“Time of Flight”

Final Report

Ranos

William P. O'Connor
December 9, 1998

EEL 5666
Intelligence Machine Design Laboratory
Aamir Qaiyumi / Scott Jantz / Ivan Zapata
A. A. Arroyo

Table of Contents

1. Abstract 3
2. Executive Summary 4
3. Introduction 5
    - 3.1 Background Information 5
    - 3.2 Scope and Objectives 5
    - 3.3 Overview 5
4. Integrated System 7
    - 4.1 Theory of operation 7
    - 4.2 Electronic connections 8
5. Mobile Platform 9
6. Actuation 10
7. Sensors 11
    - 7.1 Infrared (IR) 11
    - 7.2 Bumpers 11
    - 7.3 Sonar 11
        - 7.3.1 Calibration 13
        - 7.3.2 Analog port 14
        - 7.3.3 Input Capture port 15
        - 7.3.4 Pulse Accumulator 16
    - 7.4 Sensor Integration 17
8. Behaviors 18
    - 8.1 Obstacle Avoidance 18
    - 8.2 Front bumper 18
    - 8.3 Back bumper 18
    - 8.4 Take coordinates 18
    - 8.5 Find coordinates 19
    - 8.6 Randomly finding coordinates 19
9. Experimental Layout and Results 20
10. Conclusion 21
11. Documentation 23

Appendix A: Vendors 24
Appendix B: Ranos' main program 25
Appendix C: Sonar test code 39
- Analog Port Code 39
- Pulse Accumulator Code 40
- Input Capture Code 41

1. Abstract

Ranos is a mapping robot that uses sonar to take and find coordinates of where it is located in a room. Ranos uses a “time of flight” algorithm by pinging a signal against an object and returning to Ranos.

2. Executive Summary

The purpose of Ranos is to create an accurate and precise “time of flight” algorithm by pinging a signal against an object and returning to Ranos. This was done by testing the analog, input capture, and pulse accumulator ports. The results of this test determined that the input capture and pulse accumulator ports gave the most accurate and precise results.

To test this “time of flight” algorithm and meet the IMDL requirements of an autonomous robot, a mapping robot was created. Ranos uses collision and obstacle avoidance behaviors combined with taking coordinates, randomly finding coordinates, and finding coordinates to create the mapping robot.

Ranos uses the Motorola 68HC11 EVBU board with the Mekatronix ME11 board. The platform used is Mekatronix Talrik body with a newly created head. Ranos uses the ME11 board for the power input, power save mode, motor controls, memory, and the 40kHz-output port for the infrared and sonar emitters.

Ranos was successful in the “time of flight” algorithm and performing all its behaviors. The biggest problem occurs when Ranos tries to find the coordinates. Due to lack of time, the wall following routine has not been tested well. This routine would be very useful for zeroing in on coordinates. The other major issue is that a better algorithm can be created if direction is known by the use of a digital compass.

3. Introduction

The purpose of this robot is to explore sonar carefully and come up with a good "time of flight" algorithm. Since using sonar is the main intent of this robot, its name is Ranos, sonar spelled backwards. Ranos incorporates sonar to become a mapping robot.

3.1 Background Information

Michael Apodaca, a former student in IMDL (Intelligent Machine Design Laboratory) during spring 1998, created a charging station, WobbleHead, which tracked his robot Odin using sonar and infrared (IR) sensors. He used sonar because he needed long range detection, which he was able to produce results from almost 25 ft. through the air. The length of the low pulse determined the readings, which is the distance between the emitter and the receiver. The emitter was mounted on Odin while the three receivers were mounted on a charging station.

The next step using sonar is to proceed with a "time of flight" algorithm pinging a signal off of an object back to the source. The circuits he designed for the emitter and receiver are being used for the sonar. These figures are included in section 7, Sensors.

3.2 Scope and Objectives

Ranos must incorporate different behaviors since robots for IMDL must be autonomous. The basic behaviors that must be included are obstacle avoidance and collision. The third behavior tests the "time of flight" algorithm. A microphone will be added to allow commands to be sent to Ranos, and it would have to behave accordingly.

In order to experiment with sonar, Ranos must have a specific function. This function serves as a means of exhibiting Ranos and allows for the experimentation of combining behaviors, thus making Ranos autonomous.

Once Ranos turns on, it will use sonar to take its coordinates in a room. Then, Ranos will move randomly throughout the room. If Ranos happens to move across these coordinates, it will play a sound using a piezo speaker. Every time its microphone detects one clap, Ranos will re-take its coordinates. To allow for a second method of checking whether it remembers its coordinates, if Ranos hears two claps, it must return back to the most recent set of coordinates and play another sound.

3.3 Overview

This paper includes the documentation on the integrated system, mobile platform, actuation, sensors, behaviors, and experimental layout and results of Ranos. These sections demonstrate the components that make Ranos a mapping robot. Accomplishments, limitations, future work, and references

conclude this report. All documentation, code, parts list, head design, and data have been included in the appendices to aid in any future endeavors with sonar or a mapping robot.

4. Integrated System

The main electronics of Ranos includes the Motorola MC68HC11 EVBU board with Mekatronix's ME11 board that includes memory, motor drivers, 40khz clock for sonar and IR, plus other features. Ranos was programmed using ICC11 and includes a data buffer that catches the last 50 values of each sonar, the current coordinates taken, and the last coordinates that were found.

4.1 Theory of operation

Ranos incorporates many behaviors while operating. After reset, Ranos initializes itself. If the front bumper is held down, Ranos display to a terminal program the sonar buffer, current coordinates taken, and last coordinates found. If the front bumper is not pressed, Ranos continues with its normal behaviors. During every stage, Ranos uses obstacle avoidance and collision avoidance with all of the other behaviors. First, Ranos tries to find valid coordinates. If valid coordinates are not obtained after a period of time, Ranos beeps and stops in place. If Ranos finds valid coordinates, it randomly moves about the room. While moving, if Ranos happens to find the coordinates, it will beep to let the user know this. Once the back bumper is hit, Ranos goes into the algorithm to find the original coordinates. If Ranos finds valid coordinates, it will beep, stop in place for several seconds, and then randomly move about the room. Once the back bumper is hit, Ranos will take new coordinates and continue in this loop. Hitting the back bumper to change behaviors was incorporated into Ranos instead of using a microphone due to lack of time and other students having problems implementing a microphone due to noise from other sources. The flowchart for Ranos' operation is in Figure 4.1.

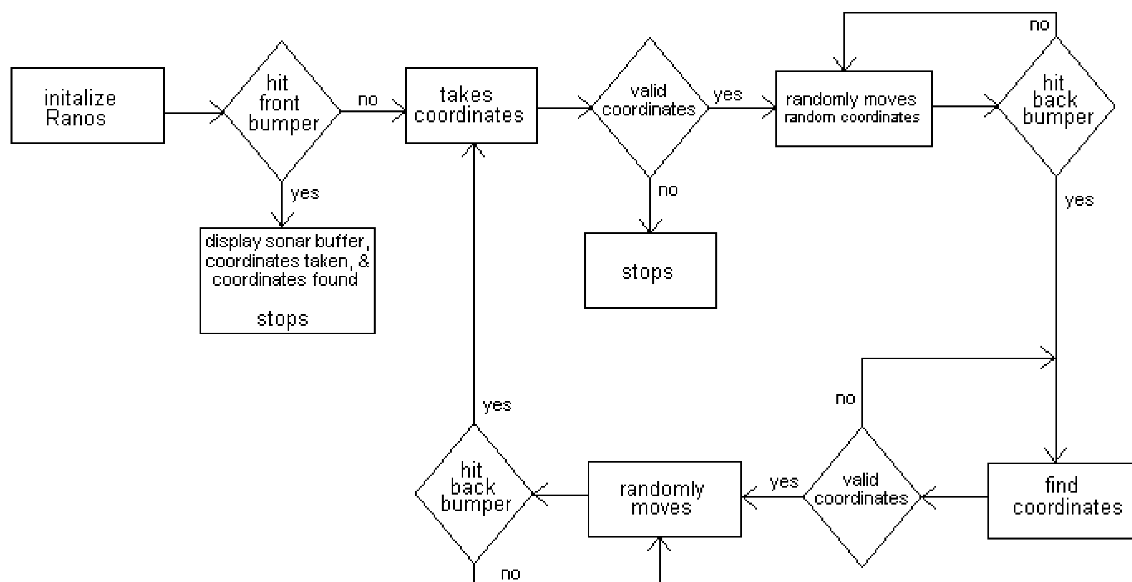


Figure 4.1

4.2 Electronic connections

The bumpers for Ranos are divided into the front and back bumper, which is connected to the analog port pins 6 and 7 respectively. The left and right IR receivers are hooked up to the analog port pins 5 and 4 respectively. The IR emitters are connected to pins 4-7 of the 40kHz port on the ME11 board. The front, left, and right sonar receivers are hooked up to Port A input captures IC1, IC2, and IC3 respectively. The back sonar receiver uses the Pulse Accumulator of Port A. The sonar emitters are connected to pins 0-3 of the 40kHz port on the ME11 board. The piezo speaker is connected to one of the Output Compare pins (OC4) of Port A. The servo for the head to OC5 of Port A and uses power directly from the batteries instead of the board. The servo has been tested, but no algorithm has been created to use the head. The two motors use the ME11 motor ports.

