FINAL DESIGN FOR ROSIE THE ROBOT: A DETAILED TECHNICAL REPORT

Prepared for: Dr. Kevin Wedeward
Dr. Stephen Bruder

Prepared by: Rebecca Brown
Adam Garcia
Manuel Jaramillo
Michael Jones

May 8, 2000

Abstract

The Electrical Engineering Department at the New Mexico Institute of Mining and Technology gave the junior class an objective at the beginning of the spring semester when the students enrolled in the Junior Design course. The objective for Spring 2000 was to design a robot that is able to navigate through a maze, find a candle flame and extinguish the flame using closed-loop control and various other sensors. Each of the sensors comprise a different subsystem of the design and each subsystem is then implemented together with a Motorola 68HC12 processing board that interfaces each of the subsystems together. In addition to accomplishing this goal, each group of students learned to allocate manpower and communicate with one another. Each group needed to complete this goal in a timely manner and with the mentality that a good quality product would be designed.

The goal for Group 5 was accomplished with all the descriptions that were stated above. The design of our robot "Rosie" not only describes each of the subsystems and their integration, but the hard work and dedication that the group had towards accomplishing our goals. The finalized product of hard work and dedication was greatly rewarded because Rosie attended an International Competition in Hartford, Connecticut, where she placed eighth out of fifty-three teams. In addition, Rosie competed in the local competition and placed second. The following technical report presents the results of our design for Rosie the robot.

Introduction

The design of Rosie involved many different subsystems including chassis and motor design, distance detection, fire detection, white-line detection, tone activation and fire extinguishing. All of these subsystems must then be integrated together so that the robot can navigate the maze and put out the flame. The design was divided into four areas so that each member of the group had smaller subsystems to design and build. The following report gives a detailed discussion of what each member did to meet the requirements specified for the design. The goals of the group were to design make a design which was compact, robust and clean in a timely fashion. The following is a review of our design and what made Rosie a success.

Chassis

We decided that the most efficient method for maneuvering the maze would be the differential drive system. By positioning two drive motors opposite each other on the round 10-inch aluminum plate, along with a caster wheel for support, the robot could easily make 180-degree turns in place. As shown below in the simplified Figure 1, the robot's caster wheel was placed at the front of the robot versus the rear in order to keep the robot's center of gravity (C.G.) in the forward triangle (created between the two side wheels and the front caster). As the robot traveled, the C.G. remained in the triangle especially when the robot de-accelerated, thus preventing the robot from tipping forward as it approached a wall; however, we did run into some problems with the robot tipping backward and hitting a wall when it quickly turned to leave a room. This was approximately 95% corrected using accurate robot navigation. We felt that the chances for the robot to tip forward and strike a wall were greater than the tipping backward and

striking a wall. Therefore, we felt the robot would be better at navigating the maze with the caster wheel in front (Plus, it was fun to watch the robot pop wheelies).

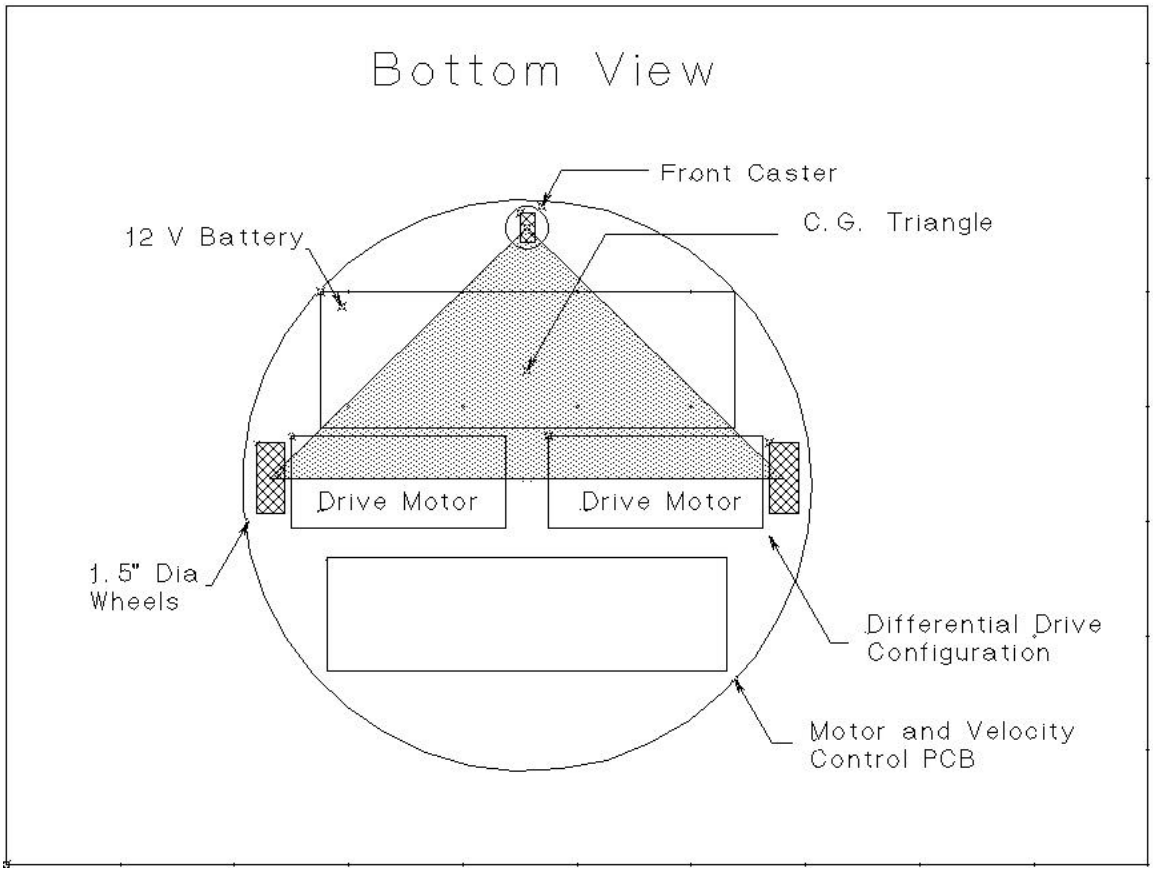


Fig 1: Bottom View of Robot Showing Robot Center of Gravity

In another strategy for our robot, we implemented a low-profile compact design. By mounting hardware as close to the drive wheels as possible, the height of the robot would be minimized and aid in the robot's tendency to tip. Also, we considered the placement of components depending on their weight, thus ensuring that the robot's C.G. was inside the triangle. As shown in Figure 2, we offset the HC12 and a small shelf mounted directly over the HC12 to the rear of the robot. We mounted some of the external control circuitry to the bottom rear of the robot. We placed the heavy 12-volt battery just behind the front caster wheel. The weight distribution worked well to

balance the robot. There was enough weight provided by the 12-volt battery to keep the robot‘s front caster on the ground. The offset HC12 and miscellaneous control circuitry counter balanced the weight of the battery.

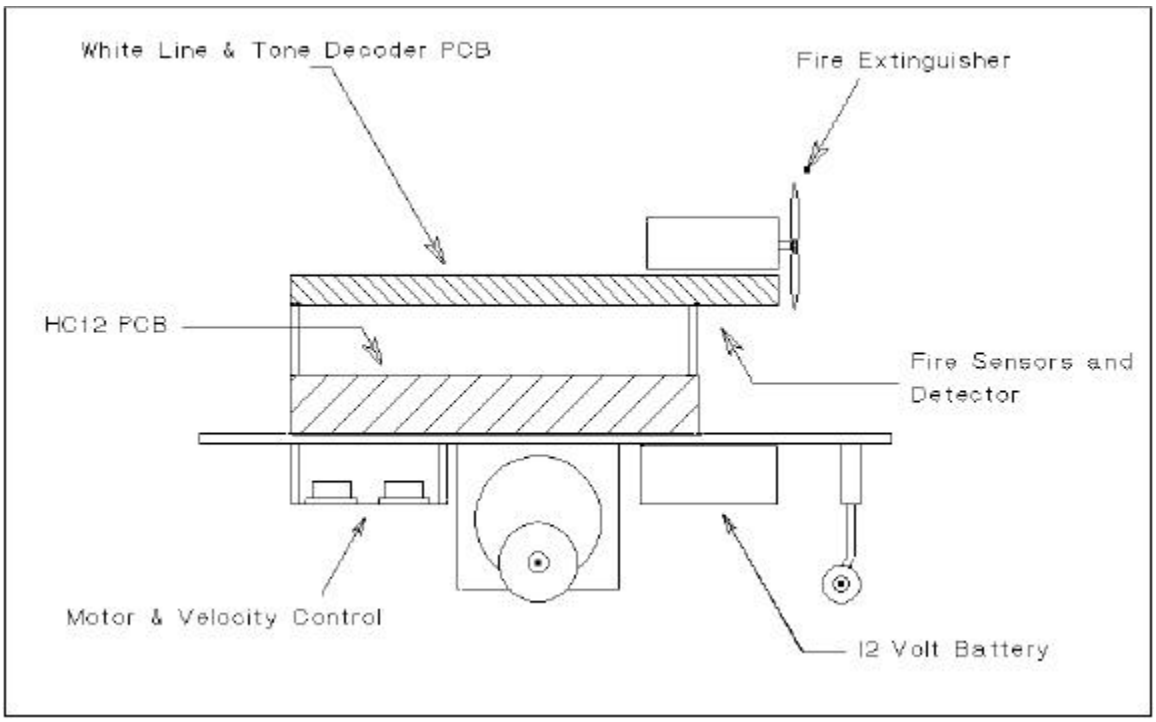


Fig 2: Side View of the Robot

Power Distribution

The power distribution was broken up into three systems, as shown below in Figure 3. First, 18 volts were used to power the fire extinguisher. We used two 9-volt alkaline batteries wired in series. Initially, the robot was equipped with a single 9-volt battery. This was about 95% successful in blowing out the candle, but this was not satisfactory. After wiring-in the second battery, the fan became 100% successful in extinguishing the candle. The second source for power was a 9.6-volt radio controlled toy-car type battery with a rating of one amp-hour. The positive lead from the battery was connected to each sub-system, and the ground was directly connected to the aluminum

chassis, which reduced wiring. It also provided a nice ground plane for the HC12 and other circuitry. For a more detailed schematic of the power distribution, see Appendix B.

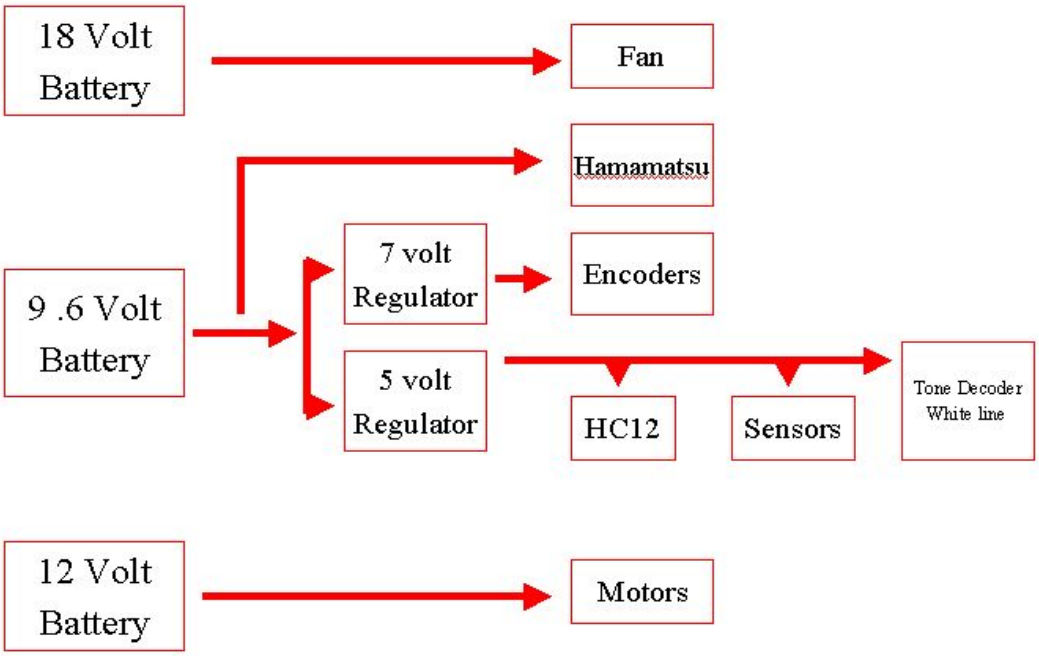


Fig 3: Block Diagram of Power Distribution.

