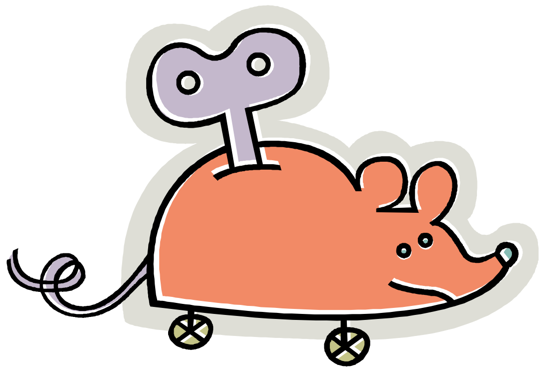
Final Report

For the MicroMouse



Submitted April 29th, 2003.
GE 498

Team Members:

Brian Ladewig
Matt McReynolds
Bill Sustr
Katie Williams

Faculty Advisors:
Prof. Kempf
Prof. Stone

Executive Summary

This final report summarizes the final progress of the MicroMouse senior project team and clarifies the ultimate design solution by first discussing the problem definition, posed solution, and benefits of implementation. Secondly the as built specifications address the mechanical and electrical technical specifications which include drawings, diagrams, and schematics that illustrate the final constructed MicroMouse. Each design requirement is then addressed and verified by test results. Lastly, the final report includes a concluding cost analysis of the project. Overall, the MicroMouse meet most of the system design requirements and was under budget with a cost of only $394.60.

Honor Code

"I have neither given nor received nor have I tolerated any others use of unauthorized aid"

Brian Ladewig

Matt McReynolds

Bill Sustr

Katie Williams

Table of Contents

I. Introduction

The MicroMouse is a miniature robot whose purpose is to find its way through a maze (just like a real mouse) in the shortest amount of time. The MicroMouse adheres to contest rules and specifications for the APEC, the Applied Power Electronics Conference and Exposition, which are similar to other popular MicroMouse competitions. APEC is a leading worldwide event for practicing power professionals. However, the MicroMouse will compete in the local IEEE (Institute of Electrical and Electronics Engineers) sponsored MicroMouse competition on May 3rd, 2003 at Valparaiso University.

The MicroMouse is capable of navigating a random maze of selected dimensions and completing ten runs in ten minutes without human intervention. The basic function of the MicroMouse is to travel from the start square (located at one of the corners of the maze) to the destination square (center of the maze). This is otherwise known as a "run". The time it takes to travel from start to destination is called the run time. However, the maze time is the total time from the first activation for the MicroMouse until the start of each run is measured. The scoring of the contest rewards speed, efficiency of maze solving, and self-reliance of the MicroMouse.

The ultimate fully assembled MicroMouse robot is a combination of electrical and mechanical ingenuity. Measuring only 14 cm in length, 10 cm in width, and only 10 cm in height, the miniature robot combines efficiency of space and functionality. Five sensors on the sides and front of the robot are the "eyes" for seeing and judging wall distances in the maze. The "eyes" are non-other than infrared light detectors which detect the infrared light emitted parallel to the sensors which bounce off the maze walls. A custom-built encoder measures the distance traversed by the robot. As the robot moves in the maze, beams of infrared light either pass through (detected by the receiver) or bounce off (not detected by the receiver) the small openings on the wheel. The number of pulses detected corresponds to a distance traveled by the robot.

This final report first explains the rationale and significance of the MicroMouse project in the problem definition. The problem definition addresses our project problem, the posed solution, the benefits of our implementation, and the impact of our project. The as built specifications are the blue prints of the MicroMouse, illustrating the technical solution and final design of the MicroMouse so that posterity may rebuild or improve on the MicroMouse project. The next important section of this final report is the requirements verification. This section describes how the MicroMouse final design addresses and successfully completes each of the system design requirements. Each requirement is stated and then given sufficient proof of verification by referencing results from our test report. This section proves that the project design was successful in certain requirements and unsuccessful in others. Lastly the budget of the project is discussed. It consists of an overview listing the major categories and a complete parts list of each component, including the suppliers and final overall budget.

II. Problem Definition

The project problem was to design and build a robot that autonomously navigates a random maze of specified dimensions with accordance to APEC contest rules. The scoring of MicroMouse contests rewards speed, efficiency of maze solving, and self-reliance of the MicroMouse.

A.) Significance and Benefits

The MicroMouse competition has been in existence since the late 1970's. Teams from around the world enter into regional competition in efforts of creating the best maze-solving robot available. In 1986, the Institute of Electrical and Electronics Engineers (IEEE) created the first U.S. MicroMouse competition held in Atlantic City. Now competitions exist all over the country, where educational institutions and clubs compete against one another. For the first time Valparaiso University will have the opportunity of entering their own version of the MicroMouse this year, and our team has been chosen to tackle the responsibility of creating such a robot.

Using references from previous MicroMouse teams, our group came up with our own solution to complete our project goal. Our design, using a configuration of infrared sensors, a Handy Board processing system, poly-carbonate chassis and servo motors, allowed the robot to navigate, store position data, and retrace the optimal path in the shortest time possible. Our design will hopefully set a benchmark for future Valparaiso MicroMouse teams to change and improve on in years to come and allow those teams to continue to make Valparaiso University a defining force in the educational engineering world

B. System Design Requirements Document

1.0 Scope

This document is intended to disclose the goals, objectives, and requirements of the MicroMouse project.

2.0 Referenced Documents

Official MicroMouse APEC, the Applied Power Electronics Conference and Exposition Contest, Rules Webpage
http://www.apec-conf.org/APEC_MicroMouse_Contest_Rules.html

3.0 Requirements Definition

3.1 Project Goal Statement

Our goal is to construct a robot that is capable of navigating a selected maze and completing ten runs in ten minutes without human intervention.

3.2 Objectives

To achieve the project goal, the system must be able to:

3.2.1 Meet the APEC specifications for robot construction.
3.2.2 Meet the APEC rules for the competition.
3.2.3 Navigate a randomly selected maze of standard dimensions according to APEC rules.
3.2.4 Must be a sustainable resource.
3.2.5 Navigate the maze in the shortest amount of time possible.

3.3 System Requirements

3.3.1 Meet the APEC specifications requirements for robot construction

3.3.1.1 The MicroMouse shall be self-contained.
3.3.1.2 The MicroMouse shall use a power source not driven by combustion.

3.3.2 Meet the APEC rules for the competition

3.3.2.1 The MicroMouse shall run and return unassisted to the start position after completion of each run.
3.3.2.3 The MicroMouse must be capable of operating in a dry environment at STP.

3.3.3 Navigate a randomly selected maze of standard dimensions according to APEC rules

3.3.3.1 The MicroMouse shall have dimensions no greater than 25cm X 25cm square at any point during navigation of the maze.
3.3.3.2 The MicroMouse shall remain intact and leave nothing behind during navigation of the maze.
3.3.3.3 The MicroMouse shall be able to navigate without jumping over, climbing, scratching, damaging, or destroying the walls of the maze.
3.3.3.4 The MicroMouse battery should run at minimum 10 minutes.

3.3.4 Must be a sustainable resource

3.3.4.1. The MicroMouse must use recyclable batteries.
3.3.4.2. The MicroMouse must be able to be manufactured in the Valparaiso University labs.
3.3.4.3. The MicroMouse main microcontroller must be reusable.

3.3.5 Navigate the maze in the shorted amount of time possible

3.3.5.1 The MicroMouse shall complete 10 runs within the 10 minute time limit.

C.) Posed Solution Analysis

The posed solution was implemented after careful analysis from our alternate solution design analysis report, which included a preliminary hazards analysis of the optimal solution. To achieve the MicroMouse design goal, the project team had to find the proper design solutions for the robot construction. After looking at previous designs of MicroMouse robots and performing research, possible methods and components that could help in designing and building the robot were determined. From this research, the team constructed three separate alternative solutions. Each of the three solutions had different electrical and mechanical components, which were evaluated based on their capabilities of meeting the project requirements. The evaluation focused mainly on the chassis material, the wheel configuration, the driving motors, the microcontroller, the sensors, and the required software components of the design. Each of the three chosen design solutions met all of the requirements of the project, however the second solution scored the highest after the down-selection analysis in meeting the project requirements and was chosen to be our design solution. A risk and hazard assessment table was then constructed for that chosen solution to find the possible hazards associated with its design. From this assessment, possible changes or additions that could reduce the risks of those hazards were created.

Proposed Solutions

In order to achieve the MicroMouse design goal of navigating a selected maze, three alternative solution combinations were created and analyzed. Each of these unique solution combinations had their own individual positive and negative aspects, however all three were still capable of reaching the main goal of the design problem.

<u>Alternate Solution 1</u>

For our first alternative solution, the robot chassis is to be built out of circuit board material. The system will be designed with four wheels configuration using servomotors for driving and steering. The robot will work with a custom printed circuit board (PCB) as a micro controller and it will use a Basic X Language for the software. To sense the walls of the maze, the robot will use sonar sensors.

Sonar Sensors:

In this solution we chose to use sonar sensors for wall detection. Sonar sensors prove to be very good in detecting long distances with great accuracy. This would be helpful in detecting long corridors for the MicroMouse to travel down. Sonar sensors also provide a linear output. This is beneficial because it is easier to implement in the software algorithm. One large disadvantage of sonar sensors is that they are not very accurate for short ranges and some do not even detect short ranges at all. This would cause navigation problems due to the miniature dimensions of the maze. However, sonar sensors are also more expensive compared to infrared sensors.

Circuit Board Chassis:

The circuit board chassis is the lightest of the three construction material possibilities. This will allow for maximum acceleration and velocity. It is also relatively easy to integrate electrical and mechanical components into this material. One draw back of this material is that it is not very strong. If the robot were to crash during operation, the circuit boards could shatter or crack. As a result of a collision of this magnitude the MicroMouse would be unable to achieve the required main goal of the project, to successfully navigate the maze.

Custom PCB:

This method would involve using a custom printed circuit board (PCB) and an X microcontroller. Using a custom PCB has several advantages. First of all it would be designed specifically for our project, so it would have no excess, unnecessary functionality. This also leads to a smaller circuit board, which is ideal for minimizing size and weight specifications. Having a custom PCB costs only about fifty dollars, which is less expensive than our other options and would provide increased funds for other parts of the project. Using a PCB would, however, have a few downfalls including the need to learn the associated design software as well as laying out the entire schematic to be fabricated. This could be time consuming, since no one in the group is familiar with this software. A PCB would also require that all circuitry layouts are included and fully functional when it is sent out for fabrication. We would no longer be able to make changes to any of the circuitry contained on the PCB once the design is ordered. With regard to the X microcontroller, while it is completely capable of giving us the functionality we would need for this project it requires the use of a programming language that no one in the team is familiar with, which would not be an ideal situation.

Four Wheel Configuration:

For designing a very stable MicroMouse, using four wheels is highly recommended over using a three-wheel configuration. The robot with four wheels will have less risk of tipping over because the robot will have four contact points with the ground. Also, using four wheels will result in a high-speed MicroMouse because we will not be concerned with weight being transferred forwards or backwards under braking or accelerating. Unfortunately, accurate steering is difficult to achieve with four wheels because it demands a good steering design. Also using four wheels will result in a larger turning radius preventing sharp turns in the maze.

Servomotors:

Servomotors run using a control loop and require feedback of some kind. The control system uses feedback from the motor to compensate for errors in the states of the motor, such as position and velocity. The control loop constantly checks to see if the motor is on the right path and, if it is not, makes the necessary adjustments. The servomotor requires tuning since it is using a feedback system. Tuning a motor can be a very difficult process, but it allows the user to have more control over the behavior of the motor. This makes servomotors more reliable than stepper motors. In general, servomotors run more smoothly than stepper motors except when micro-stepping is used. Also, as speed increases, the torque of the servo remains constant, making it better than the stepper at high speeds (usually above 1000 RPM), but it is more expensive than the stepper motor and it require more maintenance due to brushes on a brushed DC motors.

Basic X Language:

This solution requires a simple basic program to interface with the basic x-microcontroller. This software language has many features such as a floating point math library, multitasking, and networking. BasicX also allows easy programming for interface to servo ready robot wheels and serial LCD screen. This programming language is low-level and will take more time to learn and develop than the higher level languages considered in the other alternate solutions. The primary advantage is that it easily interfaces with the basic x microcontroller.

Alternate solution one is excellent for making the robot fast and light in weight. Disadvantages of this solution are that it will be time consuming in the learning of the software associated with the microcontroller, it will not be very accurate in sensing the walls of the maze in short ranges, it will need a large turning radius, and the motor will require maintenance due to brushes on brushed DC motor. Also some of the main components in this design are expensive compared to the other designs, such as the sonar sensors and the servomotors.

Alternative Solution 2

In this solution, the MicroMouse will function by using infrared (IR) sensors, polycarbon plastic chassis, HC11 Handy Board, and Interactive C software language. The infrared sensors are very accurate for short ranges and will therefore sense accurately below the wall. For designing the chassis, the polycarbon plastic is a strong material and easy to manufacture by using CNC (Computer Numerical Control) machine. The HC11 is a specially designed circuit board that has useful functions for MicroMouse applications. Using stepper motor will provide the MicroMouse with precise positioning control. One of the best features of Interactive C software language is that the compiler and debugger are integrated into one package. By using the above choice the MicroMouse will be applied to navigate and solve the maze in shortest time.

Infrared Sensors:

In this solution we chose to use infrared (IR) sensors that sense below the wall level. Infrared sensors are very accurate for short ranges. This is beneficial because we can use these sensors to easily sense the walls on the sides of the MicroMouse and keep it centered along the corridor between the side walls. In general, IR sensors are also cheaper than sonar sensors. IR sensors are readily available with analog or digital outputs. If a digital output is selected, it will be very easy to implement in the software algorithm. The analog option has more of a logarithmic output and therefore must implement a lookup table in the software algorithm to determine distances. This option is also more difficult to implement. The IR sensor is also less accurate than sonar for sensing longer distances.

Polycarbon Plastic Chassis:

This material is the middleweight of the three materials. This option allows for increased acceleration and velocity. Polycarbon is also a very strong material. It can withstand relatively high-speed impacts, allowing for any accidental collisions. An advantage as well as a drawback to Polycarbon is that it is produced in sheets of raw material. This allows for the use of CNC machines to cut exact parts to specific measurements. This procedure, however, takes time and causes a great deal of waste product. CNC precision must also be taken to an exact degree ensuring that all dimensions fit to the desired dimensions. Without following close detail to milling precision proper assembly will be impossible resulting in a large waste in raw material.

HC11 Handy Board:

This solution involves using a semi-custom, HC11 based circuit board referred to as the Handy Board. This board is a specially designed circuit board that has only functions that would be considered useful to robotics applications such as the MicroMouse. It contains only necessary functionality and therefore is smaller in size, which is ideal for our size and weight specifications. This kit has built in 32K of battery backed RAM as well as a rechargeable power supply. As mentioned, the Handy Board is HC11 based which is ideal since our team members are familiar with assembly or C programming languages that can be used to program this microcontroller. The major downfall of the Handy Board is that it is about 220 dollars to purchase one, which depletes funds for other materials. At this point the Handy Board's advantageous functionality and size is outweighing the somewhat excessive price.

Three Wheel Configuration:

Since the MicroMouse will have to make many turns during navigation of the maze, the three-wheel configuration is highly recommended to achieve this task. The MicroMouse will then turn by using a separate motor for the each wheel, and by slowing down or reversing one of the motors, the robot is able to steer. Also, we can use servomotors to turn the front wheel instead of the pervious method. Three wheels will have small turning radius but more risk of tipping over if the weight of the MicroMouse does not distribute, as it should, and also we have to lower the center of gravity of the MicroMouse to provide high weight transferred forwards or backwards under braking or accelerating.

Stepper Motor:

Stepper motors are less expensive and easier to use than a servomotor of a similar size. They are controlled by providing the drive with a step and direction signal. Their step movements make them excellent for precise positioning control. Stepper motors have the capability of running in an open loop configuration (no feedback), hence no tuning is required, but if they are used with an open-loop system, they can stall or lose position. In general, a stepper motor has high torque capability at low speeds, but low torque capability at high speeds. Also, it has a rough movement at low speeds unless the drive has micro-stepping capability. A stepper motor has a higher holding torque than a servo motor of similar size when idle, because the current is continuously flowing in the stepper motor windings, but this makes it a high current consumer. Stepper motors require low maintenance since they are brushless. Generally stepper motors are good for low-cost applications.

Interactive C:

The Interactive C (IC) language is a multi-tasking C language based compiler designed to run on boards based on the Motorola 68HC11. One of the best features of Interactive C is that the compiler and debugger are integrated into one package. Individual expressions can be typed at the command line where they are instantly compiled and executed immediately by the controller board. This feature makes trying out expressions easy without having to edit, compile and download an entirely new program. The language development program also includes some convenient built-in routines for printing to the

LCD screen, operating motors and accepting user inputs through the pushbuttons. The language is high-level, which cuts down on development time since most of our ECE team members have had advanced high-level software programming experience. The disadvantage is that the team is unfamiliar with Interactive-C and would have to learn this as a second language.

This solution proposes that the MicroMouse will function by using the following components: infrared (IR) sensors, polycarbon plastic chassis, HC11 handyboard, three wheels and Interactive C software language. This solution will provide the MicroMouse with ability to navigate and solve the maze in the shortest time. Also, this solution will minimize the problems that the MicroMouse might face.

<u>Alternative Solution 3</u>

Another set of possible options for each of the individual areas of the MicroMouse utilizes infrared sensors that are placed on the underside of arms that protrude out from the side of the robot. The sensors will sense downward onto the tops of the walls of the maze. These arms are used in conjunction with a steel metal chassis that will provide a strong, stable base structure. In this case, the MicroMouse will be controlled by a 68HC11 based evaluation board. The wheels will be driven by stepper motors, which allow for easy and accurate control. Low-level assembly language will be the type of software that is used for programming.

Infrared Sensors:

This solution again utilizes the IR sensors over sonar. The IR sensors are placed above the wall and look down at the tops of the wall. This makes it simple to keep the MicroMouse in the center of the corridor. The IR sensors are excellent for short ranges, but lack the desired accuracy for longer distances. The implementation of the sensors above the wall will also require construction of arms that hang over the walls to house the sensing units. This may incorporate more problems if one of those arms fail during operation and will also add more weight to our robot decreasing its speed. The software implementation of the sensors will be the same as that in solution two.

Sheet Metal Chassis:

Sheet metal is by far the heaviest of the three materials. Because of this deficiency, larger motors will be needed to attain comparable speeds with the circuit board and polycarbon plastic materials. Sheet metal is also very difficult to shape and assemble. Working sheet metal into desired shapes takes a great deal of skill to reduce error. Precision must be used when assembling sheet metal primarily due to the fact that the material properties do not allow for error in dimensions. However when constructed properly, sheet metal is nearly indestructible. The same material properties that inhibits assembly creates a structure that would allow for a very durable robot.

HC11 Evaluation Board:

This solution would use an HC11 based evaluation board to control the MicroMouse. An evaluation board has all the functionality that would be necessary for this project and very likely, much more than would be required. Our team is familiar with software languages that can be used to program this processor, which is also a positive point. An evaluation board however, is quite large in size and would be approaching our size and weight requirements. This type of circuit board is also about 150 dollars to purchase, which is right in the middle of the price range we are considering.

Three Wheel Configuration:

Since the MicroMouse will have to make many turns during navigation of the maze, the three-wheel configuration is highly recommended to achieve this task. The MicroMouse will then turn by using a separate motor for the each wheel, and by slowing down or reversing one of the motors, the robot is able to steer. Also, we can use servomotors to turn the front wheel instead of the pervious method. Three wheels will have small turning radius but more risk of tipping over if the weight of the MicroMouse does not distribute, as it should, and also we have to lower the center of gravity of the MicroMouse to provide high weight transferred forwards or backwards under braking or accelerating.

Stepper Motor:

Stepper motors are less expensive and easier to use than a servomotor of a similar size. They are controlled by providing the drive with a step and direction signal. Their step movements make them excellent for precise positioning control. Stepper motors have the capability of running in an open loop configuration (no feedback), hence no tuning is required, but if they are used with an open-loop system, they can stall or lose position. In general, a stepper motor has high torque capability at low speeds, but low torque capability at high speeds. Also, it has a rough movement at low speeds unless the drive has micro-stepping capability. A stepper motor has a higher holding torque than a servo motor of similar size when idle, because the current is continuously flowing in the stepper motor windings, but this makes it a high current consumer. Stepper motors require low maintenance since they are brushless. Generally stepper motors are good for low-cost applications.