5. Mobile Platform

Since this project concentrates of different sensors and robot behaviors, the structure is not the main area of concentration. Therefore, the TALRIK platform from Mekatronix is used, which contains a 10-inch diameter for its circular base, 10 inches tall, slots for two servos which will contain 3-inch airplane wheels, and slots for five servos on its bridge above the base.

The central slot of the bridge contains a servo to move its head. The head contains four sets of transducers for the sonar “time of flight” algorithm. These transducers are attached to the head at ninety-degree angles for this algorithm so coordinates can be taken in x, y axis. Mounting the transducers in this fashion allows Ranos to take coordinates while moving about the room.

The servos that were used are inexpensive. This means that each servo can operate differently; consequently, Ranos moves towards the left instead of moving straight.

6. Actuation

The actuation for Ranos is not a major concern for this project. Three servos were obtained, two for the motors and one for the head. The two servos for the motors were bought from Novasoft and hacked to become DC motors. The servo for the head was bought from Towerhobbies since they were cheaper. These were not used for the motors since they were of lesser quality.

Section 6.1, Hacking the Servos into DC Gearhead motors, of the Talrik Assembly manual shows how to hack servos. Section 2.3, Actuation, of the Talrik Assembly manual explains the requirements needed for the Talrik platform. No special servos were needed for Ranos. The head can contain the cheapest servo possible, while the motors must meet the minimum specifications needed for the Talrik platform.

7. Sensors

Ranos uses three types of sensors. The IR sensors are used for obstacle avoidance. The front bumper is used for outputting the data buffer to a terminal program and collision algorithm while the back bumper is used to change behaviors and collision algorithm. Sonar is used to take and search for coordinates.

7.1 Infrared (IR)

Ranos uses 2 hacked Sharp GPIU58Y IR receivers on the front of the Talrik platform connected the analog port. Section 5.1, IR Analog Hack, of the Talrik Assembly manual, shows how to hack the IR receivers. Four IR emitters are mounted under the platform, and are attached to the 40kHz port of the ME11 board.

7.2 Bumpers

Ranos uses 5 bump switches for each section (front and back) for the bumper. A circular clear plastic surrounds Ranos where the bump switches are located so that everywhere the plastic is touched (except for where the sides meet up with the bridge), the front or back bumper signal is low.

7.3 Sonar

Sonar was chosen as the special sensor for Ranos since "time of flight" has not been explored in IMDL. Ranos pings a 40kHz signal out of the emitter's transducer against an object and receive the signal through the receiver's transducer. The emitter will use the ME11's 40kHz-output port while the receiver will use the most precise input port.

Three different input methods were tested to receive the sonar's signal. The analog port, which has been used before with sonar, is the least precise port for a "time of flight" algorithm. The 8-bit counter for the pulse accumulator and the 16-bit counters for the three input captures allow for more precise results.

In order to use these sensors, behaviors for Ranos are associated to incorporate the "time of flight" algorithm. Ranos has four sets of sonar sensors, mounted 90 degrees apart, so it can determine its location in a room. Another behavior returns Ranos to its original location by zeroing in on that location using only sonar. These four sets of sensors are synchronized to each other to take these readings. In order for Ranos to know where it is all the time, the room size would need to be no larger than the maximum range of the sensor.

Using Michael Apodaca's circuits of figures 7.1 and 7.2, a maximum distance of 8.5 feet is obtained. For the behavior of determining position within a room, a room 8.5 feet x 8.5 feet is ideal. If Ranos is in a corner, two readings would be at a maximum 8.5 feet. Ranos can get readings in any size room, but all sensors will not contain readings at the same time since the room exceeds maximum dimensions. Two sensors, 90 degrees from each other, are enough to know

where it is in a room if the other two sensors exceeded the maximum readings. Since this is the case, Ranos knows exactly where it is in a room if it knew its direction and the room did not exceed 17 feet by 17 feet.

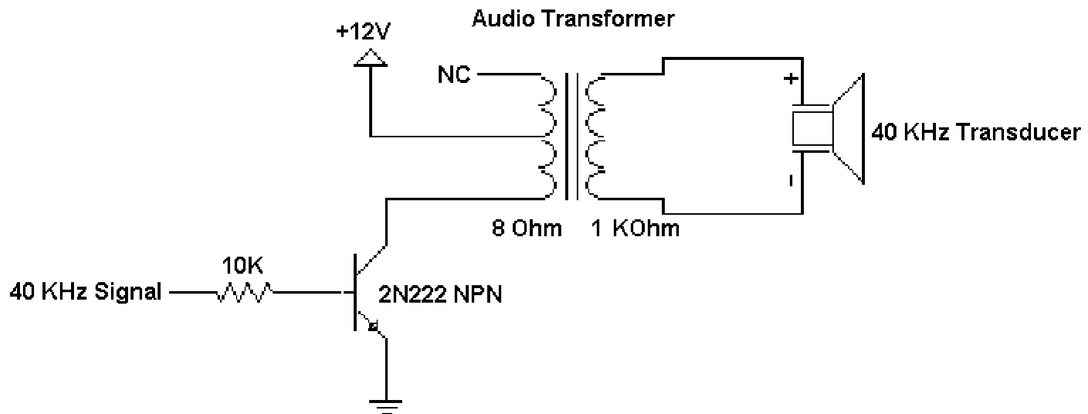


Figure 7.1: Sonar Emitter Circuit

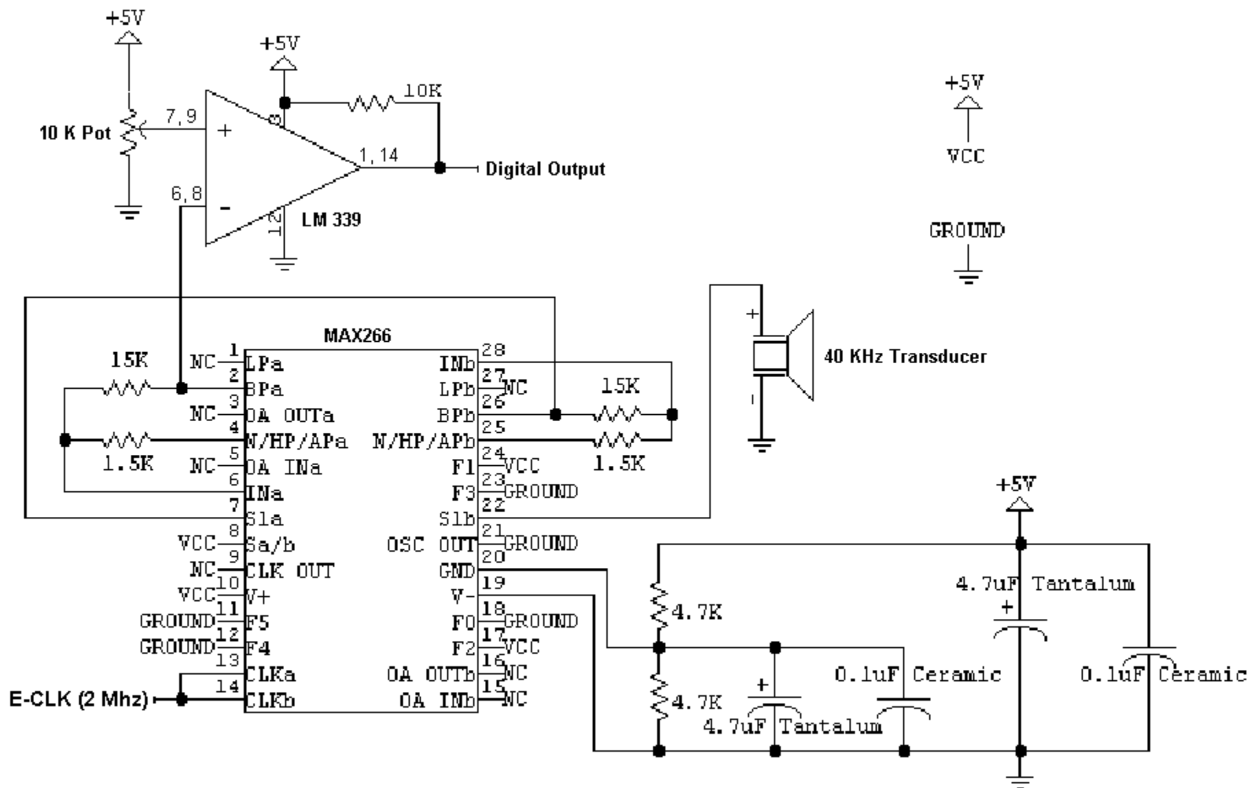


Figure 7.2: Sonar Receiver Circuit

7.3.1 Calibration

To calibrate the receivers, the emitter needs to run at a constant 40 kHz signal. The transducers for the emitter and receiver need to face each other and even connected together. The oscilloscope probe connects to ground and the digital output of the op amp (LM339). The following figures were taken from an oscilloscope set at 5μs and 2 volt/div. The potentiometer needs to be adjusted by turning it right so that the voltage reading between ground and the output is around 3V (shown in figure 7.3).