The majority of the robot's external control circuitry ran from the 9.6-volt battery. We acquired a Hamamatsu fire detector and control board that needed 10-volts. Fortunately, the 9.6-volt battery usually would have about 10.5-volts after a full charge which was adequate to power the Hamamatsu. Next, a 5-volt regulator was connected to the 9.6-volt battery to supply power to the HC12, encoders, tone decoder, white-line sensors, and the flame detectors. Also, the two frequency-to-voltage converters received their power from an adjustable voltage regulator that was set to 7-volts. In both cases, each of the voltage regulators was rated at one amp. We came close to this rating on the five-volt regulator, but we did not have any problems with the regulator overheating. Since it was mounted to a piece of metal, it was kept cool and any excessive heat was drawn away.

The third battery system was the 12-volt battery, which only powered the H-bridges, and the Pittman Drive motors. This 12-volt VCR-type battery was impressively rated at 2amps/20 hours, but the disadvantage to the battery was its weight. One item to note is the Drive motors did not have their returns connected to the aluminum ground plane, as did the 9.6-volt system. They each had a heavy-duty wire for their ground, and they were directly connected to the H-bridges, thus ensuring the drive motors were isolated from the rest of the robot's power system.

The main motivation for breaking up the power distribution was our concerns that the drive motors and fan would generate a lot of noise. We were especially worried that the HC12 and sensors would have poor performance because of motor noise. We are not sure if the isolation was beneficial because we did not have time to try any other power schemes; however, the one we used presented no troubles.

Power Budget

Power consumption was estimated at the beginning of the robot's initial design and we found that there was sufficient energy in the batteries to power the robot. Since the power distribution was broken into three parts, there was less of a demand on any one battery. In Table 1 below, is the estimated power budget for each battery.

Table 1: Estimated Power Budget.

12 Volt System			9.6 Volt System			18 Volt System	
Item	Watts		Item	Watts		Item	Watts
Motors	8.80		HC-12	0.43		Fan	18
H-Bridge	6.00		Fire Sensors	0.46			
			Freq/VoltConv.	0.10			
			Hamamatsu	0.15			
			Encoders	0.29			
			Regulators	2.00			
			Tone Decoder	2.00			
			White Line	0.15			
Total	**14.80**		**Total**	**5.58**		**Total**	**18.00**

From experience, the 18-volt system would last about 15 minutes if operated continuously. We went through many 9-volt batteries during testing. The 9.6-volt battery would last about 1 hour and frequently, during long sessions of robot algorithm development and testing, this battery would die. It was mandatory for the battery to be connected to the batter charger a few hours before Junior Design class, or we would not be able to test the robot for the whole period. It took about five hours for the battery to charge completely. Finally, the 12-volt battery was estimated to last approximately 2 hours. We never had any troubles with the battery life primarily because of its high-energy rating, and we usually charged the 12-volt battery the same time we charged the 9.6-volt battery.

Motor Control

There were several H-bridges to choose from: Some with two H-bridges on one chip, others that needed little or no external circuitry and others with high power capabilities. Since we were not completely certain what our motor current demands were going to be, we decided to play it safe by using the H-bridge with the high power rating. The National brand LMD18200T is rated for 3 amps, and it can briefly handle 6 amps peak (see Appendix A). It comes with the standard features such as direction, PWM, and brake control. It also has some other bells and whistles we did not use, such as the ability to send a warning when the chip is overheating, and it has the ability to monitor the current being delivered to the motors. Once we wired-up the two H-bridges, one for each motor, we forgot about them because we had absolutely no trouble with them.

Velocity Control

We initially intended to use the HC12's pulse-accumulator to count the number of pulses received from the encoders. We thought this would be the most accurate method to determine the robot's velocity, but we needed two accumulators, one for each motor. Since the HC12 only has one accumulator, we attempted to program a second one on our previously installed Altera chip. Since our group is better suited to solving engineering problems via hardware, it only took a few frustrating hours of attempting to implement a pulse-accumulator in Altera before we decided to use frequency to voltage converters (FTVC) to monitor the robot's velocity.

The FTVC are a little tricky to set up due to external circuitry need. The Data sheet on the LM2907 gave some "cook book" formulas to determine component values. At first, we were not happy with the performance of the FTVC as there was a lot of ac ripple (about 60mV) at the output. We thought this might lead to large errors in determining the velocity. Scratching our heads and researching further into the problem, we discovered that one of the external capacitors was the culprit so we changed from a small value to a much larger size. This change eliminated the large output ac ripple, but the response time of the output was not acceptable. When we simulated the speed of the motors, we fed in a 13khz square wave that simulated the motors at the maximum speed we wanted to travel. However, when we simulated the motor stopping by changing the input frequency to zero hertz, it took about 2 seconds for the output to drop to zero volts. To make a long story short, we experimented with different capacitor values and found one which resulted in about a 5-mv output ripple, and with approximately a .25 second response time changing from 5 to 0 volts. Once we figured out the component values, we

built a second FTVC. After calibrating both, we found that there is a very nice linear relationship between the input frequency and the output voltage. As shown below in Figure 4, with an input frequency ranging from 0 to 13 kHz there is a nice linear 0 to 5-volt output. Actually, both traces are on the graph, only one is visible because both traces overlap each other perfectly. Our final judgement concerning the performance of the FTVC has been outstanding. Appendix C gives a detailed schematic of the motor and velocity control circuitry.

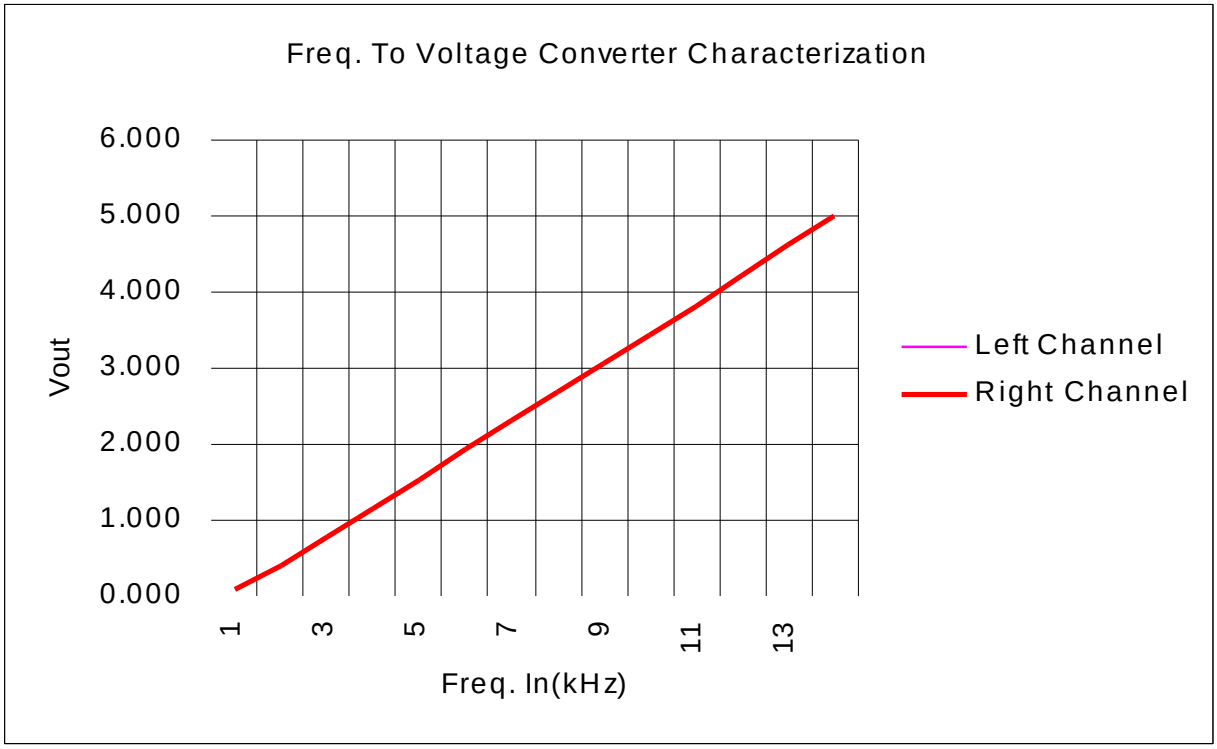


Fig 4: Frequency to Voltage Response.

Fire Detection Subsystems

When looking at options for our fire sensors we had some criteria that needed to be met in order for a sensor to be considered. First of all the sensor had to be small to fit the low profile design of our robot. Second, the sensors had to be fairly oblivious to any

changes in lighting schemes. We did not want to worry about problems with calibration and reflections due of changes from florescent lighting to Sodium vapor lighting for example. After looking at all our options we decided on two different sensors to optimize our detection of the flame and separated our fire detection into two parts; long-range and short-range detection. For long-range fire detection we chose the UVtron R2868 Photomultiplier and the C3704 companion driving board made by Hamamatsu for its reliability and extreme sensitivity. Usually the UVtron along with its driving board would cost roughly $70.00, but we were able to receive a sample from the Hamamatsu Corporation. For short-range detection we used PN168 phototransistors by Panasonic for their versatility and their ease of use.

Ultraviolet Detection

For long range detection, we took advantage of the sensitivity and reliability of the UVtron as well as its use of ultraviolet detection to speed up our candle search. With the UVtron we were able to pass by a room or peek into the entrance of a room to know if a flame was present. The UVtron is able to detect Ultraviolet light from all directions and is sensitive enough to detect reflections from long distances as shown in figure 5.

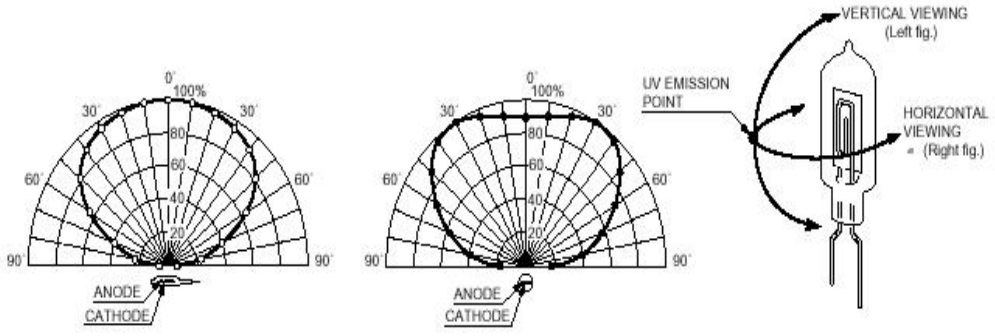


Fig 5: Angular Sensitivity of the Hamamatsu UVtron.
Source: http://usa.hamamatsu.com/

This sensor gives the option of different levels of sensitivity and a pulse digital output that can easy manipulated. Usually the UVtron driving circuit takes 11-25Vdc for power, but we took advantage of the fact that it can work on as little as 9Vdc. When the UVtron detects a flame its output is a stream of digital pulses 10ms in duration. The closer the UV source is the tighter the duty cycle of the pulses. The stream of pulses was then fed into the HC12 pulse accumulator, which counted the amount of pulses and determined if a valid flame was being detected. The proper subroutine was then called in the HC12 program.

Infrared Detection

For our short-range detection we decided to build a sensor that would be able to pinpoint the flame with as much accuracy as possible. To get the highest amount of accuracy we decided that two PN168 phototransistors in a binocular vision configuration would be the best way to pinpoint the fire as shown below.

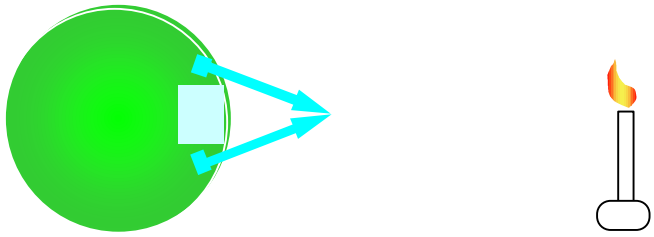


Fig 6: Binocular Vision Configuration Used to Pinpoint the Flame

The usual emitter voltage output of the phototransistors was in the millivolt range. In order to calibrate the two sensors to give as close an output voltage as possible a 15 turn 10K? potentiometer was used to increase or decrease the emitter voltage of the phototransistors. The emitter voltage was fed into a MAX492 dual micropower single-supply op-amp in a non-inverting configuration with a gain of 1001 to give us a voltage

range between 1V and 5V (see Fig 7). The outputs of the amplifier were then fed into two different A/D channels on the HC12.