Assembly Language:

The assembly language used by this solution will be familiar to all ECE team members since they have completed a microcontrollers course based on specific assembly programming of the Motorola 68HC11. The main advantage is the familiar programming language. The disadvantage of the assembly language is the low-level development and testing of our software algorithms. It might take more time to develop routines to perform the same functions that could be easily programmed in high-level languages (such as Interactive C).

This combination of the type of sensors, chassis, microcontroller, motor and programming language gives us a very plausible solution for the MicroMouse. This particular solution might not be optimal, but it would function at a somewhat acceptable level.

Down Selection and Analysis Matrix

Now that three alternate solutions have been presented, there must be a method to determine which one will best fulfill our requirements. To determine this, we have created two matrices. The first matrix will determine if any of the solutions will fail to meet any of the required criteria. If a solution fails to meet any required criteria, that solution will be excluded immediately from final selection. The second matrix will determine how well each of the solutions meet our desired criteria. After all rankings are complete, we may down-select our optimal final solution.

Solutions Matrix Table

Each one of our alternate solutions needed to satisfy the list of required criteria for our project. These are the absolute minimum requirements to consider a solution a possibility. Each of the three solutions that we came up with satisfies all of the requirements. A short description is given for the list of criteria to more clearly explain each requirement.

Table 1 – Alternate Solutions that Satisfy Required Criteria

Required Criteria	*Alternate Solution 1*		*Alternate Solution 2*		*Alternate Solution 3*	
	Description	Y/N	Description	Y/N	Description	Y/N
MicroMouse must be self-contained	All electrical and mechanical components needed are mounted and attached to the robot body.	Y	All electrical and mechanical components needed are mounted and attached to the robot body.	Y	All electrical and mechanical components needed are mounted and attached to the robot body.	Y
Power source not driven by combustion	Using batteries	Y	Using batteries	Y	Using batteries	Y
MicroMouse runs and returns unassisted to the start position after completion of each run	Memorize path	Y	Memorize path	Y	Memorize path	Y
Operates in a dry environment at STP	Operate in normal room conditions	Y	Operate in normal room conditions	Y	Operate in normal room conditions	Y
Dimensions no greater than 25 cm X 25 cm square at any point during navigation of maze	Can be built to meet the required dimension	Y	Can be built to meet the required dimension	Y	Can be built to meet the required dimension	Y
Remains intact and leave nothing behind during navigation of maze	All components shall be fixed to the body	Y	All components shall be fixed to the body	Y	All components shall be fixed to the body	Y
Navigates without jumping over, climbing, scratching, damaging, or destroying the walls of the maze.	Uses wheels, and uses sonar sensors to sense walls with leaving enough tolerance	Y	Uses wheels, and uses infrared sensors to sense walls with leaving enough tolerance	Y	Uses wheels, and uses infrared sensors to sense walls with leaving enough tolerance	Y
Battery should run at minimum 10 minutes	Using a long life batteries	Y	Using a long life batteries	Y	Using a long life batteries	Y
Must Use recyclable batteries	Batteries are recyclable	Y	Batteries are recyclable	Y	Batteries are recyclable	Y
Manufactured in the Valparaiso university labs	Parts to be manufactured and assembled are of VU manufacturing and design capabilities	Y	Parts to be manufactured and assembled are of VU manufacturing and design capabilities	Y	Parts to be manufactured and assembled are of VU manufacturing and design capabilities	Y
Microcontroller must be reusable	Uses custom PCB	Y	Uses HC11 Handyboard	Y	Uses HC11 evaluation board	Y
Shall complete 10 runs within the 10 minute time limit		N/A		N/A		N/A

Down Selection and Analysis Matrix Table

Each member of the team ranked each of the desired criteria and we then averaged the numbers to come up with a final value. The higher the importance of the criteria, the higher the value it was given. That number was then multiplied by a score that was determined by how well each solution satisfied the criteria. This gave us a weighted score for each item on the list. We then added up the weighted scores for each of the possible solutions to obtain a final score.

Table 2 – Down Selection Matrix

		Alternate Solution 1		Alternate Solution 2		Alternate Solution 3	
Desired Criteria	*Value*	*Sc*	*Wt Sc*	*Sc*	*WtSc*	*Sc*	*Wt.Sc*
Light weight	8	9	72	8	64	5	40
Maximum speed	8	4	32	9	72	6	48
Rechargeable batteries	3	10	30	10	30	10	30
Easy hardware interface	4	10	40	8	32	6	24
Battery backed ram	6	6	36	10	60	6	36
High-level software (easy to develop)	3	6	18	8	24	4	12
Familiar software	3	3	9	7	21	9	27
Small turning radius	7	5	35	10	70	10	70
Maximum acceleration	7	6	42	9	63	8	56
Stability (does not fall over)	6	10	60	7	42	7	42
Aesthetically pleasing	2	5	10	8	16	8	16
Totals			384		**494**		401

As the results from our matrices show, all of the solutions meet our desired criteria. Therefore, we ranked all three solutions in our desired criteria matrix. After analyzing all the totals, it was determined that our optimal configuration was alternate solution two. As it can be seen from the totals, alternate solution two best met all of our desired criteria. This is the solution we will choose to implement to achieve our project goal.

Hazard Analysis of Optimal Solution

With the purpose of fully guaranteeing that a particular design configuration is in fact optimal, all possible hazards related to that solution must be identified. An in depth analysis of these known hazards allows for each alternate solution to receive both severity and probability assignments. From this analysis, the best course of action can be taken to ensure that each risk and hazard due to hazard is either reduced or eradicated completely.
The matrix below shows the possible hazards along with their possible causes. It then illustrates the effects of these hazards and any actions that must be taken to reduce the probability of that hazard occurring. It is important to determine these hazards ahead of time so that we may incorporate certain safety measures into our design.

Table 3 – Hazards and Preventive Measures

Hazard	***Cause***	***Effect***	***Corrective or Preventive Measures***
1. Electrical Shock	Human contact with electrical circuits.	- May cause painful, undesirable sensation in area of contact.	Isolate and enclose all electrical circuitry in order to eliminate risk of human contact. Solder all connections and enclose them so

		- May also damage the circuitry of the MicroMouse	that they are not accessible to touch with something the size of a pencil.
2. Fire	Circuitry drawing too much power.	- Components may melt down and possibly ignite if they get too hot.	Regulate power expenditure the keep temperatures down and reduce risk of combustion.
3. Explosion	Excessive temperatures.	- Letting circuitry become too hot.	Control temperatures of circuitry.
4. Sharp Edges	Corners of integrated circuits or pin connectors may be sharp. Using a sheet metal chassis.	- Can cause physical injury to humans. - Cut skin. - Damage the maze.	Make certain that no hazardous circuitry, connectors or wiring is exposed by properly encasing them. Debur edges. Cover edges with tape. Use plastic chassis.
5. Battery Acid	Batteries punctured	- Could corrode circuitry - Burn skin if touched	Provide Casing for batteries. Use non-corrosive batteries.
6. Sharp Gears	Exposed gears	- Wires and skin could be cut by gears.	Cover gears with a shield.
7. Tipping Over	3 wheel design with too much acceleration or too fast of a stop	- The MicroMouse will be penalized a 3 second bonus for being “touched” to stand it up again	Place a spring on the base to prevent the MicroMouse from tipping forward or backward when either accelerating or stopping.
8. Collision	Loss of position in motor. Slippery path.	- May damage parts of the MicroMouse and will also incur a penalty in the competition. - Might also damage maze wall.	Limit the top speed of the MicroMouse, properly test all sensors before use, proper software algorithm implementation. Using good quality sensors. Use a feedback system. If using stepper motor, use a good angle resolution for the steps. Make sure that the robot path is not slippery.
9. Overheating	Circuitry using large amounts of power.	- Circuitry may become damaged and/or not work properly. - Can be harmful if touched by a person. - Can also cause circuitry meltdown.	Limit the amount of current to the motors, limit speed of the motors. Properly encase the circuitry to avoid human contact. Regulate power output so as to minimize heat release.
10. Noise	Stepper Motors	- It might be noisy for people attending the robot operation.	Reducing the noise by slowing the motors down. Use a servo motor. Mount the stepper motor inside a box that will eliminate part of the noise.
11. Mechanical Vibration	Using Stepper motor at low speed without having a micro-stepping capability in the system.	- It could lead to disconnection in the electric components like wires and sensors. - It could cause the robot to tip over.	Use a system with a micro-stepping capability. Fix the electrical connections. Provide the robot base with springs that will prevent the robot from tipping over.
12. Short	Reach of liquid,	- It could damage the	Seal the circuit board, electrical

	like water, to the electrical connections and components.	circuit board and electric components. - It could damage the motor.	components, and wire connections.

Risk and Hazard Assessment Table

After determining all of the possible hazards, it is important to assess the nature of them. The matrix below includes a hazard assessment and a risk assessment of each hazard. The hazard levels include catastrophic hazards (CA), critical hazards (CR), and controlled hazards (CN). Catastrophic is the most serious, possibly resulting in death or system destruction. Critical hazards may cause severe injury or major property damage, and controlled hazards may cause minor injury or property damage. There are four different severity classes ranking from Class I, which correlates to a catastrophic hazard, down to Class IV, which correlates to a controlled hazard that may be negligible. Also included in the table is a probability estimate which predicts the probability of each hazard actually occurring. These range from Estimate A, which is likely to occur immediately, to Estimate D, which is unlikely to occur.

Table 4 – Hazard and Risk Assessment

Hazard Assessment		***Risk Assessment***	
Hazard #	**Hazard Level**	**Severity Class**	**Probability Estimate**
1. Electrical Shock	CR	II	C
2. Fire	CA	I	D
3. Explosion	CA	I	D
4. Sharp Edges	CN	IV	C
5. Battery Acid	CN	IV	D
6. Sharp Gears	CR	II	D
7. Tipping Over	CR	II	C
8. Collision	CN	III	C
9. Overheating	CR	II	C
10. Noise	CN	IV	D
11. Mechanical Vibration	CN	III	C
12. Short	CR	II	C

Conclusions

The MicroMouse project alternate solutions design analysis document covered three main topics: the alternate solutions and their descriptions, the down-selection and analysis matrix, and the hazard analysis and risk assessment components. Three distinct solutions were discussed in detail considering the advantages and shortcomings of each. The down-selection and analysis matrix section described the methodology and results of considering each of the three alternate design solutions. This analysis helped us quantitatively rank and score our desired criteria for each solution. Alternate solution two was chosen because it scored the highest on the down-selection and analysis matrix and presented the least amount of hazards and risks. This solution incorporates the Handy Board kit with high-level interactive C software, a custom polycarbon plastic chassis, infrared sensors, and a stepper motor for accurate navigation. The last portion of the alternate design document considered the hazard and risk aspects of the MicroMouse project's best solution, alternate solution two. Table three displays possible hazards, their causes, effects, and possible preventive measures to counteract the hazards. Table four categorizes each hazard to a certain hazard level, severity class, and probability of occurrence. These tables clarify the hazard and risk assessments with the MicroMouse project. This document clearly shows that different solutions to the problem can be presented, but after critical analysis, one optimal solution can be chosen.

.

D.) Final Solution Changes

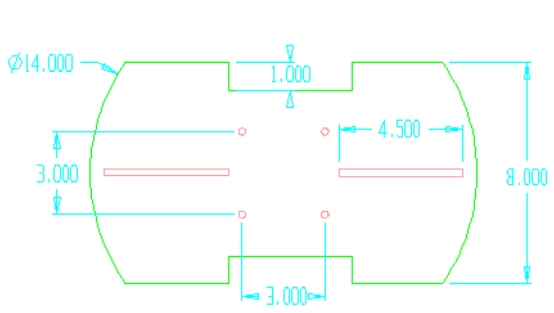
The final as-built design solution differed from the original optimal design from our alternate solutions analysis. We encountered numerous obstacles throughout the construction and testing phases of the robot. The design failures encouraged us to try different products and new creative design solutions. The major difference between the original design and the fully assembled robot was the installation of different motors. We replaced the DC motors and plastic gearbox with just servomotors. The gearbox had a lot of slop, which caused the gears to bind and thus setting our robot off course from the one wheel binding. The servomotors proved to provide more reliable mobility throughout the maze, and thus a better chance in solving the maze and winning the contest. Also, the original design had two side digital sensors for steering. However, we determined from testing that the digital sensors were not conductive to our steering needs. The analog sensors replaced the digital steering sensors and proved to perform better steering in the maze during our tests. The analog sensors were placed not at the rear of the chassis but in the front of the robot, so that the MicroMouse could sense as soon as it entered the next cell and steer better. This new placement decreased the amount of time spent exploring the maze and increased our maze solving efficiency. Another change was the encoder assembly. We custom-built an encoder using an infrared transmitter and receiver and an encoder wheel from a computer mouse.

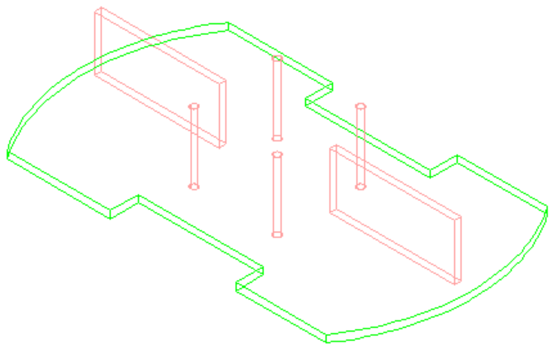
III. As Built Specifications

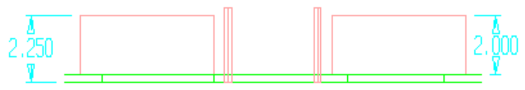
A.) Mechanical Specifications

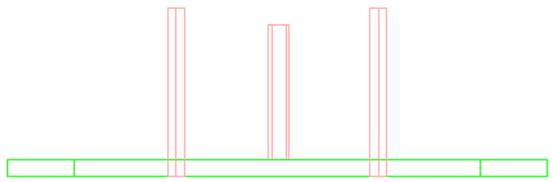
Chassis

The chassis of the MircoMouse is constructed entirely out of polycarbonate plastic, ¼ inch thick. The raw material was milled on a CNC machine using CADKEY and SurfCam as code writing software. Four holes where drilled through the top and bottom levels to allow for aluminum spacers to be inserted. These spacers, made of ¼ inch stock, were milled on a lathe to exact lengths. Two polycarbonate walls were installed to allow for added stability and mounting of sensors. Holes were also drilled in the chassis to allow for mounting of hardware components.





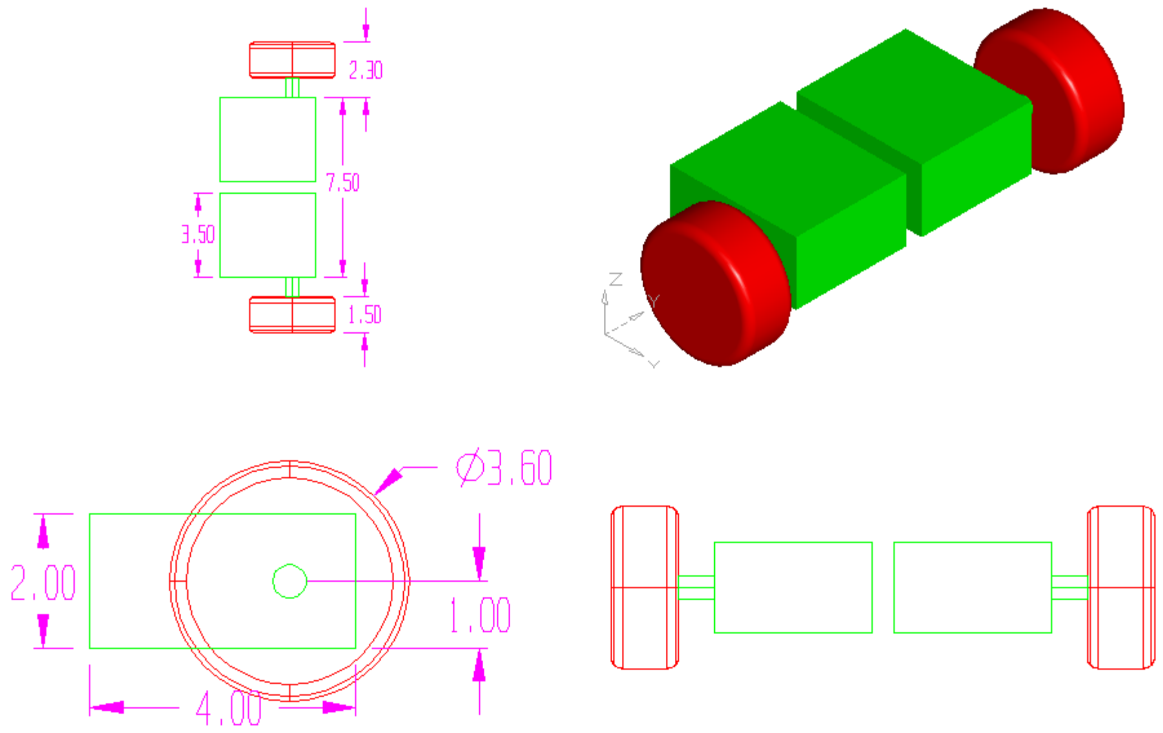




(Note Top Level same as Bottom without wheel wells)

Motor System

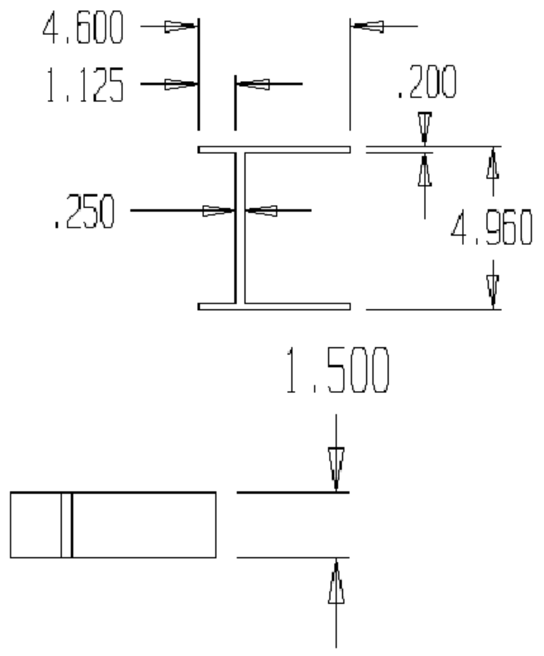
Servomotors were chosen because of their high torque ratings and precise movement. These motors were mounted in parallel on the underside of the lower level of the chassis. Rubber wheels were attached to the shafts with machine screws. The motors were reconfigured to rotate 360° by removing plastic stops within the motor housing.



Motor System

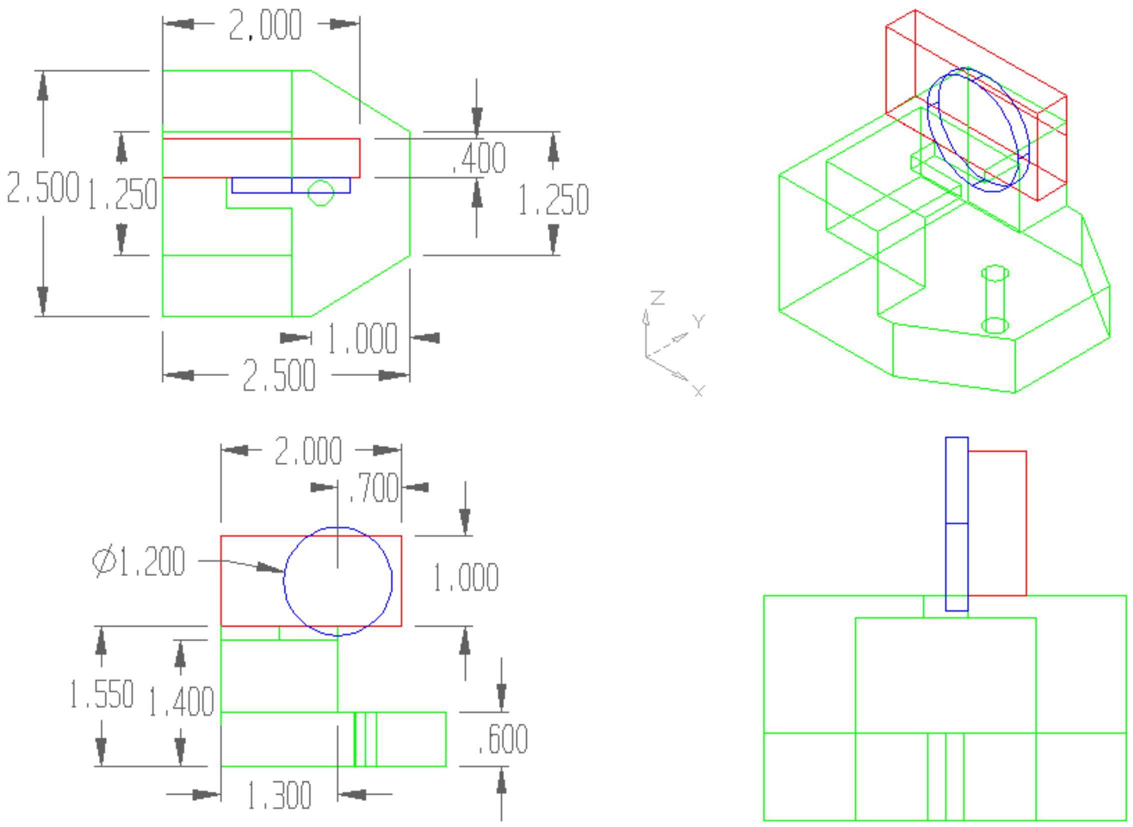
Mounting Bracket for Sensors

A bracket for mounting the is made out of 1/8 and ¼ inch polycarbonate plastic. This is mounted to the underside of the lower level of the chassis with 4-40 machine screws.. The bracket itself is held together by 4-40 machine screws.