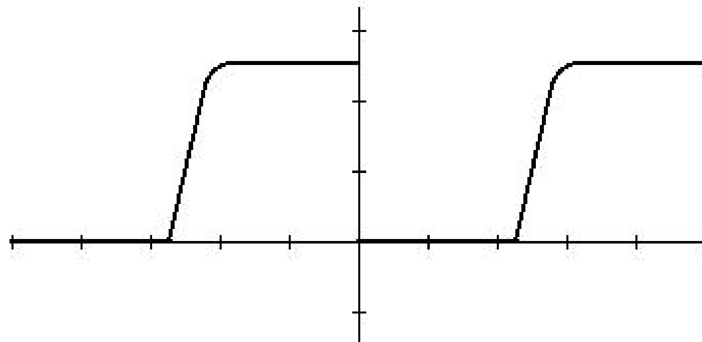


Figure 7.3

When turning the potentiometer to the left, the signal becomes weaker decreasing the positive value (shown in figure 7.4) until the waveform is no longer noticeable.

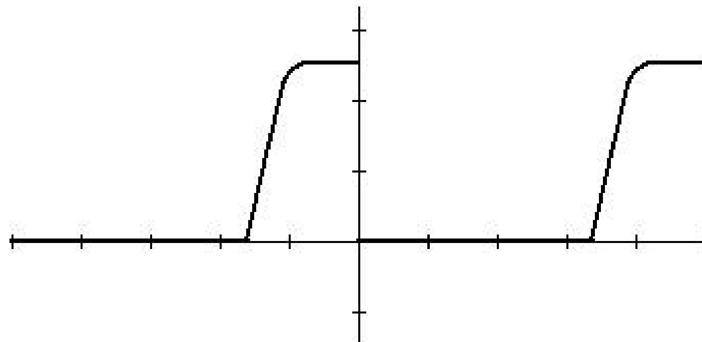


Figure 7.4

7.3.2 Analog port

Several tests were taken to determine the qualities that these sonar circuits possess. The first two tests output data using the analog port code, written by Megan Grimm, listed in Appendix C. Since this code was already written and the main focus is concentrated on the results from the pulse accumulator and input capture pins, this code was used for the initial data readings for the “time of flight” algorithm. This code sends a 40 kHz signal for 1ms and turns the emitter off. The receiver waits for a “falling edge” or for the counter to time-out, which means that the signal did not reflect back, and returns the value of the counter.

The first test determines the accuracy and preciseness of the analog port. The first set of data determines that the receiver produces accurate readings since a linear relationship and repeatable results occurred. To take these tests, the signal reflected off of a white wall in IMDL. A table was marked while using a yardstick every ½ inch. The data was collected when the front of Ranos was located at appropriate distances. The receiver detects a maximum distance of 8.5 feet, which makes the total distance the signal traveled 17 feet. One of the transducers needed to be moved to the base due to interference from the emitter. The receiver was moved to the front of the base since this is where the distance is being measured from for these tests. The receiver was detecting immediate feedback from the emitter while it was waiting for the return signal even though the emitter was turned off after 1ms. The average analog readings were plotted in Figure 7.5. If the analog port is used, the greatest accuracy would be 8.5 feet / 16 (max obtained analog reading) = 0.53125 feet.

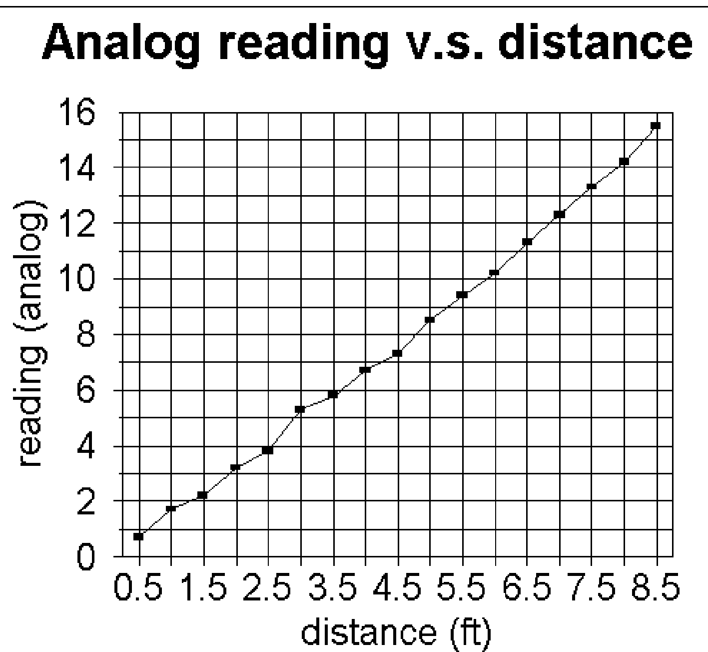


Figure 7.5

The second test determined the angles at several distances where data can be obtained. The average number of non-readings was calculated at different degrees for certain distances, which is plotted in Figure 7.6. At certain distances, a piece of tape was placed on the table to be used as an index mark of a compass. The bumper switches are placed every 15 degrees from IR and CDS cell holders. Measurements were taken 0 degrees and 30 degrees located at bump switches, 15 degrees at the CDS cell holders, and 22½ degrees

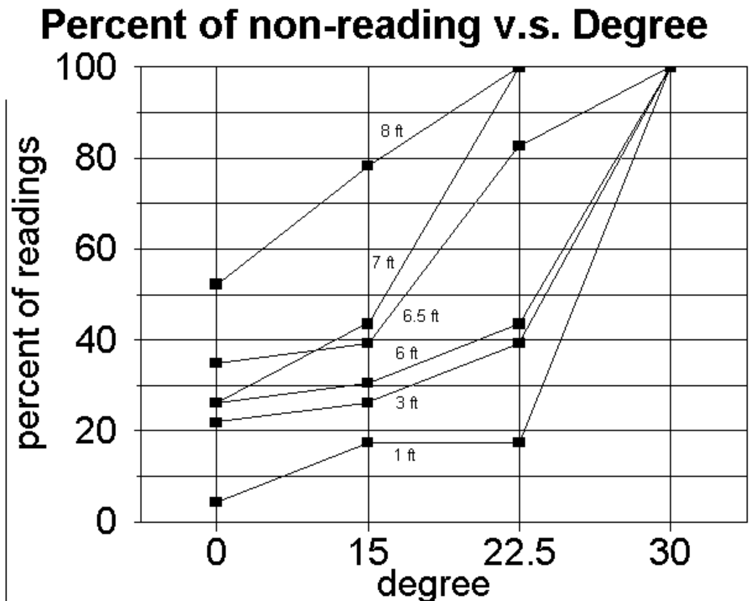


Figure 7.6

approximately half way between a bump switch and CDS cell holder. These marks were lined up with a piece of tape for each reading. At close distances, the number of non-readings is low at 15 and 30 degrees. The further away from the wall, the larger number of non-readings is obtained. At 30 degrees, the receiver can not receive any readings. The further away from the wall Ranos goes, this angle decreases. After 7 feet, the receiver could not obtain a reading beyond an angle of 22.5 degrees.

7.3.3 Input Capture port

After testing the analog port, greater preciseness is desired; therefore the input capture is used to increase the preciseness of the readings. The input capture port captures results off of the free running counter, which increments every 500ns, every time a falling edge occurs. Just like the analog program, the pulse accumulator sends out a sonar signal for 1 ms then looks for a “falling edge” or the counter to time-out. This port captures a 16-bit value; therefore overflows do not need to be accounted for due to the sonar’s maximum distance. This code can be found in Appendix C.

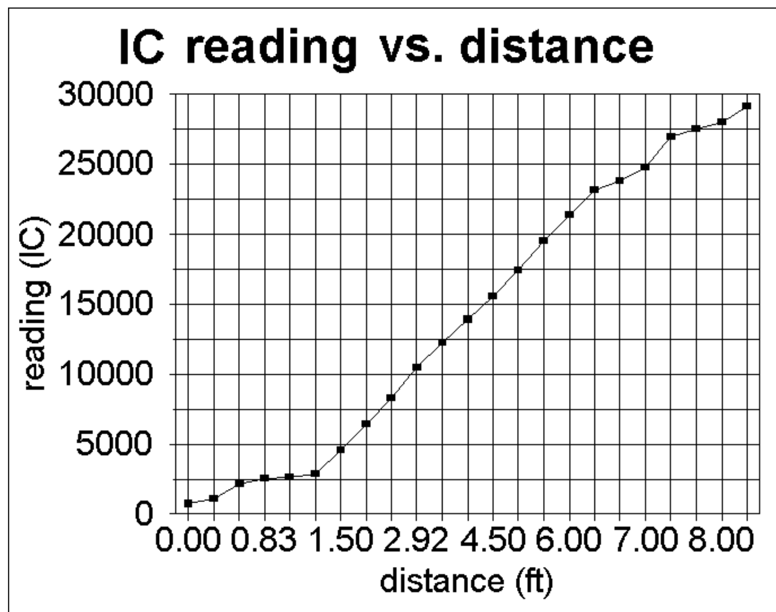


Figure 7.7

The input capture test determines the accuracy and preciseness of the digital port. This data determines that the receiver produces accurate readings since a linear relationship and repeatable results occurred. These tests were taken in the same manner as the analog port test. The receiver detected a maximum distance of 8.25 feet. Ranos may receive readings from farther away since as distance from an object increases, more readings are not read by the receiver. The input capture appeared to be accurate to an inch, or even smaller distances. The average input capture readings were plotted in figure 7.7.

