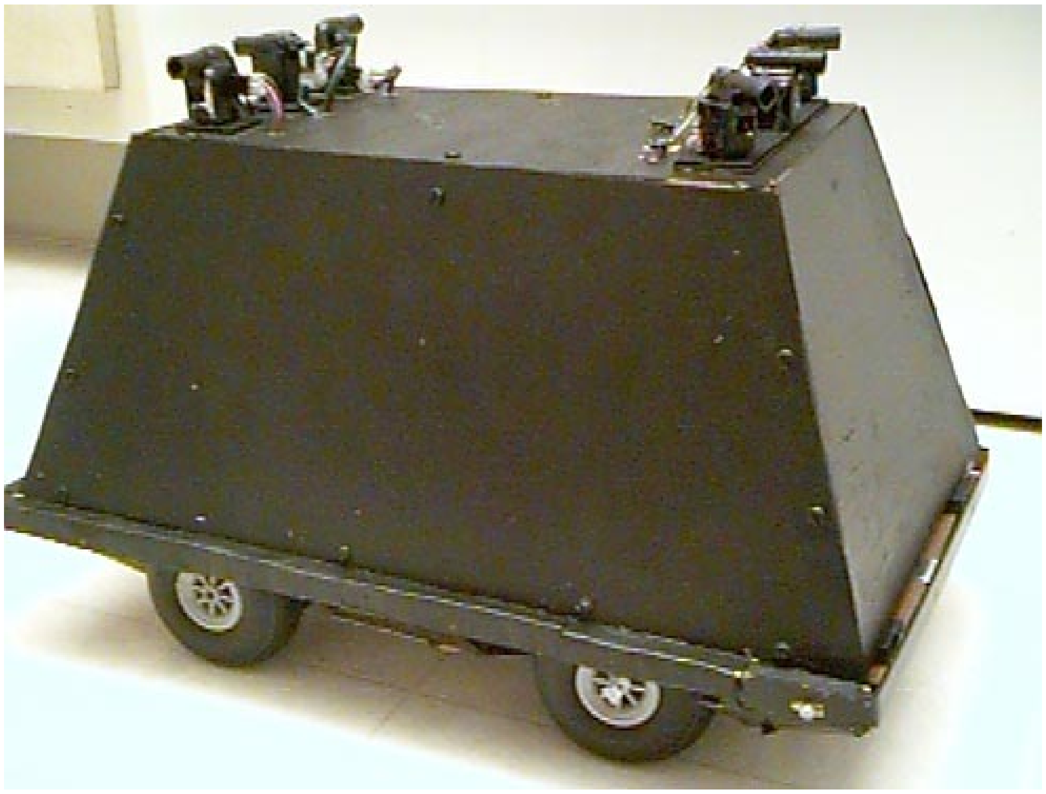
MSE-6 'Mouse' Droid

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



Date: April 28, 1997
Name: Michael Fiyak
Instructor: Keith L. Doty

TABLE OF CONTENTS

ABSTRACT

The robot was designed to look and act like the 'Mouse' Droid from *Star Wars* and be able to go from one location from a map to another location by using the information from the map and information from its environment. I made the body out of aluminum and used RC car type steering to give it that 'Mouse' Droid look. The 'Mouse' has the following behaviors that help it accomplish its overall goal: obstacle avoidance, fear, map interpretation/navigation, and auto location correction. These behaviors are controlled by the internal code that gets information from the outside world using the following sensors: bump, IR, Wookie Sensor, and a battery voltage detector. The droid also contains an internal map of its current location and the proper corner coordinates to be able to navigate through the map by the shortest distance. Timing and the servo angle maintain the droid's internal location and heading, but this did not perform very well.

The 'Mouse' droid performed many of its original design parameters admirably, but because of the great amount of error within its location calculations it was not able to find its destination consistently. The map interpretation/navigation behavior and the auto location correction behavior worked very well in trying to overcome this error but could only increase the performance to about 40% effective. Though the robot did not meet its pre-designed objectives, overall, this project was a success.

Executive Summary

Since this is my first robot I wanted to make it uncomplicated and fun, so I chose to replicate the MSE-6 'Mouse' Droid from *Star Wars*. To add originality to the design I added Map Interpretation and Navigation behavior that would ideally navigate from one location to another, given the layout.

After the simple task of making the robot look like the Star Wars robot was finished, I began the second, and most important, task of getting the robot to be able to Interpret and Navigate through maps while correcting its own error using correction algorithms. These correction algorithms are categorized under the Auto Location Correction behavior.

The behaviors that where implemented are: obstacle avoidance, fear of load noises, map interpretation, and auto location correction. The obstacle avoidance uses simple IF/THEN else statements along with a subsumption architecture to avoid obstacles consistently. The fear of load noises behavior made the droid scurry away when it detects a loud noise. The map interpretation and auto location correction both worked fine in there own respects, but together they where not able to overcome the error from the location maintenance design. This design was intentionally designed to contain error thus needing a stronger dependence on the Auto Location Correction behavior to overcome this error. Unfortunately, the design location maintenance using timing, not practical things like shaft encoders, was so error prone the robot could not always overcome the errors.

INTRODUCTION

The overall goal of this design is two parts. The robot was designed to look and act like the 'Mouse' Droid from Star Wars. The robot was also designed to be able to go from one location from a map to another location by using the information from the map and the information from its environment.

To make the task more difficult on myself and for experimental purposes, I also imposed that I use a sub-standard method of maintaining the robots internal coordinate system and thus forcing the software and behaviors to be more adaptive and adjust for error.

Unfortunately, after developing all of the error correction software I could (I ran out of space because I was using IC.). I could not properly overcome the error from my sub-standard method of coordinate maintenance. The maintenance system used the robots internal timing and the servo angles to determine and maintain the robot's coordinate system. Obviously, this method is not ideal and has a lot of error overtime. One of the biggest errors that I could not compensate for was one that I never even thought of before I designed it. My geometry calculations used the headings, time changes, and turning radii to determine its heading. Unfortunately, the radii are not consistent because from the robot's perspective the actual change in the servo heading happens very slowly and thus heading calculation are overshot which then ruins the coordinates.

Even though I was unable to overcome this error and I was not able to make the robot reliably find its final destination (worked about 40% of the time), I did achieve a lot of useful algorithms for map manipulation and error correction. These algorithms used on a robot specifically designed for the task will work great. I will mostly like pursue making a map manipulation/navigation/making robot next semester in EEL5840.

INTEGRATED SYSTEM

Hardware

The MSE-6 'Mouse' Droid uses a Motorola 68HC11E9 EVBU board coupled with a ME11 daughter board by Novasoft. This board controls the motors, servos, and all of the various sensors on the robot.

Software

The code was used with the Interactive C, version 2.85, freeware program that was written by Randy Sargent. Also, the map interpretation was compiled for WIN32 system using the Cygnus WIN32 GNU C compiler. Besides my own software, my system used the lib_rw11.c library, servo.c, servo.icb, and the pcoderwl.s19 to operate. These files are usually found in the IC libraries.

System Overview

The overall goal of the robot of being able to navigate from one location to another from a given map while acting and looking like a 'Mouse' droid is achieved by using multiple behaviors together. These behaviors where designed so that they each had a precedence depending on the order in which they where started as a process. Since each of the behaviors can write to all of the global variables, the last behavior that writes to the variable before the arbitration process is the variable that is used in actuation. Ideally I would have like to been able to have sub-processes and their arbitration as a process under a larger arbitration, but with the current design of IC this was not possible.

MOBILE PLATFORM

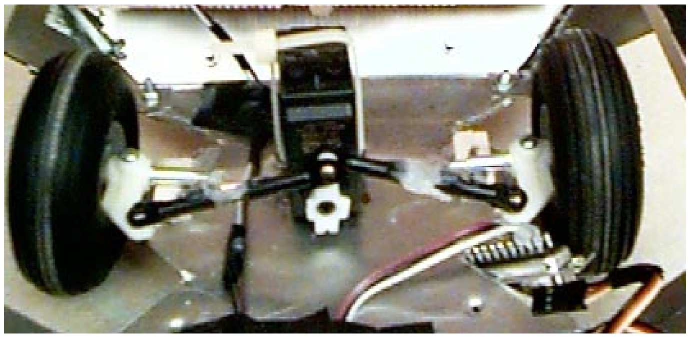
Since I needed to make my droid look the one from Star Wars I decided to make its body out of aluminum and have RC car type steering to give it that 'Mouse' Droid look. The body of the MSE-6 is constructed of aluminum and wood. Aluminum is very strong and light weight, so it is a great material to use. The only draw back is that it is expensive and not the easiest to work with. I kept in mind what the overall body needed to look like, the MSE-6 in Star Wars, and created a base platform for the board, the steering mechanism, the servo, and the motors. The rest of the body required some actual measurements and calculations to put together. I chose a wood rim on the inside to attach the aluminum to and control the shape. The wood rim also holds in place the bump sensor and it's associated piece of brass bumper. The very top surface, where the IR sensors are attached, is made of wood. Using Aluminum requires a lot of work but the originality made it rewarding.

I really like using the Aluminum as my material for my robot's body. I highly recommend it to others to use if they can spare the money and the time to work with it. The 'Mouse' Droid body is not very practical though. I had to keep the outside clear of wires and I could not add on any extras. Keeping control of the wires and electronics inside also became very tedious because of all the wires that needed to go from the top of the robot to the bottom half of the robot.

ACTUATION

Motors

For locomotion the MSE-6 'Mouse' droid uses two hacked MS410 Servos. These servos are converted to motors by removing the servo controller board and removing a stop tab to allow the servo to output shaft to rotate freely. These motors attach to 2.75" DU-BRO wheels. These motors are driven from a SN754410 motor driver attached to the MC68HC11 through the PA6, PA5, PD5, and PD4 pins. This motor driver was part of the ME11 board from NOVASOFT.



Steering Servo

The MSE-6 moves much like a Remote Control car. It has steering mechanics attached to the front DU-BRO wheels and a MS410 Servo attached to the mechanics. This way I can control the angle the 'Mouse' droid heads by setting an angle in the servo. This is controlled by the MC68HC11 through IC by using the *servo.icb* and *servo.c* files.

The steering servo ended up being an extreme pain for me. The servo would not accurately return to the same location every time thus my directions where not consistent and this damaged my overall control of the robot and its ability to know its own location. Using a car like steering base reduces the robot's moveability because it usually has a poorer turning radius then your typical two wheels and a caster design. This makes the programming more difficult because you have to program to back up and know which direction to back up.

Playback IC

I used an ISD1000A Voice Record/Playback IC to make the appropriate 'Mouse' Droid noises. The ISD uses Direct Analog Storage Technology (DAST) to write analog data directly into a single cell without A/D or D/A conversion. This allows for a greater, non-volatile storage capacity. The chip only required a couple of extra components to construct a working Recording and Playback device. This chip stores 20 seconds of sound sampled at 6.4kHz. I controlled the playback of two different sounds and the control of the chip by using some digital outputs mapped to address 0x6000 of my HC11 board. This way the noises where controlled in software by setting the correct address of the sound on the ISD1000A and strobing the chip to play.

The overall quality of this chip was pretty good. I found that it was easier to use a normal microphone over an electret microphone for recording. Also, the volume level from the internal amplifier on the ISD1000A was not high enough for practical robot uses without using a second amplifier.

SENSORS

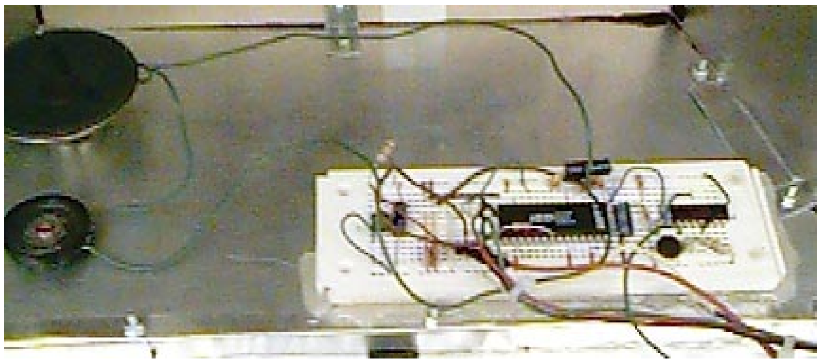
Bump Sensor

The bump sensor is simply two switches in parallel attached through a pull-down resistor to ground and +5V. Thus, when the switches are not pressed in, the output of the circuit is ground. When either one or both of the switches if pressed the output is +5 volts. The bump sensor output is sent into the analog(2) port (port E(2)).