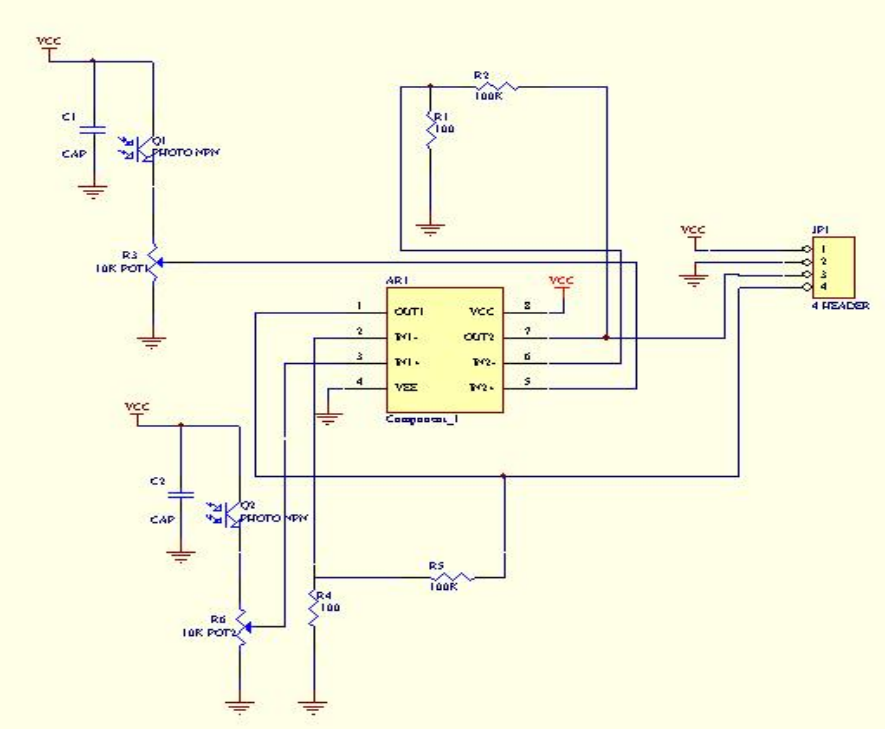


Fig 7: Schematic of the Infrared Fire Sensors

To implement pinpoint accuracy with the phototransistors we had to reduce their angular sensitivity and filter their input. To do this we housed each of them in a one and a half-inch piece of a pen case that acted like a blind. Since the PN168 has a very wide spectral range, we filtered the input to the phototransistors using a small piece of a 5¼-inch floppy disk. The filter was able to block all ambient light and limit the amount of reflections we saw while trying to pinpoint the flame. To implement binocular vision we mounted the two sensors angled in toward front center of the chassis. When scanning a room the difference between the value of the two sensors was taken until the result was close to zero. This design proved to be reliable, accurate and was oblivious to any changes in lighting schemes thus meeting our criteria for a good fire detector.

Distance Measuring Sensors

To navigate through the maze we found that it would be best to implement wall following. To wall follow we needed to find a sensor that could accurately tell us our distance away from the right and left walls as well as keep us from hitting any obstructions in front of us. The two sensors that were seriously considered for this job were the GP2D12 and the GP2D120 both made by Sharp. Although both sensors proved to be accurate and reliable we decided to use the GP2D120 for its extra accuracy at close range. The following graph shows the characteristic curve of output voltage vs. distance for the GP2D120.

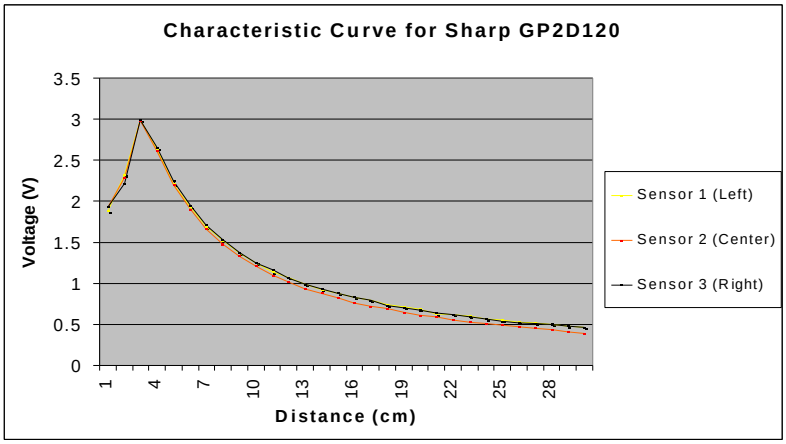
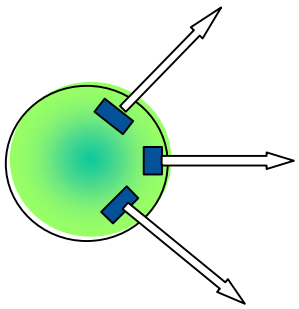


Fig. 8: Output of the GP2D120 Distance Measuring Sensors.

The above graph was used to determine the range of the sensors and to mount them on the chassis. When an object is about 3cm. away the distance sensor output peaks at 3V and follows a smooth curve as the object farther away. In order to minimize any confusion we decided to mount the sensors 4cm. inside the main plate at 45 degree angles from each other on small aluminum L-brackets as shown below.

Fig 9: Mounting Configuration of the GP2D120 Sensors



Fire Extinguishing Subsystem

To extinguish the flame we thought of a few different ways to implement an extinguisher. Some options we researched included using a pressured CO_2 cartridge, a small water hose, and a motor controlled fan. Our final decision was to use a motor controlled fan powered by two 9V batteries in series to make 18V. The motor was controlled by an opto-isolated relay that was controlled by PORTEA3 in the HC12. When the extinguisher was to be used PORTEA3 went high and allowed current to travel from the batteries to the motor and power the motor for a set duration. Once PORTEA3 went low the relay cut the current flow to the motor and shut off the extinguisher. For a fan blade we used a 5in. model airplane propeller. The propeller and the whole extinguishing system proved to work well and effectively extinguished even a large candle flame.

White-Line Sensor Subsystem

The white-line sensor is a very important subsystem of Rosie's design because the sensor indicates when she is at home, in a room, or in front of the candle. The design of the white-line sensor went through three preliminary designs before the final design was implemented, etched, and mounted.

The first design consisted of using a red light emitting diode (LED) with a phototransistor along with an operational amplifier (op amp). This first design had problems finding the white surfaces because the LED did not emit enough light. The second design consisted of the same components except for this design used a green LED. This design was worse than the first design because it did not find any white surfaces. Some of the other groups used these designs and they had consistent problems

with their white-line sensor. The final design that was used consisted of the same components except an incandescent lamp, comparator and a Schmidt trigger. The incandescent lamp emitted a sufficient amount of light to reflect off the white surface that enabled the phototransistor to detect it accurately. Another part of the final design was the implementation of an adjustable comparator, which discriminates between two unequal voltages. The adjustable comparator enables the white-line sensor to be adjusted for different lighting. This helps distinguish between imaginary and real white lines. I made a more accurate design that would enable us to have an efficient white-line sensor. The Schmidt trigger enabled us to send a digital output to the 68HC12. The output would be a 1 for all black surfaces and a 0 for all white surfaces. The white-line senor is contained in a film canister wrapped with black electrical tape to shield out all ambient light. The final design proved to be very efficient and did not cause any problems.

In addition, to designing an accurate and precise white-line sensor there also had to be a good representation of the design. Protel was used to design the board for the white-line sensor. This program takes a great deal of time to learn. The first etched white-line sensor circuit board was nice but it could have been better. The group decided that since the circuit board could have been done better we decided to implement the tone decoder circuitry and white-line sensor circuitry all on one circuit board (see Appendix D). This idea also conserved space and connectors.

Tone Decoder

The tone decoder was an added accessory to Rosie because it was not required. This subsystem enabled Rosie to be activated by a 3.5kHz signal generated by a piezo buzzer. The reason we decided to add the tone decoder was to get a time reduction for

the national and regional competitions. We decided to use the LM567NC tone decoder chip manufactured by National Semiconductor. The chip is very accurate and precise and can be easily implemented. Below is a diagram showing the circuitry for the tone decoder.

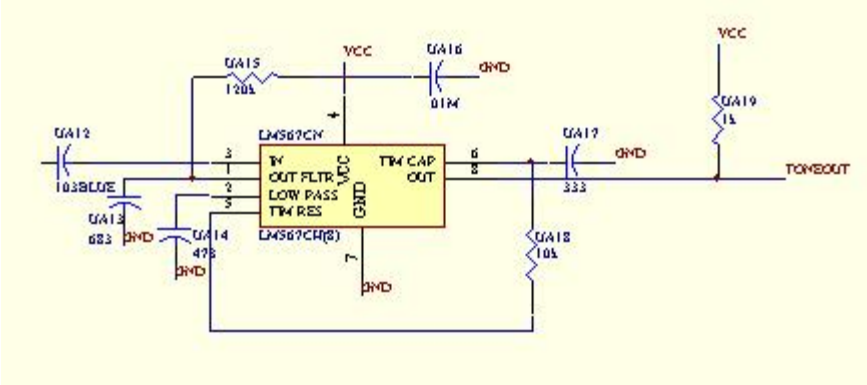


Fig 10: Tone Decoder Wiring Diagram

The tone decoder chip has a timing capacitor and a timing resistor that determines the frequency that will trigger its output low. The capacitor and resistor values can be calculated using the equations in the specification sheet. The bandwidth (BW) is determined by the capacitor on pin 2. We used a 10% bandwidth to enable the tone decoder to only go low on the specified signal. All the other capacitors were used to filter out noise. The specification sheet also noted that to further reduce the amount of noise we needed to add a resistor from pin 4 to pin 1. This enabled Rosie to be activated from a distance of 12 inches. The capacitor at the input on pin 3 was to get rid of any DC offset from the microphone condenser.

We used an inverting operational amplifier (op-amp) to amplify the signal approximately 500 times. The single supply op-amp only gives the positive half of the signal, so we added another offset at V+ by adding 2.5V. This enabled us to have either a

low or high output. The amplified signal was the reason Rosie could be activated at a further distance. The circuitry for amplification is show below.

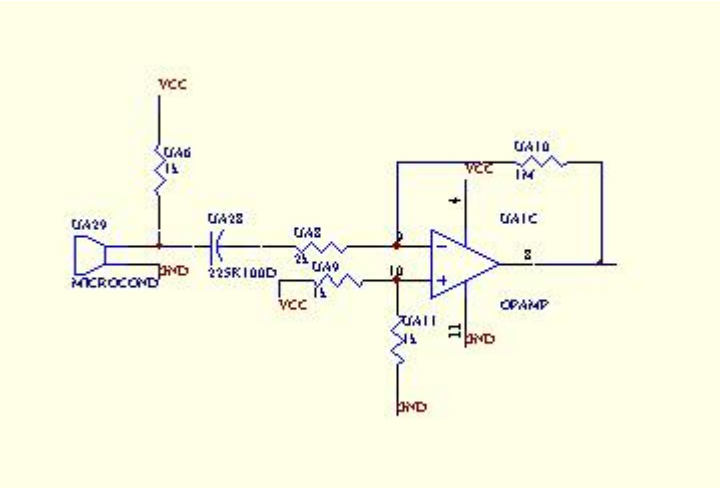


Fig 11: Microphone Condenser Amplification

The problems with the tone decoder were minor except when implementing its design with the white-line sensor circuitry. Protel was used for designing the board so we encountered some problems. Some problems were getting all the traces right and making sure that nothing was going to cause each of the different subsystems to work incorrectly. Finally, after hours of hard work the tone decoder and white-line sensor worked accurately all together on the board.

Subsystems Integration

In order to integrate all of the different subsystems of the robot together, the Motorola 68HC12 was used. The HC12 was programmed to control all of the robots actions, including navigating the maze and detecting and extinguishing the candle flame. We have the HC12 setup with a real-time interrupt of 16ms. When the HC12 gets this interrupt, all of the data from each of the subsystems is read and stored. Using this data, the HC12 decides which routine it should implement, and does so. Each routine used

many similar subroutines. Therefore, we made smaller functions that controlled each separate subsystem. Each routine calls the subsystem functions needed to complete the current routine. The different subroutines that we used for each subsystem are discussed below.

Tone Decoder

The robot doesn't do anything until it receives the correct tone from the tone emitter. Therefore, it just sends a desired speed of zero to its motors, and waits for the tone. Once a tone is detected from the PORTEB, the robot waits for the next interrupt to double check. If the tone is detected for three consecutive interrupts, then the tone is assumed to be correct. Hence, the robot can begin the general navigation of the maze.