7.3.4 Pulse Accumulator

The pulse accumulator, in gated-time mode, was used since the old M68HC11 manuals stated that there were three input capture ports. The newer manuals show that there is a fourth input capture port, but since the pulse accumulator is precise enough and already wired to Ranos, the fourth port will not be used. The pulse accumulator counter takes increments every 64 E-clocks, or 32μs. Since sonar travels 1 foot every 1ms, the pulse accumulator, in theory, gives more precise readings. Just like the input capture program, the pulse accumulator sends out a sonar signal for 1 ms then looks for a “falling edge” or the counter to time-out. The pulse accumulator counter overflows every 8.19 ms; therefore, around two overflows need to be added to the result of the counter. This code is included in Appendix C.

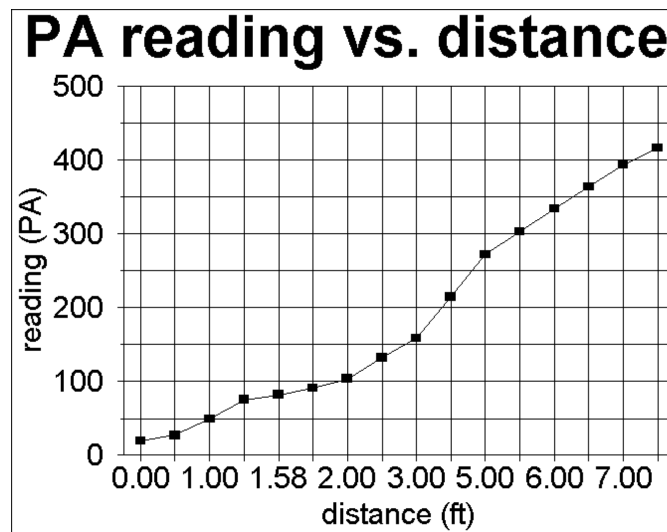


Figure 7.8

The pulse accumulator test determines the accuracy and preciseness of the digital port. This data determines that the receiver produces accurate readings since a linear relationship and repeatable results occurred. These tests were taken in the same manner as the input capture test. The receiver detected a maximum distance of 7.5 feet; however Ranos will probably receive readings from farther away. The pulse accumulator also appeared to be accurate to an

inch, or even smaller distances. The average pulse accumulator readings were plotted in figure 7.8.

7.4 Sensor Integration

When trying to integrate three Input Captures and the Pulse Accumulator together, the receivers sometime got interference from the ringing of the emitters. Approximately a 1½ ms delay was added after the emitters were turned off before receiving any data. This solved the problem, but Ranos cannot receive signals closer than 1¼ feet ((1ms signal + 1¼ delay) / 2). The sensor integration first incorporated integrating the three input captures and the pulse accumulator algorithms. This integration can be seen by looking at ranos.c at the sonar() procedure. The main program had an infinite loop calling sonar (), writing the values to the screen, and a small delay. The second sensor integration was adding the sonar() routine to obstacle and collision avoidance.

8. Behaviors

Figure 4.1 shows the basic flow chart for Ranos' behaviors and briefly described in section 4.1, Theory of operation. These behaviors include obstacle avoidance, front bumper, back bumper, take coordinates, find coordinates, and randomly finding coordinates. The main program for Ranos, ranos.c, is included in Appendix B.

8.1 Obstacle avoidance

Ranos avoids obstacles by reading analog values through the hacked IR receivers. A value above a certain threshold, 100, means that there is an object close by and Ranos needs to try to avoid it. The right and left IR emitters are read through the analog port. If the right value is greater than the threshold, the left direction is set to go at half speed backwards. If the left value is greater than the threshold, the right direction is set to go at half speed backwards. If both values are greater than the threshold, Ranos turns then both directions are set to go forward. The motor commands are then called with the appropriate direction. See the routine avoid_obsticle() in ranos.c for the code. Ivan Zapata gave the outline to this algorithm.

8.2 Front bumper

Ranos knows it has collided with an object when the front bumper is pressed. If the bumper is pressed, Ranos backs up, turns in a random direction for a random amount of time, and then continues moving forward. See the routine front_bumper_hit() in ranos.c for the code. Ivan Zapata gave the outline to this algorithm.

If the front bumper is pressed after reset (during the initialization routine), Ranos outputs the sonar buffer, sonar coordinates taken, and sonar coordinates found.

8.3 Back bumper

Ranos knows an object hit itself from the back if the back bumper is pressed. If the bumper is pressed, Ranos moves forwards, turns in a random direction for a random amount of time, and then continues moving forward. This algorithm is called when Ranos is trying to get coordinates and find coordinates. See the routine back_bumper_hit() in ranos.c for the code.

The back bumper changes Ranos' behaviors. If the back bumper is pressed when Ranos is randomly moving about the room after it has taken coordinates, Ranos begins to try to find those coordinates. If the back bumper is pressed when Ranos is randomly moving about the room after it has found coordinates, Ranos takes new coordinates.

8.4 Take coordinates

When Ranos takes coordinates, Ranos will first spin then randomly move about the room to try to take coordinates.

In the routine spin_for_coord() in ranos.c, coordinates are good if all four sonar receivers produces an accurate readings. Otherwise, Ranos will turn at least 90 degrees until four accurate sonar readings are obtained. At the end of the turn, the coordinates are accurate if the front and/or back sonar has an accurate reading and the left and/or right sonar has an accurate reading. If a reading is not obtained, Ranos then randomly moves about the room.

In the routine move_for_coord() in ranos.c, Ranos moves randomly about the room trying to obtain to accurate readings, at least one in the x-axis and one in the y-axis. Ranos will randomly move about the room until accurate readings are obtained or a period of time expires. If time expires, Ranos beeps and stops in its place.

8.5 Find coordinates

When Ranos enters this behavior, it beeps three times to inform the user. While running collision and obstacle avoidance, Ranos compares sonar results. When comparing sonar results, Ranos checks all possibilities in case it is facing a different direction. If at least one reading on the x and y axis exist, the coordinates are considered to be found; then, Ranos beeps, stops for a small period of time, and then moves randomly about the room until the back bumper is hit. Otherwise, Ranos tries to find a coordinate on its front sensor. Once a coordinate is found, Ranos then tries to follow the wall to try to find at least one more coordinate.

In the routine front_reading() in ranos.c, Ranos tries to obtain any coordinate on the front sensor, then turns 90 degrees. This puts that follow coordinate on the right sonar sensor.

In the routine wall_follow() in ranos.c, if the right sonar is below the threshold, it turns 180 degrees and puts the follow coordinate on the left sonar sensor. At this point, if the right sonar sensor is below the threshold, it goes back to finding a reading on its front sensor. If the sonar reading is in target, Ranos keeps moving in that direction. If Ranos loses the value, it first tries to turn right by a small increment. If there is still no reading, it then turns left by the twice the same small increment it turned right. If there is still no reading, Ranos then goes back to finding a reading on its front sensor.

8.6 Randomly finding coordinates

After Ranos finds coordinates, it randomly moves across the room using the collision and obstacle avoidance routines. If Ranos happens to find the coordinates before the back bumper is hit, it beeps once to inform the user. This routine is random_coord() found in ranos.c.

9. Experimental Layout and Results

Creating the behaviors for Ranos came in many steps. During each step, any problems in the software or hardware were fixed before continuing to the next. Many people in the past do all the hardware, then all the software. By doing this, they have a lot of debugging to do at the end. By creating programs to work with the hardware after each step, Ranos is certain to work properly after each step is completed. By doing this, there will always be something that can be displayed by Ranos.

Once the basic Talrik platform was assembled, the collision and obstacle avoidance example that Ivan gave me was modified for use with ICC11 and Ranos, version 1 of ranos.c. When sections of Ranos did not work for various reasons, IC (interpretive C) was used to test each individual component and displayed the correct value. IC can interactively turn on a particular emitter and read values from a particular receiver. The next step was to create and test the sonar circuits. The individual sonar tests were explained in section 7.3, Sonar. Then, the three input capture ports and the pulse accumulator port had to be synchronized together. Once this was accomplished, the synchronized sonar had to be incorporated into the collision and obstacle avoidance program. The integration of the sonar is explained in section 7.4, Sensor Integration, which is version 2 of ranos.c.