IR Sensor

The Infrared Sensors use an IR LED to transmit a 40kHz modulated source. Then a SHARP GPIU58Y is used to detect the reflected light of object the LED shines upon. The LED's use the output current from the 7000 Address latch IC. Also, I attached a potentiometer to the left and the right LED's so I could adjust the level of the current into the LED's. The LED's are placed inside of a one inch piece of a Paper-Mate pen to focus the point of light from the LED so it's corresponding sensor only picks up the light emanating from its respective light source. Also, for convenience I placed the emitter/detector pairs onto a Velcro like material so I can easily move and adjust their locations. The three current sensors use the Analog(0), Analog(1), and Analog(3) input ports.

Wookie Sensor



The object of this sensor is to provide the MSE-6 Droid with a behavior of fear of loud noises. This "Wookie Sensor" does two basic things. First, the sensor detects loud noises. Secondly, the sensor plays a pre-recorded sound that is part of its fear behavior. The sound detection is accomplished with an amplified microphone attached to an analog port that is scanned for a threshold. An ISD1000A Playback/Record IC produces the pre-recorded sound. This IC can also be used to play other noises and is controlled by the MC68HC11 through a memory-mapped output. More details about the Wookie Sensor can be found in Appendix Section I.

Battery Voltage Detector

I constructed a battery voltage detector so I could use data from the voltage level to adjust some battery dependent variables in my robot. The battery Voltage Detector simply measured a scaled down version of the actual battery voltage through an analog port. I ended up not using this data for anything in particular because the variables I was going to use it for where instead adjusted through calibration software instead of using the voltage data.

BEHAVIORS

Obstacle Avoidance

This first behavior is necessary in any mobile, autonomous agent so that it may move about its environment without continuously crashing into other objects. Currently, my mobile agent uses 6 infrared emitters (IR Diodes) and 6 detectors for its eyes and a bump sensor for its touch. The code for this behavior is a set of IF/ELSE statements that compare the IR detector's value with a pre-determined threshold level and then arbitrate a reaction based on all three detectors for forward navigation. The three forward IR emitter/detector pairs point left, right, and forward. These IR Diodes are powered from the current coming from a latch and are collimated for greater accuracy and range. The rear mounted IRs are powered and collimated the same way as the front mounted ones but they are used differently. The center, rear mounted IR sensor is used so the robot does not back up into things. The left and the right IRs are used for wallfollowing. The bump sensor is only placed on the front and only uses a single bit. The MSE-6 has six directions that it will choose depending on the inputs from the IR's and the bump sensor: straight, right, left, gentle right, gentle left, and reverse right. The methods in which the directions are arbitrated give the 'Mouse' droid different turning tendencies depending on which wall needs to be followed. This method works surprisingly well because the outputs from the IR's are considerably high and thus enables a longer threshold point.

The high positioning of my IR was only useful in real hallways, not in the small little "walls" in the MIL lab. I will not place my IRs that low next time but will instead place them lower and closer to the middle to be able to avoid smaller objects. The overall performance of my IRs was very good but next time I think I will add more.

Fear

The Fear behavior is simple. Basically, when the MSE-6 detects a load noise it pauses and gives a small response, then turns around and runs off. This complicates things while trying to navigate a map, so was removed from the behaviors while the robot was navigating. This characteristic would be more effective if it could move quicker but changing the motors just for this behavior would seem counter productive. When the microphone was outside of the droid (on top) this fear behavior worked nicely, but caused the robot to get lost when it was navigating.

Map Interpretation and Navigation

The map interpretation and navigation works like this. The robot is introduced a map with a starting point, finishing point, and an initial heading. Then the robot uses a program internally to keep track of its location and its heading and attempts to go from point A to point B. This requires the robot figures out the shortest path and goes there.

The map is defined by a two dimensional plan consisting of 2ft by 2ft blocks. Each block is either label: open, close, obstacle, or door. This map is then formed into an array (ideally 2 dimensional) and loaded into the robot. Unfortunately, I did not have room to place this map into the robot and then do manipulations on the array needed for proper interpretation and navigation. My solution was to move the map manipulation part of the software onto a host computer that does the manipulation, by finding the shortest path of corners for the robots desired path and then giving this data and one copy of the map to the robot. This limited the manipulations that could be done to the map while on the robot and thus decreased its performance but was necessary in this case.

One of the coolest things that came out of this project was the map-making program my friend (Matthew Perkins) wrote for me to use with my robot. The program was written in JAVA and is located on the web at quicker but changing the motors just for this behavior would seem counter productive.

They're where a couple of problems with this behavior that made it perform lousy for my demo. First of all, I ran out of code space early on in the program so I could not increase the program to compensate for its problems. Secondly, the droid used internal timing and the servo angle only, to maintain its heading and location. The problem with this timing idea was that the servo angle did not change fast enough for this to be accurate overtime. Going straight, or with very few turns this method worked pretty well, but after multiple servo angle changes the heading would get off and thus the coordinates would get off. This second problem was expected but not to the degree that it became. This incorrect location/heading problem was the reason the forth behavior was designed. The map interpretation part actually worked very well but was overshadowed in its performance by the navigational problems. When the navigation did not get off, the robot would find its destination location, but this only happened about 40% of the time.

I think this idea is a really great one and I will most likely continue it further on a new robot with ICC. I feel that a new design can both create its own map while exploring and use its map to navigate. Timing is not an appropriate way to maintain the robots internal heading and location and does not need to be tried again.

Auto Location Correction

Because of voltage changes in the motors and servo, the internal coordinates of the 'Mouse' droid and its actual coordinates will eventually get askew. Since there exists a model of the location in the memory of the 'Mouse' droid it can utilize it to correct its coordinates. The MSE-6 will follow along a wall waiting for its next location to turn. When it finds this location it takes note of the coordinate that corner is located. This corner (actually slightly off of the corner in the appropriate direction) will then be placed as the droid's current location. Also, as the 'Mouse' droid aligns with the wall for wallfollowing the heading of the wall will replace the current heading. This helps with the navigation a lot. So, much that with out the robot wouldn't have a chance of reaching its location. This was designed so that basically to travel to a corner all you need to do is get the droid to head toward the wall before the corner at an angle. The angle can actually be pretty poor and because of a pre-established side the robot should be following, the robot will tend to wallfollow toward the correct corner.

This is a different way of approaching the navigation and is what probably intrigues me most about this project. It is sort of working the same problem from both sides. First you attempt to know your location and heading the normal way by using electronic devices like shaft encoders. Then you attempt to maintain the location and heading and correct for errors from the normal devices by using more abstract methods by defining walls and corners and types of objects. Thus, the error in the electronic devices will not overcome the robot and its program.

PERFORMANCE

Mechanical

The MSE-6 'Mouse' Droid looks cool, but this is not a very good body design for a robot. I'm sure the people at Lucas Arts did not design it for the behaviors I tried to impose on it. But this is not an actual complaint because I expected as much. I chose this know it would have limitations but since real world designs for other people sometimes impose things similar, I chose to do it this way anyway. Some of the problems with the design are here. First, the IRs where mounted to high leaving the middle of the robot open for objects to collide into. Second, the body was too expensive to construct for the final usage. Thirdly, car like steering is not ideal for a truly mobile robot. Though covered robots can in some instances be a good behavior in this case you do not get the benefits, just the losses.

The motors worked relatively fine but where a bit noisy for me. The steering servo and steering mechanics did not perform to my satisfaction. The servo did not return to the same location for a give signal accurately. Also, the wobbly steering mechanics where did not perform accurately enough. Later in the semester, when I needed it, I vastly improved the steering mechanics as much as I could. This helped a lot but still was not perfect. My main problem was finding the right parts. Since I do not have access to any type of shop to build my own parts I had to use the ones I could fine. The ones I eventually found where actually pretty good but not designed for such a small wheel distance or to properly hold the tires I used.

Electrical

The electrical system of the 'Mouse' Droid worked very well. The system did not have many problems as a stable system. All of the components worked and worked reliably. There was some electrical noise in the noise detection circuitry but this was overcome by desensitizing the Mic in software.

The bump sensor and the IR worked fine though I was getting mass amounts of leakage from the IR LED's into the detector cans. I didn't realize I could fix this problem until after I already moved around it in design. I could have gotten better range with a better IR setup but the one I used worked fine.

The Wookie Sensor worked by definition but was not ideal for my use. My design had flaws in it because I did not use anyone else's help and just started making my own (not even a book) thus attempting to make it completely original in design but not as an idea. I found the performance of the Wookie Sensor was best in a small room and performed poorly in large rooms. Specifically just the noise detection had problems; the playback circuit performed fine. I guess I designed it to only work best for Wookies.

Behavioral

The behaviors worked very well. As individuals they worked fine. Together they worked really well also. The only thing I would call a problem is something I never actually decided if it was a behavior or not: the internal location maintaining software. This was part of the Map Interpretation and Navigation behavior (maybe I should have made that into the many sub-behaviors it seems to be) that attempted to keep an internal direction and location of the 'Mouse' Droid in the program. Because of the steering errors this location maintenance tended to loose the correct location and heading over time. The correcting behaviors could not always compensate well enough to avoid from messing up the whole system.

My simple obstacle avoidance worked very well. They only times it got trapped where when it got stuck on something or when it would run into something above the bump sensor and below the IRs, or if it was black. The randomness in the reverse helped it not get suck in any boxed in areas.

The fear behavior it self worked great because it was really simple. It only had any problems when the actually Wookie Sensor could not properly detect a noise but the code worked fine (it was simple). I deactivated part of this fear behavior, the running away part, when I used it to navigate through a map. I did this for the obvious reason that if the robot ran away it made it a lot less likely to find its destination because it would probably loose its location.

The Map Interpretation and Navigation behavior worked great. I tested the map manipulation/interpretation part of the program on my PC using a lot of feedback to see how well my logic worked and the result where very good. The logic in the robot itself relating to the map interpretation and navigation was good enough that it performed its part very well also. I will not say as well because feedback while using IC was not as easy and thus I could not determine exactly how well it worked.

The Auto Location Correction behavior worked great. It did a lot to enhance the overall performance with relatively little code. I fell if I had more space I could have added more features and thus improved the overall performance.

Overall

The overall performance was bad. But, the performance was not only bad; it was frustrating, because sometimes it would work perfectly. The performance was bad because it did not work when everyone was watching. ☺ After removing most/all of the errors in my logic, I tested the 'Mouse' Droid many times. About 40% of the time the 'Mouse' Droid could go to the destination area (within a couple feet). But 60% of the time I got strangely erratic behavior or the heading simply got far enough out of whack to send the Droid the wrong way. Mostly, it seemed to be a heading problem. One of my ideas for this erratic behavior is discussed in the next section: The Big Gripe. Because I used the term bad for the overall performance of my robot, do not think I was really happy. I was a bit frustrated that I had no way of increasing the robots performance near the end of the project, but the ideas and lessons I came away with where great. I think the biggest problem in this overall design was my confidence that I could some how get this silly idea to work even with its design weakness. With more space for code I may have, but that is not important because I know I actually achieved a lot of things.

As a project, the 'Mouse' was awesome. I learned so many do's and don'ts that I can not wait until I can start working on my next couple of robots (in a week!). I have also thought of so many ideas that I am dying to test out and implement on a robot.

These ideas range in scale from macro ideas of the evolution of the robot, to micro ideas of ideal platforms.

The Big Gripe

This gripe is not aimed at anyone it is just something I came across at the wrong time. I had assumed foolish me, that I could fit at least about 24k of code into IC. So I stuck to IC because I thought I would never use that much. Unfortunately, I found out the hard way that I could only fit about 12.5k of code into IC. Of course this happened right after I wrote about 25k for it. But, I did find an O.K. solution. I took the unnecessary programming off of the robot and only included the necessary parts. I also reduced the amount of arrays the robot would have to have on board. But, after I did all of this and wrote the minimal amount of code I could write to do the job, I had 12.5k. Thus, I was stuck with this amount of code and could not pursue anymore algorithms and it was too late to switch to IC (this is where my fault comes in). I also came to believe that IC does not properly handle the internal memory and that once you approach this max you increasingly risk over-writing your own code as the robot is running. Most likely, the stack is rising into the code. I think this because of a couple of reasons. First, when you load about 12.8k you don't get any error message it just never completes loading. Secondly, sometimes you would get incredibly erratic behavior like something just turned off. Thirdly, sometimes the robot would just freeze and nothing would work. Fourthly, sometimes you would restart the robot and it would go left when normally it would go right without any difference in its initial start up that could cause this and this usually only occurs after the robot has frozen up.