Motor Control

One of the most needed control algorithms is that of the motor control. This system essentially tells the robot exactly what to do and how to react to any data read off of the sensors. We made a function that would take desired left and right motor speeds and then make the robot go as desired. We were also required to use closed-loop motor control, so we also used the measured speeds of the left and right motors. As mentioned previously, the measured speed was read off of the frequency to voltage converters. To make the motors go the desired speed, we used pulse width modulated signals (PWM). The PWM signal is a series of pulses whose duty cycle at a certain frequency determines the speed of the motor. We set the PWM frequency to 1kHz, and then vary the duty cycle of a pulse to change the speed of the motor. The duty cycle is the length of the pulse for one period. In order to change the speed, a proportional control equation was used. This equation takes the desired speed and multiplies it by a constant, and then adds

it to the error of the desired speed and the measured speed also multiplied by a constant. The equation is shown below.

$$\mathbf{DC = K*S_D + K_P*(S_D - S_M)}$$

The constants K and Kp are determined through testing and experimentation. To find K, we recorded the measured speed of the motor for different duty cycles. We then plotted these values to get a linear curve representing desired speed versus duty cycle. The slope of this curve gave us the value for K. The constant Kp was entirely experimental. We just tried different values until the robot went in a straight path using the closed loop control.

We encountered some problems when first implementing the closed loop control. First, we had to use temporary variables to do calculations so that we could have more accurate integer values, and also use signed numbers. Once the calculations were completed, we separated the number into an unsigned character for the magnitude of the duty cycle along with an integer representing the sign of the duty cycle. The sign part of the number gave us the direction of the motor, which was sent to PORTEA. The reverse direction of the motor control has a problem. We haven't quite figured out what it is. This problem was corrected through other parts of the code, but in the future we would like to correct this problem directly. This explains why the robot doesn't have smooth canned turns.

Wall Following

The wall following algorithm keeps the robot centered in the maze, and also prevents it from hitting any walls or obstructions. The concept and calculations for wall following are very similar to those of the motor control. Essentially, the robot stays a

desired distance from either a right or left wall. To do this, the error of the measured and desired distances from the wall is computed. This error is then added to the desired motor speed to adjust the motion of the robot. If the robot is left wall following, then the error of the left wall is only taken into consideration. This error is added to the left motor speed and subtracted from the right motor speed to get the desired effect. The opposite happens when right wall following is implemented. The equations for left wall following are shown below.

$$\mathbf{S_{DL} = K*S_D + K_{WF}*(D_D - D_M)} \qquad \mathbf{S_{DR} = K*S_D - K_{WF}*(D_D - D_M)}$$

Again, the constants K and K_{WF} are found through testing and experimentation. K_{WF} is found by plotting the output of the distances sensors to the different distances away from the wall. The slope of the plot is K_{WF}. These constants are very sensitive to changes in the desired speed of the robot. We just kept values for several different speeds, so that we could have a variety of speeds to choose from when navigating the maze. Once we got the robot to follow left and right walls properly, we had to add in code that would take the front wall into consideration. To do this, we set a desired distance that we wanted the robot to stay away from front walls. Once the robot got inside this distance, an adjustment was made to turn the robot away from the front wall. If left wall following, interaction with a front wall would cause the robot to turn right. Of course, when right wall following, the robot turns left away from a front wall. The calculations for avoiding the front wall are exactly the same as those for the correlating wall following algorithm.

White-Line Detection

The white-line needs to be checked constantly to look for a room as well as the white line around the candle. When the white-line subroutine is called, several things are checked. If there is a white line, the robot must decide whether it is in a room or not. If not in a room, the robot sets the flag in_room and increments the number of rooms. If the robot is in a room, it must then check the fire flag to see if there is candle in the room. If the fire flag is set, the white line must be around the candle. Therefore, the white_line_candle flag is set. Otherwise, the robot is leaving a room, so the in_room flag is decremented. When a white line is seen, a flag is set so that the white line is only counted once. The robot must then see the black floor indicating no more white line, before the white line will be checked again.

Fire Detection

For fire detection the robot uses two different subsystems, ultraviolet and infrared detection. We use the ultraviolet subsystem, which consists of a series of pulses off of the Hamamatsu, for wide range fire detection. The infrared sensors are then used to pinpoint the flame. The Hamamatsu is fed into the pulse accumulator on the HC12. The pulse accumulator counts the number of pulses that the Hamamatsu gives out. We wait for three interrupts, to count the number of pulses in approximately one sec. If we get at least one pulse in that second, a flag is set telling the robot that a flame has been detected. Once the flame has been detected, the robot does the needed maneuvers to go into the middle of the room.

When the robot gets to the middle of the room, it begins using the infrared sensors to pinpoint the flame. The error between the two infrared sensors is calculated. The

robot then turns toward the flame in small increments, trying to minimize the error between the sensors. Once the error is very small, the robot goes towards the flame using the infrared sensor input. The robot goes forward at a desired speed, and adjusts the motor speeds in accordance with the error of the infrared sensors. This is done so that the robot continually centers itself on the flame. Once the robot sees the white line around the candle, it stops and put out the flame.

Fire Extinguishing

The fire-extinguishing algorithm is a simple one. A 5V signal is output on PORTEA to the fan. Then, the robot delays for a given amount of time, and shuts off the fan by putting 0V to the port. In the same function, the final room variable is set so the robot knows which go home algorithm to implement. A flag is set saying that the candle is out, and the fire flag is set back to zero.

General Navigation

Once all of the main subroutines are implemented, the navigation of the maze must be done. There are many different routines that the robot needs to navigate the entire maze and put out the flame. We decided to right wall follow through the maze, checking the island room first for a flame. Therefore, the first routine tells the robot to right wall follow at a fast pace. While doing this, the robot counts the number of left walls that it sees so that it knows when it turns the first corner and passes the first room. Once it turns the first corner, the robot continues right wall following but at a slower pace. It then starts checking the Hamamatsu for candle detection.

If the robot sees a flame, it begins to left wall follow at the faster pace to the entrance of the first room. In order to detect the entrance, it must check the white line

sensor. Once it gets to the entrance, the robot then begins to right wall follow until it gets to the center of the room. While going into the room, however, if the robot sees a white line then it begins centering on the flame. If no white line is encountered, it goes to the center of the room and then centers on the flame using the infrared subroutine. Once centered, the robot goes forward to the white line around the candle and puts out the flame.

If the robot doesn't detect a candle in the first room, it speeds up and continues right wall following. Now the robot starts checking for a white line as the entrance to a room. Once it gets to a room, it checks the Hamamatsu for a flame. If a flame is detected, the robot will perform the same operations to out the flame. If no flame is detected, the robot will do a canned turn and then continue right wall following to the next room.

Once the candle is put out, a variable is set so the robot remembers what room the candle was in. Then the robot will do the appropriate wall following algorithm to get home. If the candle was in the third room, the robot exits the room and check for the white line of the fourth room. Once it sees the line, it turns away and continues to the home spot.

Summary

The goal of the group was to build a fully functional fire-fighting robot in time for the Trinity College National Fire-fighting Robot Competition. We wanted a design that was compact, robust, and very clean. In order to accomplish our goals in the proper time, we made a time-line specifying what each member was to complete each week. This was a good idea, and the backbone of our project. Everyone completed their tasks on time and

correctly. We also built a prototype of each subsystem on the robot. Once the prototyped version was correctly interfaced with the HC12, we then built a clean final version of the subsystem. This method proved to be a successful way of getting our components to work. It also gave us a very clean and stable final product. All of the subsystems on the final robot had etched circuit boards and were mounted in a robust and stable manner on the robot.

We are very pleased with the final product of our robot. The group worked very well together and everyone put in 100%. Our goals of building a compact, robust and clean design were accomplished while keep within the $100 budget allocated to us by the Electrical Engineering Department. Appendix E gives a detailed breakdown of our budget specifying items purchased as well as those donated from various corporations. We attended the national competition and placed 8th. We also placed 2nd in the local competition. We plan to change some things on the robot in the future and hope to take Rosie back the national competition next year.

One thing we will change in the future is the size of the wheels. We would like to increase the diameter of the wheels to two inches. This will also us to go over ramps easier, a problem that we had at the national competition. The robot needs to be lightened up a little so that we can speed her up. The larger wheels will also aid in increasing Rosie's speed. We will also look into methods for maneuvering around the furniture.

Appendix A
Abbreviated H-Bridge Specifications

Absolute Maximum Ratings (Note 1)

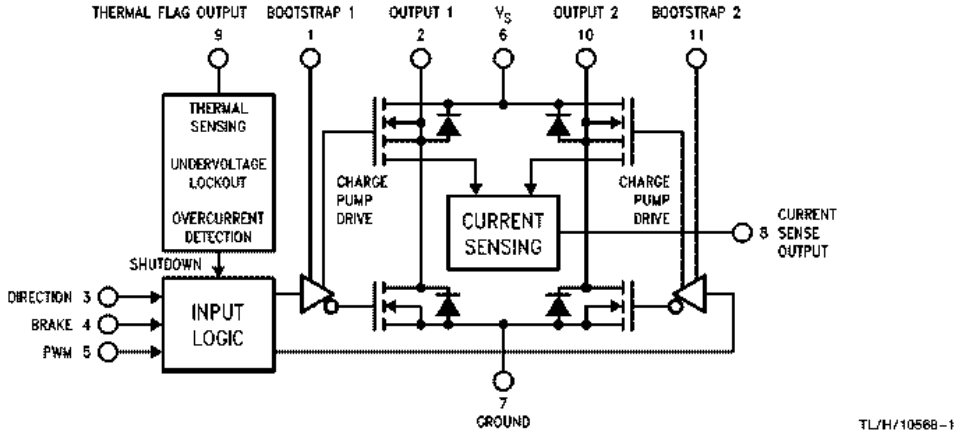
If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Total Supply Voltage (V_S, Pin 6)	60V
Voltage at Pins 3, 4, 5, 8 and 9	12V
Voltage at Bootstrap Pins (Pins 1 and 11)	V_{OUT} + 16V
Peak Output Current (200 ms)	6A
Continuous Output Current (Note 2)	3A
Power Dissipation (Note 3)	25W
Power Dissipation (T_A = 25°C, Free Air)	3W
Junction Temperature, $T_{J(max)}$	150°C
ESD Susceptibility (Note 4)	1500V
Storage Temperature, T_{STG}	−40°C to +150°C
Lead Temperature (Soldering, 10 sec.)	300°C

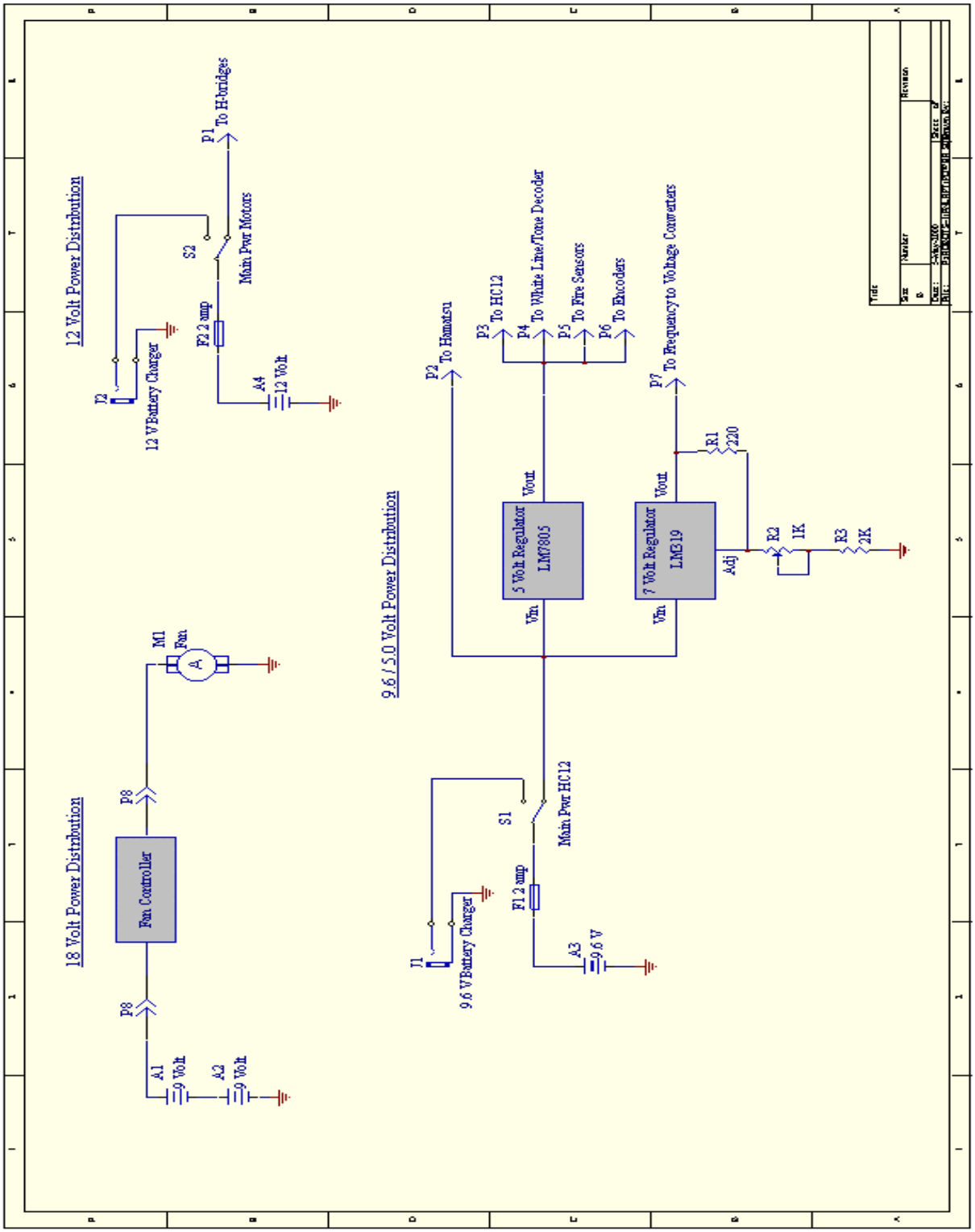
Operating Ratings (Note 1)