Version 3 adds 4 arrays to store the last 50 values of sonar readings. After this was done, the coordinates taken and found were also added as output to the screen in later versions. Since serial communication was added, later versions include feedback when Ranos is connected to a terminal. In ICC11, variables need to be initialized to any value while being declared. ICC11 puts this variable in a part of SRAM where it will not get re-initialized after reset. Version 4 adds the routines to take the initial coordinates. Version 5 adds a basic routine to find original coordinates after the back bumper is hit. Before the back bumper is hit, Ranos will beep if it randomly finds its coordinates. Version 5 also incorporates beeps for feedback so Ranos does not need to be hooked up to the terminal to figure out what behavior it is in.

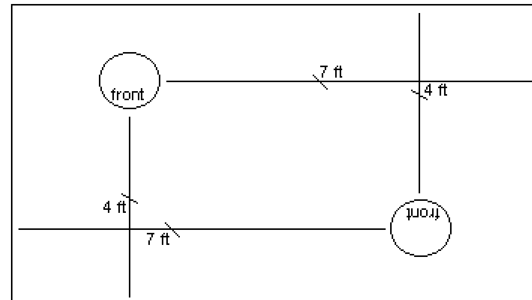
Only the final version of ranos.c is included in this report. For the most part, modular programming was used; therefore, previous versions can be obtained by taking out the newer modules. All programming files were turned in with the final report, including the previous versions.

10. Conclusion

Ranos was successful with collision and obstacle avoidance, emitting and receiving sonar from the analog, input capture, and pulse accumulator ports, storing and displaying a data buffer, taking coordinates, finding coordinates, randomly finding coordinates, and giving feedback through beeps and the terminal. When trying to create beeps, An output pin of port A was turned on and off. When the initialize motor routine is called, it sets all of the output pins of Port A to active low. Before the beep routines will work, the output pin that is used must not interfere with the motor and disable the output compare for that pin only. The code to disable the output compare pin used for the beeps can be seen in the routine initialize() in ranos.c.

The main problem with Ranos is finding coordinates. The first step would be to isolate the wall following_routine() to see how accurate it works. Due to lack of time, this was not accomplished, but the theory seems like it would work. In any demo of Ranos, this behavior was not clearly seen. The next step would be to see if the front_reading() routine works. It needs to find a valid coordinate, turn 90 degrees, and follow the wall at the distance of that coordinate. This method appears to be the best way of finding coordinates because it is difficult to zero in on readings since direction is not known.

Future work on Ranos includes knowing direction, increasing the distance of sonar readings, and limiting the interference between the emitter and receiver. If Ranos knew the direction it was facing when taking coordinates, a better algorithm could be created to find those coordinates. If Ranos found a reading on its front sensor, Ranos would know if that reading was one of the coordinates it was looking for by knowing its direction. If the reading is a valid coordinate in the right direction, Ranos could then follow the wall and find a second reading, etc… Figure 10.1 helps demonstrate this idea.



The front and left sensors are shown in both robot positions. These two sensors (as well as the right and back sensors) would take the same measurements and could not be told apart if the direction it was facing was not known.

Figure 10.1

Another step would be to increase the sensitivity of the sonar. In order for Ranos to always have a reading on all four receivers, the room must be smaller than 8 ½ feet by 8 ½ feet. In order to have a reading on at least one x-axis and one y-axis sensor, the room could be as big as 17 feet by 17 feet. Having two readings might be enough to determine its position if there are no obstacles in the way; however, four sensors would give more feedback to whether or not Ranos found the coordinates.

The last problem occurs if the receiver is turned on immediately after the emitter is turned off. The emitter interferes with the reading of the receiver. Three solutions exist to this problem. The solution that was used adds a delay after the emitter is turned off, but Ranos cannot receive signals closer than 1¼ feet ((1ms signal + 1¼ delay) / 2). The second solution is to try to limit or eliminate the ringing of the emitter by adding a foam-like or rubber-like material. One piece of rubber and foam lying around the IMDL was put around the emitter's terminals, but this did not help any. The last solution is to separate the emitter and receiver. By doing this, the head would be really tall or either the emitters or receivers would need to be moved to the base of the robot. This second option eliminates incorporating the use of the head into any algorithm.

11. Documentation

Several people in IMDL have helped with the success of Ranos. If the help was specific, they were mentioned during the report. Michael Apodaca, Aamir Qaiyumi, Scott Jantz, Ivan Zapata, Megan Grimm, Billy Eno, and Eric Anderson gave help or ideas through final reports or class sessions.

References

Doty, Keith, Talrik^{II} Assembly Manual, 1998.

Appendix A
Vendors

All parts not listed here were either obtained in IMDL or bought at Radio Shack. The two hacked servos for the wheels must meet the minimum requirement specified in the Talrik Assembly Manual. The servo used for the head can be the cheapest servo possible. This servo and the 3" airplane wheels were purchase by another class member.

<u>Electronic Goldmine</u>
Part description: 40 kHz ultrasonic transducers, 15/16" diameter
Part #: G2528
Unit price: $1.50
Websight: http://www.goldmine-elec.com

Remarks: The transducers were ordered by e-mail. They sent out the package the next day, but they never sent a confirmation that they had received the order. Someone else in the class ordered a part that they were out of. Instead of calling about the situation, they just shipped the order with what they had.

<u>Maxim Integrated Products</u>
Part description: switched-capacitor active filters
Part #: MAX266ACPI
Unit price: unknown
Websight: http://www.maxim-ic.com

Remarks: Two free samples can be ordered from their websight. Other friends ordered samples since four chips were needed for four receiver circuits.

<u>Radio Shack</u>
Part Description: audio output transformer, 1K ohm to 8 ohm
Part #: 273-1380
Unit price: $1.99

Part Description: 10K ohm, 15-turn cermet potentiometer
Part #: 271-343
Unit price: $1.49