So, those of you who are going to actually write some code for your robot, start with ICC! I sure wish I would have.

CONCLUSION

In conclusion this project was a blast. This blast was mostly all mental, because the work required sometimes became tedious. But, mentally all the physical work was all worth it. The robot did not work, but this did not really bother me since I knew why it did not work and I could not fix the problem with out completely changing my original design. I suppose some may say I should have allocated enough time incase this happen, but because I know I worked really hard on it I do not fell I could have allocated much more time the project. **(Warning to all UF EE's. Do NOT take both this course and a course by Dr. Lynch (EEL 4713, etc) in the same semester unless you are masochistic.)**

The true gem of this project was the many things I learned and the ideas I developed. I plan on using some of the map manipulation and navigation and the correcting algorithms in one of my next robots (EEL 5840). I now know that I better not use some platform I saw in a movie, for a job that needs a more specific platform. I'm also excited to be able to attempt to use all of the space in the 64k board with a real C compiler instead of an interpreter. I would really like to pursue improving the performance of the existing platforms and sensors with software instead of making new sensors for robots that don't do anything or making a body that does not have any code that makes the body design useful.

DOCUMENTATION

Reid Harrison, *Robot Programming*, UF Intelligent Machines Design Lab, Gainesville, FL, 1994.

Erik de la Iglesia, *Sharp IR Sensor Hack for Analog Distance Measurement*, UF Intelligent Machines Design Lab, Gainesville, FL, 1996.

APPENDICES

I. Wookie Sensor

Sensor Description

Basic Design

The Wookie Sensor was designed to be cheap and simple to put together. The sensor is composed of two parts: the noise sensor and the noise player. The noise sensor uses an analog port on the MC68HC11 to detect noises. The noise player is controlled by the MC68HC11 through an address port at $6000.

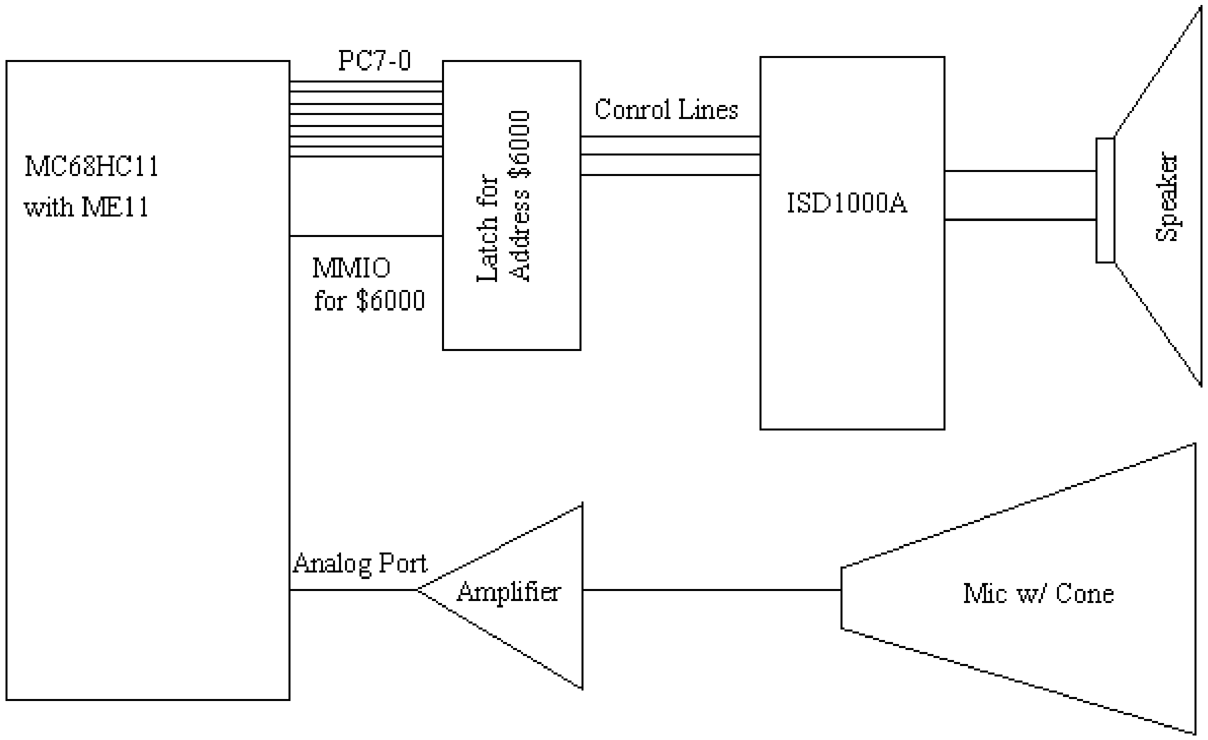


Figure 1: **Basic Wookie Sensor Design**

Microphone Amplifier

The noise detection part of the "Wookie Sensor" was made from a microphone and an amplifier. I tried about three different amplifier designs. The one that worked the best was also the simplest. The amplifier consists of a 324 op-amp, some resistors, and the microphone. The 324 was chosen because of its single power supply and its low cost. The chip also contains four op-amps and can thus be used for other circuits or more amplification. The amplifier was set-up with non-inverting amplification and 1000 gain. No load resistor was used for this amplifier. I found the circuit worked best without any biasing or coupling capacitors. Also, the input was not biased for full voltage swing and instead clips the input and output.

This design does not maintain the original signal quality and can not be used for playback purposes. The output voltage swings from 0 volts to about 4 volts. The microphone range was not very good with just using the microphone so I added a paper cone to the microphone to focus the incoming noises. This greatly enhanced the range of the noise detection but also made the noise detection direction dependent.

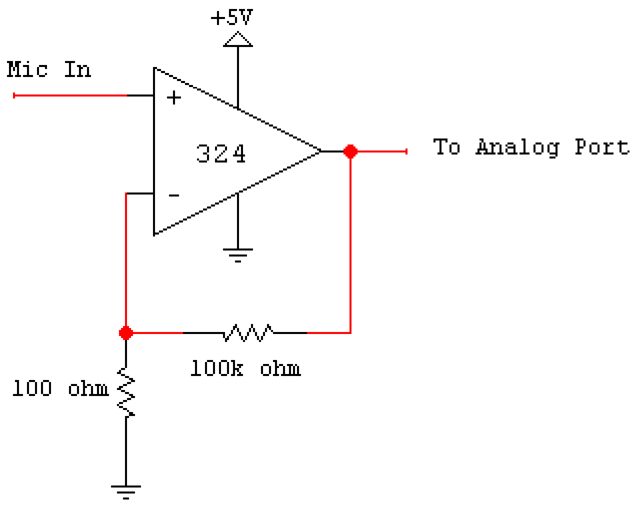


Figure 2: Microphone Amplifier for Noise Detection

Sound Player

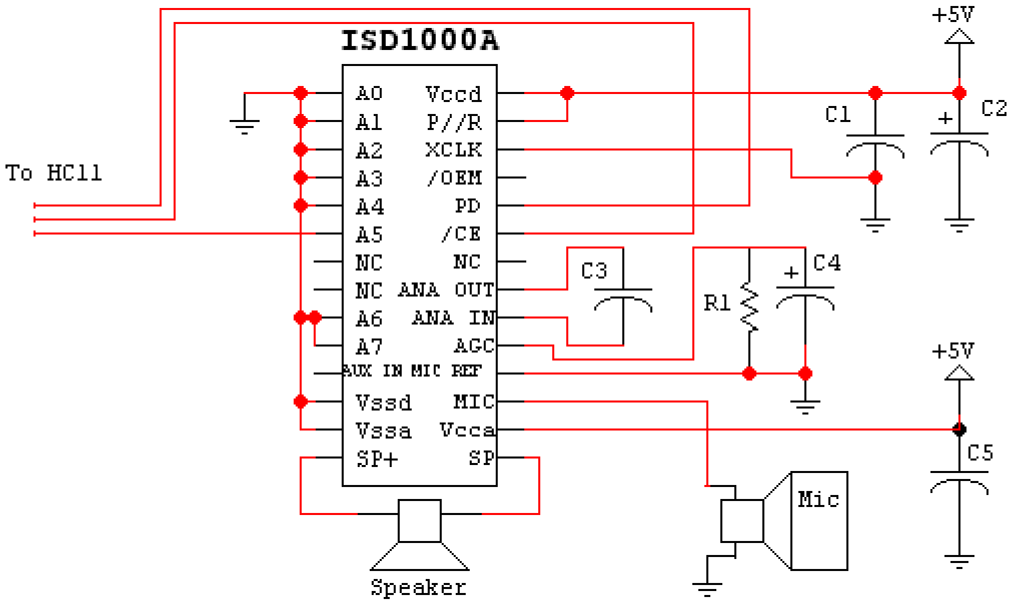
The sound playback part of the sensor was simply an ISD1000A Record/Playback IC controlled by the MC68HC11. This was the most costly part of the sensor since the IC alone cost $17. This IC was chosen because of its storage capacity for its cost and size. Also, this chip did not require too many external components and did not need power to maintain sounds stored in memory. I set up this circuit to playback pre-recorded sounds through the MC68HC11. Different sounds can be recorded to and played from different memory addressing in the ISD1000A by using it memory address lines. Since I only wanted to have two different sounds I divided the memory into two sections and used the 6th bit of the address to control, which of the two sounds are played back. The rest of the capacitors and resistors in the circuit where used for noise reduction. The speaker used is actually two 8-ohm speakers in series since the ISD1000A requires a 16-ohm load.

Three lines from the MC68HC11 control the playback. The first line controls which sound to play. The second line strobes high once to begin playback. The third line is used to reset the IC to ensure the sounds begin playing from the beginning. I found that this IC works really well but required very loud input sounds for it to playback at an appropriate level. Note the microphone in the circuit below is not used for playback but is used when recording new sounds into the IC.

Figure 2: Hardware Interface for Compass Electronics

Experimental Procedure

I don't think Experiment Procedure is a proper title for this section since what is contained is just a couple of tests with very little scientific measurements. The problem with measuring the results is that I did not have any type of device to measure the actual noise levels used in these tests so vague definitions are used instead. The tests compare the different values obtained by the noise sensor under different conditions. The three conditions that where tested where: microphone amplifier without cone, with testsnd.c, microphone amplifier with cone, with testsnd.c, and microphone amplifier with cone running complete MSE-6 code.



Results

The Microphone without the cone gave very poor results and thus led me to use the cone. The test below was done to show why the cone was needed. Without the cone the microphone could not detect even a loud clap beyond a foot of the microphone. The microphone could detect some loud low frequency sounds at distances greater then a foot but this was not very practical.

Table 1: Microphone Amplifier without Cone running testsnd.c

Analog Reading	Noise
0-1	No Noise
1-2	Motors On
195	Loud Clap within 1 foot
195	Loud, Deep Noise (Bass) within a yard
Not Detected	Any Loud Clap greater then 1 foot

The Microphone Amplifier with the cone gave the best results when running the testsnd.c code of all three tests. The noise sensor could detect loud claps at a surprisingly long distance away. It could also tell different levels of noises at closer distances. This method had a very low reading when there was no noise and wasn't effected much by the motor noises. Running testsnd.c gave this set-up an advantage because the refresh rate of reading the analog port was much faster then with the mse6101.c program. This gave it a better chance of capturing the peek of a noise and thus triggering more accurately.

Table 2: Microphone Amplifier with Cone running testsnd.c

Analog Reading	Noise
0-1	No Noise
2	Motors On, no other noise
6-195	Various, low frequency noises within a yard
195	Loud Clap within 4 yards inside a small room.
Not Detected	Any Loud Clap greater then 2 yard in a large room,

	or greater then 4 yards in a small room.

Once I attached my sensor to the actual robot and embedded the controlling software into my previous obstacle avoidance software, I noticed a peculiar noise develop in the noise detector. The detector worked much like the previous test except when the noise detector was on, it usually output a signal around 70-80 on the analog port. This noise made me have to change the threshold level of the noise detection to above that it made the detector less sensitive to lower noises. I think this noise is coming from my IR detector system, but since I adjusted the software to still work with this noise, I did not bother investigating. Also, the detector usually requires multiple claps to trigger the noise detector because of the different refresh rate on the analog port.