Stabilizers (Casters and Encoder)

Two Tamyia ball casters are positioned to the rear of the motor assembly. They are fastened to the lower level of the chassis with 4-40 nuts and bolts. An encoder assembly made out of polycarbonate and a PC mouse wheel is fastened toward the front of the robot between the mounting brackets for the sensors. It is attached to the lower chassis by a single 4-40 screw and is spring loaded.



Ball Caster and Encoder Assembly

B.) Electrical Specifications

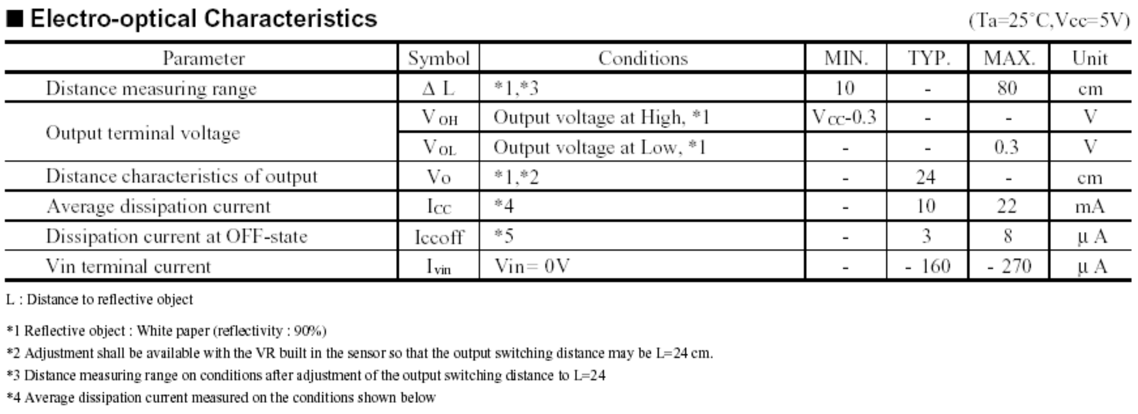
Sensors

Our MicroMouse contains five Sharp Infrared Rangers. Two of these are the Sharp GP2D05 Infrared Ranger, two are the Sharp GP2D120 Infrared Ranger, and one is the Sharp GP2D12 Infrared Ranger. The GP2D05 Rangers are used to detect the side walls of the maze for corridor detection and they will face outwards from the sides of the robot. The GP2D120 sensors are used for side wall detection for steering purposes. The GP2D12 are used to detect a wall in front of the robot and it is mounted on the front of the MicroMouse.

The output of the Sharp GP2D05 is digital. This means that it will output either a logic one or logic zero. The GP2D05 uses a fixed sensing range, and if an object is sensed inside of that range a logic zero will be output, otherwise a logic one is output if the object is out of range. This sensing range is adjustable with a small potentiometer on the sensing unit.

Interfacing the GP2D05 to the HandyBoard will be quite simple. The sensors use a four-pin connector. One pin is used for ground, one for the supply power (rated 4.4-7 volts), one for the output to the HC11 and one for the input from the HC11. To obtain a reading from this type of sensor, it must be triggered by an external source, which in our case will be the HC11. The input signal must be set low for typically 28-56ms, and then a reading can be taken from the output pin for 1ms or more.

The Sharp GP2D05 has the following characteristics:



■ Electro-optical Characteristics (Ta=25°C,Vcc=5V)

Parameter	Symbol	Conditions	MIN.	TYP.	MAX.	Unit
Distance measuring range	ΔL	*1,*3	10	-	80	cm
Output terminal voltage	V_{OH}	Output voltage at High, *1	V_{CC}-0.3	-	-	V
	V_{OL}	Output voltage at Low, *1	-	-	0.3	V
Distance characteristics of output	V_O	*1,*2	-	24	-	cm
Average dissipation current	I_{CC}	*4	-	10	22	mA
Dissipation current at OFF-state	Iccoff	*5	-	3	8	μ A
Vin terminal current	I_{vin}	Vin= 0V	-	- 160	- 270	μ A

L : Distance to reflective object

*1 Reflective object : White paper (reflectivity : 90%)

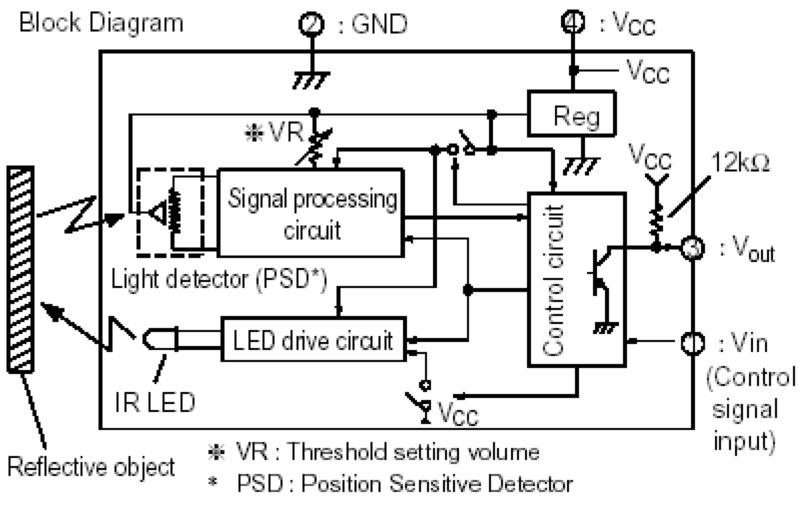
*2 Adjustment shall be available with the VR built in the sensor so that the output switching distance may be L=24 cm.

*3 Distance measuring range on conditions after adjustment of the output switching distance to L=24

*4 Average dissipation current measured on the conditions shown below

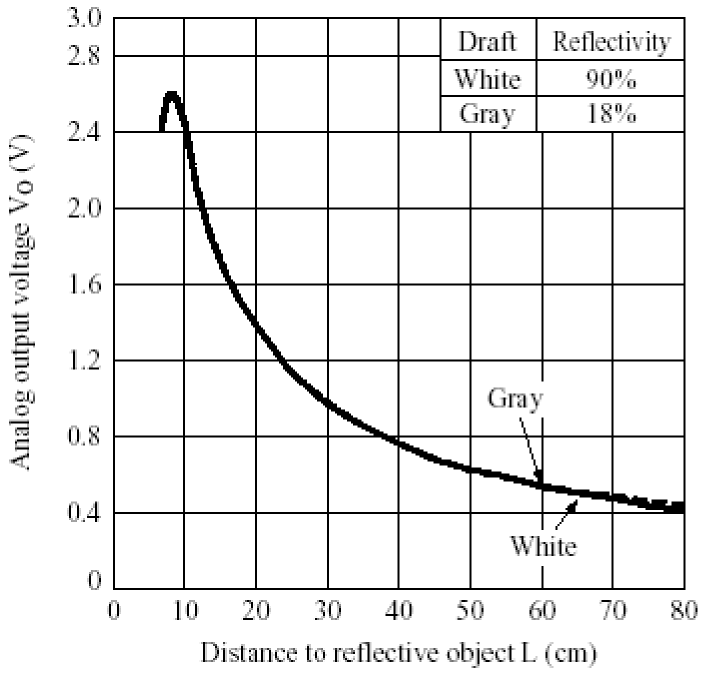
General Characteristics of Sharp GP2D05 Infrared Ranger

A schematic representation of the Sharp GP2D05 is shown below:



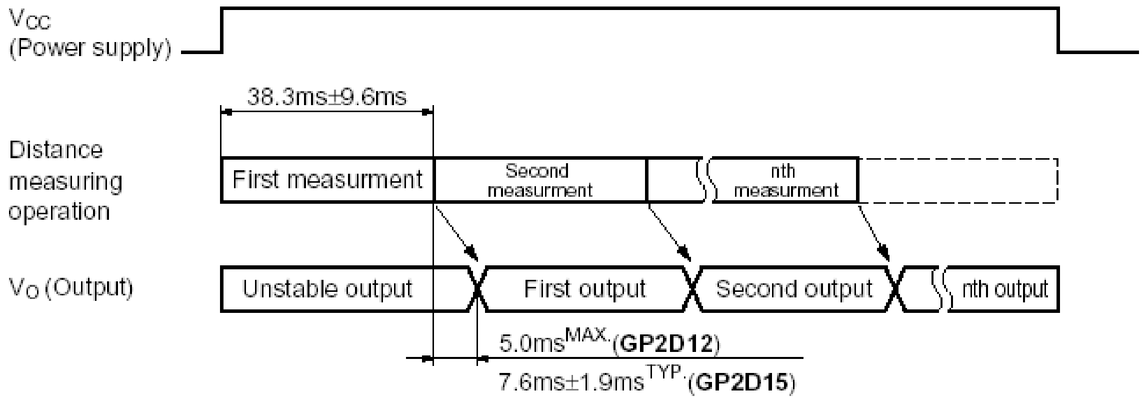
Schematic Diagram of Sharp GP2D05

The Sharp GP2D12, similar to the GP2D05, has a range of 10cm to 80cm. However, the output of the GP2D12 is analog and not digital. Since this sensor shall be mounted on the front of the robot, the analog characteristic will benefit us in the fact that we will be able to detect a wall in front of the MicroMouse at varying distances so that we may decelerate in time to avoid crashing into any walls. The analog output sends a voltage to the output pin corresponding to the distance to the object. As you can see in the graph below, a voltage near 0.4 volts corresponds to an object 80cm away, and a voltage near 2.4 volts corresponds to an object 10cm away. Also notice that this is not a linear scale.



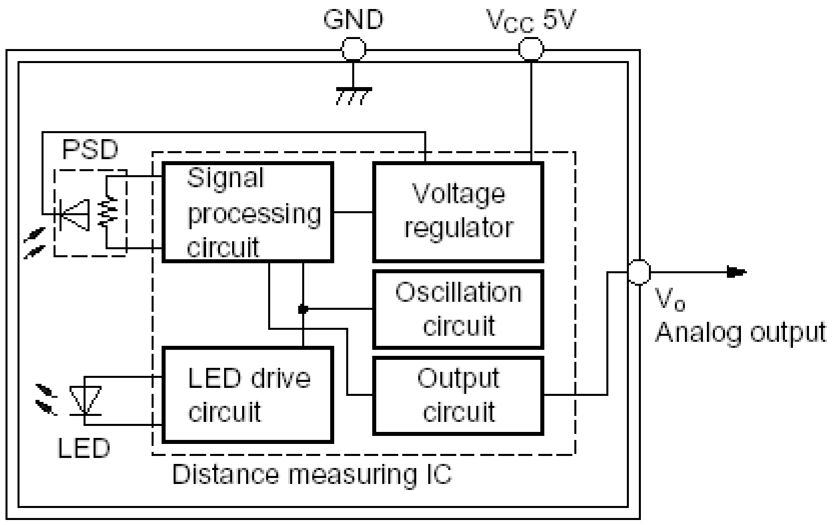
Output Voltage vs. Distance to reflected object for GP2D12

Interfacing the GP2D12 with the HC11 will also be pretty easy. This sensor uses a 3 pin connector. One pin is used for ground, one for Vcc (rated 4.5-5 volts), and one for the output. Since the GP2D12 fires continuously, no external input signal is needed to trigger the output. Readings can be taken constantly. However, we must keep in mind the delay of the readings to the output pin, as shown in the timing chart below.

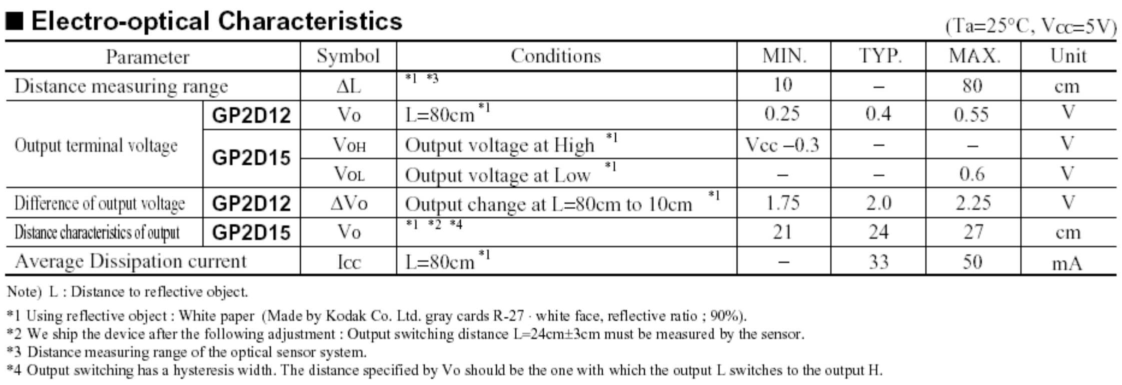


Timing chart for GP2D12

Another benefit of this sensor is that we do not have to worry about noise signals from the sensor. This sensor package contains its own processing unit and will output a crisp analog signal with very little noise interference. To obtain a usable sample, we must take the analog voltage from the sensor and send it to the HC11's Analog to Digital converter. This will return a digital hex value with which we can easily incorporate into our software. Below is the schematic for the GP2D12 and its characteristics.



Schematic Diagram of Sharp GP2D12 Infrared Ranger



■ Electro-optical Characteristics (Ta=25°C, V_{CC}=5V)

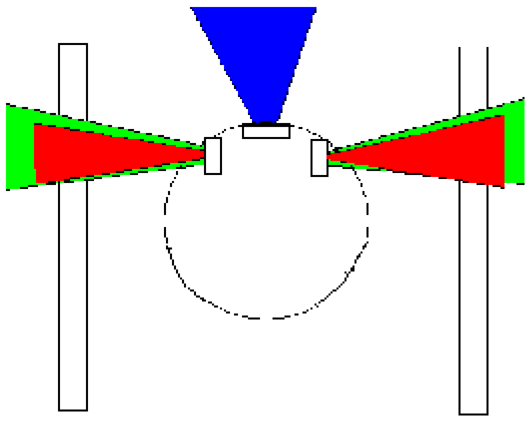
Parameter		Symbol	Conditions	MIN.	TYP.	MAX.	Unit
Distance measuring range		ΔL	*1 *3	10	–	80	cm
Output terminal voltage	GP2D12	Vo	L=80cm *1	0.25	0.4	0.55	V
	GP2D15	V_{OH}	Output voltage at High *1	Vcc –0.3	–	–	V
		V_{OL}	Output voltage at Low *1	–	–	0.6	V
Difference of output voltage	GP2D12	ΔVo	Output change at L=80cm to 10cm *1	1.75	2.0	2.25	V
Distance characteristics of output	GP2D15	Vo	*1 *2 *4	21	24	27	cm
Average Dissipation current		Icc	L=80cm *1	–	33	50	mA

Note) L : Distance to reflective object.

*1 Using reflective object : White paper (Made by Kodak Co. Ltd. gray cards R-27 · white face, reflective ratio ; 90%).
*2 We ship the device after the following adjustment : Output switching distance L=24cm±3cm must be measured by the sensor.
*3 Distance measuring range of the optical sensor system.
*4 Output switching has a hysteresis width. The distance specified by Vo should be the one with which the output L switches to the output H.

General Characteristics of GP2D12

The GP2D120 sensors have the same characteristics as the GP2D12 sensors. The only difference is that a special lens is used to decrease the range to between 4cm and 30cm for valid readings. We use these sensors on the sides of the robot in the front to aid in steering the robot away from any corridor walls.



Sensor Locations on MicroMouse

Using all these sensors together, as shown above, we will guide the MicroMouse through the maze. We will have a total of 2 Sharp GP2D05 sensors. These will be the green sensors seen above. The green sensors will detect the absence of a wall, i.e. detect corridors on the left or right of the MicroMouse. This is useful in turning situations and in the implementation of our flood-fill algorithm. The red sensors are the 2 Sharp GP2D120 sensors. These are used for steering purposes. We will continuously poll these sensors and read their output values. If a sensor finds a wall it will adjust the path of the MicroMouse accordingly. For example, if a wall is sensed on the right side, this must mean the robot has drifted too far right. We must then momentarily slow down the left wheel motor and speed up the right wheel motor to send the robot back on the right path. The blue sensor on the front of the MicroMouse will be the GP2D12. This senses the walls in front of the robot. This will help us to slow down before the ends of the corridor so that we don't crash into any walls, and allow us to see ahead to implement our algorithm more efficiently.

Hardware and Circuitry Schematics

The hardware and circuitry are two very important parts of the MicroMouse robot. There are several different circuits that must be implemented to create a fully functional and efficient MicroMouse. The Handy Board will prove to be ideal for the construction of the circuitry. The circuits that will be used include the CPU and memory, as well as motor drivers, digital inputs, analog inputs, infrared transmitter/receiver, power supply and interface/charger.

CPU and Memory

The main components that we are concerned with in this schematic are the 68HC11 microcontroller and the associated memory integrated circuits. All of the pin in/outs of the HC11 are explicitly labeled and routed. Outputs from the HC11 are routed to the memory elements, which include the 74HC373, a transparent octal latch for temporarily retaining data values and the 62256-100LP, 32K of static CMOS RAM for more permanent storage of software and data. Pins from the memory elements are then routed back to the HC11 inputs so that stored data can be accessed.

Motor Driver Circuit

The motor driver circuit is comprised of three main parts. First, digital outputs come from the HC11 and go into a 74HC374 integrated circuit, which is an octal latch to temporarily retain data values. From there the signals go into one of two L293D motor driver circuits. Those outputs are then run to a twelve-pin female header where the motors will be connected. This circuit allows four separate DC motors to be driven at 9.6 volts and as much as 1.1 amps of current. Each output line of this circuit has an LED connected to indicate if it is being driven or not.

Digital Input Circuit

The digital input circuit takes inputs from an external source through a nine-pin female header and the signals run into a 74HC244 tri-state bus driver IC. Each line has a 47 KOhm pull up resistor. The signals then go into the HC11 to be processed. There will be four digital infrared sensors that will use these digital input circuits. Also using digital inputs are the start and stop buttons for the Handy Board.

Analog Input Circuit

The analog input circuit is simply a seven-pin female connector to take signals from external sources. The signal lines have 47 KOhm pull up resistors and go straight into the HC11 for processing. The analog infrared sensor will use this circuitry.

Power Supply Circuit

The power supply circuit is basically the 8-cell AA nicad rechargeable battery pack that runs to a 2.5 amp Poly Switch fuse. There then is an LM2931Z-5.0 voltage regulating IC, which supplies power to memory. There is also a LM7805CTB voltage regulator for powering the motors. The power supply with give 9.6 volts to the Handy Board for powering all components including the HC11, memory, sensors, and motors.

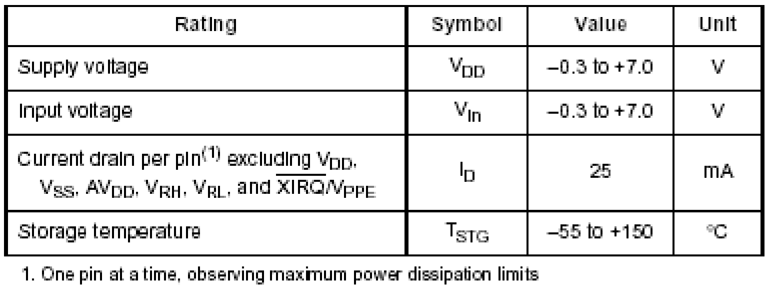
Interface/Charger Circuit
The charger unit in this circuit is comprised of a 2.1 mm coax power jack that takes power from an AC adapter that plugs into a standard wall outlet. At that point there is a wave rectifier circuit, which then goes into a LM2931Z voltage regulator IC. There is also an LED connected to indicate when power is applied. As for the interface portion, there is a DB 25-pin female connector that is connected to a MAX232CPE RS232 converter IC. There is also a RJ-12 connector available. These connectors are used for downloading the software onto the Handy Board.