Junction Temperature, T_J	−40°C to +125°C
V_S Supply Voltage	+12V to +55V

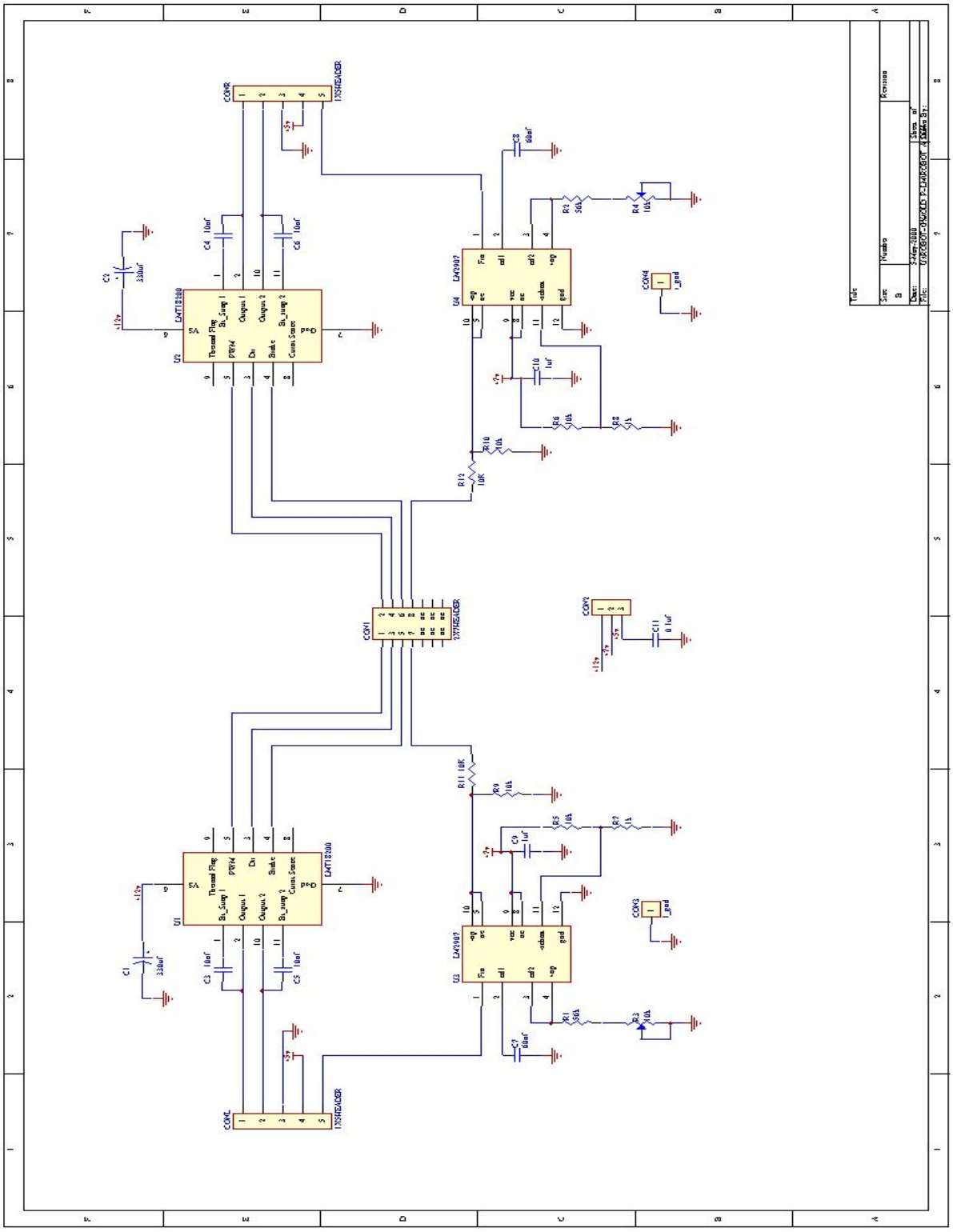
Functional Diagram



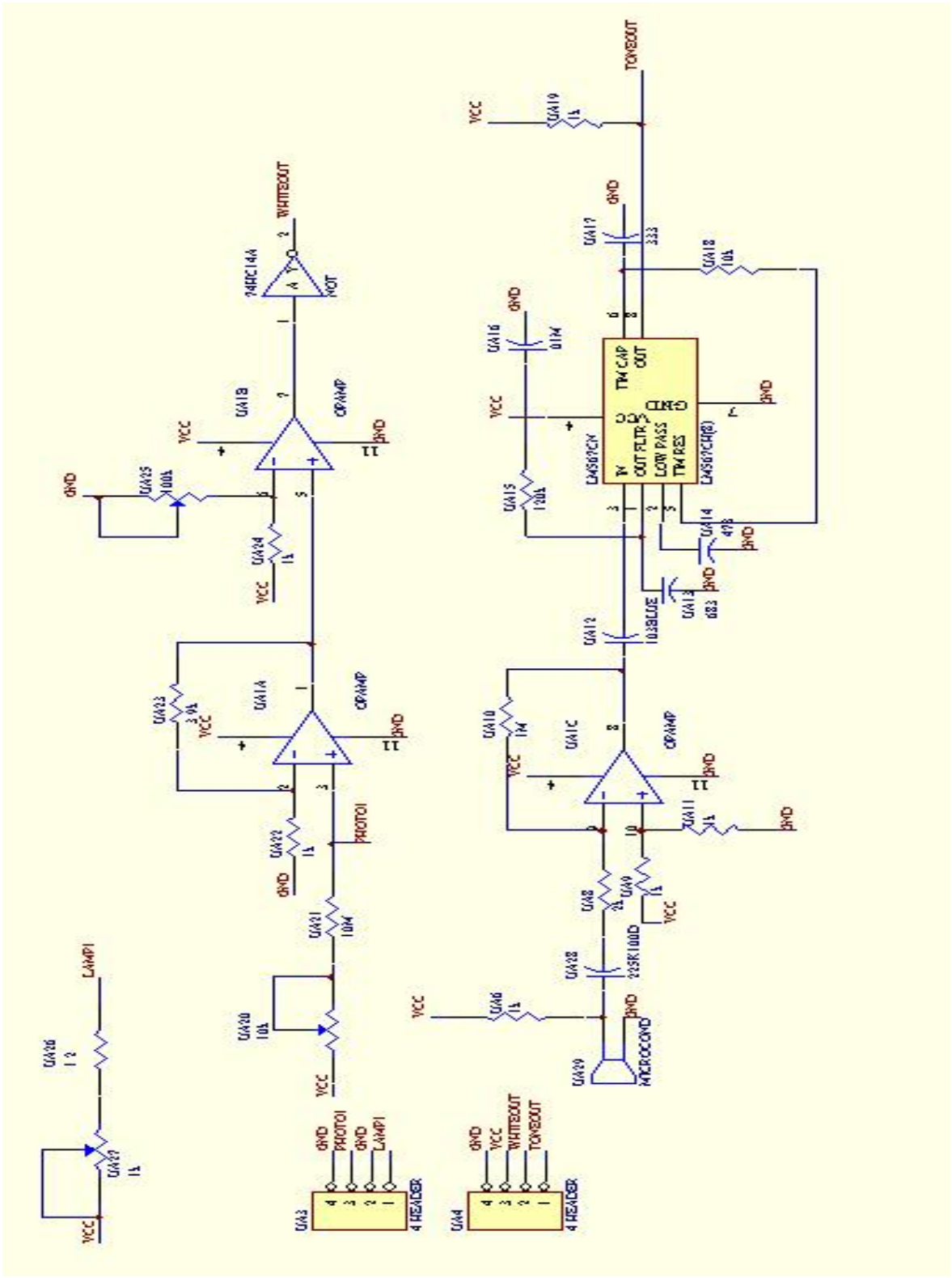
Appendix B
Schematic of the Robot's
Power Distribution System



Appendix C
Schematic of the Motor and Velocity Control Board



Appendix D
Schematic for White-Line Sensor and Tone Decoder



Appendix E
Final Detailed Budget

Quantity	Cost per Item	Total	Donated
2	$ 1.00	$ 2.00	No
2	$ 1.50	$ 3.00	No
2	$ 1.07	$ 2.14	No
2	$ 14.06	$ 28.12	No
1	$ 1.98	$ 1.98	No
2	$ 250.00	$ 500.00	Yes
		$ 20.00	No
1	$ 0.50	$ 0.50	Yes
1	$ 1.99	$1.99	No
1	$ 3.29	$ 3.29	No
1	$ 1.69	$ 1.69	No
3	$ -	$ -	Yes
1	$ 70.00	$ 70.00	Yes
1	$ 4.00	$ 4.00	Yes
1	$ 0.49	$ 0.49	No
1	$ 1.99	$ 1.99	No
1	$ 3.49	$ 3.49	No
1	$ 1.00	$ 1.00	Yes
1	$ 1.00	$ 1.00	Yes
1	$ 24.99	$ 24.99	No
2	$ 3.98	$ 3.98	No
1	$ -	$ -	Yes
3	$ 5.17	$ 5.17	No
3	$ -	$ -	Yes
1	$ 6.99	$ 6.99	No
1	$ 0.99	$ 0.99	No
1	$ 1.29	$ 1.29	No
2	$ 0.99	$ 1.98	No
13	$ 0.13	$ 0.56	No
	$ 50.00	$ 50.00	No
5	$ 4.50	$ 22.50	Yes
	$ 20.00	$ 20.00	No
14	$ 1.99	$ 27.86	No
	TOTAL	**$ 812.99**	
	TECH BUDGET	**$ 97.26**	
	DONATED	**$ 715.73**	

Appendix F
Navigation Algorithm for Rosie the Robot

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


     Filename:      home_maze.c

     Written by: Rebecca Brown

      Purpose:      This is a program to control Rosie the robot
                    through the home maze.  The robot should
                    right-wall follow at a fast pace until it
                    turns the first corner of the maze.  Then,
                    it continues to right-wall follow but slows
                    down and starts checking the hamamatsu for a
                    candle in the first room.  After it passes
                    the first room, it speeds up, and checks the
                    hamamatsu once it detects the entrance to a
                    room.  Once the hamamatsu detects a candle,
                    the robot enters the room for a bit, and
                    then uses the ir sensors to get centered on
                    the flame.  Once centered, the robot goes
                    forward until a white line is detected, and
                    then turns the fan on to extinguish the
                    flame.
   *********************************************************/

/* Include files */
 #include "math.h"
 #include "hc12.h"
 #include "Dbug12.h"

/* Define constants */
 #define Kir 1           /* Ir detection constant      */
 #define Kp 1            /* Motor control constant     */
 #define REVERSE_l 0x01  /* Direction bits for left motor */
 #define FORWARD_l 0XFE
 #define FORWARD_r 0xFD  /* Direction bits for right motor  */
 #define REVERSE_r 0x02
 #define TURN 85         /* Constant amount for turning     */
 #define D_1MS (8000/4)  /* Used for delay function         */

 /* define routine names */
 #define before_first 1
 #define check_first 7
```

```
 #define go_to_first 2
 #define put_out_first 10
 #define after_first 4
 #define check_rooms 5
 #define turn_away 6
 #define put_out 3
 #define go_home_1 12
 #define go_home_2 13
 #define go_home_3 14
 #define go_home_4 15

