.0);
        for(i=i;i<300;i++);
      }
    }
    if (sRight>200)
    {
      beep();
      beep();
      motorAvoidMod = 10.0;
    }


    /*Collision procedure*/
    poke(0x6000,1);
    if (analog(7)==0)
    {
      beep();
      collision();
    }

}
}

int calibrateIR(int i0, int i1, int i2, int i3, int i4)
{
int i,j;
int ia, noise;

noise=0;

motor(0,0.0);
motor(1,0.0);
motor(0,20.0);
motor(1,20.0);

for(i=1;i<400;i++)
{
ia=analog(0)-i0;
if (noise<ia) noise=ia;
ia=analog(1)-i1;
if (noise<ia) noise=ia;
ia=analog(2)-i2;
if (noise<ia) noise=ia;
ia=analog(3)-i3;
```

```
if (noise<ia) noise=ia;
ia=analog(4)-i4;
if (noise<ia) noise=ia;

}
motor(0,0.0);
motor(1,0.0);

return(noise);
}

void collision(void)
{
int i;

motor(0,-20.0);
motor(1,20.0);
for(i=1; i<4000; i++);
motor(0,0.0);
motor(1,0.0);
motor(0,10.0);
motor(1,10.0);
}

void checkSonar(int *sl, int *sc, int *sr)
{
int i;
 /*Turn sonar signal on. Sonar transmitter
   overheats quickly, so the procedure tries
   to use it only when necessary*/
   poke(0x7000,0xff);

 /*Sonar receiver is connected through relays to
   all three receiving transducers. Relays are
   run by the multiplexer latch.
   Left transducer   - 4Q - pin 15
   Center Transducer - 6Q - pin 17
   Right Transducer  - 5Q - pin 16*/

 /*Check left sonar (sum of 5 readings)*/
   *sl=0;
   poke(0x6000,9);
   for(i=0;i<5;i++)
   {
      *sl=*sl+analog(6);
```

```
  }

 /*Check center sonar (sum of 5 readings)*/
  *sc=0;
  poke(0x6000,33);
  for(i=0;i<5;i++)
  {
     *sc=*sc+analog(6);
  }

 /*Check right sonar (sum of 5 readings)*/
  *sr=0;
  poke(0x6000,17);
  for(i=0;i<5;i++)
  {
     *sr=*sr+analog(6);
  }

 /*Switch transmitter signal off*/
  poke(0x7000,0x00);
}
```

```
/*******************************************************
 *Program:       sonar.c                                 *
 *Programmer:  Marcin Owczarz                            *
 *Description:   Sonar testing procedure.                *
 *                                                       *
 *Date:          4/10/1999                               *
 *******************************************************/

void main(void)
{
int sl, sc, sr, i, j;

init_serial();


while(1)
{
poke(0x7000,0xff);

/*Check left sonar (sum of 5 readings)*/
  sl=0;
  poke(0x6000,9);
  for(i=0;i<5;i++)
  {
    sl=sl+analog(6);
  }

 /*Check center sonar (sum of 5 readings)*/
  sc=0;
  poke(0x6000,33);
  for(i=0;i<5;i++)
  {
    sc=sc+analog(6);
  }

 /*Check right sonar (sum of 5 readings)*/
  sr=0;
  poke(0x6000,17);
  for(i=0;i<5;i++)
  {
    sr=sr+analog(6);
  }

 /*Switch transmitter signal off*/
  poke(0x7000,0x00);
```

```
print(“\n%d “, sl);
print(“%d ”, sc);
print(“%d”,sr);
}
}
```