Encoder Assembly
The encoder consisted of a very simple assembly and circuit. The small structure itself came right out of a PC scroll wheel mouse. We took only one of the two wheels inside the PC mouse and used that as a drag wheel underneath the MicroMouse. The drag wheel had very thin slits running from the center outward that allowed an infrared signal to pass through when the beam was not blocked. This created an infrared pulse that allowed us to track distance. Each pulse corresponded to a small distance and it was possible to add the pulses up and know exactly how far the MicroMouse had traveled. The infrared transmitter/receiver circuit itself consisted of a small infrared transmitter connected to power (+5v) and ground (0v), as well as a infrared receiver that was connected to power and ground with the output signal coming from the ground side of the receiver.

Conclusion
All of these individual circuits will work together in order to successfully control our MicroMouse robot. These circuits were integrated onto the Handy Board for projects like this and it will prove to be an efficient electronic controller device.

HC11 Microcontroller

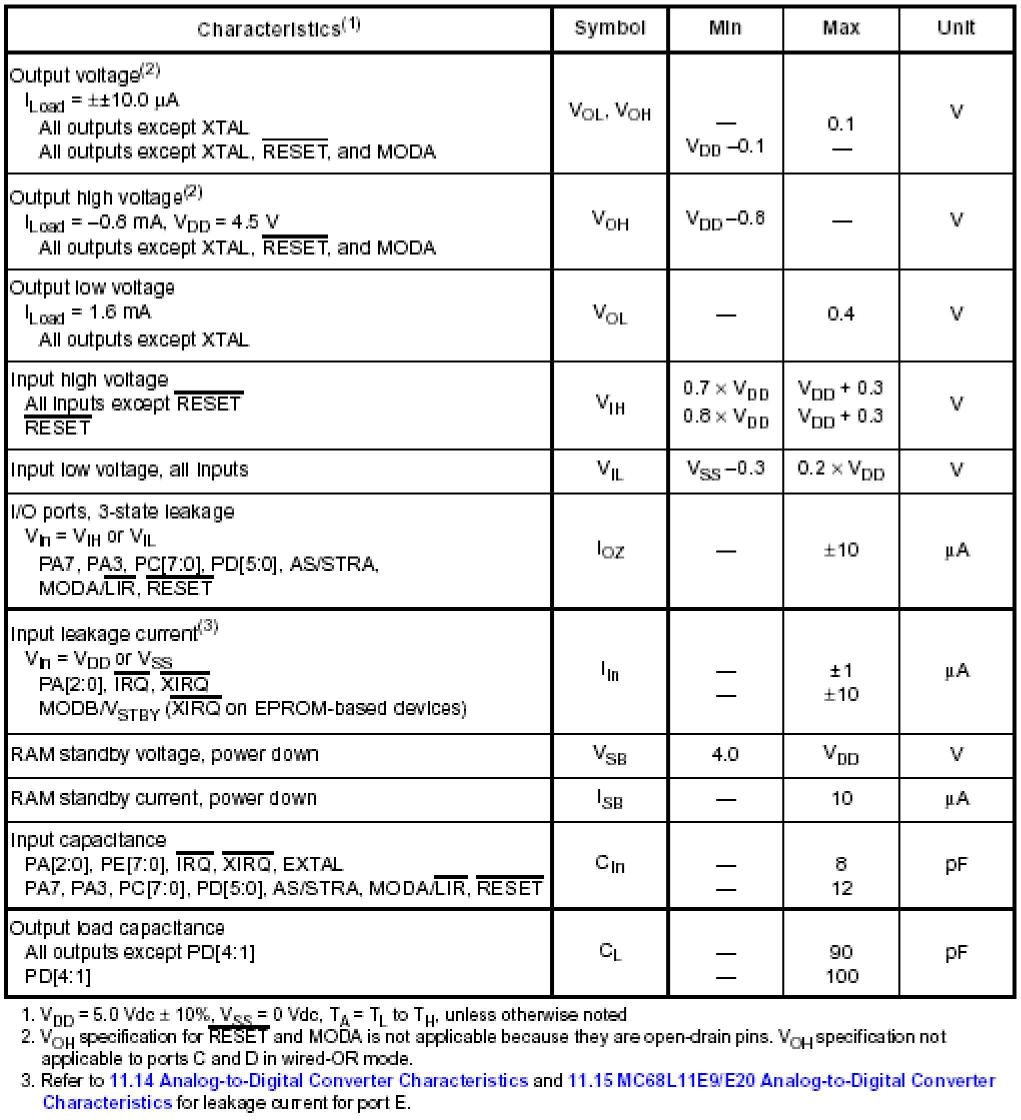
In order to execute the algorithm and control the MicroMouse, the Motorola 68HC11 microcontroller will be used. This microcontroller has a 2MHz clock frequency. It also contains a 5MHz bus at 5V for quick data transfer. This can also be reduced to 3MHz at 3V to save power. The HC11 contains two 8-bit accumulators for data operations or they can be combined for one 16-bit accumulator. It also has two 16-bit index registers and a 16-bit stack pointer. The HC11 manipulates many powerful bit instructions including add, subtract, and, or, shift commands, branch commands, plus many others. It is also capable of 16-bit by 16-bit integer and fractional division as well as 8-bit by 8-bit multiplication. These commands will be more than sufficient to execute our algorithm successfully. The microcontroller also has memory mapped I/O which is very useful for efficiently reading inputs and providing outputs. The HC11 also has a very extensive interrupt scheme, which can also prove to be very useful. There are 22 total interrupts including a real time interrupt, timer input capture, timer output compare, software interrupt, and pulse accumulator for a few examples that might be useful in our applications. For addressing, it has six very powerful modes including, immediate, direct, extended, indexed, inherent, and relative. The HC11 contains an 8-bit analog to digital converter, which will be useful to convert our analog input from our front sensor to a digital value to be manipulated within our algorithm. Overall the HC11 contains 11 dedicated input pins, 11 dedicated output pins, and 16 bi-directional. Some of the electrical characteristics for the HC11 are listed below.



Rating	Symbol	Value	Unit
Supply voltage	V_{DD}	–0.3 to +7.0	V
Input voltage	V_{In}	–0.3 to +7.0	V
Current drain per pin[1] excluding V_{DD}, V_{SS}, AV_{DD}, V_{RH}, V_{RL}, and $\overline{\text{XIRQ}}/V_{PPE}$	I_D	25	mA
Storage temperature	T_{STG}	–55 to +150	°C

1. One pin at a time, observing maximum power dissipation limits

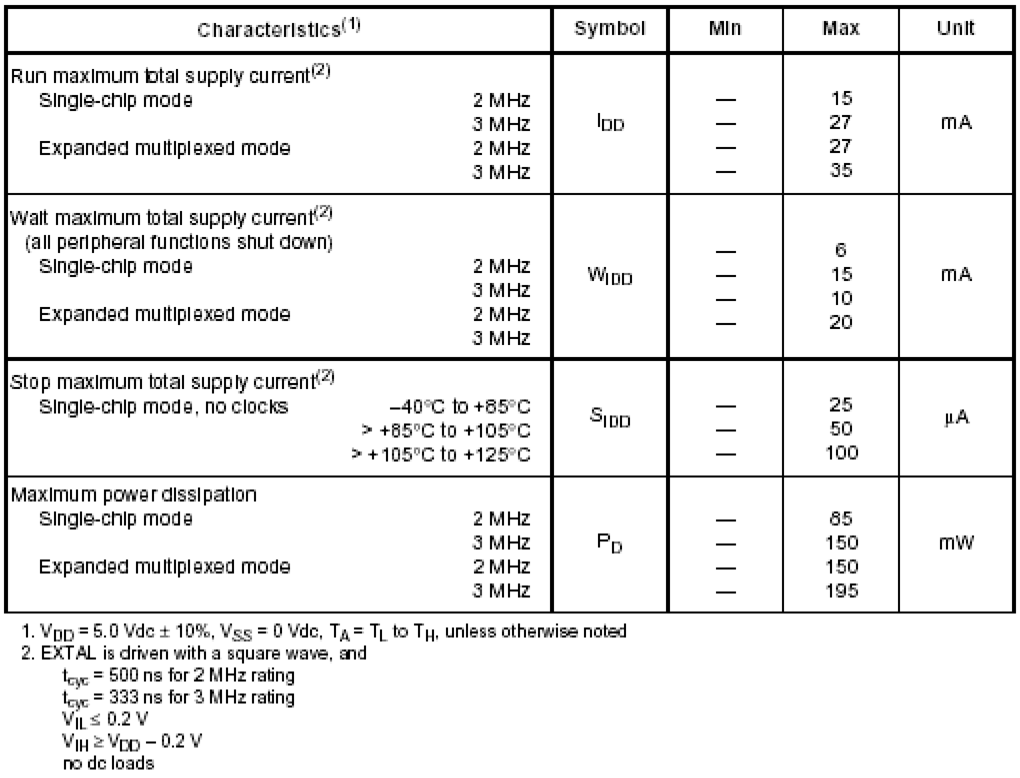
Maximum ratings for exposure without damage



Characteristics[1]	Symbol	Min	Max	Unit
Output voltage[2] I_{Load} = ±±10.0 µA All outputs except XTAL All outputs except XTAL, $\overline{\text{RESET}}$, and MODA	V_{OL}, V_{OH}	— V_{DD} –0.1	0.1 —	V
Output high voltage[2] I_{Load} = –0.8 mA, V_{DD} = 4.5 V All outputs except XTAL, $\overline{\text{RESET}}$, and MODA	V_{OH}	V_{DD} –0.8	—	V
Output low voltage I_{Load} = 1.6 mA All outputs except XTAL	V_{OL}	—	0.4	V
Input high voltage All inputs except $\overline{\text{RESET}}$ $\overline{\text{RESET}}$	V_{IH}	0.7 × V_{DD} 0.8 × V_{DD}	V_{DD} + 0.3 V_{DD} + 0.3	V
Input low voltage, all inputs	V_{IL}	V_{SS} –0.3	0.2 × V_{DD}	V
I/O ports, 3-state leakage V_{In} = V_{IH} or V_{IL} PA7, PA3, PC[7:0], PD[5:0], AS/STRA, MODA/$\overline{\text{LIR}}$, $\overline{\text{RESET}}$	I_{OZ}	—	±10	µA
Input leakage current[3] V_{In} = V_{DD} or V_{SS} PA[2:0], $\overline{\text{IRQ}}$, $\overline{\text{XIRQ}}$ MODB/V_{STBY} ($\overline{\text{XIRQ}}$ on EPROM-based devices)	I_{In}	— —	±1 ±10	µA
RAM standby voltage, power down	V_{SB}	4.0	V_{DD}	V
RAM standby current, power down	I_{SB}	—	10	µA
Input capacitance PA[2:0], PE[7:0], $\overline{\text{IRQ}}$, $\overline{\text{XIRQ}}$, EXTAL PA7, PA3, PC[7:0], PD[5:0], AS/STRA, MODA/$\overline{\text{LIR}}$, $\overline{\text{RESET}}$	C_{In}	— —	8 12	pF
Output load capacitance All outputs except PD[4:1] PD[4:1]	C_L	— —	90 100	pF

1. V_{DD} = 5.0 Vdc ± 10%, V_{SS} = 0 Vdc, T_A = T_L to T_H, unless otherwise noted
2. V_{OH} specification for $\overline{\text{RESET}}$ and MODA is not applicable because they are open-drain pins. V_{OH} specification not applicable to ports C and D in wired-OR mode.
3. Refer to 11.14 Analog-to-Digital Converter Characteristics and 11.15 MC68L11E9/E20 Analog-to-Digital Converter Characteristics for leakage current for port E.

DC Electrical Characteristics

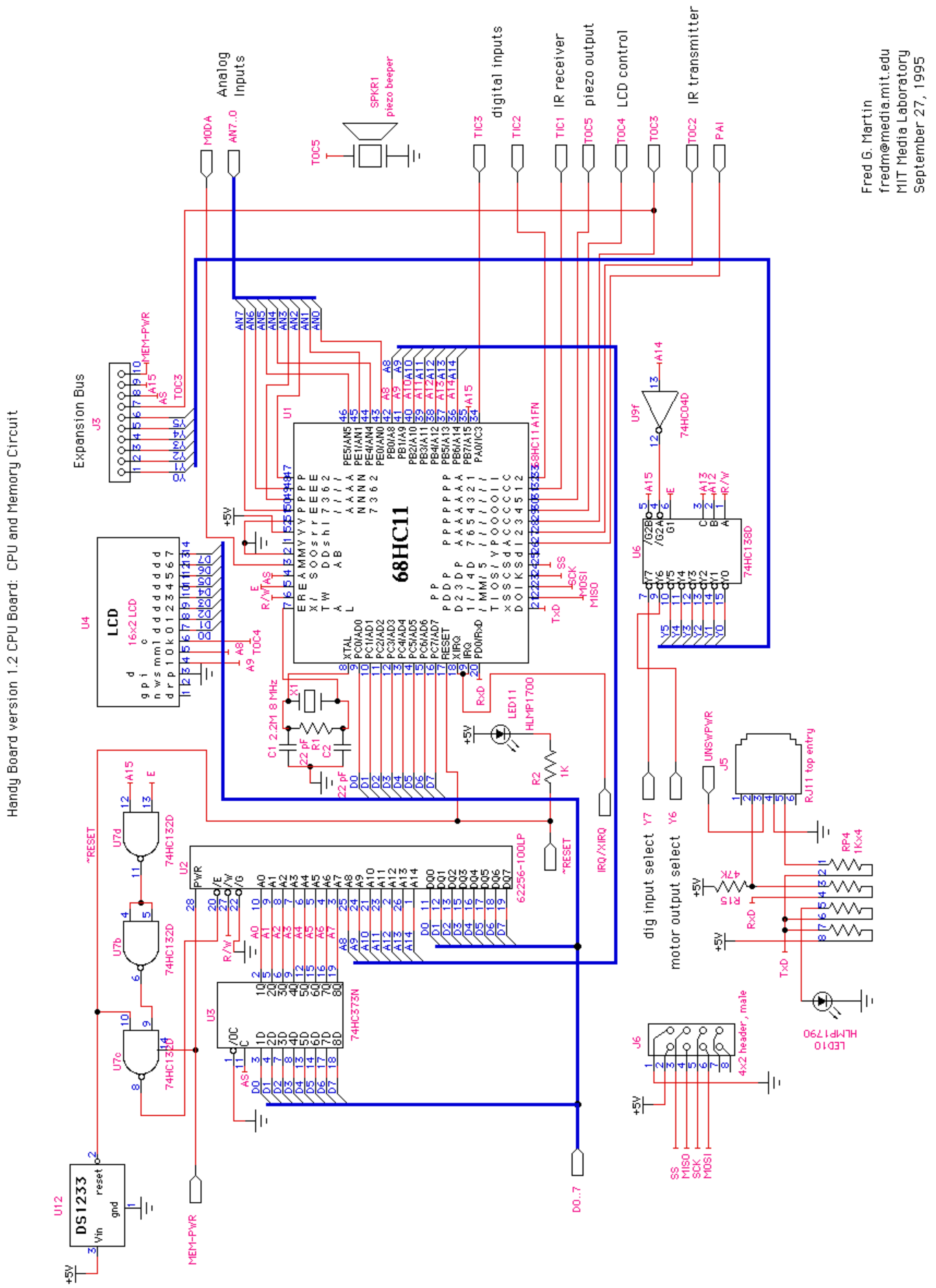


Characteristics[1]		Symbol	Min	Max	Unit
Run maximum total supply current[2]					
Single-chip mode	2 MHz	I_{DD}	—	15	mA
	3 MHz		—	27	
Expanded multiplexed mode	2 MHz		—	27	
	3 MHz		—	35	
Wait maximum total supply current[2] (all peripheral functions shut down)					
Single-chip mode	2 MHz	W_{IDD}	—	6	mA
	3 MHz		—	15	
Expanded multiplexed mode	2 MHz		—	10	
	3 MHz		—	20	
Stop maximum total supply current[2]					
Single-chip mode, no clocks	–40°C to +85°C	S_{IDD}	—	25	µA
	> +85°C to +105°C		—	50	
	> +105°C to +125°C		—	100	
Maximum power dissipation					
Single-chip mode	2 MHz	P_D	—	85	mW
	3 MHz		—	150	
Expanded multiplexed mode	2 MHz		—	150	
	3 MHz		—	195	

1. V_{DD} = 5.0 Vdc ± 10%, V_{SS} = 0 Vdc, $T_A = T_L$ to T_H, unless otherwise noted
2. EXTAL is driven with a square wave, and
   t_{cyc} = 500 ns for 2 MHz rating
   t_{cyc} = 333 ns for 3 MHz rating
   $V_{IL} \leq 0.2$ V
   $V_{IH} \geq V_{DD} - 0.2$ V
   no dc loads

Supply currents and power dissipation

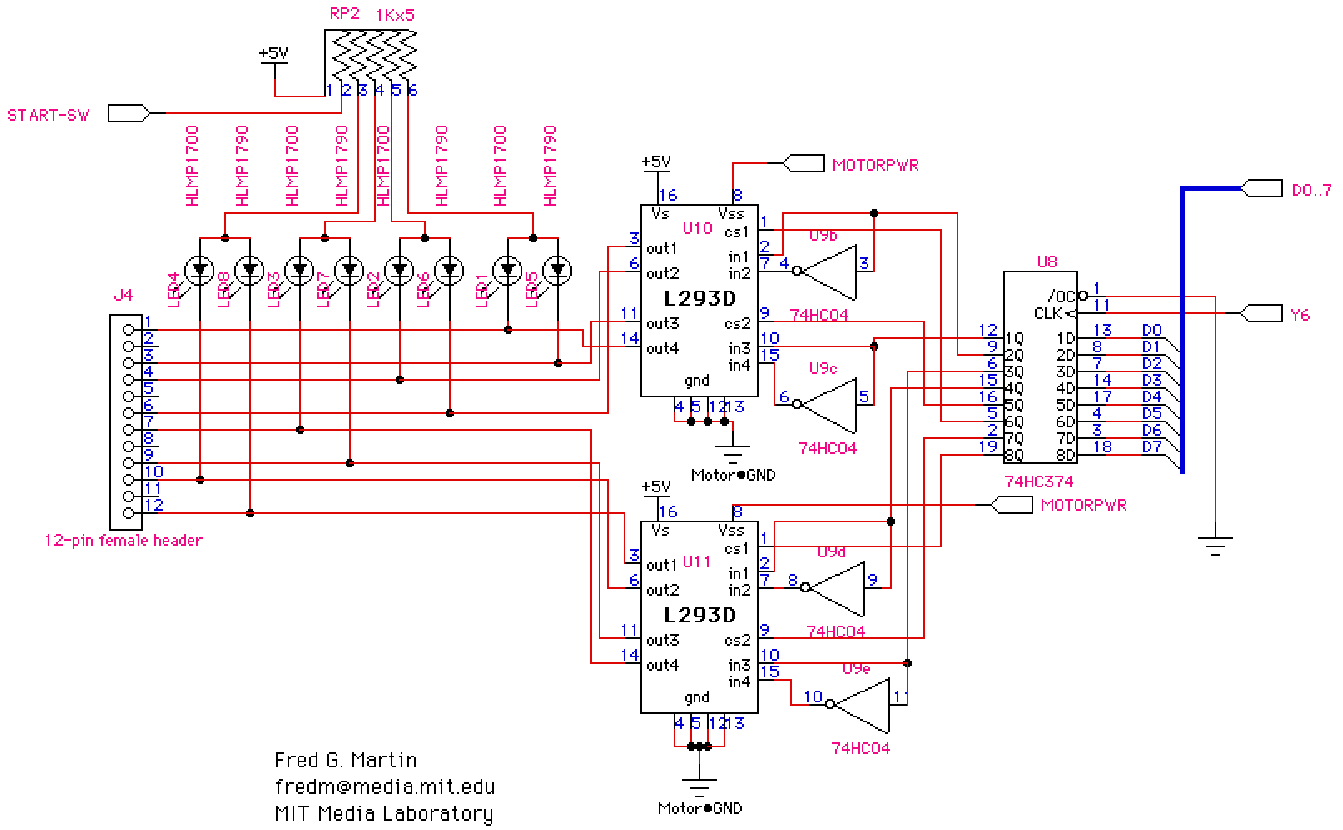
Electrical Circuits



Handy Board version 1.2 CPU Board: CPU and Memory Circuit

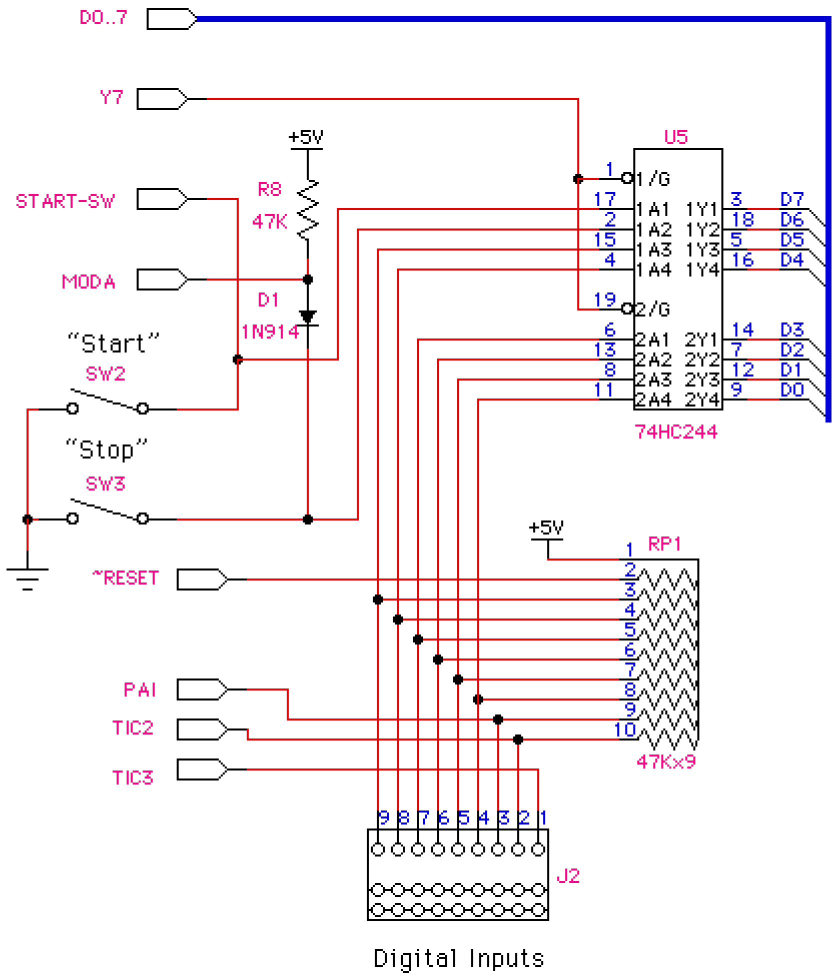
Fred G. Martin
fredm@media.mit.edu
MIT Media Laboratory
September 27, 1995