/* Global variables */
 /* Flags */
 int routine;            /* Says which routine to use       */

 int reflection = 1;     /* Is it a candle or a reflection  */
 int turned_left = 0;    /* Says whether you turned left    */
 int turned_right = 0;   /* Says whether you turned right   */
 int candle_check_done = 0; /* For checking rooms with hama */
 int in_room = 0;        /* Set to 1 when in a room         */

 int white_line_candle = 0;/* Set for white line in a room  */
 int w = 1;              /* Just a loop flag in white line  */
 int temp_fire=0;        /* Set when hama detects a candle  */
 int centered = 0;       /* Set when robot centers on flame */
 int left_wall_flag = 0; /* Used when looking for left walls*/
 int candle_out = 0;     /* Set after fan goes on and off   */
 int data_found;         /* Set when rti interrupts         */
 int first_room_check_done = 0;/* Set when first room passed*/
 int at_candle = 0;      /* Set when white line candle seen */


 /* Counters */
 int go_forward = 0;     /* Counter to go towards the candle*/
 int num_of_rooms = 0;   /* Counts the number of rooms      */
 int go_in_room = 0;     /* Counter to go into the room     */
 int turn_around = 0;    /* Counter to turn around          */
 int left_wall_count = 0;/* Counts number of left walls     */
 int count3 = 0;
 int count2 = 0;         /* Simple counters */
 int count4 = 0;
 int backup = 0;         /* Counter backup after flame out  */
 int to_candle = 0;      /* Make sure your inside white line*/

 /* Values from A/D */
 unsigned char measured_l = 0;/* Meas speed of left motor   */
 unsigned char measured_r = 0;/* Meas speed of right motor  */
```

```
 unsigned char meas_dist_r = 0;/* Meas dist from right wall */
 unsigned char meas_dist_l = 0;/* Meas dist from left wall  */
 unsigned char meas_dist_f = 0;/* Meas dist from front wall */
 unsigned char hama_pulses = 0;/* Read of hama off of PACNT */
 int ir_left = 0;        /* Left ir sensor reading          */
 int ir_right = 1;       /* Right ir sensor reading         */
 int white_line = 0;     /* Read off PORTEB pin 0           */
 int tone = 2;           /* Read off PORTEB pin 1           */

 /* Other variables */
 unsigned char hama = 0; /* Total number of hamamatsu pulses*/
 unsigned char threshold = 0;/* Min value to avoid reflectn */
 int final_room = 0;     /* Where did we end up?            */


 main()
 {
     /* Desired values for hamamatsu */
     unsigned char des_pulses, des_pulses_room_1;

     /* Desired speeds */
unsigned char canned_speed, slow_speed, fast_speed_1,
     slow_speed_2, fast_speed_2;

     /* Desired distances from the wall */
     unsigned char des_dist_left_side_1, des_dist_left_side_2;
     unsigned char des_dist_left_side_3, des_dist_left_side_4;
     unsigned char des_dist_right_side_1;
     unsigned char des_dist_right_side_2;
     unsigned char des_dist_right_side_3;
     unsigned char des_dist_right_side_4;
     unsigned char des_dist_front_1, des_dist_front_2;
     unsigned char des_dist_front_3, des_dist_front_4;

     /* Wall following constants */
     int Kwfl_1, Kwfl_2;
     int Kwfr_1, Kwfr_2, Kwfr_3, Kwfr_4, Kwfr_5;
     int Kwfr_fast, Kwfr_fast_2;
     int Kwff_1, Kwff_2, Kwff_3, Kwff_4, Kwff_fast_2;

     /* Miscellaneous variables */
     int start = 0;
     int count1,error,b;
     int tone_flag = 0;
     int ir_check;
     int canned_turn = 0;
     int far = 0;
```

```
DBug12FNP->printf("Here We Go\n\r");

/* Setup the rti, pulse accumulator, pwm, and A/D */
setup();

/* Setup ports (PORTEA-output, PORTEB-input) */
DDREA = DDREA | 0x01;
DDREB = DDREB & 0x00;

/* Enable interrupts */
enable();

/* Intialize speeds */
fast_speed_2 = 0xc0;
fast_speed_1 = 0x85;
slow_speed = 0x50;
slow_speed_2 = 0x10;
canned_speed = 0x50;

/* Initialize desired distances */
des_dist_left_side_1 = 0x2a;
des_dist_left_side_2 = 0x02;
des_dist_left_side_3 = 0x23;
des_dist_left_side_4 = 0x1a;
des_dist_right_side_1 = 0x2a;
des_dist_right_side_2 = 0x20;
des_dist_right_side_3 = 0x10;
des_dist_right_side_4 = 0x2c;
des_dist_front_1 = 0x10;
des_dist_front_2 = 0x08;
des_dist_front_3 = 0x13;
des_dist_front_4 = 0x09;

/* Initialize pulse variables */
des_pulses_room_1 = 1;
des_pulses = 1;

/* Make sure fan is off */
PORTEA = PORTEA & ~0x04;

count1 = 0;
ir_check = 0;

/* Initialize wall-following constants */
Kwff_1 = 8;
Kwff_2 = 14;
```

```
Kwff_3 = 11;
Kwff_4 = 13;
Kwff_fast_2 = 15;
Kwfl_1 = 4;
Kwfl_2 = 2;
Kwfr_1 = 3;
Kwfr_2 = 5;
Kwfr_3 = 8;
Kwfr_4 = 7;
Kwfr_5 = 6;
Kwfr_fast = 11;
Kwfr_fast_2 = 10;

/* Wait for the correct tone */
while(tone_flag < 4)
{
     /* Rti has interrupted */
     if(data_found == 1)
     {
          choose_routine();

          /* Beginning of program */
          if(routine == before_first)
          {
               /* Decide whether correct tone */
               tone_flag = tone_decoder(tone_flag);
          /* If no tone stay on home spot */
               go(0,0);
               PACNT = 0;
          }
     }
}

/* Begin main loop */
while(1)
{
     /* Wait for RTI to interrupt */
     if(data_found == 1)
     {
          /* Decide where we are in the maze */
          choose_routine();

          /* Go fast before first room */
          if(routine == before_first)
          {
               /* Check for a white line */
               white_line_detection();
```

```
/* Wait for three interrupts and check
     hama reading*/
                    if(count1 == 3)
                    {
                         hama_detection(des_pulses_room_1);
                         count1 = 0;
                         hama = 0;
                         PACNT = 0;
                     }

/* Start out slow to avoid making a
     wheelie */
                    if(start < 5)
                    {
                         start= start +1;
                         right_wall_follow(slow_speed,
des_dist_right_side_2,Kwfr_1, des_dist_front_2, Kwff_1);
                    }
                    /* Go fast */
                    else
                         right_wall_follow(fast_speed_2,
des_dist_right_side_2,Kwfr_fast, des_dist_front_2, Kwff_2);

                    count1 = count1 + 1;
                    hama = hama+hama_pulses;

                    /* Count number of left walls */
                    count_left_wall();

               }

               /* After second left wall, go slow to check
     first room */
               if(routine == check_first)
               {
                    /* Check for a white line */
                    white_line_detection();

                    /* Wait for three interrupts and check
     hama reading */
                    if(count1 == 3)
                    {
                         hama_detection(des_pulses_room_1);
                         count1 = 0;
                         hama = 0;
                         PACNT = 0;
```

```
                    }

                    /* Go slow to check first room */
                    right_wall_follow(slow_speed,
des_dist_right_side_2,Kwfr_1, des_dist_front_1, Kwff_1);
                    count1 = count1 + 1;
                    hama = hama+hama_pulses;
                    count_left_wall();
               }


               /* If hama in first room, left-wall follow to
                    the room */
               if(routine == go_to_first)
               {
                    /*Check for a white line*/
                    white_line_detection();

                    /* Left wall folow into room */
                    left_wall_follow(fast_speed_1,
des_dist_left_side_1, Kwfl_1, des_dist_front_1, Kwff_1);

                    PACNT = 0;
               }

/* If hama in room 2,3,or 4, this routine puts
     out candle */
               if(routine == put_out)
               {
                    /* Go into middle of room */
                    if(go_in_room < 170)
                    {
                        /*Check for white line around candle*/
                         white_line_detection();

                         /* If white line around candle, stop
                              and check ir*/
                         if(white_line_candle == 1)
                              go_in_room = 200;

                         /* Left wall follow into the room */
                         if(num_of_rooms == 3)
                              left_wall_follow(fast_speed_1,
des_dist_left_side_2, Kwfl_1, des_dist_front_1, Kwff_1);
                         else
                              left_wall_follow(fast_speed_1,
des_dist_left_side_4, Kwfl_1, des_dist_front_1, Kwff_1);
```

```
                    go_in_room = go_in_room + 1;
               }
               /* Once in room, center on candle */
               else
               {
                    /* Calibrating sensors? */
                    if(ir_check == 0)
                    {
                         /* Check for a white line */
                         white_line_detection();

                         /* If not centered, then use ir
                              to adjust position */
                         if(centered == 0)
                         {
                              /* If reflection then turn
                                   away and keep checking */
                              if(reflection == 1)
                                   reflection_check();
     /* Continue checking until ir reading is the same */
                              else
                              {
                                   go(0,0);

                                   while(data_found== 0);
                                   choose_routine();
                                   sense_candle();
                              }
                         }
     /* Once centered, check for white line and put out flame */
                         else
                         {
          /* If you see a white line, go forward to make sure you
          are inside*/
                              if(white_line_candle == 1)
                              {
                                   if(at_candle == 0)
                                   {
                                        if(far == 1)
at_candle = go_to_candle(25, slow_speed);
                                        else
at_candle = go_to_candle(30, slow_speed);
                                   }
                                   else
                                        extinguish();
                              }
```

```
          /* Go towards flame until white line detected */
                                    else
                                    {
                                         ir_detection(slow_speed);
                                         far = 1;
                                    }
                               }
                          }
     /* We are just calibrating, so just read ir sensors */
                          else
                          {
                               go(0,0);

                          while(data_found == 0);
                               choose_routine();
                               sense_candle();
                          }
                     }
                     PACNT = 0;
                }

/* Right wall follow fast, until you find a room */
                if(routine == after_first)
                {
                     /* Check for a room */
                     white_line_detection();

                     /* Right wall follow fast */
                     right_wall_follow(fast_speed_2,
des_dist_right_side_1,Kwfr_fast_2, des_dist_front_2, Kwff_fast_2);

                     PACNT = 0;
                }

/* Check for a candle in rooms 2, 3 or 4 using
     hama */
                if(routine == check_rooms)
                {
                     /* Stop entrance and check for candle */
                     go(0,0);

/* Wait for three RTI interrupts and then
     check hama */
                     if(count2 == 3)
                     {
                          hama_detection(des_pulses);
                          candle_check_done = 1;
```

```
                         PACNT = 0;
                    }
                    count2 = count2 + 1;
                    hama = hama_pulses+hama;

               }

/* If no candle in room, turn around and
     continue checking */
               if(routine == turn_away)
               {
                    /* Turn around to the left */
                    go(-canned_speed+0x25,canned_speed+0x35);

                    count3 = count3 + 1;
                    if(count3 == 7)
                    {
                         in_room = 0;
                         count3 = 0;
                    }
                    candle_check_done = 0;
                    PACNT = 0;
               }

/* Go into middle of first room and put out
     candle */
               if(routine == put_out_first)
               {
                    /* Go to middle of first room using rwf */
                    if(go_in_room < 125)
                    {
                         /* Check white line around candle */
                         white_line_detection();

/* If candle, then stop and get
     centered */
                         if(white_line_candle == 1)
                              go_in_room = 200;

right_wall_follow(fast_speed_1,
des_dist_right_side_3,Kwfr_1, des_dist_front_1, Kwff_1);
                         go_in_room = go_in_room + 1;
                    }
                    /* When middle room, center on candle */
                    else
                    {
                         /* Not calibrating sensors */
```

```
                         if(ir_check == 0)
                         {
/* Check for white line around
     candle */
                              white_line_detection();

                              /*If not centered,get centered*/
                              if(centered == 0)
                              {
/* If reflection, turn away
     and keep checking */
                                   if(reflection == 1)
                                        reflection_check();
/* Use ir sensors to center
     on flame */
                                   else
                                   {
                                        go(0,0);
                                        while(data_found== 0);
                                        choose_routine();
                                        sense_candle();
                                   }
                              }
/* If centered, go to flame and
     put out */
                              else
                              {
/* If white line found, go
inside of line and extinguish*/
                                   if(white_line_candle == 1)
                                   {
/* Get close enough to
     candle */
                                        if(at_candle == 0)
                                        {
                                             if(far == 0)
at_candle = go_to_candle(30, slow_speed);
                                             else
at_candle = go_to_candle(25, slow_speed);
                                        }
                                        /* Put out flame */
                                        else
                                             extinguish();
                                   }
                                   /* If no white line, go
     towards flame */
```