Appendix B
Ranos' main program

```
/*********************************************************************
 * Ranos.c
 * Bill O'Connor
 * November 22, 1998
 * Version 5
 *
 * This program contains the behaviors for Ranos.

 * Version 1 included a very simple collision avoidance program.
 *          Ranos will read each IR detector, and turn away from any
 *          obstacles in its path.  Also, if something hits Ranos'
 *          bumper, it will back up, turn, and go on. The structure
 *          of this program and some of the ideas came from Ivan
 *          Zapata's avoid.c program with many modifications
 *
 * Version 2 adds the synchronization of 4 sonar tranducers
 *
 * Version 3 adds 4 arrays to store the last 50 values of sonar readings
 *
 * Version 4 adds the routines to take the initial coordinates
 *
 * Version 5 adds a basic routine to find original coordinates after
 *          back bumper is hit.  Before the back bumper is hit, Ranos
 *          will beep if it randomly finds its coordinates.  It also
 *          incorporates beeps for feedback
 *
*********************************************************************/

/**************************** Includes *********************************/
#include <me11.h>
#include <hc11.h>
#include <analog.h>
#include <vectors.h>
#include <serial.h>
/************************ End of includes ******************************/

/**************************** Constants ********************************/
#define OUTPUT_LATCH   *(unsigned char *)(0x7000)

#define DATA           50
#define BEEP_LENGTH    10000

#define IR_BITS               0xf0
#define IR_THRESH      100

#define TURN180_CONST  4
#define TURN90_CONST   8
#define TURN_R_CONST   30000
#define TURN_L_CONST   30000
#define F_BUMP_CONST   5
#define B_BUMP_CONST   5
#define TURN180_BASE   10000
#define TURN90_BASE    9500
#define F_BUMP_BASE    10000
#define B_BUMP_BASE    5000
#define BUMPER_BACK    7
#define BUMPER_FRONT   6
#define IR_LEFT               5
#define IR_RIGHT       4

#define IC_THRESH      1500   /* minimum reading for IC */
#define PA_THRESH      15     /* minimum reading for PA */
#define COORD_NUM      10
#define COORD_NUM2     100
#define IC_MAX         1000   /* +/- IC reading can be off */
#define PA_MAX         15     /* +/- PA reading can be off */
```

```
#define LEFT            0
#define RIGHT           1
#define FWRD_FULL_LEFT 50
#define FWRD_HALF_LEFT 25
#define BACK_FULL_LEFT -50
#define BACK_HALF_LEFT -25
#define FWRD_FULL_RIGHT        100
#define FWRD_HALF_RIGHT        50
#define BACK_FULL_RIGHT        -100
#define BACK_HALF_RIGHT        -50
#define MOTOR_OFF       0

#define RECEIVE_DELAY  90
#define MAX_IC          0xFFFF
#define MAX_PA          0xFF
#define FRONT_BIT       0x08
#define LEFT_BIT        0x04
#define RIGHT_BIT       0x02
#define BACK_BIT        0x01
#define CLEAR_BITS      0x00

#define BIT7    0x80    /* 10000000 */
#define BIT6    0x40    /* 01000000 */
#define BIT5    0x20    /* 00100000 */
#define BIT4    0x10    /* 00010000 */
#define BIT3    0x08    /* 00001000 */
#define BIT2    0x04    /* 00000100 */
#define BIT1    0x02    /* 00000010 */
#define BIT0    0x01    /* 00000001 */
#define BIT65   0x60    /* 01100000 */
#define INV6    0xCF    /* 10111111 */
#define INV5    0xDF    /* 11011111 */
#define INV4    0xEF    /* 11101111 */
#define INV3    0xF7    /* 11110111 */
#define INV2    0xFC    /* 11111011 */
#define INV0    0xFE    /* 11111110 */

/************************ End of Constants ****************************/

/*************************** Prototypes *********************************/
void init_sonar(void);
void initialize(void);
void avoid_obsticle(void);
void front_bumper_hit(void);
void back_bumper_hit(void);
void turn(void);
void turn180(void);
void turn90(void);
void turn_right(void);
void turn_left(void);
void beep(void);
void sonar_IC1(void);
void sonar_IC2(void);
void sonar_IC3(void);
void sonar_PA(void);
void sonar(void);
void update_coord(void);
void spin_for_coord(void);
void move_for_coord(void);
void get_coord(void);
void sonar_compare(void);
void front_reading(void);
void wall_follow(void);
void find_coord(void);
void random_coord(void);
void collect_data(void);
void send_data(void);

/************************ End of Prototypes ******************************/
```

```
/**************************** Globals ********************************/
unsigned int front_sonar[DATA] = {0};
unsigned int left_sonar[DATA] = {0};
unsigned int right_sonar[DATA] = {0};
unsigned int back_sonar[DATA] = {0};
unsigned int front_coord = 0;
unsigned int left_coord = 0;
unsigned int right_coord = 0;
unsigned int back_coord = 0;
unsigned int found_front = 0;
unsigned int found_left = 0;
unsigned int found_right = 0;
unsigned int found_back = 0;
int coord_flag, coord_count, x_axis, y_axis, x_current, y_current;
int data_counter = 0;
int over, stop, found, num_found_x, num_found_y, num_x, num_y;
int front_flag, follow_value, follow_count;
unsigned int first, last, sonar_front, sonar_left, sonar_right, sonar_back;
int r_dir, l_dir;
int r_IR, l_IR, f_bump, b_bump;
int i, j;

/************************** End of Globals *****************************/

/****************************** Main **********************************/
int main(void)
{
  initialize();
  if (!stop)
    get_coord();
  while(!stop)
    {
      avoid_obsticle();
      front_bumper_hit();
      random_coord();
      b_bump = analog(BUMPER_BACK);
      if (b_bump < 100)
        {
          find_coord();
          b_bump = analog(BUMPER_BACK);
          while (b_bump < 100)
            b_bump = analog(BUMPER_BACK);
          write("roaming!\n\r");
          while (b_bump > 100)
            {
              avoid_obsticle();
              front_bumper_hit();
              for(i=1; i < 2500; i++);
              b_bump = analog(BUMPER_BACK);
            }
          get_coord();
        }
      for(i=1; i < 2500; i++);
    }
}

/*************************** End of Main *****************************/

void init_sonar()
{
  /* front sonar (IC1) */
  TCTL2 = TCTL2 | BIT5; /* capture on falling edge */
  TCTL2 = TCTL2 & INV4;

  /* left sonar (IC2) */
  TCTL2 = TCTL2 | BIT3; /* capture on falling edge */
  TCTL2 = TCTL2 & INV2;

  /* right sonar (IC3) */
  TCTL2 = TCTL2 | BIT1; /* capture on falling edge */
  TCTL2 = TCTL2 & INV0;
```

```
  /* back sonar (PA) */
  PACTL = BIT65;  /* PAMOD = gated, PEDGE = falling, & enable */
}

void initialize()
{
  init_motors();       /* Initialize necessary registers, etc. */
  TCTL1 = TCTL1 & (INV3 & INV2);  /* Turn OC4 off for beep() */
  beep();
  init_analog();
  DDRD = 0x30;         /* Initializes data direction of Port D pins 4, 5 */
  init_sonar();
  init_serial();
  write ("\n\rstart\n\r");

  stop = 0;
  f_bump = analog(BUMPER_FRONT);
  if (f_bump < 100)
    {
      send_data();
      stop = 1;
    }
  else
    {
      for (i=0; i<DATA; i++)
        {
          front_sonar[i] = 0;
          left_sonar[i] = 0;
          right_sonar[i] = 0;
          back_sonar[i] = 0;
        }
      data_counter = 0;
    }
}

void avoid_obsticle()
/*******************************************************************
 * The following block will read the ir ports, and decide whether   *
 * Ranos needs to turn to avoid any obstacles                       *
 *******************************************************************/
{
  OUTPUT_LATCH = IR_BITS;     /* Turn IR on */

  r_IR = analog(IR_RIGHT);
  l_IR = analog(IR_LEFT);

  if (r_IR > IR_THRESH)
    l_dir = BACK_HALF_LEFT;
    else l_dir = FWRD_FULL_LEFT;
  if (l_IR > IR_THRESH)
    r_dir = BACK_HALF_RIGHT;
    else r_dir = FWRD_FULL_RIGHT;

  if ((r_IR > IR_THRESH) && (l_IR > IR_THRESH))
    {
      turn180();
      l_dir = FWRD_FULL_LEFT;
      r_dir = FWRD_FULL_RIGHT;
    }

  motor(RIGHT, r_dir);
  motor(LEFT, l_dir);
}

void front_bumper_hit()
/*******************************************************************
 * This "if" statement checks the bumper. If the bumper is pressed, *
 * Ranos will reverse directions, and turn.                         *
 *******************************************************************/
{
```

```
  f_bump = analog(BUMPER_FRONT);

  if (f_bump < 100)
    {
      motor(RIGHT, BACK_HALF_RIGHT);
      motor(LEFT, BACK_HALF_LEFT);
      for (i=0; i < F_BUMP_BASE; i++)
        for (j=0; j < F_BUMP_CONST; j++);
      turn();
    }
}

void back_bumper_hit()
{
  b_bump = analog(BUMPER_BACK);

  if (b_bump < 100)
    {
      motor(RIGHT, FWRD_HALF_RIGHT);
      motor(LEFT, FWRD_HALF_LEFT);
      for (i=0; i < B_BUMP_BASE; i++)
        for (j=0; j < B_BUMP_CONST; j++);
      turn();
    }
}

void turn()
/**************************************************************************
 * Function: Will turn in a random direction for a fixed amount of
 * time.  This amount of time will try to equal a 90 degree turn.
 * The acuraccy will depend on the type of surface and wheel slipage.
**************************************************************************/
{
  int i, j;
  unsigned rand;
  rand = TCNT;
  if (rand & 0x0001)
    {
      motor(RIGHT, FWRD_HALF_RIGHT);
      motor(LEFT, BACK_HALF_LEFT);
    }
    else
      {
        motor(RIGHT, BACK_HALF_RIGHT);
        motor(LEFT, FWRD_HALF_LEFT);
      }
  for (i = 0; i < rand; i++);
}

void turn180()
{
  int i, j;
  motor(RIGHT, FWRD_HALF_RIGHT);
  motor(LEFT, BACK_HALF_LEFT);
  for (i = 0; i < TURN180_BASE; i++)
    for (j = 0; j < TURN180_CONST; j++);
}

void turn90()
{
  int i, j;
  motor(RIGHT, FWRD_HALF_RIGHT);
  motor(LEFT, BACK_HALF_LEFT);
  for (i = 0; i < TURN90_BASE; i++)
    for (j = 0; j < TURN90_CONST; j++);
}

void turn_right()
{
  int i, j;
  motor(RIGHT, BACK_HALF_RIGHT);
```

```
  motor(LEFT, FWRD_HALF_LEFT);
  for (i = 0; i < TURN_R_CONST; i++);
}

void turn_left()
{
  int i, j;
  motor(RIGHT, FWRD_HALF_RIGHT);
  motor(LEFT, BACK_HALF_LEFT);
  for (i = 0; i < TURN_L_CONST; i++);
}

void beep()
{
  PORTA = PORTA | BIT4;
  for (i = 0; i < BEEP_LENGTH; i++);
  PORTA = PORTA & INV4;
  for (i = 0; i < BEEP_LENGTH; i++);
}

void sonar_IC1()
{
  over = 0;

  OUTPUT_LATCH = FRONT_BIT; /* enables emitters */
  for (i=1; i<60; i++);
  OUTPUT_LATCH = CLEAR_BITS; /* disables emitters after 1ms */
  for (i=1; i < RECEIVE_DELAY; i++);

  TFLG1 = BIT2; /* Clear IC1 flag */
  TFLG2 = BIT7;

  first = TCNT;

  while (!(TFLG1 & BIT2) && (over < 3))
     if (TFLG2 & BIT7)
       {
         over++;
         TFLG2 = BIT7; /* clears TOF */
       }

  last = TIC1;

  TFLG1 = BIT2; /* Clear IC1 flag */

  if (last > first)
    sonar_front = last - first;
    else
      sonar_front = abs((MAX_IC+1) - first + last);
  if (over==3)
    sonar_front = 0;
}

void sonar_IC2()
{
  over = 0;

  OUTPUT_LATCH = LEFT_BIT; /* enables emitters */
  for (i=1; i<60; i++);
  OUTPUT_LATCH = CLEAR_BITS; /* disables emitters after 1ms */
  for (i=1; i < RECEIVE_DELAY; i++);

  TFLG1 = BIT1; /* Clear IC2 flag */
  TFLG2 = BIT7;

  first = TCNT;

  while (!(TFLG1 & BIT1) && (over < 3))
     if (TFLG2 & BIT7)
       {
         over++;
```

```
        TFLG2 = BIT7; /* clears TOF */
       }

  last = TIC2;

  TFLG1 = BIT1; /* Clear IC2 flag */

  if (last > first)
    sonar_left = last - first;
    else
      sonar_left = abs((MAX_IC+1) - first + last);