Handy Board version 1.2 CPU Board: Motor Output Circuit



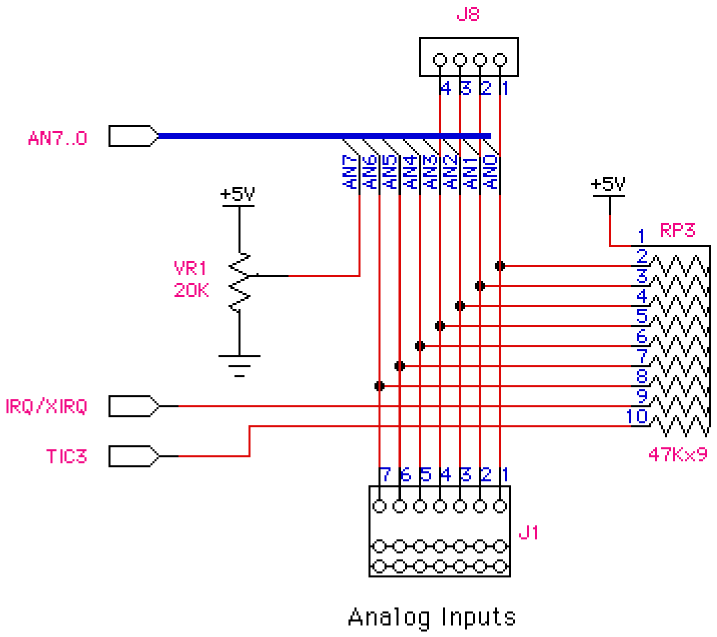
Fred G. Martin
fredm@media.mit.edu
MIT Media Laboratory
November 30, 1995

Handy Board version 1.2 CPU Board: Digital Input Circuit



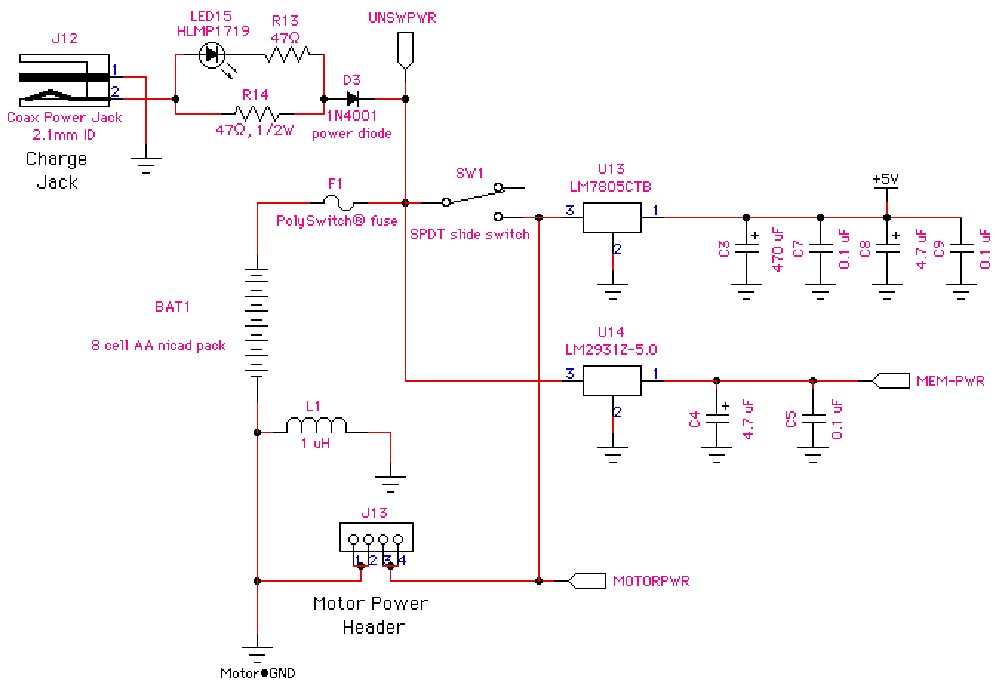
Fred G. Martin
fredm@media.mit.edu
MIT Media Laboratory
March 28, 1996

Handy Board version 1.2 CPU Board: Analog Input Circuit



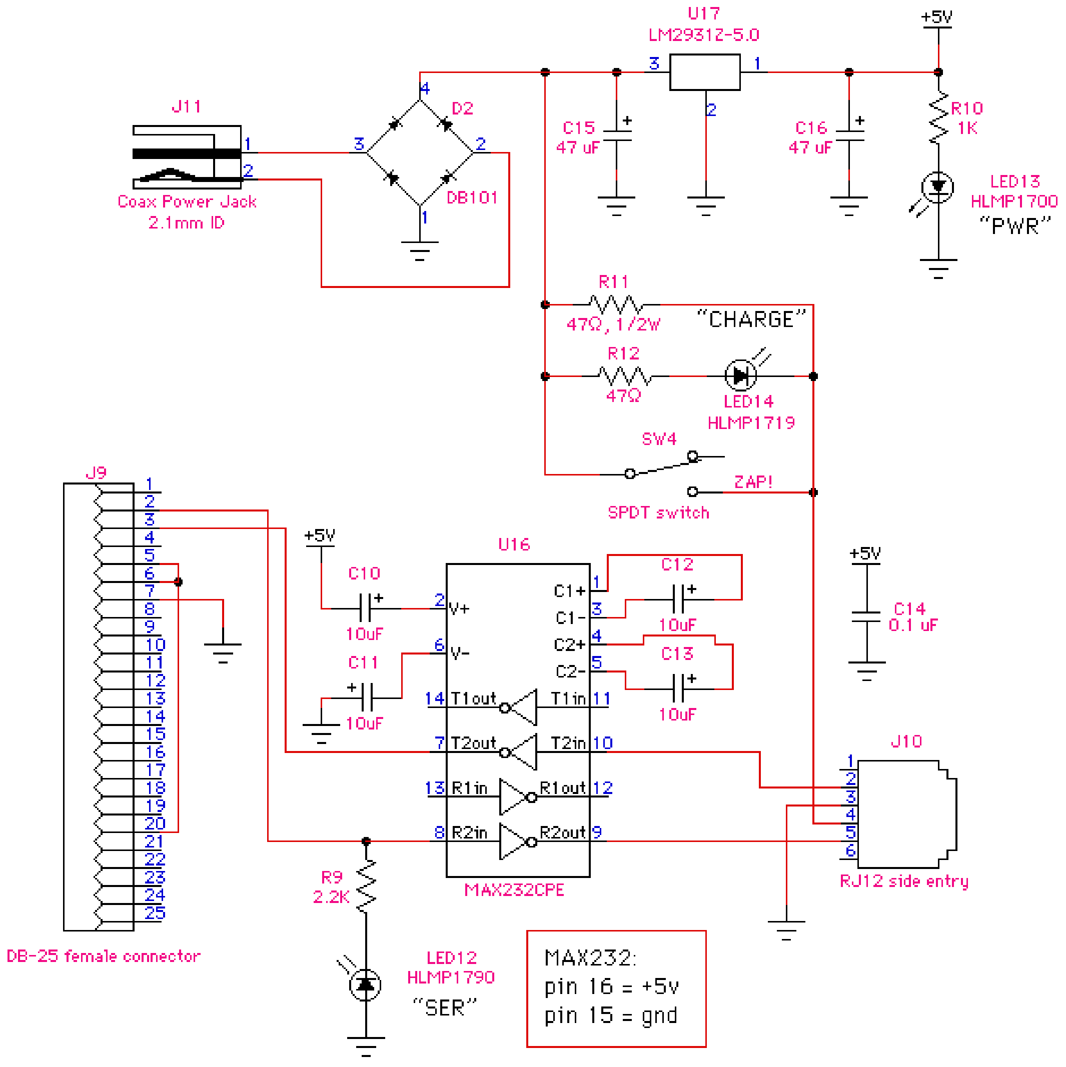
Fred G. Martin
fredm@media.mit.edu
MIT Media Laboratory
September 27, 1995

Handy Board version 1.2 CPU Board: Power Supply Circuit



Fred G. Martin
fredm@media.mit.edu
MIT Media Laboratory
September 27, 1995

Handy Board Interface/Charger Unit, version 1.0



Fred G. Martin
fredm@media.mit.edu
MIT Media Laboratory
September 27, 1995

C. Software Engineering

The software portion will discuss the setup of the maze, navigation algorithms, as well as a least-path or "flood-fill" algorithm and functions used to sense in the maze.

Interactive C and Features

Interactive C Version 3.2, by Newton Research Labs, is a windows based program developed for the compilation environment for many Motorola 6811 based robots and embedded systems. This program allows for easy user interaction. Users can type in expressions and have them compiled on the fly and run immediately, rather than waiting for lengthy compile and download cycles. Most importantly, it is fully compatible with the HandyBoard.

Interactive C v 3.2 was chosen for this project because it is a windows based program with a built-in editor, designed for easy editing and compiling. This feature eliminates the need to manually compile and edit using the freeware. Interactive C also has the ability to use predefined macros at the command line, which allow us to easily debug our MicroMouse software. Multidimensional arrays, (which are a necessity for describing the MicroMouse) are featured in this version. Also, it has an automatic interface to the HandyBoard for downloading the compiled code to the MicroMouse.

Maze Algorithm

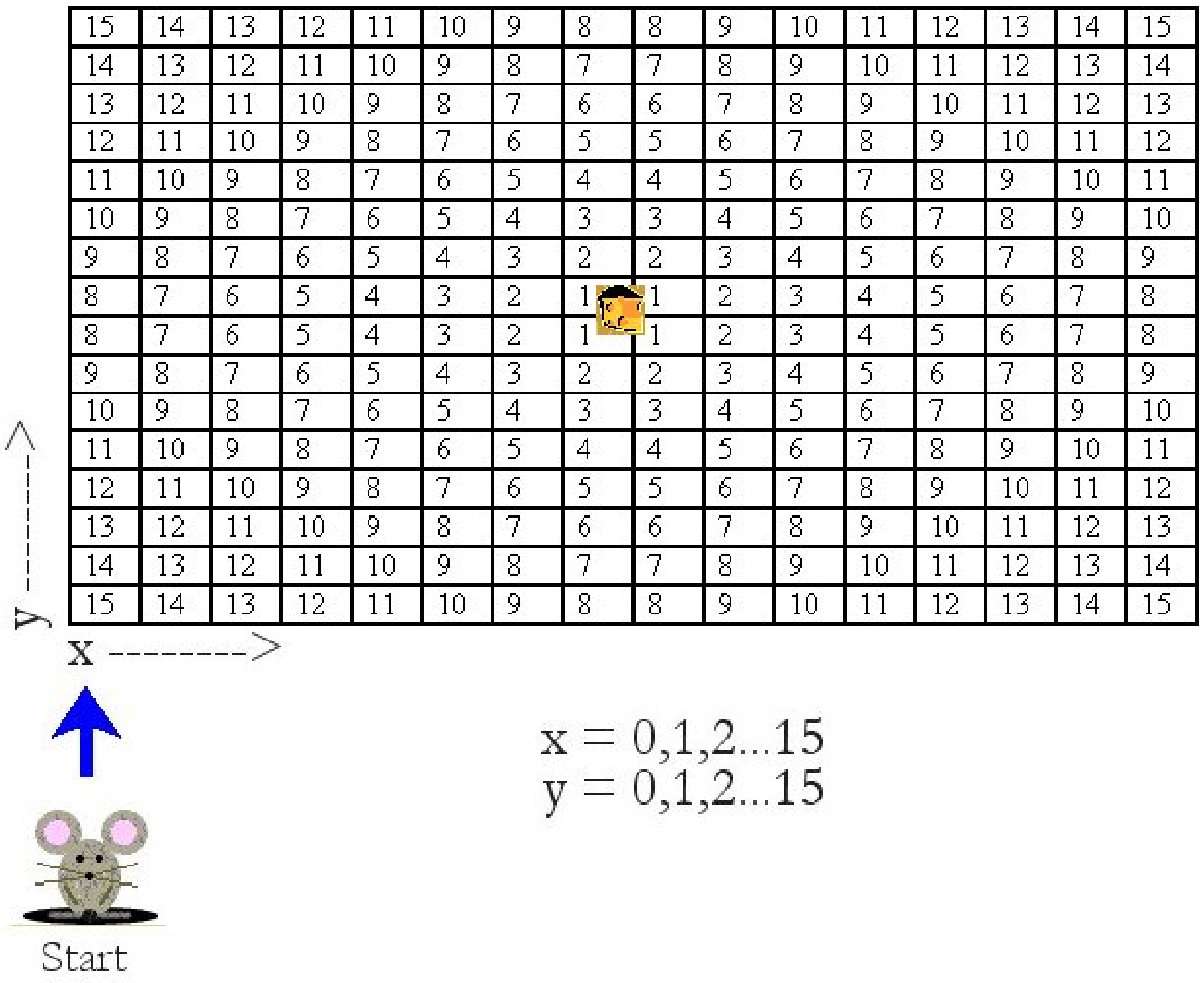
The MicroMouse maze is comprised of 16X16 or 256 squares as shown in the figure below. Each side of the square can have a wall opening except for the squares on the outer maze walls. The MicroMouse will start from the southwest corner of the maze and proceed northeast towards the center of the maze. A Cartesian coordinate system, with the origin centered at the southwest corner, will describe the current position and remaining maze squares. The positive x and y coordinates may have valid values between 0 and 15. The center or solution of the maze, designated in the figure as the cheese slice, will have 4 squares having a priority value of '1'. Each square is numbered according to its proximity and accessibility to the center of the maze. For example, the squares surrounding the corners of the four center squares are considered a '3' because although they may be near the center of the maze, the MicroMouse cannot go through corners to arrive at the center. Therefore, the MicroMouse must go through a surrounding square to reach the center of the maze. Thus a corner square is considered less desirable than the other squares.

When the program is initialized each square will begin with default variables and given a permanent location on the maze. The variables will be organized in an array for each square. The array for each square will be Cn[x,y,P,N,S,E,W,t]. The variables 'x' and 'y' represent the positive x-axis and y-axis Cartesian coordinate of the cell in the maze. The variable P is the priority number or ranking of the square relative to the center of the maze considering both proximity and accessibility. The variables 'N','S' ,'E' ,'W' are the absolute cardinal directions north, south, east, and west, with respect to the maze, of the possible wall openings in that particular cell. The variable 't' designates whether or not the cell has been "tried" or explored. This is important for our least-path searching algorithm because it is important to check and reorganize the cell's priority values depending on barrier walls. Also, the program will have stacks to determine which cells have been traversed and searched.

The program's operational block diagram has been oversimplified to allow for a top-view of all components. Firstly, when the program initializes, each cell is given a default array of values. The position values 'x' and 'y' are permanently defined, and the priority ranking 'P' is temporarily assigned. Each cell is initially given values of '0' for all other variables. Since the MicroMouse starts at the southwest corner of the maze, the shortest path and thus preferred path should be northeast, but turning the MicroMouse takes significant time and thus, it is preferred to go one direction for as long as it is possible. The robot starts moving straight North from the starting square. It senses and records whether there is an opening on each possible wall. The MicroMouse can also sense ahead one cell determining whether the cell north to the present cell has a wall open on its north wall.