```
else

  ir_detection(slow_speed);
                              }
                         }
                         /* If we are calibrating sensors,
     just check the ir readings */
                         else
                         {
                              go(0,0);
                              while(data_found== 0);
                              choose_routine();
                              sense_candle();
                         }
                    }
                    PACNT = 0;
               }

               /* Routine to go home from first room */
               if(routine == go_home_1)
               {

                    /* Turn away from flame to the right */
                    if(canned_turn < 10)
                    {

                         if(canned_turn == 0)
                              go(canned_speed,-canned_speed);
                         canned_turn = canned_turn + 1;
                         in_room = 3;
                    }
                    /* Left wall follow home */
                    else
                    {
                         /* Count white lines */
                         white_line_detection();
                         /* Stop at the home spot */
                         if(in_room == 1)
                              go(0,0);
                         else
                              left_wall_follow(fast_speed_1,
des_dist_left_side_3, Kwfl_2, des_dist_front_1, Kwff_3);
                    }
               }
```

```
          /* Routine to go home from room 4 */
               if(routine == go_home_4)
               {

                    /* Turn away from flame to the left */
                    if(canned_turn < 15)
                    {
                         if(canned_turn == 0)
                              go(-canned_speed,canned_speed);
                         canned_turn = canned_turn + 1;
                         in_room = 3;
                    }
                    /* Right wall follow home */
                    else
                    {
                         /* Count number of white lines */
                         white_line_detection();

                         /* Stop at home spot */
                         if(in_room == 1)
                              go(0,0);
                         else
                              right_wall_follow(fast_speed_1,
des_dist_right_side_1, Kwfr_4, des_dist_front_3, Kwff_4);
                    }

               }

               /* Routine to go home from room 2 */
               if(routine == go_home_2)
               {
                    /* Turn away from candle to the left */
                    if(canned_turn < 9)
                    {

                         if(canned_turn == 0)
                              turn_left();
                         canned_turn = canned_turn + 1;
                         in_room = 3;
                    }
                    /* Go home */
                    else
                    {
                         /* Check white line */
                         white_line_detection();
```

```
                    /* Stop on home spot */
                         if(in_room == 1)
                              go(0,0);
                         else
                         {
     /* Left wall follow when out of room */
                              if(in_room == 2)

  left_wall_follow(fast_speed_1,
     des_dist_left_side_1,
     Kwfl_1, des_dist_front_1, Kwff_1);
     /* Right wall follow when in room */
                              else
                               right_wall_follow(fast_speed_1,
des_dist_right_side_2, Kwfr_2, des_dist_front_1, Kwff_1);
                         }
                    }
               }

               /* Routine to go home from room 3 */
               if(routine == go_home_3)
               {
                    /* Turn away from candle to the left */
                    if(canned_turn < 9)
                    {

                         if(canned_turn == 0)
                              turn_left();
                         canned_turn = canned_turn + 1;
                         in_room = 5;
                    }
                    /* Right wall follow home */
                    else
                    {
                         /* Check white line */
                         white_line_detection();
                         if(in_room == 1)
                              go(0,0);
                         else
                         {
                              /* Turn away from room 4 */
                              if(in_room == 3)
                              {
                                   if(turn_around < 10)
                                   {
                                   turn_around = turn_around + 1;
                                   turn_left();
```

```
                                }
                                     else
                                          in_room = 2;
                                }
                                /* Right wall follow home */
                                else
                                 right_wall_follow(fast_speed_1,
des_dist_right_side_1, Kwfr_2, des_dist_front_1, Kwff_1);
                           }
                     }

               }
          }
     }
     DBug12FNP->printf("end of prg\n\r");
 }

/*************************************************************
 *                                                           *
 * Function:        rti_isr()                                *
 *                                                           *
 *escription:  Interrupt service routine for the rti   *
          interrupt.  When the rti interrupts,    *
*                   data is read off of the A/D, ports and  *
*                   the pulse accumulator.                  *
 *************************************************************/

 @interrupt void rti_isr(void)

 {
     /* Set the data found flag */
     data_found = 1;

     /* Read the ir sensors */
     ir_right = ADR0H;
     ir_left = ADR1H;

     /* Read the motor speeds */
     measured_l = ADR3H;
     measured_r = ADR4H;

     /* Read the distance sensors */
     meas_dist_r = ADR5H;
     meas_dist_l = ADR7H;
     meas_dist_f = ADR6H;
```

```
/* Read the hama pulses off the pulse accumulator */
     hama_pulses = PACNT;

     /* Read data off of the ports */
     white_line = PORTEB & 0x01;
     tone = PORTEB & 0x02;

     /* Clear the rti interrupt */
     RTIFLG = 0x80;
 }

 /*************************************************************
 *                                                           *
 * Function:        go(int des_l, int des_r)                 *
 *                                                           *
 * Description:     This function takes a desired right and  *
 *                  left motor speed and computes the        *
 *                  correct duty cycle need to make the      *
 *                  motors go the desired speed.  In order   *
 *                  to do this the current measured speed of*
 *                  each motor is also needed                *
 *************************************************************/

 void go (int des_l, int des_r)
 {
     /* Calculated duty cycles of left and right motors */
     unsigned char dc_l, dc_r;
     /* Direction of left and right motors */
     unsigned char direction_l, direction_r;
     /* Temporary integer values, used for calculations */
     int temp_l, temp_r;

/* Continuously calculate the duty cycle from desired and
     measured speeds */
     /* Do forward calculation for left motor*/
     if(direction_l == FORWARD_l)
          temp_l = (0x14*des_l/0x2f)
     + Kp * (des_l - measured_l)/6;

     /* Do reverse calculation for left motor */
     else
          temp_l = (0x14*des_l/0x2f)
     + Kp * (des_l + measured_l)/6;


     /* Do forward calculation for right motor */
     if(direction_r == FORWARD_r)
```

```
     temp_r = (0x14*des_r/0x2f)
+ Kp * (des_r - measured_r)/6;

/* Do reverse calculation for left motor */
else
     temp_r = (0x14*des_r/0x2f)
+ Kp * (des_r + measured_r)/6;


/* Set the duty cycle as an unsigned number */
dc_l = temp_l;
dc_r = temp_r;

/* Make sure duty cycle doesn't go over max value */
if(temp_l > 0xfe)
     dc_l = 0xfe;
if(temp_l < -254)
     dc_l = 0xfe;

if(temp_r > 0xfe)
     dc_r = 0xfe;
if(temp_r < -254)
     dc_r = 0xfe;

/* Set direction bit for left motor */
if(temp_l >= 0)
{
     direction_l = FORWARD_l;
     PORTEA = PORTEA & direction_l;
}
else
{
     direction_l = REVERSE_l;
     PORTEA = PORTEA | direction_l;
}

/* Set direction bit for right motor */
if(temp_r >= 0)
{
     direction_r = FORWARD_r;
     PORTEA = PORTEA & direction_r;
}
else
{
     direction_r = REVERSE_r;
     PORTEA = PORTEA | direction_r;
}
```

```
     /* Reset the routine and data flags to 0 */
     routine = 0;
     data_found = 0;

     /* Send motor the current duty cycle */
     PWDTY3 = dc_l;
     PWDTY2 = dc_r;

}

/*************************************************************
 *                                                           *
 * Function:        left_wall_follow()                       *
 *                                                           *
 * Description:     This function takes a desired speed,     *
 *                  desired distances from left and front    *
 *                  walls, and left and front wall following*
 *                  constants.  It computes the desired left*
 *                  and right motor speeds needed to keep    *
 *                  the robot at the desired distance from   *
 *                  the left and front walls. The walls      *
 *                  input is only used when the robot is     *
 *                  close enough to the front wall.          *
 *************************************************************/

void left_wall_follow(unsigned char des_speed, unsigned char
des_dist_left_side, int Kwfl, unsigned char des_dist_front, int
Kwff)
 {
     /* Desired left and right motor speeds */
     int des_left, des_right;
     /* Calculated front wall error */
     int front_error;

     /* Calculate the desired left speed based on the left
     wall data */
     des_left = des_speed
+ Kwfl*(meas_dist_l - des_dist_left_side)/2;

     /* Calculate the desired right speed based on the left
     wall data */
     des_right = des_speed
     -  Kwfl*(meas_dist_l - des_dist_left_side)/2;

     /* Add in front data if robot gets close to front wall */
     if(meas_dist_f >= des_dist_front)
```

```
     {
          front_error = meas_dist_f - des_dist_front;
          des_left = des_left + Kwff*front_error/2;
          des_right = des_right - Kwff*front_error/2;
     }

     /* Send desired speeds to the go function */
     go(des_left,des_right);
 }


/*********************************************************
 *                                                       *
 * Function:        right_wall_follow()                  *
 *                                                       *
 *  Discription:    This function takes a desired speed,    *
 *                  desired distances from right and front  *
 *                  walls, and right and front wall         *
 *                  following constants.  It computes the   *
 *                  desired left and right motor speeds     *
 *                  needed to keep the robot at the desired *
 *                  distance from the right and front walls.*
 *                  The walls input is only used when the   *
 *                  robot is close enough to the front wall.*
 *********************************************************/

 void right_wall_follow(unsigned char des_speed, unsigned char
des_dist_right_side, int Kwfr, unsigned char des_dist_front, int
Kwff)
 {
     /* Desired left and right motor speeds */
     int des_left, des_right;
     /* Calculated front wall error */
     int front_error;

     /* Calculated the desired right speed based on the right
     wall data */
     des_right = des_speed
     + Kwfr*(meas_dist_r - des_dist_right_side)/3;

     /* Calculated the desired left speed based on the right
     wall data */
     des_left = des_speed
     -  Kwfr*(meas_dist_r - des_dist_right_side)/3;

     /* Add in front data if robot gets close to front wall */
     if(meas_dist_f >= des_dist_front)
```

```
    {
         front_error = meas_dist_f - des_dist_front;
         des_right = des_right + Kwff*front_error/2;
         des_left = des_left - Kwff*front_error/2;
    }

    /* Send desired speeds to the motors */
    go(des_left,des_right);
}

/*************************************************************
 *                                                           *
 * Function:        hama_detection()                         *
 *                                                           *
 * Description:     If number of pulses is greater than the  *
 *                  desired pulses set the temp_fire flag.   *
 *************************************************************/

void hama_detection(unsigned char pulses)
{

    if(hama >= pulses)
         temp_fire = 1;

    PACNT = 0;

}
```

```
/*************************************************************
 *                                                           *
 * Function:        white_line_detection()                   *
 *                                                           *
 * Description:     This function checks for a white line.   *
 *                  If it sees a white line, it waits until  *
 *                  a black is seen before checking for      *
*                   another white line.  If a white line is  *
*                   seen outside of a room, the flag in_room*
*                   is set and the number of rooms is        *
*                   incremented.  If in a room with a candle*
*                   and a white line is seen, the            *
*                   white_line_candle flag is set. If in a   *
*                   room and not candle is in the room a     *
*                   white line will decrement the in_room    *
*                   flag back to 0.                          *
*************************************************************/

 void white_line_detection(void)
 {
     int i = 0;

     /* Check the white line sensor */
     if(white_line == 1)
     {
          /* If first time line is seen */
          if(w == 0)
          {
/* If entrance to room, increment num_of_rooms and in_room */
               if(in_room == 0)
               {
                    in_room = in_room + 1;
                    num_of_rooms = num_of_rooms + 1;
               }
               /* If already in room */
               else
               {
          /* If fire in room, set white_line_candle flag */
                    if(temp_fire == 1)
                         white_line_candle = 1;
                    /* If no fire, decrement in_room  */
                    else
                        in_room = in_room-1;
               }
          }
```

```
    /* Set this flag so only one line will be counted */
         w = 1;
    }
    /* Wait for end of line */
    else
         w = 0;
}

/*************************************************************
 *                                                           *
 * Function:        ir_detection(des_speed)                  *
 *                                                           *
 * Description:     This function goes toward the flame with*
 *                  a desired speed, adjusting speed using   *
 *                  ir sensor input to direct the robot      *
 *                  towards the flame.                       *
 *************************************************************/

void ir_detection(unsigned char des_speed)
{

    int ir_error, des_left, des_right;
    int front_error;

    /* Compute the difference between ir sensor reading */
    ir_error = ir_left - (ir_right-0x02);

    /* Compute the desire left and right speeds using this ir
    error */
    des_left = des_speed + Kir*(ir_error)/3;
    des_right = des_speed  -  Kir*(ir_error)/3;