Table 3: Microphone Amplifier with Cone running mse6101.c

Analog Reading	Noise
70-80	No Noise
70-80	Motors On, no other noise
70-80	Various, low frequency noises within a yard
195	Loud Clap within 4 yards inside a small room.
Not Detected	Any Loud Clap greater then 2 yard in a large room, or greater then 4 yards in a small room.

Conclusion

The Wookie Sensor does what it was designed to do. Granted, this is not the most elegant designs and could be vastly improved on, but it was simple and it does the job. The Wookie Sensor was relatively cheap costing around $25 to build. The design is also simple to put together and control from the MC68HC11. My particular design only required three output pins and one analog port to control. The noise problem may only be present in my particular robot and the Wookie Sensor may work better in other designs. A problem I see that could develop is the rate at which the analog port is read. If there was a large amount of code with time consuming processes, the analog port would not be read very frequently and would greatly reduce the chance of noise detection. But if the noise detector was connected to an interrupt then the Wookie Sensor would be much more accurate. My only real complaint is that the cone on the microphone is not very convenient and occupies a lot of space inside of my droid.

II. *The MSE-6 'Mouse' Droid Code*

Hope know one is ever forced to read this. ☺

mse6101.c

```
/**********************************************************************
*******/
/**********************************************************************
*******/
/**
**/
/**   MSE6101.C (version 0.99) for MSE-6 'Mouse' Droid  Last Mod:
04/25/97   **/
/**   CODE/LOGIC: Michael J. Fiyak        EE at University of Florida
**/
/**               hess@atlantic.net       (352) 377-7725
**/
/**
**/
/**   FILES NEEDED TO RUN: servo.icb, servo.c, map.c, cal.c
**/
/**
**/
/**   A MAP.C file can be created by using Matthew Perkin's JAVA map
maker   **/
/**   at http://www.cise.ufl.edu/~msperkin/javar/droidmap.html
**/
/**   and then use the Makemap.exe program to create the map.c file or
use   **/
/**   makemap.c to compile your own converter program.
**/
/**
**/
/**********************************************************************
*******/
/**********************************************************************
*******/


/**********************************************************************
***/
/****************************** GLOBAL VARIABLE
***************************/
/**********************************************************************
***/

/* Switch Variables, to turn on and off certain things */
int cal = 0;          /* If True (1) then the robot will Calibrate
itself  */
int sound = 1;        /* If True (1) then the robot will have noise
module */

/* Just plain variables :) */
int bump, center_eye, right_eye, left_eye;
int tap = 0;
int left_side, back_eye, right_side;
int ran;                  /* for a random number */
int thresh = 122;
int cthresh = 120;
int x_cell;
int y_cell;
int bumphit = 0;
int revhit = 0;
int backhit = 0;
int error = 0;

/* Sound Variables */
int mic, max;
int micthresh = 122;
int scared = 0;
int soundc = 0;
int sv = 2000;     /* how often robot makes noise, larger is less often
*/

/* Corner Recallibration Variables */
int r_s_thresh_c = 129;
int r_s_thresh_f = 129;
int l_s_thresh_c = 129;
int l_s_thresh_f = 129;
int old_left_side;
int old_right_side;
int delta_right = 0;
int delta_left = 0;
int del_rig_thresh = 5;
int del_lef_thresh = 5;
int wallfollowing = 0;
int close_to_corner = 0;
int countc = 0;

/* Maping Variables */
float corner_heading;
float heading_offset = 0.174533;  /* 10 degrees */
float fheading_offset = 0.034907; /* 2 degrees */
int current;
float tester;

/* Direction Variables */
float offset = -4.5;
float angle, direction;
float straight = 98.0 + offset;
float turn = 38.5;
float gturn = 22.0;
float left = straight + turn;
float right = straight - turn;
float gentle_left = straight + gturn;
float gentle_right = straight - gturn + 2.0;
float slight_left = straight + 1.0;
float slight_right = straight - 12.0;
float forward = 100.0;
float backward = -100.0;
float stopp = 0.0;
float current_side;
float opp_side;

/* Direction Control Variables */
/* float x_coor = 0.0; */ /* in inches */
/* float y_coor= 0.0; */
/* float heading = 0.0; */ /* in radians */
float old_angle = straight;
float old_direction = forward;
int time = (int) mseconds();
int delta_t = 0;
int max_t = 32767;
/*
float dist_straight = (87.0/9000.);
float right_deg_time = (6.2832/13800.0);
float left_deg_time = (6.2832/15900.0);
float gright_deg_time = (6.2832/23100.0);
float gleft_deg_time = (6.2832/22800.0);  */
/* dieing batteries
float dist_straight = 0.008491;
float right_deg_time = 0.000499;
float left_deg_time = 0.000486;
float gright_deg_time = 0.000253;
float gleft_deg_time = 0.000347;
*/
float dist_straight = 0.008979;
float right_deg_time = 0.000497;
float left_deg_time = 0.000302;
float gright_deg_time = 0.000261;
float gleft_deg_time = 0.000224;

float rad_right = 14.0;
float rad_left = 21.75;
float rad_gright = 34.0;
float rad_gleft = 32.0;
float delta_xp;
float delta_yp;
int servo_latency = 200;
float deg_head = 0.0;


/************************************************************************
***/
/** MAIN:  Main itializes the robot and then starts all of the processes
**/
/** and then exits.  The processes are set up so that only the behavior
**/
/** arbitrate controlls the actual motors and the steering servo.  The
**/
/** other processes that deal with the motors, give opinions of what the
**/
/** direction and angle of the robot should be. The modules are set up
in**/
/** such a way that each module can step on the previous modules opinion
**/
/** and thus each module has a degree of opinion based on the order of
**/
/** the process execution.  Thus the bump sensor opinion will overide
**/
/** any of the above processes opinions (obstacle avoidance, map_interp
**/
/** and etc.).
**/
/************************************************************************
***/

void main()
{
  beep();
  init();
  start_process(sensor_module());
  start_process(update_module());
  start_process(map_interp_module());
  start_process(side_module());
  start_process(finish_seek());
  if (sound == 1){start_process(noise_module());}
  start_process(obstacle_avoidance());
  start_process(bump_sense());
  start_process(back_check());
  start_process(behavior_arbitrate());
}

/**************************** ROBOT INITILIZATION
*************************/

void init()
{
  servo_on();                   /* Turns on Servos.  Duh...      */
  wait(500);
  poke(0x7000,0b11100000);      /* Sets Latch for FRONT IR LEDs  */
  poke(0x6000,0b00000010);      /* Sets Latch for Sound Controll */
  angle = straight;             /* Why not start off going straight? */
  direction = forward;          /* and forward                      */
  current = 0;
  side_set();
  if (cal == 1)                 /* Will Calibrate Timing Variables if Cal
is True */
  {
          calibrate_time();
          dist_straight = dis_strc;
          right_deg_time = ri_deg_tc;
          left_deg_time = le_deg_tc;
          gright_deg_time = gr_deg_tc;
          gleft_deg_time = gl_deg_tc;
  }
  time = (int) mseconds();       /* Starts initial system timing */

}

/*********************************************************************
***/
/** SENSOR_MODULE:  Updates the front and rear sensors at 5 Hz.
**/
/** These updates are alternated to conserve power.
**/
/*********************************************************************
***/

void sensor_module() /* Read sensors into globals at 5 Hz */
{
  while(1)  {                      /* Front IR’s are on for 100msecs
*/
      bump       = analog(2);   /* then Rear IR’s are on for 100msecs.
*/
      poke(0x7000,0b11100000);
      wait(100);
      center_eye = analog(0);
      right_eye  = analog(1);
      left_eye   = analog(3);
      poke(0x7000, 0b00011111);
      wait(100);
      left_side  = analog(6);
      back_eye   = analog(4);
      right_side = analog(5);
  }
}

/*********************************************************************
***/
/** UPDATE_MODULE:  Used to update different variables that are needed
**/
```

```
/** for the program.
**/
/*********************************************************************
***/

void update_module()
{
  while(1) {
       update();           /* Updates current cells */
       rads_to_deg();      /* Converts Radian Heading to Degrees */
       find_c_heading();  /* Finds the heading to the corner */
       defer();
       if(current > 2){tap = 1;}
  }
}

/*********************************************************************
***/
/** MAP_INTERP_MODULE:  Takes the data from the where to go array and
**/
/** decides where the droid should be heading based on where it is.
**/
/*********************************************************************
***/

void map_interp_module()
{
  while(1)
  {
      angle = straight;
      if((corner_heading - heading) > heading_offset)
      {
            if((corner_heading - heading) < 3.1416)
            {
                  angle = right;
            }
            else
            {
                  angle = left;
            }
      }
      else if((heading - corner_heading) > heading_offset)
      {
            if((heading - corner_heading) < 3.1416)
            {
                  angle = left;
            }
            else
            {
                  angle = right;
            }
      }
  else{angle = current_side;}
  /* wait(1500); */
  }
}

/*********************************************************************
***/
/** SIDE_MODULE:  The robot while have a tendency for a particular side
**/
/** determined by the map_interp and thus while follow along a wall on
**/
```

```
/** the given side.  Sets robot into wallfollowing. This module also
**/
/** detects the corner when it is reached and then has the location set.
**/
/*********************************************************************
***/

void side_module()
{
  while(1) {

      if((right_side > r_s_thresh_f) && (current_side == right))
      {
            wallfollowing = 1;
      }
      if((left_side > l_s_thresh_f) && (current_side == left))
      {
            wallfollowing = 1;
      }
      if(wallfollowing)
      {
            if(current_side == right)
            {
                        if(countc == 50){heading_update();}

                        delta_right = old_right_side - right_side;

                        if((x_cell < (map_x[current] + 4)) && (x_cell >
(map_x[current] - 4)))
                        {
                                    if((y_cell < (map_y[current] + 4))
&& (y_cell > (map_y[current] - 4)))
                                    {
                                           close_to_corner = 1;
                                    }
                                    else{close_to_corner = 0;}
                        }
                        else{close_to_corner = 0;}

                        if((delta_right > del_rig_thresh) && (countc >
50))
                        {
                              next_corner_setup();
                             /*   tap = 1;  */
                        }
                        if(right_side > r_s_thresh_c)
                        {
                              angle = slight_left;
                        }
                        else if(right_side < r_s_thresh_f)
                        {
                              angle = slight_right;
                        }
                        else {angle = straight;}
                        old_right_side = right_side;
                        ++countc;
            }
            else
            {
                  if(countc == 50){heading_update();}
                  delta_left = old_left_side - left_side;
                  if((x_cell < (map_x[current] + 4)) && (x_cell >
(map_x[current] - 4)))
                  {
                          if((y_cell < (map_y[current] + 4)) && (y_cell
> (map_y[current] - 4)))
                          {
                          close_to_corner = 1;
                          }
                          else{close_to_corner = 0;}
                  }
                  else{close_to_corner = 0;}
                  if((delta_left > del_rig_thresh) && (countc > 50))
                  {
                        next_corner_setup();
                        /*  tap = 1; */
                  }
                  if(left_side > l_s_thresh_c)
                  {
                          angle = slight_right;
                  }
                  else if(left_side < l_s_thresh_f)
                  {
                          angle = slight_left;
                  }
                  else{angle = straight;}
                  old_left_side = left_side;
                  ++countc;
            }
      }
      else
      {
            countc = 0;
      }
      defer();
  }
}

/**********************************************************************
***/
/** FINISH_SEEK:  If can reach the end will start heading to the end.
**/
/** When it reaches the end the robot will stop.
**/
/**********************************************************************
***/

void finish_seek()
{
  while(1) {
      if((map_x[current] == x_finish) && (map_y[current] == y_finish))
      {
            angle = straight;
            if((corner_heading - heading) > fheading_offset)
            {
                  if((corner_heading - heading) < 3.1416)
                  {
                        angle = right;
                  }
                  else
                  {
                        angle = left;
                  }
            }
            else if((heading - corner_heading) > fheading_offset)
            {
                  if((heading - corner_heading) < 3.1416)
                  {
                        angle = left;
                  }
                  else
                  {
                        angle = right;
                  }
            }

            /*  tap = 1;  */
            beep();
            beep();
            play(2);

      }
      if((walkable(map, x_cell, y_cell, x_finish, y_finish)) > 0)
      {
            if((map_x[current] != x_finish) || (map_y[current] !=
y_finish))
            {
                  ++current;
            }
      }
      if((y_cell) > (y_finish - 2))
      {
            if(y_cell < (y_finish + 2))
            {
                  tap = 1;
                  direction = stopp;
                  beep();
                  beep();
                  beep();
            }
      }
      if((x_cell == x_finish) && (y_cell == y_finish))
      {
            tap = 1;
            play(1);
      }
  }
}

/************************************************************************
***/
/** NOISE_MODULE:  This controls the robots fear behavior by detecting
**/
/** a noise and then playing a retort and turning around and running
**/
/** away.  This also plays a 'Mouse' Droid noise at a given interval so
**/
/** people can no that it is coming.
**/
/************************************************************************
***/

void noise_module()
{
  while(1) {
            mic = analog(7);
            if(mic > max){max = mic;}
            if(soundc == sv)
            {
                  play(1);
                  soundc = 0;
            }
            else{++soundc;}
            if(max > micthresh)
            {
                play(2);
                max = 0;
                scared = 1;
            }
              if (scared == 1) {           /* This is what the robot
does when */
                        direction = stopp; */
                        wait(500);         */
                        if(current_side == right)
                        {
                                angle = left;
                        }
                        else{angle = right;}
                        direction = backward;
                        wait(2500);
                        angle = straight;
                        direction = forward;
                        wait(2000);
                        scared = 0;
                 }
  }
}

/***********************************************************************
***/
/** OBSTACLE_AVOIDANCE:  This controls the basic obstacle avoidance
**/
/** using simple if/then/else statements.  Based on the readings of the
**/
/** front 'Eyes', the statements change the motor direction and steering
**/
/** servo angle accordingly.
**/
/***********************************************************************
***/

void obstacle_avoidance()
{
  while(1) {
      if (left_eye < thresh && center_eye < thresh && right_eye >
thresh) {
        angle = gentle_left;
        direction = forward;
      }
      else
        if (left_eye > thresh && center_eye < cthresh && right_eye <
thresh) {
            angle = gentle_right;
            direction = forward;
        }
      else
        if (left_eye < thresh && center_eye > cthresh && right_eye <
thresh) {
            angle =  left;
            direction = forward;
        }
      else
        if (left_eye > thresh && center_eye > cthresh && right_eye >
thresh) {
                  bumphit = 1;
        }
      else
        if (left_eye > thresh && center_eye > cthresh && right_eye <
thresh) {
            angle =  right;
            direction = forward;
        }
      else
        if (left_eye < thresh && center_eye > cthresh && right_eye >
thresh) {
            angle =  left;
            direction = forward;
        }
      else {
                    /*     angle = straight;  */
            direction = forward;
      }
      if(revhit > 0){
              direction = backward;
              angle = current_side;
              ++revhit;
              if(revhit > 175)
              {
                    revhit = 0;
              }
        }
  }
}

/*********************************************************************
****/
/** BUMP_SENSE:  This will stop the robot when it bumps into something
**/
/** makes it back-up in a certain direction based on which side the
robot **/
/** currently tends toward.
**/
/*********************************************************************
****/

void bump_sense()
{
  while(1) {
            if(bump > 50) {
                  bumphit = 1;
            }
        if(bumphit > 0){
              direction = backward;
              angle = current_side;
              ++bumphit;
              if(bumphit > 150)
              {
                    bumphit = 0;
              }
        }
  }
}

/*********************************************************************
***/
/** BACK_CHECK:  This stops the robot from backing up into things.  If
**/
/** something is detected behing the robot, it will tell the motor to go
**/
```

```
/** forward and turn the servo in direction the robot currently tends
**/
/** toward.
**/
/*********************************************************************
***/

void back_check()
{
  while(1) {
      if (back_eye > thresh) {
            backhit = 1;
      }
      if(backhit > 0){
            direction = forward;
            angle = current_side;
            ++backhit;
      }
      if(backhit > 75)
      {
            backhit = 0;
      }
      defer();
  }
}

/*********************************************************************
***/
/** BEHAVIOR_ARBITRATE:  This process arbitrates what the robot should
**/
/** do based on its current variables.  If the robot is scared it will
**/
/** runaway.  If it is not it will simply move in the direction it was
**/
/** told.
**/
/*********************************************************************
***/

void behavior_arbitrate()
{
  while(1) {

      if (tap == 1) {
            motor(1, 0.);
            motor(0, 0.);
      }
      else {
            move();              /* If the robot is not scared.    */
      }
  }
}

/*********************************************************************
***/
/**************************** Various Functions
**************************/
/*********************************************************************
***/

/****************************  NEXT CORNER SETUP
************************/
/********    Does all need work when the current corner switches.
*******/
```

```
void next_corner_setup()
{
      location_update();
      side_set();
      wallfollowing = 0;
      countc = 0;
}

/*****************************     SIDE SET
**************************/
/******** Sets the current_side based on the corner and its
heading.*******/

void side_set()
{
      int temp = around_cell(map, map_x[current], map_y[current]);