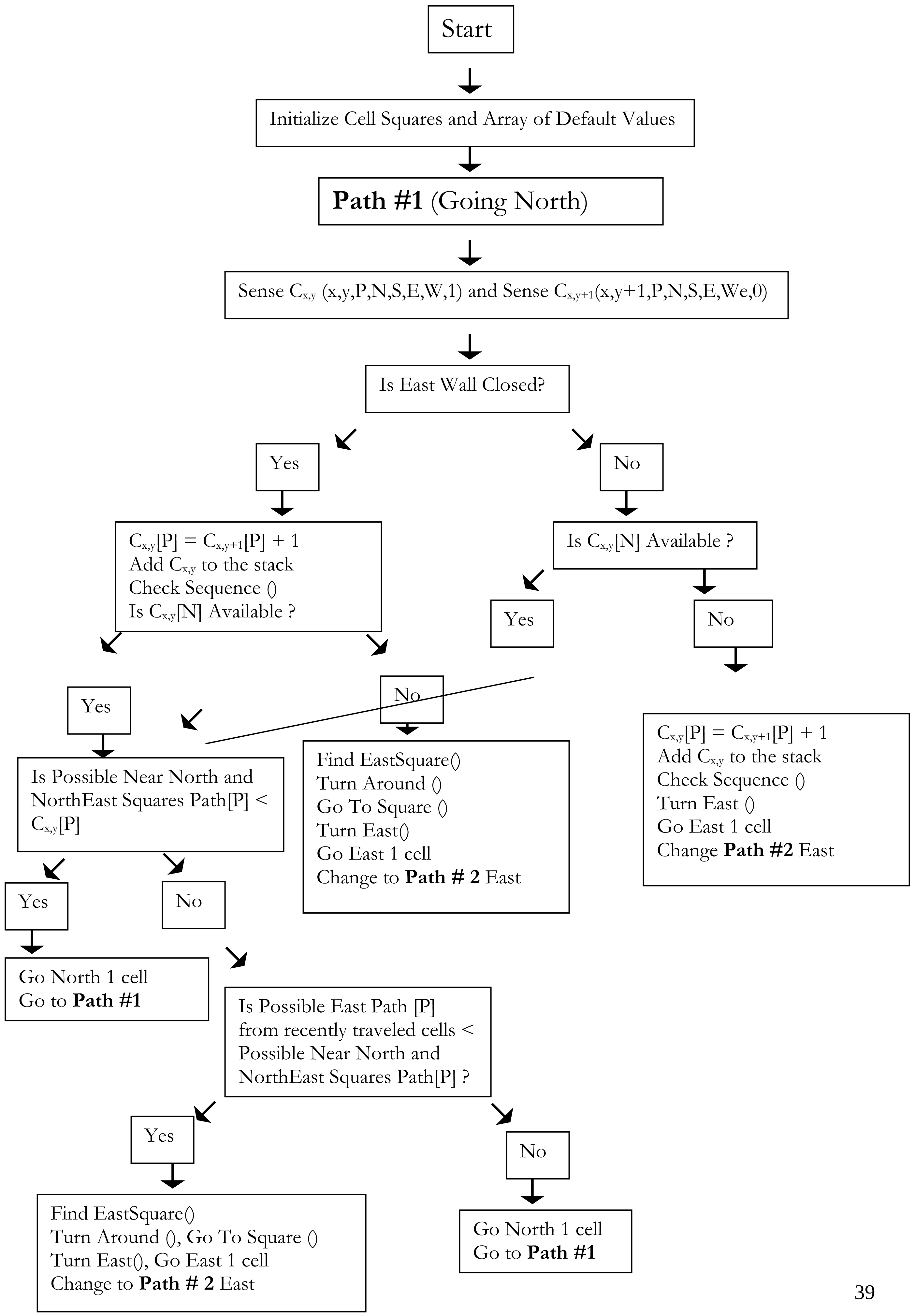
In the block diagram shown in Figure 2, $C_{x,y}$ (x,y,P,N,S,E,W,1) represents the current cell that the MicroMouse is traveling North through. After it senses each value, it must determine if there is an open wall to the East. If there is a blocked path there, the program must decrease the priority of the current cell relative to the previous cells. The priority of the current cell will be assigned one more than that of its highest ranking neighbor to the south. The current cell will be then added to a stack containing the cells already traversed. It is then necessary to run the sequence algorithm (flood fill algorithm) to check if that the cell's priority number allows a possible path to the center of the maze. For example, if the priority number is a '5', there must be a possible path that allows the mouse to travel from that square to the next which should be a '4', to the following cell, which should be a '3' to the center of the maze. The purpose of the sequence algorithm is to check this sequence, and if necessary, correct the priority of previously traversed cells to make a possible path. The sequence-checking algorithm also continually analyzes the maze and resorts the traversed cells priority values allowing the least-path to the center of the maze, is known as the flood-fill algorithm. A block diagram of this algorithm is shown below. Now if there is a path to the north available, the MicroMouse is clever enough to pick the best possible path to the center of the maze. Since it can see the north wall one cell ahead, it will consider this possible doorway and other factors to determine if its present course north is still the best choice. Other factors that the robot will consider are the priority of the cells to the north and northeast of the current cell. The mouse will also compare this to its current position priority and the result will mostly likely be to move north. If the best choice is north, the MicroMouse will then move one cell north and the north cycle will repeat. However, if the MicroMouse decides that it will not gain a better position or priority number in the maze by moving north (or if there is a wall blocking the north side of the current cell), it will consider going East. The MicroMouse will then consider if it can gain a better priority number by turning around in the maze and going back to the cell where it had an opening on its east wall previously. The robot will analyze the cells surrounding this possible cell, as it did previously, and determine if there is a possibility of gaining rank in the maze. If it is better to move east a cell, the MicroMouse will then turn around and move a cell east and then proceed on a different path cycle. If both considerations have failed, then there has been an error! The default path from the starting point is always north, even if there is an east wall available, because it takes too much time to turn. The flood-fill algorithm is displayed in on the following pages.

MicroMouse Maze Setup And Initial Priority Ranking by Location



15	14	13	12	11	10	9	8	8	9	10	11	12	13	14	15
14	13	12	11	10	9	8	7	7	8	9	10	11	12	13	14
13	12	11	10	9	8	7	6	6	7	8	9	10	11	12	13
12	11	10	9	8	7	6	5	5	6	7	8	9	10	11	12
11	10	9	8	7	6	5	4	4	5	6	7	8	9	10	11
10	9	8	7	6	5	4	3	3	4	5	6	7	8	9	10
9	8	7	6	5	4	3	2	2	3	4	5	6	7	8	9
8	7	6	5	4	3	2	1	1	2	3	4	5	6	7	8
8	7	6	5	4	3	2	1	1	2	3	4	5	6	7	8
9	8	7	6	5	4	3	2	2	3	4	5	6	7	8	9
10	9	8	7	6	5	4	3	3	4	5	6	7	8	9	10
11	10	9	8	7	6	5	4	4	5	6	7	8	9	10	11
12	11	10	9	8	7	6	5	5	6	7	8	9	10	11	12
13	12	11	10	9	8	7	6	6	7	8	9	10	11	12	13
14	13	12	11	10	9	8	7	7	8	9	10	11	12	13	14
15	14	13	12	11	10	9	8	8	9	10	11	12	13	14	15

$x = 0,1,2...15$
$y = 0,1,2...15$


Start
Initialize Cell Squares and Array of Default Values
Path #1 (Going North)
Sense $C_{x,y}$ (x,y,P,N,S,E,W,1) and Sense $C_{x,y+1}$(x,y+1,P,N,S,E,We,0)
Is East Wall Closed?
Yes
No
$C_{x,y}[P] = C_{x,y+1}[P] + 1$
Add $C_{x,y}$ to the stack
Check Sequence ()
Is $C_{x,y}[N]$ Available ?
Is $C_{x,y}[N]$ Available ?
Yes
No
Yes
No
Yes
No
Is Possible Near North and NorthEast Squares Path[P] < $C_{x,y}[P]$
Find EastSquare()
Turn Around ()
Go To Square ()
Turn East()
Go East 1 cell
Change to **Path # 2** East
$C_{x,y}[P] = C_{x,y+1}[P] + 1$
Add $C_{x,y}$ to the stack
Check Sequence ()
Turn East ()
Go East 1 cell
Change **Path #2** East
Yes
No
Go North 1 cell
Go to **Path #1**
Is Possible East Path [P] from recently traveled cells < Possible Near North and NorthEast Squares Path[P] ?
Yes
No
Find EastSquare()
Turn Around (), Go To Square ()
Turn East(), Go East 1 cell
Change to **Path # 2** East
Go North 1 cell
Go to **Path #1**

Functions and Routines

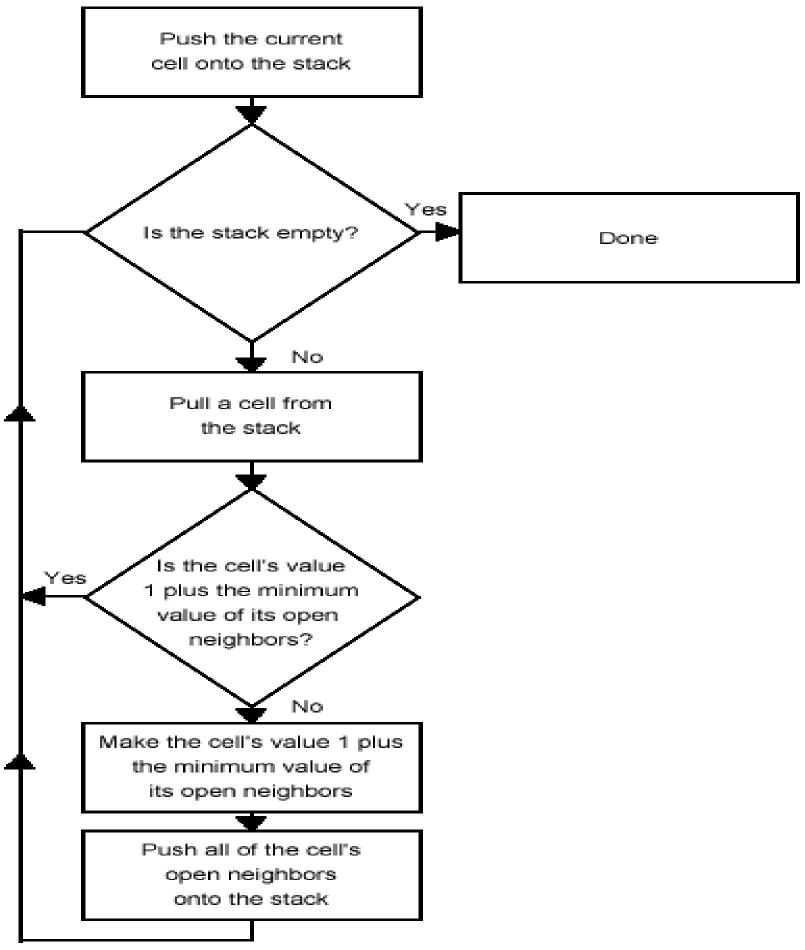
Sensors

The MicroMouse will be using both analog and digital sensors. Interactive C allows a simple function call to return the current values of the sensors.

int analog(int p)

> This function returns the value of the sensor port numbered p. The result is an integer between 0 and 255. If the analog() function is applied to a port that is implemented digitally in hardware, then the value 255 is returned if the *hardware* digital reading is 1 (as if a digital switch is open, and the pull up resistors are causing a high reading), and the value 0 is returned if the *hardware* digital reading is 0 (as if a digital switch is closed and pulling the reading near ground). Ports are numbered as marked.

Flood Fill Algorithm for Sequence Checking


Push the current cell onto the stack
Is the stack empty?
Yes
Done
No
Pull a cell from the stack
Is the cell's value 1 plus the minimum value of its open neighbors?
Yes
No
Make the cell's value 1 plus the minimum value of its open neighbors
Push all of the cell's open neighbors onto the stack

User Menu for LCD Screen

The MicroMouse will have a LCD screen and an input button for easy user interface. We will have a simple menu. This menu is needed to start the maze, test and debug the MicroMouse, and its diagnostic mode.

```
char a[3][14]={"Start ","Test","Diagnostic Mode"};
 int port,index=select_string(a,3);
if(index>-1 && index<2)
  port=select_int_value(a[index],0,27);
```

Some important diagnostic routines for the MicroMouse will be the following:

void control_panel()

This is a general purpose control panel to let the user view inputs, outputs, or set analog to digital thresholds. Pressing escape from the main menu or selecting "Quit" exits the control panel.

int view_average_port(int port,int samples)

This displays the analog reading of the given port until a button is pressed. If the button is choose, it then samples the reading at the given port, averages samples readings together, then prints and returns the average result. If the button pushed was escape, it returns -1. This will be particularly useful in allowing us to test the MicroMouse's sensors and assign sensed distance with output for use in the software maze navigation algorithm.