    /* Delay to help slow down the robot */
    delay(2);

    /* Send desired speeds to the motors */
    go(des_left,des_right);
}
```

```
/*************************************************************
 *                                                           *
 * Function:        sense_candle()                           *
 *                                                           *
 * Description:     This function computes the error between*
 *                  the ir sensors and uses this error to    *
 *                  turn in the direction of the at tiny     *
 *                  increments, until the error is less than*
 *                  2. Once this error is small, the value   *
 *                  of the ir sensors is compared with a     *
 *                  threshold to make sure that the sensors  *
 *                  aren't seeing a reflection.  If it is a  *
 *                  reflection, a flag is set to cause the   *
 *                  robot to turn away from the reflection   *
 *                  and continue checking.  Once centered    *
 *                  and no reflection, the centered flag is  *
 *                  set.                                     *
 *************************************************************/

 void sense_candle()
 {

     int ir_error, des_left, des_right;

     /* Ir sensors differ more if candle is farther away */
     if(white_line_candle == 0)
          ir_error = (ir_left-0x06) - ir_right;
     else
          ir_error = ir_left - (ir_right - 0x02);

     /* If error positive, turn right */
     if(ir_error > 2)
          turn_right();
     else
          /* If error negative, turn left */
          if(ir_error < -2)
               turn_left();
          else
/* If centered, do threshold check for
     reflection */
               if(ir_left < threshold)
                    reflection = 1;
               else
                    centered = 1;
 }
```

```
/*************************************************************
 *                                                           *
 * Function:        reflection_check()                       *
 *                                                           *
 * Description:     This function sets a threshold based on *
 *                  the room number that the candle is in.  *
 *                  Then check to see if the ir sensors are *
 *                  greater than the threshold.  If         *
 *                  reflection turn away from the reflection*
 *                  and continue to check.                   *
*************************************************************/

 void reflection_check()
 {

     /* Room 1 */
     if(num_of_rooms == 1)
     {
          /* High threshold because its a small room */
          threshold = 0x30;
          if(ir_right < threshold)
               turn_left();
          else
               reflection = 0;
     }

     /* Room 2 */
     if(num_of_rooms == 2)
     {
          /* High threshold because its a small room */
          threshold = 0x37;
          if(ir_left < threshold)
               turn_right();
          else
               reflection = 0;
     }

     /* Room 3 */
     if(num_of_rooms == 3)
     {
          /* Low threshold because large room */
          threshold = 0x10;
          if(ir_left < threshold)
               turn_right();
          else
               reflection = 0;
     }
```

```
    /* Room 4 */
    if(num_of_rooms == 4)
    {
         /* Low threshold becuase large room */
         threshold = 0x10;
         if(ir_left < threshold)
              turn_right();
         else
              reflection = 0;
    }
}

/*************************************************************
 *                                                           *
 * Function:        turn_left()                              *
 *                                                           *
 * Description:     This function just turns the robot left  *
 *                  a little bit.                            *
 *************************************************************/

void turn_left()
{
    int des_left, des_right;

    /* Sets flags saying which way it turned */
    turned_left = 1;
    turned_right = 0;

    /* Sets the desired left and right speeds to make it turn
    left */
    des_left = -TURN+0x40;
    des_right = TURN+0x50;

    /* Send desired speeds to the motors */
    go(des_left,des_right);
}
```

```
/*************************************************************
 *                                                           *
 * Function:        turn_right()                             *
 *                                                           *
 * Description:     This function just turns the robot right*
 *                  a little bit                             *
 *************************************************************/

 void turn_right()
 {
     int des_left, des_right;

     /* Set flags saying which way it turned */
     turned_right = 1;
     turned_left = 0;

/* Set the desired left and right speeds to make it turn
     right */
     des_left = TURN+0x50;
     des_right = -TURN+0x40;

     /* Send desired speeds to the motors */
     go(des_left,des_right);
 }

/*************************************************************
 *                                                           *
 * Function:        blow_fan()                               *
 *                                                           *
 * Description:     This function sends a signal to the fan *
 *                  to turn it on. Then it delays for a      *
 *                  while and turns it back off.             *
 *************************************************************/

 int blow_fan()
 {
     int out = 0;

     /* Turn on fan */
     PORTEA = PORTEA | 0x04;
     /* Delay for 8sec */
     delay(800);
     /* Turn off fan */
     PORTEA = PORTEA & ~0x04;

     return(1);
 }
```

```
/************************************************************
 *                                                          *
 * Function:        delay(int ms)                           *
 *                                                          *
 * Description:     This is a functions to delay for ms     *
 *                  milliseconds                            *
 ************************************************************/

 void delay(unsigned int ms)
 {
     int i;

     while (ms > 0)
     {
          i = D_1MS;
          while (i >0)
               i = i - 1;
          ms = ms - 1;
     }
 }

/************************************************************
 *                                                          *
 * Function:        tone_decoder()                          *
 *                                                          *
 * Description:     This function checks to see if there is *
 *                  a correct tone for a long enough time   *
 *                  period.                                 *
 ************************************************************/

 int tone_decoder(int tone_flag)
 {

     if(tone == 0)
          tone_flag = tone_flag + 1;
     if(tone == 2)
          if(tone_flag > 0)
               tone_flag = 0;

     return(tone_flag);
 }
```

```
/*************************************************************
 *                                                           *
 * Function:        count_left_wall()                        *
 *                                                           *
 * Description:     This function increments the left_wall   *
 *                  count once a new left wall is detected   *
 *                  by the left sensor.                      *
 *************************************************************/

 void count_left_wall(void)
 {

     /* The distance of 0x13 is the distance for a wall */
     if(meas_dist_l > 0x13)
     {
          /* If new wall increment left_wall_flag */
          if(left_wall_flag == 0)
          {
               /* If two walls, the first room is passed */
               if(left_wall_count == 2)
                    num_of_rooms = 1;
               left_wall_count = left_wall_count + 1;
               left_wall_flag = 1;
          }
     }
     /* Wait for a new wall */
     else
        left_wall_flag = 0;

 }

/*************************************************************
 *                                                           *
 * Function:        choose_routine()                         *
 *                                                           *
 * Description:     This function decides on the correct     *
 *                  routine based on number of left walls,   *
 *                  candle seen, and candle out data         *
 *************************************************************/

 void choose_routine(void)
 {
     /* If candle hasn't be outted */
     if(candle_out == 0)
     {
          /* If you haven't passed first room */
```

```
if(left_wall_count < 3)
{
     /* If there is no fire in first room */
     if(temp_fire == 0)
     {
          /* If you are in front of first room */
          if(left_wall_count == 2)
               routine = check_first;
          else
               routine = before_first;
     }
     /* If there is a fire in first room */
     else
     {
          /* Wait till you've turned the corner */
          if(left_wall_count == 2)
          {
     /* Left wall follow until entrance of
room */
               if(in_room == 0)
                    routine = go_to_first;
               /* Go into room and extinguish */
               else
                    routine = put_out_first;
          }
          /* Wait till you turn the corner */
          else
               routine = before_first;
     }
}
/* You have passed the first room */
else
{
     /* If not in a room, right wall follow fast */
     if(in_room == 0)
     {
          hama = 0;
          count2 = 0;
          routine = after_first;
     }
     /* Find a room, check it for hama */
     else
     {
          /* Check room with hama */
          if(candle_check_done == 0)
          {
               if(count3 < 1)
```

```
                              routine = check_rooms;
                         else
                              routine = turn_away;
                    }
                    else
                    {
                         /* If no fire, turn away */
                         if(temp_fire == 0)
                              routine = turn_away;
                         /* Fire, put it out */
                         else
                              routine = put_out;
                    }
               }
          }
     }
     /* Candle out -> go home */
     else
     {
          /* Deciding what room the candle was in, go home */
          if(final_room == 2)
               routine = go_home_2;
          else
               routine = go_home_1;
          if(final_room == 3)
               routine = go_home_3;
          if(final_room == 4)
               routine = go_home_4;
     }
 }

/*************************************************************
 *                                                           *
 * Function:        setup()                                  *
 *                                                           *
 * Description:     This function calls functions to setup   *
 *                  pwm, A/D, rti, and pulse accumulator.    *
 *************************************************************/

 void setup()
 {
     pwm_set();
     ad_set();
     rti_set();
     puls_set();
 }
```

```
/*************************************************************
 *                                                           *
 * Function:        pwm_set()                                *
 *                                                           *
 * Description:     Sets up the pwm for a 1.0kHz pulse width*
 *                  modulation on Port PP1 and PP2.          *
*                   Note: Something happened to channel      *
 *                  0 & 1 on HC12.                           *
*                   They no longer work.  So we are using    *
*                   channels 2 & 3.  PWDTY2 = right motor    *
*                   and PWDTY3 = left                        *
 *************************************************************/

 void pwm_set()
 {
     PWCLK &~ 0xC0;  /* Choose 8-bit mode  */

     PWCTL = PWCTL &~ 0x08;  /* Choose left-aligned */

PWPOL = PWPOL &~ 0xf0;  /* Select clock mode 0 for
     Channel 2 & 3      */
PWPOL = PWPOL | 0x0f;   /* Choose polarity; High to start
     Channel 2 & 3 */

     PWCLK = PWCLK | 0x28;   /* select 1 khz for channel 1 */
     PWCLK = PWCLK | 0x05;   /* select 1 khz for channel 2 */

     PWDTY1 = 0x50;          /* duty cycle for channel 1; reg
     0x51 */
     PWDTY2 = 0xf0;          /* duty cycle for channel 2; reg
     0x52 */

     PWEN = PWEN | 0x08;     /* enable channel 3 */
     PWEN = PWEN | 0x04;     /* enable channel 2 */
 }
```

```
/*************************************************************
 *                                                           *
 * Function:        ad_set()                                 *
 *                                                           *
 * Description:     This function sets up the Analog to      *
 *                  Digital Converter                        *
 *************************************************************/

 void ad_set()
 {
     int i=0;

     /* Analog to Digital Converter set-up */

     ATDCTL2 = 0xC0;        /* powerup A/D            */
     ATDCTL4 = 0x01;        /* 9 us conversions       */
     ATDCTL5 = 0x73;        /* bit 3;1 chl; only continous; do
     eight conversions*/

     for (i= 0;i<100; i++); /* AD start up delay      */
 }




/*************************************************************
 *                                                           *
 * Function:        rti_set()                                *
 *                                                           *
 * Description:     This function set up the Real-Time       *
 *                  interrupt.                               *
 *************************************************************/

 void rti_set()
 {
     /*RTI setup code */

     RTICTL = 0x05;           /* 16ms delay */
     RTICTL = RTICTL | 0x80;  /* eable RTI inturrpt */
     RTIFLG = 0x80;           /* clearing RTI flag*/
 }
```

```
/*************************************************************
 *                                                           *
 * Function:        puls_set()                               *
 *                                                           *
 * Description:     This function sets up the pulse          *
 *                  accumulator                              *
 *************************************************************/

 void puls_set()
 {
     /* code to set-up the pulse accumulator */

     PACTL = 0x50; /*enable pulse acc. ; event mode; count
     rising edge */
     PACNT = 0;
 }

/*************************************************************
 *                                                           *
 * Function:        extinguish()                             *
 *                                                           *
 * Description:     This function blows out the candle, then*
 *                  checks the ir sensors to make sure that *
 *                  the candle is out.  It sets the final   *
 *                  room based on the room that the candle  *
 *                  is in.                                   *
 *************************************************************/

 void extinguish()
 {
     int out = 0;
     int blow_done = 0;

     /* Stop while you blow out the candle */
     go(0x0000,0x0000);

     /* Do this as long as there is a flame */
     while(out == 0)
     {
          DBug12FNP->printf("flag = %d\n\r",routine);

          /* Turn on fan for a while */
          out = blow_fan();

          /* Turn off fan */
          PORTEA = PORTEA & ~0x04;
```

```
          /* Check ir sensors */
          if(ir_left < 0x10)
               out = 1;
          if(ir_right < 0x10)
               out = 1;
     }

     /* Set final room and other flags */
     final_room = num_of_rooms;
     candle_out = 1;
     temp_fire = 0;

 }

/*************************************************************
 *                                                           *
 * Function:        go_to_candle()                           *
 *                                                           *
 * Description:     This function goes a given distance at a*
 *                  given speed. It is used to go inside the*
 *                  white line around the candle.            *
 *************************************************************/

 int go_to_candle(int distance, unsigned char speed)
 {

     /* Use the ir detection function to go toward candle */
     if(to_candle < distance)
     {
          ir_detection(speed);
          to_candle = to_candle + 1;
          return(0);
     }
     else
          return(1);
 }
```