      if((heading > 5.4978) || (heading < 0.7854))
      {
            if(temp == 208){current_side = left;}
            if(temp == 104){current_side = right;}
            if(temp == 22){current_side = left;}
            if(temp == 11){current_side = right;}
      }
      if((heading > 0.7853) && (heading < 2.3562))
      {
            if(temp == 208){current_side = left;}
            if(temp == 104){current_side = left;}
            if(temp == 22){current_side = right;}
            if(temp == 11){current_side = right;}
      }
      if((heading > 2.3561) && (heading < 3.92699))
      {
            if(temp == 208){current_side = right;}
            if(temp == 104){current_side = left;}
            if(temp == 22){current_side = right;}
            if(temp == 11){current_side = left;}
      }
      if((heading > 3.92698) && (heading < 5.4979))
      {
            if(temp == 208){current_side = right;}
            if(temp == 104){current_side = right;}
            if(temp == 22){current_side = left;}
            if(temp == 11){current_side = left;}
      }
}



/*****************************  HEADING UPDATE
**************************/
/*********************************************************************
***/

void heading_update()
{
      if((corner_heading > 5.4978) || (corner_heading < 0.7854))
      {
            heading = 0.0;
      }
      if((corner_heading > 0.7853) && (corner_heading < 2.3562))
      {
            heading = 1.5707;
      }
      if((corner_heading > 2.3561) && (corner_heading < 3.92699))
      {
            heading = 3.1416;
      }
      if((corner_heading > 3.92698) && (corner_heading < 5.4979))
      {
            heading = 4.7124;
      }
}

/**********************************************************************/
/***************************   WALKABLE   ****************************/
/**********************************************************************/

int walkable(int narray[], int x1, int y1, int x2, int y2)
{
      int dy = y2 - y1;
      int dx = x2 - x1;
      int dxi, dyj;
      int tempx = x1;
      int tempy = y1;
      int temp = 0;
      int i, j;

      if(dx > 0){dxi = dx;}
      else{dxi = 0 - dx;}
      if(dy > 0){dyj = dy;}
      else{dyj = 0 - dy;}

      for(i = dxi; i > 0; i--)
      {
            if(numa_cell(narray, tempx, tempy) > 6){++temp;}
            tempx = tempx + (dx/dxi);
      }
      for(j = dyj; j > 0; j--)
      {
            if(numa_cell(narray, tempx, tempy) > 6){++temp;}
            tempy = tempy + (dy/dyj);
      }

      tempx = x1;
      tempy = y1;

      for(j = dyj; j > 0; j--)
      {
            if(numa_cell(narray, tempx, tempy) > 6){++temp;}
            tempy = tempy + (dy/dyj);
      }
      for(i = dxi; i > 0; i--)
      {
            if(numa_cell(narray, tempx, tempy) > 6){++temp;}
            tempx = tempx + (dx/dxi);
      }

      if(temp == 0){return dxi + dyj;}
      else{return 0;}
}

/**********************************************************************/
/***************************  NUMA_CELL  ****************************/
/**********************************************************************/
```

```
int numa_cell(int narray[], int x, int y)
{
      int total = 0;
      int testfor = 3;

      if(garray(narray, x-1, y+1) == testfor){++total;}
      if(garray(narray, x,   y+1) == testfor){++total;}
      if(garray(narray, x+1, y+1) == testfor){++total;}
      if(garray(narray, x-1, y  ) == testfor){++total;}
      if(garray(narray, x+1, y  ) == testfor){++total;}
      if(garray(narray, x-1, y-1) == testfor){++total;}
      if(garray(narray, x,   y-1) == testfor){++total;}
      if(garray(narray, x+1, y-1) == testfor){++total;}

      return total;
}

/****************************  LOCATION UPDATE
***************************/
/*************************************************************************
***/

void location_update()
{
      int check;

      check = around_cell(map, map_x[current], map_y[current]);
      if(check == 208)
      {
           x_coor = (24.0*(float)map_x[current]) + 6.0;
           y_coor = (24.0*(float)map_y[current]) - 6.0;
      }
      if(check == 22)
      {
          x_coor = (24.0*(float)map_x[current]) + 6.0;
          y_coor = (24.0*(float)map_y[current]) + 6.0;
      }
      if(check == 104)
      {
          x_coor = (24.0*(float)map_x[current]) - 6.0;
          y_coor = (24.0*(float)map_y[current]) - 6.0;
      }
      if(check == 11)
      {
         x_coor = (24.0*(float)map_x[current]) - 6.0;
         y_coor = (24.0*(float)map_y[current]) + 6.0;
      }
      ++current;
}

/**********************************************************************/
/************************** AROUND_CELL ***************************/
/**********************************************************************/

int around_cell(int narray[], int x, int y)
{
/* 765    330
      4C3 so 0C3 = 11001101b or 205;
      210     303    */
      int total;
      int testfor = 3; /* can be changed to something else */
      int bit7 = 0;
      int bit6 = 0;
      int bit5 = 0;
      int bit4 = 0;
      int bit3 = 0;
      int bit2 = 0;
      int bit1 = 0;
      int bit0 = 0;
      int maskb7 = 0x80;
      int maskb6 = 0x40;
      int maskb5 = 0x20;
      int maskb4 = 0x10;
      int maskb3 = 0x08;
      int maskb2 = 0x04;
      int maskb1 = 0x02;
      int maskb0 = 0x01;

      if(garray(narray, x-1, y+1) == testfor){bit7 = 1;}
      if(garray(narray, x,   y+1) == testfor){bit6 = 1;}
      if(garray(narray, x+1, y+1) == testfor){bit5 = 1;}
      if(garray(narray, x-1, y  ) == testfor){bit4 = 1;}
      if(garray(narray, x+1, y  ) == testfor){bit3 = 1;}
      if(garray(narray, x-1, y-1) == testfor){bit2 = 1;}
      if(garray(narray, x,   y-1) == testfor){bit1 = 1;}
      if(garray(narray, x+1, y-1) == testfor){bit0 = 1;}

      total = bit7*maskb7 + bit6*maskb6 + bit5*maskb5 + bit4*maskb4 +
                  bit3*maskb3 + bit2*maskb2 + bit1*maskb1 + bit0*maskb0;

      return total;
}

/*********************************************************************/
/***************************   MARRAY   ****************************/
/*********************************************************************/

void marray(int tarray[], int x, int y, int value)
{
       int location;
       location = ((26-y)*27) + x;
       tarray[location] = value;
}

/*********************************************************************/
/***************************   GARRAY   ****************************/
/*********************************************************************/

int garray(int narray[], int x, int y)
{
       int location;

       if(((x > 26) || (x < 0)) || ((y > 26) || (y < 0)))
       {
            error = 1;
            tap = 1;
            wait(5000);
       }
       location = ((26-y)*27) + x;
       return narray[location];
}

/****************************   WAIT FUNCTION
***************************/
/**************************** waits in mseconds
***************************/

void wait(int milli_seconds)
```

```
{
      long timer_a;

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

/*****************************  RANDOM FUNCTION
**************************/
/***************** creates a random variable between 0-7
***************/

void random()
{
      ran = (((int) mseconds())&0b111);

}

/*****************************  PLAY FUNCTION
**************************/
/*********************** plays a particular sound
**********************/

void play(int p)
{
      poke(0x6000, 0b00000011);
      poke(0x6000, 0b00000010);
      if(p == 1){
            poke(0x6000, 0b00000000);
            poke(0x6000, 0b00000010);
      }
      else if(p == 2){
            poke(0x6000, 0b00001111);
            poke(0x6000, 0b00001110);
            wait(200);
            poke(0x6000, 0b00001100);
            wait(1000);
            poke(0x6000, 0b00001110);
      }
}

/************************* FIND CORNER HEADING
****************************/
/******** finds the heading to the next corner from current local
*********/

void find_c_heading()
{
              if((map_x[current] - x_cell) < 0)
      {
                               if((map_y[current] - y_cell) < 0)
            {
                  tester = (((float)map_x[current]) - ((float)x_cell) +
0.01)/(((float)map_y[current]) - ((float)y_cell) + 0.01);
                  corner_heading = 3.1416 + atan(tester);
            }
            else
            {
                  tester = (((float)x_cell) - ((float)map_x[current]) +
0.01)/(((float)map_y[current]) - ((float)y_cell) + 0.01);
                  corner_heading = 6.2832 - atan(tester);
            }
      }
      else
      {
                               if((map_y[current] - y_cell) < 0)
            {
                          tester = (((float)map_x[current]) -
((float)x_cell) + 0.01)/(((float)y_cell) - ((float)map_y[current]) +
0.01);
                  corner_heading = 1.5708 + atan(tester);
            }
            else
            {
                                                tester =
(((float)map_x[current]) - ((float)x_cell) +
0.01)/(((float)map_y[current]) - ((float)y_cell) + 0.01 );
                                                corner_heading =
atan(tester);
            }
      }
}

/***************************** RADS_TO_DEGREES
****************************/
/*********** finds the degrees heading from the radians heading
***********/

void rads_to_deg()
{
      deg_head = 360.0*heading/6.2832;
}

/***************************** UPDATE FUNCTION
****************************/
/*********** finds which cell the robot is in from the coords
*************/

void update()
{
       x_cell = (int) (x_coor/24.0);
       y_cell = (int) (y_coor/24.0);
}

/**********************************************************************
***/
/** MOVE FUNCTION:  This function updats the robots location and heading
**/
/** in a rectangular coordinate system.  This funciton uses the time
that**/
/** the robot was heading in a certain direction and calculates with
**/
/** geometry the new location and heading.  This function also sets the
**/
/** new direction and heading for the robot.  This function was written
**/
/** in this way for timing reasons and thus is not optimized for memory
**/
/** space.
**/
/**********************************************************************
***/

void move()
{
      if (old_angle != angle)  /* Sets a latency in the direction
changes */
             {                          /* because of the time it takes
the servo  */
            motor(0, direction); /* to change directions.
*/
            motor(1, direction);
            servo_deg(angle);
            wait(servo_latency);
      }

/* STRAIGHT-FORWARD */
      if (old_angle == straight && old_direction == forward)
      {
            delta_t = (int) mseconds() - time;
            if (time > 0 && (int) mseconds() < 0) {
                  delta_t = (max_t - time) + (max_t + (int) mseconds());
            }
            if (time < 0 && (int) mseconds() > 0) {
                  delta_t = (0 - time) + (int) mseconds();
            }
            time = (int) mseconds();
            x_coor = x_coor + (sin(heading)*((float) delta_t) *
dist_straight);
            y_coor = y_coor +  (cos(heading)*((float)
delta_t)*dist_straight);
            heading = heading;
            motor(0, direction);
            motor(1, direction);
            servo_deg(angle);
            old_angle = angle;
            old_direction = direction;
      }

/* SLIGHT RIGHT-FORWARD */
              if (old_angle == slight_right && old_direction == forward)
      {
            delta_t = (int) mseconds() - time;
            if (time > 0 && (int) mseconds() < 0) {
                  delta_t = (max_t - time) + (max_t + (int) mseconds());
            }
            if (time < 0 && (int) mseconds() > 0) {
                  delta_t = (0 - time) + (int) mseconds();
            }
            time = (int) mseconds();
            x_coor = x_coor + (sin(heading)*((float) delta_t) *
dist_straight);
            y_coor = y_coor +  (cos(heading)*((float)
delta_t)*dist_straight);
            heading = heading;
            motor(0, direction);
            motor(1, direction);
            servo_deg(angle);
            old_angle = angle;
            old_direction = direction;
      }

/* SLIGHT LEFT-FORWARD */
              if (old_angle == slight_left && old_direction == forward)
      {
            delta_t = (int) mseconds() - time;
            if (time > 0 && (int) mseconds() < 0) {
                  delta_t = (max_t - time) + (max_t + (int) mseconds());
            }
            if (time < 0 && (int) mseconds() > 0) {
                  delta_t = (0 - time) + (int) mseconds();
            }
            time = (int) mseconds();
            x_coor = x_coor + (sin(heading)*((float) delta_t) *
dist_straight);
            y_coor = y_coor +  (cos(heading)*((float)
delta_t)*dist_straight);
            heading = heading;
            motor(0, direction);
            motor(1, direction);
            servo_deg(angle);
            old_angle = angle;
            old_direction = direction;
      }




/* STRAIGHT-BACKWARD */
      if (old_angle == straight && old_direction == backward) {
            delta_t = (int) mseconds() - time;
            if (time > 0 && (int) mseconds() < 0) {
                  delta_t = (max_t - time) + (max_t + (int) mseconds());
            }
            if (time < 0 && (int) mseconds() > 0) {
                  delta_t = (0 - time) + (int) mseconds();
            }
            time = (int) mseconds();
            x_coor = x_coor - (sin(heading)*((float) delta_t) *
dist_straight);
            y_coor = y_coor -  (cos(heading)*((float)
delta_t)*dist_straight);
            heading = heading;
            motor(0, direction);
            motor(1, direction);
            servo_deg(angle);
            old_angle = angle;
            old_direction = direction;
      }

/* RIGHT-FORWARD */
      if (old_angle == right && old_direction == forward) {
            delta_t = (int) mseconds() - time;
            if (time > 0 && (int) mseconds() < 0) {
                  delta_t = (max_t - time) + (max_t + (int) mseconds());
            }
            if (time < 0 && (int) mseconds() > 0) {
                  delta_t = (0 - time) + (int) mseconds();
            }
            time = (int) mseconds();
            delta_xp = rad_right -
rad_right*(cos((float)delta_t*right_deg_time));
            delta_yp = rad_right*(sin((float)delta_t*right_deg_time));
            x_coor = x_coor + cos(heading)*delta_xp
+sin(heading)*delta_yp;
            y_coor = y_coor + sin(heading)*delta_xp +
sin(heading)*delta_yp;
            heading = heading + (float)delta_t*right_deg_time;
            motor(0, direction);
            motor(1, direction);
            servo_deg(angle);
            old_angle = angle;
            old_direction = direction;
      }

/* LEFT-FORWARD */
      if (old_angle == left && old_direction == forward) {
            delta_t = (int) mseconds() - time;
            if (time > 0 && (int) mseconds() < 0) {
                  delta_t = (max_t - time) + (max_t + (int) mseconds());
            }
            if (time < 0 && (int) mseconds() > 0) {
                  delta_t = (0 - time) + (int) mseconds();
            }
            time = (int) mseconds();
            delta_xp = rad_left*(cos((float)delta_t*left_deg_time)) -
rad_left;
            delta_yp = rad_left*(sin((float)delta_t*left_deg_time));
            x_coor = x_coor + cos(heading)*delta_xp
+sin(heading)*delta_yp;
            y_coor = y_coor + sin(heading)*delta_xp +
sin(heading)*delta_yp;
            heading = heading - (float)delta_t*left_deg_time;
            motor(0, direction);
            motor(1, direction);
            servo_deg(angle);
            old_angle = angle;
            old_direction = direction;
      }

/* GENTLE_LEFT-FORWARD */
      if (old_angle == gentle_left && old_direction == forward) {
            delta_t = (int) mseconds() - time;
            if (time > 0 && (int) mseconds() < 0) {
                  delta_t = (max_t - time) + (max_t + (int) mseconds());
            }
            if (time < 0 && (int) mseconds() > 0) {
                  delta_t = (0 - time) + (int) mseconds();
            }
            time = (int) mseconds();
            delta_xp = rad_gleft*(cos((float)delta_t*gleft_deg_time)) -
rad_gleft;
            delta_yp = rad_gleft*(sin((float)delta_t*gleft_deg_time));
            x_coor = x_coor + cos(heading)*delta_xp
+sin(heading)*delta_yp;
            y_coor = y_coor + sin(heading)*delta_xp +
sin(heading)*delta_yp;
            heading = heading - (float)delta_t*gleft_deg_time;
            motor(0, direction);
            motor(1, direction);
            servo_deg(angle);
            old_angle = angle;
            old_direction = direction;
      }

/* GENTLE_RIGHT-FORWARD */
      if (old_angle == gentle_right && old_direction == forward) {
            delta_t = (int) mseconds() - time;
            if (time > 0 && (int) mseconds() < 0) {
                  delta_t = (max_t - time) + (max_t + (int) mseconds());
            }
            if (time < 0 && (int) mseconds() > 0) {
                  delta_t = (0 - time) + (int) mseconds();
            }
            time = (int) mseconds();
            delta_xp = rad_gright -
rad_gright*(cos((float)delta_t*gright_deg_time));
```

```
            delta_yp = rad_gright*(sin((float)delta_t*gright_deg_time));
            x_coor = x_coor + cos(heading)*delta_xp
+sin(heading)*delta_yp;
            y_coor = y_coor + sin(heading)*delta_xp +
sin(heading)*delta_yp;
            heading = heading + (float)delta_t*gright_deg_time;
            motor(0, direction);
            motor(1, direction);
            servo_deg(angle);
            old_angle = angle;
            old_direction = direction;
      }

/* RIGHT-BACKWARD */
      if (old_angle == right && old_direction == backward) {
            delta_t = (int) mseconds() - time;
            if (time > 0 && (int) mseconds() < 0) {
                  delta_t = (max_t - time) + (max_t + (int) mseconds());
            }
            if (time < 0 && (int) mseconds() > 0) {
                  delta_t = (0 - time) + (int) mseconds();
            }
            time = (int) mseconds();
            delta_xp = rad_right -
rad_right*(cos((float)delta_t*right_deg_time));
            delta_yp = 0.0 -
rad_right*(sin((float)delta_t*right_deg_time));
            x_coor = x_coor + cos(heading)*delta_xp
+sin(heading)*delta_yp;
            y_coor = y_coor + sin(heading)*delta_xp +
sin(heading)*delta_yp;
            heading = heading - (float)delta_t*right_deg_time;
            motor(0, direction);
            motor(1, direction);
            servo_deg(angle);
            old_angle = angle;
            old_direction = direction;
      }

/* LEFT-BACKWARD */
      if (old_angle == left && old_direction == backward) {
            delta_t = (int) mseconds() - time;
            if (time > 0 && (int) mseconds() < 0) {
                  delta_t = (max_t - time) + (max_t + (int) mseconds());
            }
            if (time < 0 && (int) mseconds() > 0) {
                  delta_t = (0 - time) + (int) mseconds();
            }
            time = (int) mseconds();
            delta_xp = rad_left*(cos((float)delta_t*left_deg_time)) -
rad_left;
            delta_yp = 0.0 -
rad_left*(sin((float)delta_t*left_deg_time));
            x_coor = x_coor + cos(heading)*delta_xp
+sin(heading)*delta_yp;
            y_coor = y_coor + sin(heading)*delta_xp +
sin(heading)*delta_yp;
            heading = heading + (float)delta_t*left_deg_time;
            motor(0, direction);
            motor(1, direction);
            servo_deg(angle);
            old_angle = angle;
            old_direction = direction;
      }
```

```
/* STOPPED */
              if (old_direction == stopp) {
            time = (int) mseconds();
            motor(0, direction);
            motor(1, direction);
            servo_deg(angle);
            old_angle = angle;
            old_direction = direction;
      }

                  /* Heading Correction */
      if (heading > 6.2832)
      {
            heading = heading - 6.2832;
      }
      if (heading < 0.0)
      {
            heading = 6.2832 + heading;
      }
}

/********************************* THE END
********************************/
/********************************* THE END
********************************/
/********************************* THE END
********************************/
/********************************* THE END
********************************/
/********************************* THE END
********************************/
```