Software

```
int N,N1,N2,E,W=0;
float error,cutoff,error_cutoff,front_sensor,right_sensor,left_sensor,thetaright,thetaleft;
int wallleft,wallright;
int startcell;
int steps,encoder_process,steer_process,maze_process;
float orientation_error=0.035;
int stop=282;
int stop_servo=2560;
int right_full=3000;    //2920 with 5 volts
int left_full=2000;

//#use "lib_hb2.ic"
//#use "mousesteering8.ic"
// Global Variables (all variables must be declared globally)
int map[16][9];     // keeps track of the cell layout for each cell in the maze
int rank[16][9];    // keeps track of cell rank (distance from the center) in the maze
int temp=0;          // dummy variable
int temp1=0;         // dummy variable
int stack[300];      // array (quasi-stack) for the flood fill algorithm
int top;             // keeps track of the stack position
int j=0;             // another dummy variable
int k=0;             // another dummy variable
int a=0;             // dummy variable for current x-coordinate
int b=0;             // dummy variable for current y-coordinate
int c=0;             // dummy variable for flood-fill algorithm
int d=0;             // dummy variable for flood-fill algorithm
int o=8;             // orientation
int path=1;          // variable that keeps track of next desired cell location x,y coordinate
int setup=0;         // variable that keeps tracks of whether the mouse is 1.) going toward center 2.) returning to start
int solved=0;        // set at 1 if maze has been solved
int center=0;
int moved=0;
// ***********************************************************
// map[a][b] = value;
// value (byte) =  _ _ _ _  _ _ _ _  =  _ _ C T N S E W
// C = Current Cell, T = Traversed
// N = North, S = South, E = East, W = West
// Directions of cell are absolute
// Cell Wall Directions Sensed Are Relative
// Avoid using MSBit because of the signed integer problem
// ***********************************************************
void main(){
   poke(0x1009, 0x3c);            //set up digital output for digital sensors
   bit_set(0x1008, 0x08);         //set bits for digital sensors
   bit_set(0x1008, 0x10);
   encoder_process=start_process(encoder());
   //steer_process=start_process(steer());
   maze_process=start_process(maze());
}


void maze() {
   initialize_maze(); // sets the maze locations all to default values
   initialize_rank(); // sets the default cell rankings of the maze
   while (1) {
      // waits for user to press start
      while(!start_button());while(start_button());
      // main program loop
      startcell=1;
      reset_encoder(0);
      tone(.5);
      sense();
      set_vars();
      //printf("\n%d %d %d %d",N,N1,E,W);
      //while(!start_button());
      //check();
      update();
```

```
        move();
        printf("\nmove to %d %d o=%d Rank=%d", convert_x(path), convert_y(path), o,retrieve_rank_element(a,b));
        //while(!start_button());while(start_button());
        //printf("\n");

        set_cell();
        check();

        //printf("a %d  b %d", a, b);
        //while(!start_button());
        //printf("\n");
        //printf("\n");

        while(!stop_button()) {
           printf("\n%d",steps);

           if(steps>=stop){
              reset_encoder(0);
              tone(.5);
              sense();
              set_vars();
              //printf("\n%d %d %d %d",N,N1,E,W);
              //while(!start_button());
              update();
              //printf("%d,%d",a,b);
              move();
              printf("\nmove to %d %d o=%d Rank=%d", convert_x(path), convert_y(path),o,retrieve_rank_element(a,b));
              //while(!start_button());while(start_button());
              set_cell();
              check();

              //printf("a %d  b %d", a, b);
              //while(!start_button());
              //printf("\n");
              //printf("\n");


           }
        }
     }
  }

void set_motors(int left,int right){
   servo_a5_pulse=left;
   servo_a7_pulse=right;
}

void enable_motors(){
   servo_a7_init(1);
   servo_a5_init(1);
}

void forward(){
   set_motors(left_full,right_full);
   enable_motors();
}

void stopm(){
   servo_a7_init(0);
   servo_a5_init(0);
}

void digitalsense(){
   bit_clear(0x1008, 0x08);          //clear bits for digital sensors
   bit_clear(0x1008, 0x10);
   msleep(56L);                      //wait 56 ms before reading digital sensor

   if(digital(8)==0 && analog(4)<75){
      E=0;                           //store variables from digital sensor
      tone(.1);
```

```
        //while(!start_button());
    }
    if(digital(8)==1){
        E=1;
    }
    if(digital(10)==0 && analog(5)<75){
        W=0;                            //store variables from digital sensor
        tone(.1);
        //while(!start_button());
    }
    if(digital(10)==1){
        W=1;
    }

    bit_set(0x1008, 0x08);                  //reset bits for digital sensors
    bit_set(0x1008, 0x10);
}

void sense() {

    right_sensor=(float)analog(4);          //store values from analog sensors -- A4=right side sensor
    left_sensor=(float)analog(5);           //A5=left side sensor
    front_sensor=(float)analog(6);          //A6=front sensor
    thetaright=atan(front_sensor/right_sensor);     //find angles to check for orientation error
    thetaleft=atan(front_sensor/left_sensor);
    error=thetaright-thetaleft;                 //determine orientation error
    digitalsense();
}

void sense2(){

    if (left_sensor < 122.0) {                          /* A wall is detected on right side */
        wallright = 1;
        if (right_sensor < 136.0 && wallleft==1){           //if wall is too close to sensor dont sense wall
            wallright = 0;
        }
    }
    else {                                          /* A wall is not detected on right*/
        wallright = 0;
    }

    if (right_sensor < 122.0) {
        wallleft = 1;
        if (left_sensor < 136.0 && wallright==1){
            wallleft = 0;
        }
    }
    else {
        wallleft = 0;
    }
}

void steering_normal(){
    if(wallright==1){
        steer_left();
    }
    if (wallleft==1){
        steer_right();
    }

}

void steering_onewall(){
    if(E==1){
        if(right_sensor<123.){
            steer_left();
        }
        else{
            if(right_sensor>142.){
            }
        }


    }
    if(W==1){
        if(left_sensor<123.){
            steer_right();
        }
    }
}

void steer_left(){
    cutoff=left_sensor;
    while(wallright==1 && left_sensor<=cutoff+0.50){
        if(error<-.12){
            set_motors(left_full-50,right_full-50);
        }
        else{
            set_motors(left_full-25,right_full-25);
        }
        sense();
        sense2();
    }
    if(E==1 && W==1){
        correct_orientation();
    }
    //set_motors(15,11);
    stopm();
    msleep(20L);
    forward();
}

void steer_right(){
    cutoff=right_sensor;
    while(wallleft==1 && right_sensor<=cutoff+0.50){
        if(error>.12){
            set_motors(left_full+50,right_full+50);
        }
        else{
            set_motors(left_full+25,right_full+25);
        }
        sense();
        sense2();
    }
    if(E==1 && W==1){
        correct_orientation();
    }
    //set_motors(11,15);
    stopm();
    msleep(20L);
    forward();
}

void correct_orientation(){
    if(right_sensor<155.0 && left_sensor<155.0){
        //if (error<-orientation_error && front_sensor<135.){
        while (error<-orientation_error){           //facing right, must turn left to correct
            if(error<-.15){
                set_motors(left_full-50,right_full-50);
            }
            else{
                set_motors(left_full-25,right_full-25);
            }
            sense();
        }
        //set_motors(15,12);
        stopm();
        msleep(20L);
        forward();
        //}
        //if (error>orientation_error && front_sensor<135.){
        while(error>orientation_error){                  //facing left, must turn right to correct
            if(error>.15){
                set_motors(left_full+50,right_full+50);
            }
            else{
                set_motors(left_full+25,right_full+25);
            }
            sense();
        }
        stopm();
        //set_motors(12,15);
        msleep(20L);
        forward();
        //}
    }
}

// go one cell at a time
void go_straight() {
    if (startcell==1){
        printf("\ngo_straight");
        forward();
        //steer_process=start_process(steer());
        while(steps<125);
        //kill_process(steer_process);
        stopm();
        startcell=0;
        stop=125;
    }
    else{
        printf("\ngo_straight");
        forward();
        //steer_process=start_process(steer());
        if(E==0 && W==0){
            //kill_process(steer_process);
        }
        while(steps<282){
            printf("\n%d",steps);
        }
        //kill_process(steer_process);
        stopm();
        stop=282;
        //while(!start_button());
    }
    startcell=0;
}

void steer() {
    sense();
    sense2();
    //correct_orientation();
    if (W==1 && E==1){
        steering_normal();
    }
    //else{
    //  if(!(W==0 && E==0)){
    //    steering_onewall();
    //}
    //}
}

void turn_left() {
    printf("\nturn_left");
    //while(!start_button());
    reset_encoder(0);
    forward();
    while(steps<100);
    stopm();
    msleep(800L);
    set_motors(right_full,right_full);
    enable_motors();
    msleep(800L);
    stopm();
    msleep(450L);
    //kill_process(steer_process);
    reset_encoder(0);
    forward();
    while(steps<100);
    stopm();
    //reset_encoder(0);
    stop=100;
}

void turn_right() {

    printf("\nturn_right");
    //while(!start_button());
    reset_encoder(0);
    forward();
    while(steps<100);
    stopm();
    msleep(800L);
    set_motors(left_full,left_full);
    enable_motors();
    msleep(800L);
    stopm();
    msleep(450L);
    //kill_process(steer_process);
    reset_encoder(0);
    forward();
    while(steps<100);
    stopm();
    //while(!start_button());
    //reset_encoder(0);
    stop=100;
}

void turn_around(){
    printf("\nturn_around");
    reset_encoder(0);
    forward();
    while(steps<100);
    stopm();
    msleep(500L);
    set_motors(right_full,right_full);
    enable_motors();
    msleep(1700L);
    stopm();
    msleep(450L);
    reset_encoder(0);
    forward();
    while(steps<75);
    stopm();
    stop=75;
}

void set_vars(){
    if(front_sensor<53. && front_sensor>33.){              //2 walls ahead reading A6=38
        N2=1;
    }
    else{
        N2=0;
    }
    if(front_sensor<89. && front_sensor>54.){          //1 wall ahead reading A6=59
        N1=1;
    }
    else{
        N1=0;
    }
    if(front_sensor<155. && front_sensor>92.){
        N=1;
    }
    else{
        N=0;
    }
    /*    if(left_sensor<38. && left_sensor>24.){
        W1=1;
    }
    else{
        W1=0;
    }
*/
    /*
    if(right_sensor<44. && right_sensor>26.){
        E1=1;
    }
    else{
        E1=0;
    }
*/}

void encoder(){                    //enables encoder on digital port 7
    enable_encoder(0);
    while(1){
        steps=read_encoder(0);        //reads encoder on digital port 7
    }
}

void set_cell() {
    a = convert_x(path);
    b = convert_y(path);

}

void check() {
    if ((a==7 && b==7) || (a==7 && b==8) || (a==8 && b==7) || (a==8 && b==8)) {
        // the mouse has just solved the maze
        center=1;
        solved=1;
        setup=2;
        printf("\nMAZE SOLVED!!!");
        tone(2.);
        while(!start_button());while(start_button());

        // optional, play music
    }
    else{
        center=0;
    }
    if (a==0 && b==0 && solved==1) {
        // the mouse has returned the home
        setup=0;
        printf("\nHOME!!!");
        tone(2.);
        while(!start_button());while(start_button());
        // play music
    }
}

// speedier run depending on whether or not the maze has been solved
void update() {
    if ((solved==0) || ((solved==1) && (traversed(a,b)==0) && center==0)) {
        update_cell(a,b);
        update_cell_ranks();
        path = suggestpath();
    }
    else {
        path = suggestpath();
    }
```

```
}


void move() {
   // determine movement based on desired path variable and current orientation
   if (o==8 && moved==0) {
      if ((a==convert_x(path)) && (b+1 == convert_y(path))) {
         go_straight();
         moved=1;
      }
      if ((a==convert_x(path)) && (b-1 == convert_y(path))) {
         // turn 180
         turn_around();
         o = 4;
         moved=1;
         //go_straight();
      }
      if ((a+1==convert_x(path)) && (b == convert_y(path))) {
         turn_right();
         o = 2;
         moved=1;
         //go_straight();
      }
      if ((a-1==convert_x(path)) && (b == convert_y(path))) {
         turn_left();
         o = 1;
         moved=1;
         //go_straight();
      }
   }
   if (o==4 && moved==0) {
      if ((a==convert_x(path)) && (b+1 == convert_y(path))) {
         // turn 180
         turn_around();
         o = 8;
         moved=1;
         //go_straight();
      }
      if ((a==convert_x(path)) && (b-1 == convert_y(path))) {
         go_straight();
         moved=1;
      }
      if ((a+1==convert_x(path)) && (b == convert_y(path))) {
         turn_left();
         o = 2;
         moved=1;
         //go_straight();
      }
      if ((a-1==convert_x(path)) && (b == convert_y(path))) {
         turn_right();
         o = 1;
         moved=1;
         //go_straight();
      }
   }
   if (o==2 && moved==0) {
      if ((a==convert_x(path)) && (b+1 == convert_y(path))) {
         turn_left();
         o = 8;
         moved=1;
         //go_straight();
      }
      if ((a==convert_x(path)) && (b-1 == convert_y(path))) {
         turn_right();
         o = 4;
         //go_straight();
         moved=1;
      }
      if ((a+1==convert_x(path)) && (b == convert_y(path))) {
            go_straight();
            moved=1;
        }
        if ((a-1==convert_x(path)) && (b == convert_y(path))) {
            // turn 180
            turn_around();
            o = 1;
            moved=1;
            //go_straight();
        }
    }
    if (o==1 && moved==0) {
        if ((a==convert_x(path)) && (b+1 == convert_y(path))) {
            turn_right();
            o = 8;
            moved=1;
            //go_straight();
        }
        if ((a==convert_x(path)) && (b-1 == convert_y(path))) {
            turn_left();
            o = 4;
            //go_straight();
            moved=1;
        }
        if ((a+1==convert_x(path)) && (b == convert_y(path))) {
            // turn 180
            turn_around();
            o = 2;
            moved=1;
            //go_straight();
        }
        if ((a-1==convert_x(path)) && (b == convert_y(path))) {
            go_straight();
            moved=1;
        }
    }
    moved=0;
}



int retrieve_map_element(int a1, int b1) {
    return map[a1][b1];
}

int retrieve_rank_element(int a1, int b1) {
    return rank[a1][b1];
}

void set_map_element(int a1, int b1, int c1) {
    map[a1][b1] = c1;
}

void set_rank_element(int a1, int b1, int c1) {
    rank[a1][b1] = c1;
}

void initialize_maze() {
    for (j = 0; j < 16; j++)
      for (k = 0; k < 9; k++)
        set_map_element(j,k,0);
}

void initialize_rank() {

    temp = 15;
    temp1 = temp;
    for (j = 0; j < 8; j++){
        for (k = 0; k < 8; k++) {
            set_rank_element(j,k,temp1);
            temp1 = temp1-1;
        }
        temp = temp-1;
        temp1 = temp;
    }

    temp = 15;
    temp1 = temp;
    for (j = 15; j > 7; j--){
        for (k = 0; k < 8; k++) {
            set_rank_element(j,k,temp1);
            temp1 = temp1-1;
        }
        temp = temp-1;
        temp1 = temp;
    }

    temp=8;
    for(j=0;j<8;j++){
        set_rank_element(j,8,temp);
        temp=temp-1;
    }
    temp=8;
    for(j=15;j>7;j--){
        set_rank_element(j,8,temp);
        temp=temp-1;
    }
    /*
    set_rank_element(0,8,8);
    set_rank_element(1,8,7);
    set_rank_element(2,8,6);
    set_rank_element(3,8,5);
    set_rank_element(4,8,4);
    set_rank_element(5,8,3);
    set_rank_element(6,8,2);
    set_rank_element(7,8,1);
    set_rank_element(8,8,1);
    set_rank_element(9,8,2);
    set_rank_element(10,8,3);
    set_rank_element(11,8,4);
    set_rank_element(12,8,5);
    set_rank_element(13,8,6);
    set_rank_element(14,8,7);
    set_rank_element(15,8,8);
    */

    /*
    temp = 15;
    temp1 = temp;
    for (j = 15; j > 7; j--){
        for (k = 8; k > 7; k--) {
            set_rank_element(j,k,temp1);
            temp1 = temp1-1;
        }
        temp = temp-1;
        temp1 = temp;
    }

    temp = 15;
    temp1 = temp;
    for (j = 0; j < 8; j++){
        for (k = 8; k > 7; k--) {
            set_rank_element(j,k,temp1);
            temp1 = temp1-1;
        }
        temp = temp-1;
        temp1 = temp;
    }
*/
}
```

```
void update_cell(int a, int b) {
    // set cell as current and travesered
    temp = retrieve_map_element(a,b);
    temp1 = 48;
    temp = temp | temp1;
    set_map_element(a,b,temp);

    if (o == 8) {
        if (N == 1) {
            temp = retrieve_map_element(a,b);
            temp1 = 8;
            temp = temp | temp1;
            set_map_element(a,b,temp);
            /*if (b+1 < 9) {
                temp = retrieve_map_element(a,b+1);
                temp1 = 4;
                temp = temp | temp1;
                set_map_element(a,b+1,temp);
            }
            else {
            }*/
        }
        else {
            temp = retrieve_map_element(a,b);
            temp1 = 247;
            temp = temp & temp1;
            set_map_element(a,b,temp);
            /*if (b+1 < 9) {
                temp = retrieve_map_element(a,b+1);
                temp1 = 251;
                temp = temp & temp1;
                set_map_element(a,b+1,temp);
            }
            else {
            }  */
        }
        if (E == 1) {
            temp = retrieve_map_element(a,b);
            temp1 = 2;
            temp = temp | temp1;
            set_map_element(a,b,temp);
            /*if (a+1 < 16) {
                temp = retrieve_map_element(a+1,b);
                temp1 = 1;
                temp = temp | temp1;
                set_map_element(a+1,b,temp);
            }
            else {
            }*/
        }
        else {
            temp = retrieve_map_element(a,b);
            temp1 = 253;
            temp = temp & temp1;
            set_map_element(a,b,temp);
            /*if (a+1 < 16) {
                temp = retrieve_map_element(a+1,b);
                temp1 = 254;
                temp = temp & temp1;
                set_map_element(a+1,b,temp);
            }
            else {
            }*/
        }
        if (W == 1) {
            temp = retrieve_map_element(a,b);
            temp1 = 1;
            temp = temp | temp1;
            set_map_element(a,b,temp);
        /*if (a-1 >=0) {
            temp = retrieve_map_element(a-1,b);
            temp1 = 2;
            temp = temp | temp1;
            set_map_element(a-1,b,temp);
        }
        else {
        }*/
    }
    else {
        temp = retrieve_map_element(a,b);
        temp1 = 254;
        temp = temp & temp1;
        set_map_element(a,b,temp);
        /*if (a-1 >=0) {
            temp = retrieve_map_element(a-1,b);
            temp1 = 253;
            temp = temp & temp1;
            set_map_element(a-1,b,temp);
        }
        else {
        }*/
    }
    if ((N1==1) && (b+1<9) && (N==0)) {
        temp = retrieve_map_element(a,b+1);
        temp1 = 8;
        temp = temp | temp1;
        set_map_element(a,b+1,temp);

        /*        if (b+2<9) {
            temp = retrieve_map_element(a,b+2);
            temp1 = 4;
            temp = temp | temp1;
            set_map_element(a,b+2,temp);
        }
        else {
        }*/
    }
    else {
    }
    if ((N2==1) && (b+2<9) && (N1==0)) {
        temp = retrieve_map_element(a,b+2);
        temp1 = 8;
        temp = temp | temp1;
        set_map_element(a,b+2,temp);

        /*if (b+3<9) {
            temp = retrieve_map_element(a,b+3);
            temp1 = 4;
            temp = temp | temp1;
            set_map_element(a,b+3,temp);
        }
        else {
        }    */

    }
    else {
    }
}

if (o == 4) {
    if (N == 1) {
        temp = retrieve_map_element(a,b);
        temp1 = 4;
        temp = temp | temp1;
        set_map_element(a,b,temp);

        /*if (b-1>=0) {
            temp = retrieve_map_element(a,b-1);
            temp1 = 8;
        temp = temp | temp1;
        set_map_element(a,b-1,temp);
    }
    else {
    }*/

}
else {
    temp = retrieve_map_element(a,b);
    temp1 = 251;
    temp = temp & temp1;
    set_map_element(a,b,temp);

    /* if (b-1>=0) {
        temp = retrieve_map_element(a,b-1);
        temp1 = 247;
        temp = temp & temp1;
        set_map_element(a,b-1,temp);
    }
    else {
    }*/

}
if (E == 1) {
    temp = retrieve_map_element(a,b);
    temp1 = 1;
    temp = temp | temp1;
    set_map_element(a,b,temp);

    /*if (a-1>=0) {
        temp = retrieve_map_element(a-1,b);
        temp1 = 2;
        temp = temp | temp1;
        set_map_element(a-1,b,temp);
    }
    else {
    }*/

}
else {
    temp = retrieve_map_element(a,b);
    temp1 = 254;
    temp = temp & temp1;
    set_map_element(a,b,temp);

    /*if (a-1>=0) {
        temp = retrieve_map_element(a-1,b);
        temp1 = 253;
        temp = temp & temp1;
        set_map_element(a-1,b,temp);
    }
    else {
    }*/
}
if (W == 1) {
    temp = retrieve_map_element(a,b);
    temp1 = 2;
    temp = temp | temp1;
    set_map_element(a,b,temp);

    /*if (a+1 < 16) {
        temp = retrieve_map_element(a+1,b);
        temp1 = 1;
        temp = temp | temp1;
        set_map_element(a+1,b,temp);
    }
    else {
    }*/

}
```

```
    else {
        temp = retrieve_map_element(a,b);
        temp1 = 253;
        temp = temp & temp1;
        set_map_element(a,b,temp);

        /*if (a+1 < 16) {
            temp = retrieve_map_element(a+1,b);
            temp1 = 254;
            temp = temp & temp1;
            set_map_element(a+1,b,temp);
        }
        else {
        }*/
    }
    if ((N1==1) && (b-1>=0) && (N==0)) {
        temp = retrieve_map_element(a,b-1);
        temp1 = 4;
        temp = temp | temp1;
        set_map_element(a,b-1,temp);

        /*        if (b-2>=0) {
            temp = retrieve_map_element(a,b-2);
            temp1 = 8;
            temp = temp | temp1;
            set_map_element(a,b-2,temp);
        }
        else {
        }*/
    }
    else {
    }
    if ((N2==1) && (b-2>=0) && (N1==0)) {
        temp = retrieve_map_element(a,b-2);
        temp1 = 4;
        temp = temp | temp1;
        set_map_element(a,b-2,temp);

        /*if (b-3>=0) {
            temp = retrieve_map_element(a,b-3);
            temp1 = 8;
            temp = temp | temp1;
            set_map_element(a,b-3,temp);
        }
        else {
        } */

    }
    else {
    }
}

if (o == 2) {
    if (N == 1) {
        temp = retrieve_map_element(a,b);
        temp1 = 2;
        temp = temp | temp1;
        set_map_element(a,b,temp);
        /*if (a+1<16) {
            temp = retrieve_map_element(a+1,b);
            temp1 = 1;
            temp = temp | temp1;
            set_map_element(a+1,b,temp);
        }
        else {
        }*/
    }
    else {
        temp = retrieve_map_element(a,b);
        temp1 = 253;
    temp = temp & temp1;
    set_map_element(a,b,temp);

    /* if (a+1<16) {
        temp = retrieve_map_element(a+1,b);
        temp1 = 254;
        temp = temp & temp1;
        set_map_element(a+1,b,temp);
    }
    else {
    }*/
}
if (E == 1) {
    temp = retrieve_map_element(a,b);
    temp1 = 4;
    temp = temp | temp1;
    set_map_element(a,b,temp);

    /*if (b-1>=0) {
        temp = retrieve_map_element(a,b-1);
        temp1 = 8;
        temp = temp | temp1;
        set_map_element(a,b-1,temp);
    }
    else {
    }*/
}
else {
    temp = retrieve_map_element(a,b);
    temp1 = 251;
    temp = temp & temp1;
    set_map_element(a,b,temp);

    /*if (b-1>=0) {
        temp = retrieve_map_element(a,b-1);
        temp1 = 247;
        temp = temp & temp1;
        set_map_element(a,b-1,temp);
    }
    else {
    }  */
}
if (W == 1) {
    temp = retrieve_map_element(a,b);
    temp1 = 8;
    temp = temp | temp1;
    set_map_element(a,b,temp);

    /*if (b+1<9) {
        temp = retrieve_map_element(a,b+1);
        temp1 = 4;
        temp = temp | temp1;
        set_map_element(a,b+1,temp);
    }
    else {
    }*/
}
else {
    temp = retrieve_map_element(a,b);
    temp1 = 247;
    temp = temp & temp1;
    set_map_element(a,b,temp);

    /*if (b+1<9) {
        temp = retrieve_map_element(a,b+1);
        temp1 = 251;
        temp = temp & temp1;
        set_map_element(a,b+1,temp);
    }
    else {
            }  */

        }
        if ((N1==1) && (a+1<16) && (N==0)) {
            temp = retrieve_map_element(a+1,b);
            temp1 = 2;
            temp = temp | temp1;
            set_map_element(a+1,b,temp);
            /*if (a+2<16) {
                temp = retrieve_map_element(a,b+1);
                temp1 = 1;
                temp = temp | temp1;
                set_map_element(a,b+1,temp);
            }
            else {
            }  */
        }
        else {
        }
        if ((N2==1) && (a+2<16) && (N1==0)) {
            temp = retrieve_map_element(a+2,b);
            temp1 = 2;
            temp = temp | temp1;
            set_map_element(a+2,b,temp);

            /*      if (a+3<16) {
                temp = retrieve_map_element(a,b+1);
                temp1 = 1;
                temp = temp | temp1;
                set_map_element(a,b+1,temp);
            }

            else {
            }  */
        }
        else {
        }
    }

    if (o == 1) {
        if (N == 1) {
            temp = retrieve_map_element(a,b);
            temp1 = 1;
            temp = temp | temp1;
            set_map_element(a,b,temp);

        }
        else {
            temp = retrieve_map_element(a,b);
            temp1 = 254;
            temp = temp & temp1;
            set_map_element(a,b,temp);
        }
        if (E == 1) {
            temp = retrieve_map_element(a,b);
            temp1 = 8;
            temp = temp | temp1;
            set_map_element(a,b,temp);

        }
        else {
            temp = retrieve_map_element(a,b);
            temp1 = 247;
            temp = temp & temp1;
            set_map_element(a,b,temp);
        }
        if (W == 1) {
            temp = retrieve_map_element(a,b);
            temp1 = 4;
            temp = temp | temp1;
            set_map_element(a,b,temp);
        }
        else {
            temp = retrieve_map_element(a,b);
            temp1 = 251;
            temp = temp & temp1;
            set_map_element(a,b,temp);
        }
        if ((N1==1) && (a-1>=0) && (N==0)) {
            temp = retrieve_map_element(a-1,b);
            temp1 = 1;
            temp = temp | temp1;
            set_map_element(a-1,b,temp);
        }
        else {

        }
        if ((N2==1) && (a-2>=0) && (N1==0)) {
            temp = retrieve_map_element(a-2,b);
            temp1 = 1;
            temp = temp | temp1;
            set_map_element(a-2,b,temp);
        }
        else {

        }
        /*        if ((E1==1) && (b+1<9) && (E==0)) {
            temp = retrieve_map_element(a,b+1);
            temp1 = 8;
            temp = temp | temp1;
            set_map_element(a,b+1,temp);
        }
        if ((W1==1) && (b-1 >=0) && (W==0)) {
            temp = retrieve_map_element(a,b-1);
            temp1 = 4;
            temp = temp | temp1;
            set_map_element(a,b-1,temp);
        }
*/
    }
}

/* Flood Fill Algorithm Routines Begin*/

// Flood Fill Algorithm (NOT ANY CODE!!) technique was learned from
// http://homepage.mac.com/SBenkovic/MicroMouse/introduction/mfloodfill.html

void update_cell_ranks() {
    // empty the stack
    for (j=0; j<300; j++) {
        stack[j] = 0;
    }
    top = 0;
    push(return_location(a,b));
    // the current cell in which the mouse is in
    while(top != 0) {
        temp = pop();
        c = convert_x(temp);     // the current x coordinate converted from location byte temp
        d = convert_y(temp);     // the current y coordinate converted from location byte temp
        // check to see if there are any accessible neighboring cells
        if(open_neighbor(c,d) == 1) {
            temp = min_neighbor_rank(c,d); // the minimum rank of the open neighbors
            if(temp+1 == retrieve_rank_element(c,d)){
                // ok , do nothing
            }
            else{
                // change this cell to 1 + the minimum value of its neighbors
                set_rank_element(c,d,temp+1);

                // push all of the cell's open neighbors onto the stack to be checked
                push_open_neighbors(c,d);
            }
        }
        else {
            // no accesible cells error
        }
    }
}

/* Flood Fill Algorithm End */

// returns the minimum value of all the accessible neighbors to the cell
int min_neighbor_rank(int c,int d) {
    // get any neighboring rank value
    if(open_north(c,d) == 1) {
        temp=retrieve_rank_element(c,d+1);
    }
    if(open_south(c,d) == 1) {
        temp=retrieve_rank_element(c,d-1);
    }
    if(open_east(c,d) == 1) {
        temp=retrieve_rank_element(c+1,d);
    }
    if(open_west(c,d) == 1) {
        temp=retrieve_rank_element(c-1,d);
    }

    // determine minimum and return it
    if ((open_north(c,d) == 1) && (retrieve_rank_element(c,d+1) < temp)) {
        temp = retrieve_rank_element(c,d+1);
    }
    if (((open_south(c,d)==1)) && (retrieve_rank_element(c,d-1) < temp)) {
        temp = retrieve_rank_element(c,d-1);
    }
    if ((open_east(c,d)==1) && (retrieve_rank_element(c+1,d) < temp)) {
        temp = retrieve_rank_element(c+1,d);
    }
    if ((open_west(c,d)==1) && (retrieve_rank_element(c-1,d) < temp)) {
        temp = retrieve_rank_element(c-1,d);
    }
    return temp;
}

// returns the maximum cell rank of all the accessible neighbors to the cell
int max_neighbor_rank(int c,int d) {
    // get any neighboring rank value
    if(open_north(c,d) == 1) {
        temp=retrieve_rank_element(c,d+1);
    }
    if(open_south(c,d) == 1) {
        temp=retrieve_rank_element(c,d-1);
    }
    if(open_east(c,d) == 1) {
        temp=retrieve_rank_element(c+1,d);
    }
    if(open_west(c,d) == 1) {
        temp=retrieve_rank_element(c-1,d);
    }

    // determine maximum and return it
    if ((open_north(c,d) == 1) && (retrieve_rank_element(c,d+1) > temp)) {
        temp = retrieve_rank_element(c,d+1);
    }
    if (((open_south(c,d)==1)) && (retrieve_rank_element(c,d-1) > temp)) {
        temp = retrieve_rank_element(c,d-1);
    }
    if ((open_east(c,d)==1) && (retrieve_rank_element(c+1,d) > temp)) {
        temp = retrieve_rank_element(c+1,d);
    }
    if ((open_west(c,d)==1) && (retrieve_rank_element(c-1,d) > temp)) {
        temp = retrieve_rank_element(c-1,d);
    }
    return temp;
}

int open_north(int c, int d) {
    if((d+1 < 9) && ((retrieve_map_element(c,d) & 8) != 8)) {
        return 1;
    }
    else {
        return 0;
    }
}

int open_west(int c,int d) {
    if((c-1 >= 0) && ((retrieve_map_element(c,d) & 1) != 1)) {
        return 1;
    }
    else {
        return 0;
    }
}

int open_south(int c,int d) {
    if((d-1 >= 0) && ((retrieve_map_element(c,d) & 4) !=4 )) {
        return 1;
    }
    else {
        return 0;
    }
}

int open_east(int c, int d) {
    if((c+1 < 16) && ((retrieve_map_element(c,d) & 2) != 2 )) {
        return 1;
    }
    else {
        return 0;
    }
}

int open_neighbor(int c,int d) {
    if ((open_north(c,d) == 1) || (open_south(c,d) == 1) || (open_east(c,d) == 1) || (open_west(c,d) == 1)) {
        return 1;
    }
    else {
        return 0;
    }
}

void push_open_neighbors(int c,int d) {

    if (open_north(c,d) == 1)  {
        push(return_location(c,d+1));
    }
    if (open_south(c,d) == 1 ) {
        push(return_location(c,d-1));
    }
    if (open_east(c,d) == 1)  {
        push(return_location(c+1,d));
    }
    if (open_west(c,d) == 1) {
        push(return_location(c-1,d));
    }
}

int convert_x(int temp1) {
    return (temp1&240)/16;
}
```

```
int convert_y(int temp1) {
    return (temp1)&15;
}

void push(int x) {
    top = top+1;
    stack[top] = x;
}

int return_top() {
    return stack[top];
}

int pop() {
    if (top >= 0) {
        temp = return_top();
        top = top-1;
        return temp;
    }
    else {
        return 0;
    }
}

int return_location(int a, int b) {
    temp = a*16;
    temp1 = b;
    return (temp | temp1);
}


// returns a 1 if the cell was traversed, 0 otherwise
int traversed(int c, int d) {
    if ((retrieve_map_element(c,d) & 16) == 16 ) {
        return 1;
    }
    else {
        return 0;
    }
}

int suggestpath(){
    // returns the suggested next cell to move to depending on
    // its current orientation and takes consideration that
    // turning takes time, prefers going straight, and
    // prefers not doing 180 turns

    // If The Mouse is Going Towards the Center Exploring The Maze
    if (setup == 0) {
        if (o == 8) {
            if ( (open_north(a,b)==1) && min_neighbor_rank(a,b) == retrieve_rank_element(a,b+1) ) {
                return  return_location(a, b+1);
            }
            if ( (open_east(a,b)==1) && min_neighbor_rank(a,b) == retrieve_rank_element(a+1,b) ) {
                return  return_location(a+1, b);
            }
            if ( (open_west(a,b)==1) && min_neighbor_rank(a,b) == retrieve_rank_element(a-1,b) ) {
                return  return_location(a-1, b);
            }
            if ( (open_south(a,b)==1) && min_neighbor_rank(a,b) == retrieve_rank_element(a,b-1) ) {
                return  return_location(a, b-1);
            }
        }
        if (o == 4) {
            if ( (open_south(a,b)==1) && min_neighbor_rank(a,b) == retrieve_rank_element(a,b-1) ) {
                return  return_location(a, b-1);
            }
            if ( (open_east(a,b)==1) && min_neighbor_rank(a,b) == retrieve_rank_element(a+1,b) ) {
                return  return_location(a+1, b);
            }
            if ( (open_west(a,b)==1) && min_neighbor_rank(a,b) == retrieve_rank_element(a-1,b) ) {
                return  return_location(a-1, b);
            }
            if ( (open_north(a,b)==1) && min_neighbor_rank(a,b) == retrieve_rank_element(a,b+1) ) {
                return  return_location(a, b+1);
            }
        }
        if (o == 2) {
            if ( (open_east(a,b)==1) && min_neighbor_rank(a,b) == retrieve_rank_element(a+1,b) ) {
                return  return_location(a+1, b);
            }
            if ( (open_north(a,b)==1) && min_neighbor_rank(a,b) == retrieve_rank_element(a,b+1) ) {
                return  return_location(a, b+1);
            }
            if ( (open_south(a,b)==1) && min_neighbor_rank(a,b) == retrieve_rank_element(a,b-1) ) {
                return  return_location(a, b-1);
            }
            if ( (open_west(a,b)==1) && min_neighbor_rank(a,b) == retrieve_rank_element(a-1,b) ) {
                return  return_location(a-1, b);
            }
        }
        if (o == 1) {
            if ( (open_west(a,b)==1) && min_neighbor_rank(a,b) == retrieve_rank_element(a-1,b) ) {
                return  return_location(a-1, b);
            }
            if ( (open_north(a,b)==1) && min_neighbor_rank(a,b) == retrieve_rank_element(a,b+1) ) {
                return  return_location(a, b+1);
            }
            if ( (open_south(a,b)==1) && min_neighbor_rank(a,b) == retrieve_rank_element(a,b-1) ) {
                return  return_location(a, b-1);
            }
            if ( (open_east(a,b)==1) && min_neighbor_rank(a,b) == retrieve_rank_element(a+1,b) ) {
                return  return_location(a+1, b);
            }
        }
        return return_location(0,0);
    }

    // The MicroMouse Is Returning to Its Start Position From the Center
    // edit so that it follows the same least path back
    if (setup == 2) {
        if (o == 8) {
            if ( (open_north(a,b)==1) && max_neighbor_rank(a,b) == retrieve_rank_element(a,b+1) ) {
                return  return_location(a, b+1);
            }
            if ( (open_east(a,b)==1)  && max_neighbor_rank(a,b) == retrieve_rank_element(a+1,b) ) {
                return  return_location(a+1, b);
            }
            if ( (open_west(a,b)==1)  && max_neighbor_rank(a,b) == retrieve_rank_element(a-1,b) ) {
                return  return_location(a-1, b);
            }
            if ( (open_south(a,b)==1) && max_neighbor_rank(a,b) == retrieve_rank_element(a,b-1) ) {
                return  return_location(a, b-1);
            }
        }
        if (o == 4) {
            if ( (open_south(a,b)==1) && max_neighbor_rank(a,b) == retrieve_rank_element(a,b-1) ) {
                return  return_location(a, b-1);
            }
            if ( (open_east(a,b)==1)  && max_neighbor_rank(a,b) == retrieve_rank_element(a+1,b) ) {
                return  return_location(a+1, b);
            }
            if ( (open_west(a,b)==1)  && max_neighbor_rank(a,b) == retrieve_rank_element(a-1,b) ) {
                return  return_location(a-1, b);
            }
            if ( (open_north(a,b)==1) && max_neighbor_rank(a,b) == retrieve_rank_element(a,b+1) ) {
                return  return_location(a, b+1);
            }
        }
        if (o == 2) {
            if ( (open_east(a,b)==1)  && max_neighbor_rank(a,b) == retrieve_rank_element(a+1,b) ) {
                return  return_location(a+1, b);

            }
            if ( (open_north(a,b)==1) && max_neighbor_rank(a,b) == retrieve_rank_element(a,b+1) ) {
                return  return_location(a, b+1);
            }
            if ( (open_south(a,b)==1) && max_neighbor_rank(a,b) == retrieve_rank_element(a,b-1) ) {
                return  return_location(a, b-1);
            }
            if ( (open_west(a,b)==1)  && max_neighbor_rank(a,b) == retrieve_rank_element(a-1,b) ) {
                return  return_location(a-1, b);
            }
        }
        if (o == 1) {
            if ( (open_west(a,b)==1)  && max_neighbor_rank(a,b) == retrieve_rank_element(a-1,b) ) {
                return  return_location(a-1, b);
            }
            if ( (open_north(a,b)==1) && max_neighbor_rank(a,b) == retrieve_rank_element(a,b+1) ) {
                return  return_location(a, b+1);
            }
            if ( (open_south(a,b)==1) && max_neighbor_rank(a,b) == retrieve_rank_element(a,b-1) ) {
                return  return_location(a, b-1);
            }
            if ( (open_east(a,b)==1)  && max_neighbor_rank(a,b) == retrieve_rank_element(a+1,b) ) {
                return  return_location(a+1, b);
            }
        }
    }
}
```