  if (over==3)
    sonar_left = 0;
}

void sonar_IC3()
{
  over = 0;

  OUTPUT_LATCH = RIGHT_BIT; /* enables emitters */
  for (i=1; i<60; i++);
  OUTPUT_LATCH = CLEAR_BITS; /* disables emitters after 1ms */
  for (i=1; i < RECEIVE_DELAY; i++);

  TFLG1 = BIT0; /* Clear IC3 flag */
  TFLG2 = BIT7;

  first = TCNT;

  while (!(TFLG1 & BIT0) && (over < 3))
     if (TFLG2 & BIT7)
       {
        over++;
        TFLG2 = BIT7; /* clears TOF */
       }

  last = TIC3;

  TFLG1 = BIT0; /* Clear IC3 flag */

  if (last > first)
    sonar_right = last - first;
    else
      sonar_right = abs((MAX_IC+1) - first + last);
  if (over==3)
    sonar_right = 0;
}

void sonar_PA()
{
  OUTPUT_LATCH = BACK_BIT; /* enables emitters */
  for (i=1; i<60; i++);
  OUTPUT_LATCH = CLEAR_BITS; /* disables emitters after 1ms */
  for (i=1; i < RECEIVE_DELAY; i++);

  over = 0;
  TFLG2 = BIT4;  /* clears pulse accumulator flag */
  TFLG2 = BIT5;  /* clears overflow flag */
  PACNT = 0;

  while (!(TFLG2 & BIT4) && (over < 10))
    if (TFLG2 & BIT5)
      {
        over=over+1;
        TFLG2 = BIT5; /* clears overflow flag */
      }
  if (over < 10)
    {
      sonar_back = PACNT;
      for (; over > 0; over--)
        sonar_back += (MAX_PA+1);
```

```
    }
    else sonar_back = 0;
}

void sonar()
/* The following block will emit and transmit sonar */
{
  sonar_IC1();
  for (i=1; i<300; i++);  /* 5ms delay to make sure signal has stopped */
  sonar_IC2();
  for (i=1; i<300; i++);  /* 5ms delay to make sure signal has stopped */
  sonar_IC3();
  for (i=1; i<300; i++);  /* 5ms delay to make sure signal has stopped */
  sonar_PA();
  OUTPUT_LATCH = IR_BITS;
}

void update_coord()
{
  sonar();
  collect_data();
  x_current = 0;
  y_current = 0;
  if (sonar_front > IC_THRESH)
    x_current++;
  if (sonar_left > IC_THRESH)
    y_current++;
  if (sonar_right > IC_THRESH)
    y_current++;
  if (sonar_back > PA_THRESH)
    x_current++;

  /* Coordinates are changed if the number of sonar that produced
     accurate readings is greater or equal to the previous number,
     if the front and/or back sonar has an accurate reading, and if
     the left and/or right sonar has an accurate reading
  */
  if (((x_current + y_current) >= (x_axis + y_axis)) &&
      (x_current > 0) && (x_current >= x_axis) && (y_current > 0) &&
      (y_current >= y_axis))
    {
      front_coord = sonar_front;
      left_coord = sonar_left;
      right_coord = sonar_right;
      back_coord = sonar_back;
      x_axis = x_current;
      y_axis = y_current;
    }
}

void spin_for_coord()
{
  while ((coord_count < COORD_NUM) && (!coord_flag))
    {
      update_coord();

      /* Coordinates are good if all 4 sonar produces accurate readings.
         Otherwise, Ranos will turn at least 90 degrees until 4 accurate
         sonar readings are obtained.  At the end of the turn, the
         coordinates are accurate if the front and/or back sonar has an
         accurate reading and the left and/or right sonar has an accurate
         reading [i.e. ((x_axis > 0) & (y_axis > 0)) ]
      */
      if (((x_axis + y_axis) == 4) ||
          ((coord_count > COORD_NUM) && (x_axis > 0) && (y_axis > 0)))
        coord_flag = 1;
        else if (coord_count < COORD_NUM)
               {
                 r_dir = BACK_HALF_RIGHT;
                 l_dir = FWRD_HALF_LEFT;
               }
```

```
      if ((coord_count > COORD_NUM) || ((x_axis + y_axis) == 4))
        {
          r_dir = MOTOR_OFF;
          l_dir = MOTOR_OFF;
        }

      motor(RIGHT, r_dir);
      motor(LEFT, l_dir);
      coord_count++;
    }
}

void move_for_coord()
{
  while ((coord_count < COORD_NUM2) && (!coord_flag))
    {
      update_coord();
      if ((x_axis > 0) && (y_axis > 0))
        coord_flag = 1;
      for(i=1; i < 750; i++);
      avoid_obsticle();
      front_bumper_hit();
      back_bumper_hit();
      coord_count++;
    }
  motor(RIGHT, MOTOR_OFF);
  motor(LEFT, MOTOR_OFF);
}

void get_coord()
{
  coord_flag = 0;
  coord_count = 0;
  x_axis = 0;
  y_axis = 0;
  spin_for_coord();
  move_for_coord();
  if (!coord_flag)
    {
      stop = 1;
      write("coordinates not taken!\n\r");
    }
    else write("coordinates taken!\n\r");
  beep();
  beep();
}

void sonar_compare()
{
  num_found_x = 0;
  num_found_y = 0;
  num_x = 0;
  num_y = 0;
  if ((sonar_front < (front_coord + IC_MAX)) &&
      (sonar_front > (front_coord - IC_MAX)) &&
      (front_coord != 0))
    num_x++;
  if ((sonar_left < (left_coord + IC_MAX)) &&
      (sonar_left > (left_coord - IC_MAX)) &&
      (left_coord != 0))
    num_y++;
  if ((sonar_back < (back_coord + PA_MAX)) &&
      (sonar_back > (back_coord - PA_MAX)) &&
      (back_coord != 0))
    num_x++;
  if ((sonar_right < (right_coord + IC_MAX)) &&
      (sonar_right > (right_coord - IC_MAX)) &&
      (right_coord != 0))
    num_y++;
  if (num_x > num_found_x)
    num_found_x = num_x;
```

```
if (num_y > num_found_y)
  num_found_y = num_y;

num_x = 0;
num_y = 0;
if ((sonar_front < (left_coord + IC_MAX)) &&
    (sonar_front > (left_coord - IC_MAX)) &&
    (left_coord != 0))
  num_x++;
if ((sonar_left < ((back_coord * 64) + IC_MAX)) &&
    (sonar_left > ((back_coord * 64) - IC_MAX)) &&
    (back_coord != 0))
  num_y++;
if (((sonar_back * 64) < (right_coord + IC_MAX)) &&
    ((sonar_back * 64) > (right_coord - IC_MAX)) &&
    (right_coord != 0))
  num_x++;
if ((sonar_right < (front_coord + IC_MAX)) &&
    (sonar_right > (front_coord - IC_MAX)) &&
    (front_coord != 0))
  num_y++;
if (num_x > num_found_x)
  num_found_x = num_x;
if (num_y > num_found_y)
  num_found_y = num_y;

num_x = 0;
num_y = 0;
if ((sonar_front < ((back_coord * 64) + IC_MAX)) &&
    (sonar_front > ((back_coord * 64) - IC_MAX)) &&
    (back_coord != 0))
  num_x++;
if ((sonar_left < (right_coord + IC_MAX)) &&
    (sonar_left > (right_coord - IC_MAX)) &&
    (right_coord != 0))
  num_y++;
if (((sonar_back * 64) < (front_coord + IC_MAX)) &&
    ((sonar_back * 64) > (front_coord - IC_MAX)) &&
    (front_coord != 0))
  num_x++;
if ((sonar_right < (left_coord + IC_MAX)) &&
    (sonar_right > (left_coord - IC_MAX)) &&
    (left_coord != 0))
  num_y++;
if (num_x > num_found_x)
  num_found_x = num_x;
if (num_y > num_found_y)
  num_found_y = num_y;

num_x = 0;
num_y = 0;
if ((sonar_front < (right_coord + IC_MAX)) &&
    (sonar_front > (right_coord - IC_MAX)) &&
    (right_coord != 0))
  num_x++;
if ((sonar_left < (front_coord + IC_MAX)) &&
    (sonar_left > (front_coord - IC_MAX)) &&
    (front_coord != 0))
  num_y++;
if (((sonar_back * 64) < (left_coord + IC_MAX)) &&
    ((sonar_back * 64) > (left_coord - IC_MAX)) &&
    (left_coord != 0))
  num_x++;
if ((sonar_right < ((back_coord * 64) + IC_MAX)) &&
    (sonar_right > ((back_coord * 64) - IC_MAX)) &&
    (back_coord != 0))
  num_y++;
if (num_x > num_found_x)
  num_found_x = num_x;
if (num_y > num_found_y)
  num_found_y = num_y;
```

```
  if ((num_found_x > 0) && (num_found_y > 0))
    found = 1;
}

void front_reading()
{
  if (front_flag == 0)
    if (((sonar_front < (front_coord + IC_MAX)) &&
         (sonar_front > (front_coord - IC_MAX))) ||
        ((sonar_front < ((back_coord * 64) + IC_MAX)) &&
         (sonar_front > ((back_coord * 64) - IC_MAX))) ||
        ((sonar_front < (left_coord + IC_MAX)) &&
         (sonar_front > (left_coord - IC_MAX))) ||
        ((sonar_front < (right_coord + IC_MAX)) &&
         (sonar_front > (right_coord - IC_MAX))))
      {
        front_flag++;
        turn90();
        follow_value = sonar_front;
      }
}

void wall_follow()
{
  if ((front_flag > 0) && (r_IR < IR_THRESH) && (l_IR < IR_THRESH))
    {
      if ((sonar_front < IC_THRESH) && (sonar_front != 0))
        {
          turn90();
          turn90();
          front_flag++;
        }
      if ((sonar_right < (follow_value + IC_MAX)) &&
          (sonar_right > (follow_value - IC_MAX)) &&
          (front_flag == 1))
        follow_count = 0;
        else follow_count++;
      if ((sonar_left < (follow_value + IC_MAX)) &&
          (sonar_left > (follow_value - IC_MAX)) &&
          (front_flag == 2))
        follow_count = 0;
        else follow_count++;
      if (follow_count == 1)
        turn_right();
      if (follow_count == 2)
        {
          turn_left();
          turn_left();
        }
      if ((follow_count == 3) || (follow_count == 3))
        {
          follow_count = 0;
          front_flag = 0;
        }
    }
}

void find_coord()
{
  write("looking for coordinates!\n\r");
  beep();
  beep();
  beep();
  found = 0;
  front_flag = 0;
  follow_count = 0;
  while (!found)
    {
      avoid_obsticle();
      front_bumper_hit();
```

```
      sonar();
      collect_data();
      back_bumper_hit();
      sonar_compare();
      front_reading();
      wall_follow();
      for(i=1; i < 2500; i++);
    }
  found_front = sonar_front;
  found_left = sonar_left;
  found_right = sonar_right;
  found_back = sonar_back;
  write("coordinates found!\n\r");
  motor(RIGHT, MOTOR_OFF);
  motor(LEFT, MOTOR_OFF);
  for(i=1; i < 2500; i++);
  write("num_found_x = ");
  write_int(num_found_x);
  write("num_found_y = ");
  write_int(num_found_y);
  while(num_found_x > 0)
     {
      beep();
      num_found_x--;
     }
  while(num_found_y > 0)
     {
      beep();
      num_found_y--;
     }
  for (i = 0; i < 10; i++)    /* delay for motor being off */
    for (j = 0; j < 30000; j++);
}

void random_coord()
{
  found = 0;
  sonar();
  collect_data();
  sonar_compare();
  if (found == 1)
    {
      found_front = sonar_front;
      found_left = sonar_left;
      found_right = sonar_right;
      found_back = sonar_back;
      beep();
      write("randomly found coordinates!\n\r");
    }
}

void collect_data()
{
  front_sonar[data_counter] = sonar_front;
  left_sonar[data_counter] = sonar_left;
  right_sonar[data_counter] = sonar_right;
  back_sonar[data_counter] = sonar_back;
  if (++data_counter == DATA)
    data_counter = 0;
}

void send_data()
{
  j = data_counter-1;
  write ("\n\rfront\n\r");
  for (i=0; i<DATA; i++)
    {
      write_int(front_sonar[j]);
      if (j == 0)
        j = DATA;
      j--;
```

```
    }

  j = data_counter-1;
  write ("\n\rleft\n\r");
  for (i=0; i<DATA; i++)
    {
      write_int(left_sonar[j]);
      if (j == 0)
        j = DATA;
      j--;
    }

  j = data_counter-1;
  write ("\n\rright\n\r");
  for (i=0; i<DATA; i++)
    {
      write_int(right_sonar[j]);
      if (j == 0)
        j = DATA;
      j--;
    }

  j = data_counter-1;
  write ("\n\rback\n\r");
  for (i=0; i<DATA; i++)
    {
      write_int(back_sonar[j]);
      if (j == 0)
        j = DATA;
      j--;
    }

  write ("\n\rfront coord: ");
  write_int (front_coord);
  write ("left coord: ");
  write_int (left_coord);
  write ("back coord: ");
  write_int (back_coord);
  write ("back coord * 64: ");
  write_int (back_coord * 64);
  write ("right coord: ");
  write_int (right_coord);

  write ("\n\rlast found front coord: ");
  write_int (found_front);
  write ("last found left coord: ");
  write_int (found_left);
  write ("last found back coord: ");
  write_int (found_back);
  write ("last found back coord * 64: ");
  write_int (found_back * 64);
  write ("last found right coord: ");
  write_int (found_right);
}
```