Cal.c Calibration Program

```
float dis_strc, ri_deg_tc, le_deg_tc, gl_deg_tc, gr_deg_tc;

void calibrate_time()
{

      long curr_time;

      wait(3000);


      beep();
      servo_deg(straight);
      wait(500);
      curr_time = mseconds();
      motor(0, 100.);
      motor(1, 100.);
      while(analog(2) < 100)
      {
      }
      motor(0, 0.);
      motor(1, 0.);
      dis_strc = 72.0/(float)(mseconds() - curr_time);



      servo_deg(right);
      wait(6000);
      beep();
      curr_time = mseconds();
      motor(0, 100.);
      motor(1, 100.);
      while(analog(2) < 100)
      {
      }
      motor(0, 0.);
      motor(1, 0.);
      ri_deg_tc = 6.2832/(float)(mseconds() - curr_time);




      servo_deg(left);
      wait(6000);
      beep();
      curr_time = mseconds();
      motor(0, 100.);
      motor(1, 100.);
      while(analog(2) < 100)
      {
      }
      motor(0, 0.);
      motor(1, 0.);
      le_deg_tc = 6.2832/(float)(mseconds() - curr_time);



      servo_deg(gentle_right);
      wait(6000);
      beep();
      curr_time = mseconds();
```

```
        motor(0, 100.);
        motor(1, 100.);
        while(analog(2) < 100)
        {
        }
        motor(0, 0.);
        motor(1, 0.);
        gr_deg_tc = 6.2832/(float)(mseconds() - curr_time);



        servo_deg(gentle_left);
        wait(6000);
        beep();
        curr_time = mseconds();
        motor(0, 100.);
        motor(1, 100.);
        while(analog(2) < 100)
        {
        }
        motor(0, 0.);
        motor(1, 0.);
        gl_deg_tc = 6.2832/(float)(mseconds() - curr_time);
        beep();
        beep();
        beep();
        beep();
        wait(6000);


}
```