IV. Requirements Verification

THIS DOCUMENT WILL DESCRIBE THE TESTS AND THEIR RESULTS THAT WERE PERFORMED ON THE MICROMOUSE MAZE SOLVING ROBOT. THREE SEPARATE TESTS WERE PERFORMED AND CONCLUSIONS DRAWN. THE ROBOT WAS FOUND TO HAVE ACCOMPLISHED ALL BUT FOUR OF THE TWELVE SYSTEM DESIGN REQUIREMENTS IMPOSED ON IT.

DUE TO THE LACK OF ACCURATE MOTORS THE ROBOT WAS UNABLE TO PERFORM MANY OF THE TESTS IN THEIR ENTIRETY. SEVERAL COMPONENTS OPERATED PROPERLY, BUT WERE UNABLE AT TIMES TO WORK IN UNISON WITH THE ENTIRE ROBOT. THE ROBOT WAS ABLE TO SOLVE THE MAZE, HOWEVER NOT ON ITS OWN POWER. THE ROBOT ALSO WAS ABLE TO SHOW STEERING AND MANEUVERING CAPABILITIES. ALL SIZE REQUIREMENTS WERE MET IN ACCORDANCE TO APEC SPECIFICATIONS.

1 Test Program

Program Objective

This test program will verify whether the MicroMouse team satisfies its design requirements as identified in the System Design Requirements Document (SDRD). The tests described herein will verify or nullify each requirement of our design.

Scope of Effort

The article to be tested is the autonomous micromouse robot, as pictured in the physical depiction in this report. There are a total of 12 requirements listed in the SDRD. These requirements will be tested through the completion of three tests, as detailed in Section 2.

Administration

This verification test plan includes several major administrative tasks (responsible party shown in parentheses):

- Obtain access to area where maze can be fully constructed and put maze together.
- Gather all measurement devices, including stopwatch and rulers.
- Photograph MicroMouse before and after a successful set of runs.

Safety

The MicroMouse poses no direct hazards to bystanders. An updated version of the Hazard Analysis document is listed in posed solutions section of the report.

Test Support Requirements

The three verification tests that will measure the MicroMouse's ability to meet the requirements, as defined in the SDRD, are described in the following subsections. These three tests are: (1) Navigation, (2) Dimensions, (3) General Running Operation.

Test 1: Navigation

Design Requirements Tested

SDRD 3.3.2.1
The MicroMouse shall run and return unassisted to the start position after completion of each run.

SDRD 3.3.2.2 The MicroMouse must be capable of operating in a dry environment at STP.

SDRD 3.3.3.1 The MicroMouse shall have dimensions no greater than 25cm X 25cm square at any point during navigation of the maze.

SDRD 3.3.3.2 The MicroMouse shall remain intact and leave nothing behind during navigation of the maze.

SDRD 3.3.3.3 The MicroMouse shall be able to navigate without jumping over, climbing, scratching, damaging, or destroying the walls of the maze.

SDRD 3.3.3.4 The MicroMouse battery should run at minimum 10 minutes.

SDRD 3.3.5.1 The MicroMouse shall complete 10 runs within the 10 minute time limit.

Description of Test

A user will set the mouse down in the starting square of the maze, turn on the power switch and press the start button to start the mouse. A timer will be started when the mouse begins. The mouse will then be observed while traversing the maze. Any collisions with walls will be noted and checked for damage. Also, it will be recorded if the mouse at any time leaves something behind in the maze. Each time the mouse finds the center of the maze from the starting square, it will be recorded with a time. Also, each time the mouse returns to the starting square from the center, it will be recorded with a time. The mouse will be allowed to run for 10 minutes. Once the 10 minutes are up, the mouse will be stopped and the test will be over.

Resources Required

Mechanical Requirements

Measurement and verification of APEC specifications regarding the MicroMouse before and after the test run, analysis of mechanical performance and integrity of the maze after the run.

Electrical Requirements

Standard wall power outlet to charge MicroMouse in between runs or uses.

Video Requirements

The portable video camera equipment will be needed to film the micromouse performing according to the SDRD requirements.

Space Requirements

A room with a floor space large enough to accommodate a 10X10 foot square to place the maze.

2.1.3.5 Personnel Requirements

Timer to time MicroMouse during runs. Operator to start MicroMouse and assist it through any problems. Spotter to identify any penalties the MicroMouse might incur. Recorder to write down times and penalties.

Justification

SDRD 3.3.2.1 is tested by not intervening with the MicroMouse during its test runs.

SDRD 3.3.2.2 is tested by having the MicroMouse test runs in Gellersen hall.

SDRD 3.3.3.1 is tested by measuring the dimensions before and after each test run, and by observing the Mouse during the test runs.

SDRD 3.3.3.2 is tested by observing if the mouse leaves any "popcorn" trails or mechanical/electrical parts during the test runs.

SDRD 3.3.3.3 is tested by observing the MicroMouse's influence on the maze walls and itself during the test run. Maze walls will be inspected for damaged if there are collisions.

SDRD 3.3.3.4 is tested by measuring and estimating the battery power before and after each 10 minute run.

SDRD 3.3.5.1 is tested by observing the MicroMouse successfully complete the maze and return to its starting position 10 times.

2.1.5 Results and Conclusions

The mouse was mobile and able to steer, however the motors and gearbox caused excessive amounts of inconsistencies in steering causing the mouse to veer into the walls. The gearbox would bind at any given point and pull the mouse off course. It was impossible to fully compensate for this type of error, therefore this part of the project was not a complete success at the time of testing.

Although the mobility test of the MicroMouse failed, the maze solving logic test passed. The MicroMouse robot successfully solved the maze while *assisted.* The software algorithm successfully ranked the individual cells according to the cell proximity and accessibility (determined from the sensor inputs) to the center of the maze. The ranked MicroMouse maze was documented . This table shows that a least-path was determined as the MicroMouse traversed the maze while assisted and sensing the maze walls. This test was repeated for different maze configurations and the least-path was still successfully determined.

SDRD 3.3.2.1 failed because the MicroMouse required user intervention during its runs.

SDRD 3.3.2.2 was successful due to the normal weather in Gellersen Hall.

SDRD 3.3.3.1 was successful by observation. The MicroMouse was not observed to leave any part of itself behind in the maze during its runs.

SDRD 3.3.3.2 was also successful by observing that the robot did not leave anything behind during the maze runs.

SDRD 3.3.3.3 failed because the MicroMouse was observed to collide into the maze walls uncontrollably.

SDRD 3.3.3.4 was determined to be successful. The motor speeds (dependent on battery consumption) were observed to be operating at the desired speed after the 10 minutes of the motors running full power.

SDRD 3.3.5.1 failed because the MicroMouse was not observed to solve the maze and return to home without human intervention.

Test 2: Dimensions

Design Requirements Tested

SDRD 3.3.3.1: The MicroMouse shall have dimensions no greater than 25cm X 25cm square at any point during navigation of the maze.

Description of Test

A mechanical engineering team member will measure all physical dimensions of the MicroMouse. Each measurement will be taken three times. Wherever disassembly in needed, the tester will be authorized to break the robot down into its individual components in order to more easily accommodate the testing apparatus. After measurements are taken the robot will be reassembled and restored to full working operation.

Resources Required

2.2.3.1 Mechanical Requirements

A set of decimal calipers and micrometers will be needed to measure each dimension. In the case of disassembly, both Phillips and standard head screwdrivers will be provided.

2.2.4 Justification

SDRD 3.3.3.1 is tested by verifying all dimensions are within required tolerances.

2.2.5 Results and Conclusions

The mouse was found to be within all tolerances. At no point during the navigation of the maze will the mouse exceed the dimensions of 25 cm x 25 cm square. After measuring the mouse with a caliper it was found to be 10.23 cm at its widest and had a length of 14.3 cm. These dimensions will allow the robot to maneuver easily and avoid accidental collisions with the walls of the maze. It was noted that some measurements were not exactly the same as originally specified in the design drawings due to reconfiguration in various components.

SDRD 3.3.3.1 was successful by measuring the dimensions of the robot.

2.3 TEST 3: GENERAL RUNNING OPERATION

2.3.1 Design Requirements Tested

SDRD 3.3.3.2: The MicroMouse shall remain intact and leave nothing behind during navigation of the maze.

SDRD 3.3.3.3: The MicroMouse shall be able to navigate without jumping over, climbing, scratching, damaging, or destroying the walls of the maze.

SDRD 3.3.3.4: The MicroMouse battery should run at minimum 10 minutes.

2.3.2 Description of Test

The mouse will be operated in a pre-constructed maze of APEC specifications. During operation the mouse progress will be monitored closely. If at any time the mouse leaves an object behind or causes damage to the maze wall, the run will be considered failed. Passing status will be given after the mouse has run for a total of 10 minutes without a failed run. FAILED

2.3.3 Resources Required

2.3.3.1 Mechanical Requirements

Partial maze built to APEC specifications to operate mouse within.

2.3.3.2 Electrical Requirements

9.6 volt battery pack to operated robot circuitry and motors.

2.3.4 Justification

SDRD 3.3.3.2 is tested by observing if the mouse leaves any "popcorn" trails or mechanical/electrical parts during the test runs.

SDRD 3.3.3.3 is tested by observing the MicroMouse's influence on the maze walls and itself during the test run. Maze walls will be inspected for damaged if there are collisions.

SDRD 3.3.3.4 is tested by measuring and estimating the battery power before and after each 10 minute run.

2.3.5 Results and Conclusions

The robot was carefully monitored as it progressed through the maze. At no time during the test did any component of the mouse fall off or leave and trail behind. However while the mouse was navigating there were some instances where it collided with a wall and required human intervention to correct the misalignment.

SDRD 3.3.3.2 passed by inspecting that the mouse did not leave any "popcorn" trails or mechanical/electrical parts during the test runs.

SDRD 3.3.3.3 failed by observing that the MicroMouse collided with maze walls.

SDRD 3.3.3.4 was successful. The motor speeds (dependent on battery consumption) were observed to be operating at the desired speed after the 10 minutes of the motors running full power.

Supporting Information

Referenced Documents

APEC MicroMouse Contest Rules
http://www.apec-conf.org/APEC_MicroMouse_Contest_Rules.html

Symbols and Abbreviations

APEC	Applied Power Electronics Conference and Exposition

3.3 MAZE NAVIGATION TEST DATA

Final Numbered Cell Ranks Table

VI. Conclusion

This final report discussed the rationale and significance of the MicroMouse project in the problem definition. The MicroMouse has a potential of scoring high in competitions. The problem definition addressed our project problem, the posed solution (including how our original design was down-selected), the benefits of our implementation, and the impact of our project. A safety analysis was also included in the this section's preliminary hazard analysis. The "as built" specifications, or the blue prints of the MicroMouse, illustrated the technical solution and final design of the MicroMouse so that posterity may rebuild or improve on the MicroMouse project. The requirements verification section explained how the final design addressed and successfully completed each of the system design requirements. Each requirement was stated and then given sufficient proof of verification by referencing results from our test report. The project design was successful in certain requirements and unsuccessful in others. Lastly the budget of the project was discussed. An overview of the budget listed major categories and a complete parts list of each component, including the suppliers and final overall budget.

The MicroMouse project has been a good experience for all team members. We have learned that this project is a team effort by all. Each team member has had to work effectively and maturely with members from different backgrounds, majors, and even different cultures. However, individual members struggled with their own schedules and working near deadlines. Setbacks included faulty hardware and original design specifications. These setbacks could have been minimized by allocating more time for the assembly and testing of the project rather than the design. We found that through lack of experience, our original designs did meet our high expectations of successfully completing the system design requirements. However, by testing and trying new creative designs, our team was able to successfully complete most of our system design requirements.

VII. Appendix

Appendix A

Official MicroMouse APEC, the Applied Power Electronics Conference and Exposition Contest, Rules Webpage
http://www.apec-conf.org/APEC_MicroMouse_Contest_Rules.html

Appendix B

Official 2003 IEEE Sponsored MicroMouse Contest at Valparaiso University
http://www.valpo.edu/engineering/micromouse