Appendix C
Sonar Test Code

Analog Port Code

```
/* Megan Grimm with the invaluable assistance of Scott Jantz */
/* 10/6/98 */
/* time-of-flight calibration for Alph's sonar */


int counter;

void main()
{
   init_serial();
   counter=0;
   write("data");
   while(1)
   {
   counter=0;
      poke(0x7000,0x0f);
      msleep(1L);
      poke(0x7000,0x00);

   while((analog(0)>200)&&(counter<1000))
   {

      counter=counter+1;

   }
   write_int(counter);
   msleep(1000L);
   }
}
```

Pulse Accumulator Code

```
/* Bill O'Connor */
/* 10/21/98 */
/* "time of flight" calibration for Ranos' sonar using
   Pulse Accumulator */

#include <me11.h>
#include <hc11.h>
#include <analog.h>
#include <vectors.h>
#include <serial.h>

#define OUTPUT_LATCH      *(unsigned char *)(0x7000)

#define BIT6  0x40   /* 01000000 */
#define BIT5  0x20   /* 00100000 */
#define       BIT4   0x10   /* 00010000 */
#define       BIT54  0x30   /* 00110000 */
#define INV6  0xCF   /* 10111111 */
#define INV5  0xDF   /* 11011111 */

int counter, i;

void main()
{
   init_serial();
   PACTL = BIT54;  /* PAMOD = gated & PEDGE = rising */
   write("data");
   counter=0;
   while(1)
   {
   counter=0;
   PACNT = 0;
   PACTL = PACTL | BIT6;  /* enable pulse accumulator */

   OUTPUT_LATCH = 0x0f;  /* enables emitter */
   for(i=1; i < 60; i++);
   OUTPUT_LATCH = 0x00;  /* disables emitter */

   while (!(TFLG2 & BIT4))
   {
     if (TFLG2 & BIT5)
       {
         counter=counter+1;
         TFLG2 = BIT5; /* clears overflow flag */
       }
   }
   PACTL = PACTL & INV6; /* disable pulse accumulator */
   TFLG2 = BIT4; /* clears timer */
   write_int(counter);
   put_char(9);
   write_int(PACNT);

   for(i=1; i < 20000; i++);
   }
}
```

Input Capture Code

```
/* Bill O'Connor */
/* 10/21/98 */
/* time-of-flight calibration for Ranos' sonar */

#include <me11.h>
#include <hc11.h>
#include <analog.h>
#include <vectors.h>
#include <serial.h>

#define OUTPUT_LATCH      *(unsigned char *)(0x7000)

#define MAX_CNT     0xFFFF

#define      BIT7   0x80   /* 10000000 */
#define BIT6  0x40   /* 01000000 */
#define BIT5  0x20   /* 00100000 */
#define      BIT4   0x10   /* 00010000 */
#define BIT2  0x04   /* 00000100 */
#define      BIT54  0x30   /* 00110000 */
#define INV6  0xCF   /* 10111111 */
#define INV5  0xDF   /* 11011111 */

int old1, current1, sonar1, flag, i;

void main()
{
  init_serial();
  TCTL2 = TCTL2 & BIT4; /* capture on rising edge */
  write("data");
  while(1)
    {
      flag = 0;
      old1 = TIC1;
      OUTPUT_LATCH = 0x0f; /* enables emitters */
      for (i=1; i<60; i++);
      OUTPUT_LATCH = 0x00; /* disables emitters after 1ms */

      while (!(TFLG1 & BIT2))
        {
          write(" while loop");
          if (TIC1 == MAX_CNT)
            flag = 1;
          write_int(TIC1);
          if ((current1 > old1) && (flag == 1))
            break;
        }

      current1 = TIC1;
      TFLG1 = BIT2;
      if (current1 > old1)
        sonar1 = current1 - old1;
        else sonar1 = MAX_CNT - old1 + current1;
      write_int(sonar1);
      write_int(flag);

      for (i=1; i<20000; i++);
    }
}
```