map.c **Sample map program**

```
int x_start = 7;
int y_start = 3;
int x_finish = 12;
int y_finish = 19;

int map[729]={3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,
              3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,
              3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,
              3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,
              3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,
              3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,
              3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,
              3,3,3,3,3,3,3,3,3,3,3,0,9,0,3,3,3,3,3,3,3,3,3,3,3,3,3,
              3,3,3,3,3,3,3,3,3,3,3,0,0,0,3,3,3,3,3,3,3,3,3,3,3,3,3,
              3,3,3,3,3,3,3,3,3,3,3,0,0,0,3,3,3,3,3,3,3,3,3,3,3,3,3,
              3,3,3,3,3,3,3,3,3,3,3,0,0,0,3,3,3,3,3,3,3,3,3,3,3,3,3,
              3,3,3,3,3,3,3,3,3,3,5,0,0,0,4,3,3,3,3,3,3,3,3,3,3,3,3,
              3,3,3,3,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,3,3,3,3,3,3,
              3,3,3,3,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,3,3,3,3,3,3,
              3,3,3,3,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,3,3,3,3,3,3,
              3,3,3,3,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,3,3,3,3,3,3,
              3,3,3,3,3,0,0,0,0,5,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,
              3,3,3,3,3,0,0,0,0,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,
              3,3,3,3,3,0,0,0,0,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,
              3,3,3,3,3,0,0,0,0,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,
              3,3,3,3,3,0,0,0,0,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,
              3,3,3,3,3,0,0,0,0,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,
              3,3,3,3,3,0,0,0,0,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,
              3,3,3,3,3,0,0,8,0,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,
              3,3,3,3,3,0,0,0,0,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,
              3,3,3,3,3,0,0,0,0,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,
              3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3};



int map_x[3] = {9, 10, 12};
int map_y[3] = {10, 15, 19};

float x_coor = 168.0;
float y_coor = 72.0;
float heading = 0.0;
```

floor2.map Same map before processing from map making program of the 2nd level of Weil

```
/* map.h, contains the droid map and other arrays for the MSE-6 'Mouse'
*/
/* Droid by Michael Fiyak. This file is derived from Makemap by Michael
*/
/* Fiyak and the Droid Map JAVA script by Matthew Perkins.
*/


int x_start = 6;
int y_start = 1;
int x_finish = 12;
int y_finish = 19;

int map_x[10];
int map_y[10];

int map[729]={3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,
              3,0,0,0,0,0,0,0,0,0,3,0,0,0,3,0,0,0,0,0,3,0,0,0,0,0,3,
              3,0,0,0,0,0,0,0,0,0,3,0,0,0,3,0,0,0,0,0,3,0,0,0,0,0,3,
              3,0,0,0,0,0,0,0,0,0,3,0,0,0,3,0,0,0,0,0,3,0,0,0,0,0,3,
              3,0,0,0,0,0,0,0,0,0,3,0,0,0,3,0,0,0,0,0,3,0,0,0,0,0,3,
              3,0,0,0,0,0,0,0,0,0,3,0,0,0,3,3,3,3,3,3,3,0,0,0,0,0,3,
              3,0,0,0,0,0,0,0,0,0,3,1,1,1,3,0,0,0,0,0,3,0,0,0,0,0,3,
              3,0,0,0,0,0,0,0,0,0,3,0,0,0,3,0,0,0,0,0,3,0,0,0,0,0,3,
              3,0,0,0,0,0,0,0,0,0,3,0,0,0,1,0,0,0,0,0,3,0,0,0,0,0,3,
              3,0,0,0,0,0,0,0,0,0,3,0,0,0,3,0,0,0,0,0,3,0,0,0,0,0,3,
              3,0,0,0,0,0,0,0,0,0,3,0,0,0,3,0,0,0,0,0,3,0,0,0,0,0,3,
              3,3,3,3,3,3,3,3,3,3,3,0,0,0,3,3,3,3,3,3,3,3,3,3,3,3,3,
              3,0,0,0,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,0,0,0,0,0,3,
              3,0,0,0,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,3,
              3,0,0,0,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,3,
              3,0,0,0,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,0,0,0,0,0,3,
              3,0,0,0,3,0,0,0,0,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,
              3,0,0,0,3,0,0,0,0,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,
              3,0,0,0,3,0,0,0,0,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,
              3,0,0,0,3,0,0,0,0,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,
              3,0,0,0,3,0,0,0,0,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,
              3,0,0,0,3,0,0,0,0,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,
              3,0,0,0,3,0,0,0,0,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,
              3,0,0,0,3,0,0,0,0,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,
              3,0,0,0,3,0,0,0,0,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,
              3,0,0,0,3,0,0,0,0,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,
              3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3};
```

```
int mapt[729];

int mapt2[729];
```

mapwork.c **Program that does processing on the map (C code, not IC)**

```
/* Mapwork Program
*/
/* by Michael Fiyak, for use with the MSE-6 'Mouse' Droid
*/

#include "floor4.map"

/* ---------------GLOBAL VARIABLES------------------------ */

int maskb7 = 0x80;
int maskb6 = 0x40;
int maskb5 = 0x20;
int maskb4 = 0x10;
int maskb3 = 0x08;
int maskb2 = 0x04;
int maskb1 = 0x02;
int maskb0 = 0x01;


/* ------------------------------------------------------- */

/*******************************************************************/
/**************************    MAIN     ***************************/
/*******************************************************************/

main()
{

      printf("int x_start = %d;\n", x_start);
      printf("int y_start = %d;\n", y_start);
      printf("int x_finish = %d;\n", x_finish);
      printf("int y_finish = %d;\n\n", y_finish);

      zero(map_x, 10);
      zero(map_y, 10);
      update_array(mapt, map);
      set_cells(mapt);
      update_array(mapt2, mapt);
      map_convert(mapt, mapt2);
      print_array(mapt);
      printf("\n\n");
      printf("int map_x[3] = {%d, %d, %d};\n", map_x[0], map_x[1],
map_x[2]);
      printf("int map_y[3] = {%d, %d, %d};\n", map_y[0], map_y[1],
map_y[2]);
      printf("float x_coor = %d;\n", (float)x_start * 24.0);
      printf("float y_coor = %d;\n", (float)y_start * 24.0);
      printf("float heading = fill in here; \n");
}

/*********************************************************************/
/************************** MAP_CONVERT ***************************/
/*********************************************************************/

map_convert(int narray[], int parray[])
{
      int corners;

      cell_convert(narray);
      update_array(parray, narray);
      end_check(narray, parray);
      corner_mark(narray);
      update_array(parray, narray);
      corners = count_corners(narray);
      corn_calc(narray, corners);
      place_sol(narray);
}

/*********************************************************************/
/**************************  PLACE_SOL  ***************************/
/*********************************************************************/

place_sol(int narray[])
{
      int i = 0;

      while((map_x[i] != x_finish) || (map_y[i] != y_finish))
      {
            marray(narray, map_x[i], map_y[i], 5);
            ++i;
      }
}

/*********************************************************************/
/****************************  ZERO  ******************************/
/*********************************************************************/

zero(int narray[], int number)
{
      int i;
      for(i = 0; i < number; i++)
      {
            narray[i] = 0;
      }
}


/*********************************************************************/
/**************************  CORN_CALC  ***************************/
```

```
/*******************************************************************/

corn_calc(int narray[], int corners)
{
      int cor_a_x[corners];
      int cor_a_y[corners];
      int path_x[corners];
      int path_y[corners];
      int walk[corners];
      int walk2[corners];
      int walk3[corners];
      int walk3b[corners];
      int walktotal[corners];
      int i, j;
      int temp3;
      int temp6;
      int temp2;
      int w3;
      int min;
      int winner;
      int curr = 0;
      int solution = 0;

      zero(walk, corners);
      zero(walk2, corners);
      zero(walk3, corners);
      zero(walk3b, corners);
      zero(walktotal, corners);

      for(j = 26; j> -1; j--)
      {
            for(i = 0; i < 27; i++)
            {
                  if(garray(narray, i, j) == 4)
                  {
                        cor_a_x[curr] = i;
                        cor_a_y[curr] = j;
                        ++curr;
                  }
            }
      }
      for(i = 0; i < corners; i++)
      {
            if(walkable(narray, x_start, y_start, cor_a_x[i],
cor_a_y[i]))
            {
                  walk[i] = walkable(narray, x_start, y_start,
cor_a_x[i], cor_a_y[i]);
            }
            else
            {
                  walk[i] = 0;
            }
      }
      for(i = 0; i < corners; i++)
      {
            if(walkable(narray, x_finish, y_finish, cor_a_x[i],
cor_a_y[i]))
            {
                  walk2[i] = walkable(narray, x_finish, y_finish,
cor_a_x[i], cor_a_y[i]);
            }
            else
            {
```

```
                        walk2[i] = 0;
                  }
            }
            for(i = 0; i < corners; i++)
            {
                  if(walk[i] > 0)
                  {
                        min = 99;
                        for(j = 0; j < corners; j++)
                        {
                              if(walk2[j] > 0)
                              {
                                    temp3 = walkable(narray, cor_a_x[i],
      cor_a_y[i], cor_a_x[j], cor_a_y[j]);
                                    if(temp3 > 0)
                                    {
                                          if(temp3 < min)
                                          {
                                                walk3[i] = temp3;
                                                walk3b[i] = j;
                                                solution = 2;
                                                min = temp3;
                                          }
                                    }
                                    else{walk3[i] = 0;}
                              }
                        }
                  }
            }
            if(solution == 2)
            {
                  min = 900;
                  for(i = 0; i < corners; i++)
                  {
                        if(walk3[i] > 0)
                        {
                              w3 = walk3b[i];
                              walktotal[i] = walk[i] + walk3[i] + walk2[w3];
                              if(walktotal[i] < min)
                              {
                                    winner = i;
                                    min = walktotal[i];
                              }
                        }
                  }
                  map_x[0] = cor_a_x[winner];
                  map_y[0] = cor_a_y[winner];

                  map_x[1] = cor_a_x[walk3b[winner]];
                  map_y[1] = cor_a_y[walk3b[winner]];

                  map_x[2] = x_finish;
                  map_y[2] = y_finish;
            }
      }

      /*********************************************************************/
      /**************************   WALKABLE  ****************************/
      /*********************************************************************/

      int walkable(int narray[], int x1, int y1, int x2, int y2)
      {
            int dy = y2 - y1;
            int dx = x2 - x1;
            int dxi, dyj;
            int tempx = x1;
            int tempy = y1;
            int temp = 0;
            int i, j;

            if(dx > 0){dxi = dx;}
            else{dxi = 0 - dx;}
            if(dy > 0){dyj = dy;}
            else{dyj = 0 - dy;}

            for(i = dxi; i > 0; i--)
            {
                  if(numa_cell(narray, tempx, tempy) > 6){++temp;}
                  tempx = tempx + (dx/dxi);
            }
            for(j = dyj; j > 0; j--)
            {
                  if(numa_cell(narray, tempx, tempy) > 6){++temp;}
                  tempy = tempy + (dy/dyj);
            }

            tempx = x1;
            tempy = y1;

            for(j = dyj; j > 0; j--)
            {
                  if(numa_cell(narray, tempx, tempy) > 6){++temp;}
                  tempy = tempy + (dy/dyj);
            }
            for(i = dxi; i > 0; i--)
            {
                  if(numa_cell(narray, tempx, tempy) > 6){++temp;}
                  tempx = tempx + (dx/dxi);
            }

            if(temp == 0){return dxi + dyj;}
            else{return 0;}
      }

      /*********************************************************************/
      /************************* COUNT_CORNERS ***************************/
      /*********************************************************************/

      int count_corners(int narray[])
      {
            int corn = 0;
            int i, j;
            for(j = 26; j> -1; j--)
            {
                  for(i = 0; i < 27; i++)
                  {
                        if(garray(narray, i, j) == 4){++corn;}
                  }
            }
            return corn;
      }

      /*********************************************************************/
      /*************************** CORNER_MARK ***************************/
      /*********************************************************************/

      corner_mark(int narray[])
      {
            int i, j;
            for(j = 26; j> -1; j--)
            {
                  for(i = 0; i < 27; i++)
                  {
                        if (garray(narray, i, j) == 3);
                        {
                              if (around_cell(narray, i, j) == 208)
                              {
                                    marray(narray, i, j, 4);
                              }
                              if (around_cell(narray, i, j) == 22)
                              {
                                    marray(narray, i, j, 4);
                              }
                              if (around_cell(narray, i, j) == 104)
                              {
                                    marray(narray, i, j, 4);

                              }
                              if (around_cell(narray, i, j) == 11)
                              {
                                    marray(narray, i, j, 4);
                              }
                        }
                  }
            }
      }

      /*******************************************************************/
      /**************************  END_CHECK  ***************************/
      /*******************************************************************/

      end_check(int narray[], int parray[])
      {
            int i, j;
            for(j = 26; j > -1; j--)
            {
                  for(i = 0; i < 27; i++)
                  {
                        if(garray(narray, i, j) == 0)
                        {
                              blockt_kill(narray, parray, i, j);
                              room_kill(narray, parray, i, j);
                              update_array(narray, parray);
                        }
                  }
            }
      }

      /*******************************************************************/
      /************************* CELL_CONVERT **************************/
      /*******************************************************************/

      cell_convert(int narray[])
      {
            int i, j;
            for(j = 26; j> -1; j--)
            {
                  for(i = 0; i < 27; i++)
                  {
                        if(garray(narray, i, j) == 1)
                        {
                              marray(narray, i, j, 3);
                        }
                        if(garray(narray, i, j) == 2)
                        {
                              marray(narray, i, j, 3);
                        }
                  }
            }
      }

      /**********************************************************************/
      /************************** UPDATE_ARRAY  **************************/
      /**********************************************************************/

      update_array(int narray[], int parray[])
      {
            int i, j;
            for(j = 26; j> -1; j--)
            {
                  for(i = 0; i < 27; i++)
                  {
                        marray(narray, i, j, garray(parray, i, j));
                  }
            }
      }

      /**********************************************************************/
      /***************************  ROOM_KILL  ***************************/
      /**********************************************************************/

      room_kill(int narray[], int parray[], int i, int j)
      {
            int ti = i;
            int tj = j;
            int arount = around_cell(narray, i, j);
            int numt = numa_cell(narray, i, j);
            int kill = 0;
            if (arount == 244)
            {
                  ++ti;
                  while(around_cell(narray, ti, tj) == 224){++ti;}
                  if(around_cell(narray, ti, tj) == 233)
                  {
                        --tj;
                        while(around_cell(narray, ti, tj) == 41){--tj;}
                        if(around_cell(narray, ti, tj) == 47)
                        {
                              --ti;
                              while(around_cell(narray, ti, tj) == 7){--ti;}
                              if(around_cell(narray, ti, tj) == 151)
                              {
                                    ++tj;
                                    while(around_cell(narray, ti, tj) ==
      148){++tj;}
                                    if(around_cell(narray, ti, tj) == 244)
                                    {
                                          if(ti == i && tj == j)
                                          {
                                                kill = 1;
                                          }
                                    }
                              }
                        }
                  }
            }
            if(kill == 1)
            {
                  marray(parray, ti, tj, 3);
                  ++ti;
                  while(around_cell(narray, ti, tj) == 224)
                  {
                        marray(parray, ti, tj, 3);
                        ++ti;
                  }
                  if(around_cell(narray, ti, tj) == 233)
                  {
                        marray(parray, ti, tj, 3);
                        --tj;
                  }
                  while(around_cell(narray, ti, tj) == 41)
                  {
                        marray(parray, ti, tj, 3);
                        --tj;
                  }
                  if(around_cell(narray, ti, tj) == 47)
                  {
                        marray(parray, ti, tj, 3);
                        --ti;
                  }
                  while(around_cell(narray, ti, tj) == 7)
                  {
                        marray(parray, ti, tj, 3);
                        --ti;
                  }
                  if(around_cell(narray, ti, tj) == 151)
                  {
                        marray(parray, ti, tj, 3);
                        ++tj;
                  }
                  while(around_cell(narray, ti, tj) == 148)
                  {
                        marray(parray, ti, tj, 3);
                        ++tj;
                  }
            }
      }

      /*********************************************************************/
      /************************** BLOCKT_KILL ***************************/
      /*********************************************************************/

      blockt_kill(int narray[], int parray[], int x, int y)
      {
            int total = 0;
            int testfor = 3; // can be changed to something else

            if(garray(narray, x,   y+1) == testfor){++total;}
            if(garray(narray, x-1, y  ) == testfor){++total;}
            if(garray(narray, x+1, y  ) == testfor){++total;}
            if(garray(narray, x,   y-1) == testfor){++total;}

            if(total > 2){marray(parray, x, y, 3);}
      }

      /*********************************************************************/
      /**************************  NUMA_CELL  ***************************/
      /*********************************************************************/

      int numa_cell(int narray[], int x, int y)
```

```
{
      int total = 0;
      int testfor = 3; // can be changed to something else

      if(garray(narray, x-1, y+1) == testfor){++total;}
      if(garray(narray, x,   y+1) == testfor){++total;}
      if(garray(narray, x+1, y+1) == testfor){++total;}
      if(garray(narray, x-1, y  ) == testfor){++total;}
      if(garray(narray, x+1, y  ) == testfor){++total;}
      if(garray(narray, x-1, y-1) == testfor){++total;}
      if(garray(narray, x,   y-1) == testfor){++total;}
      if(garray(narray, x+1, y-1) == testfor){++total;}

      return total;
}

/*********************************************************************/
/************************** AROUND_CELL ****************************/
/*********************************************************************/

int around_cell(int narray[], int x, int y)
{
// 765    330
// 4C3 so 0C3 = 11001101b or 205;
// 210        303
      int total;
      int testfor = 3; // can be changed to something else
      int bit7 = 0;
      int bit6 = 0;
      int bit5 = 0;
      int bit4 = 0;
      int bit3 = 0;
      int bit2 = 0;
      int bit1 = 0;
      int bit0 = 0;

      if(garray(narray, x-1, y+1) == testfor){bit7 = 1;}
      if(garray(narray, x,   y+1) == testfor){bit6 = 1;}
      if(garray(narray, x+1, y+1) == testfor){bit5 = 1;}
      if(garray(narray, x-1, y  ) == testfor){bit4 = 1;}
      if(garray(narray, x+1, y  ) == testfor){bit3 = 1;}
      if(garray(narray, x-1, y-1) == testfor){bit2 = 1;}
      if(garray(narray, x,   y-1) == testfor){bit1 = 1;}
      if(garray(narray, x+1, y-1) == testfor){bit0 = 1;}

      total = bit7*maskb7 + bit6*maskb6 + bit5*maskb5 + bit4*maskb4 +
            bit3*maskb3 + bit2*maskb2 + bit1*maskb1 + bit0*maskb0;

      return total;
}

/*********************************************************************/
/**************************  SET_CELLS  ****************************/
/*********************************************************************/

set_cells(int narray)
{
      marray(narray, x_start, y_start, 8);
      marray(narray, x_finish, y_finish, 9);
}

/*********************************************************************/
/**************************   MARRAY   ****************************/
/*********************************************************************/
```

```
marray(int tarray[], int x, int y, int value)
{
    int location;
    location = ((26-y)*27) + x;
    tarray[location] = value;
}

/*******************************************************************/
/**************************    GARRAY    ***************************/
/*******************************************************************/

int garray(int narray[], int x, int y)
{
    int location;
    location = ((26-y)*27) + x;
    return narray[location];
}

/*******************************************************************/
/************************** PRINT ARRAY ****************************/
/*******************************************************************/

int print_array(int narray[])
{
      int i, j;
      printf("int map[729]={");
      for(j = 26; j > -1; j--)
      {
            if(j != 26)
            {
                  printf("              ");
            }
            for(i = 0; i < 27; i++)
            {
                  if(i != 26 || j != 0)
                  {
                        printf("%d,", garray(narray, i, j));
                  }
                  else{printf("%d", garray(narray, i, j));}
                  if(i == 26 && j != 0)
                  {
                        printf("\n");
                  }
            }
      }
      printf("}\n\n");
}
```

makemap.c **Just in case you need it. Converts the data from JAVA map maker to .map file.**

```
/*
File          : makemap.c
Function      : Makes the array definitions of the map from the map 
making program
              : done by Matt Perkins.  Basically converts his format to 
my array
              : format.
Logic By      : Michael J. Fiyak
Code By       : Michael J. Fiyak
Creation Date : 04/14/97
Last Modified :
*/

#include <stdio.h>
#define x = 27;
#define y = 27;

/* Print header at top of page */

header()
{
      printf("/* map.h, contains the droid map and other arrays for the 
MSE-6 'Mouse'    */\n");
      printf("/* Droid by Michael Fiyak. This file is derived from 
Makemap by Michael    */\n");
      printf("/* Fiyak and the Droid Map JAVA script by Matthew Perkins. 
*/\n");
      printf("\n\nint x_start = 6;\nint y_start = 1;\nint x_finish = 
14;\nint y_finish = 25;\n");
```

```
      printf("\nint map_x[10];\nint map_y[10];\n\n");
      printf("int
map[729]={3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,\n");

}

/* Print a variable in binary format */

prnt_bit(unsigned data, int width)
{
      int mask;
      int div = 64;
      mask = 3<< width-2;
      while(mask>2)
      {
            if(((data & mask)/div) == 0)
            {
                  printf("0,");
            }
            if(((data & mask)/div) == 1)
            {
                  printf("1,");
            }
            if(((data & mask)/div) == 2)
            {
                  printf("2,");
            }
            if(((data & mask)/div) == 3)
            {
                  printf("3,");
            }
            mask = mask >> 2;
            div = div/4;
      }
}
/* Print trailer at end of file */

trailer()
{
      printf("\n");
      printf("int mapt[729];\n\n");
      printf("int mapt2[729];");

}

/*    main program       */

main(argc, argv)
int     argc;
char    *argv[];
{
      char    dummy[16];
      unsigned  temp[32];     /* array to contain data from S-records */
      /* temporary storage for 2 S-rec chars converted to unsigned int
*/
      unsigned  tmp;
      FILE    *stream;                  /* input stream from file */
      int     i,z;
      int number;


      if (argc < 2 )
      {
            printf ("\nUsage: makemap file.map \n");
            goto terminate;
      }

      if ((stream = fopen(argv[1], "r")) != NULL)
      {
            header();
            fscanf(stream, "%d", &number);
            for (i = 25; i > 0; i--)
            {
                  printf("              ");
                  for (z = 9; z > 0; z--)
                  {

                        {
                              if (z == 9){printf("3,");}
                              else if (z == 1){printf("3,");}
                              else if (z == 2)
                              {
                                    printf("%d,", (number & 192)/64);
                                    fscanf(stream, "%d", &number);
                              }
                              else
                              {
                                    prnt_bit(number, 8);
                                    fscanf(stream, "%d", &number);
                              }
                        }
                  }
                  printf("\n");
            }
            printf("
3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3};\n\n");
            trailer();
      }
      else
      {
            printf("\n***** Could not open S-record data file:%s \n",
argv[1]);
      }
terminate:
      return(0);
}
```