# The Life and Times of Jerry Rig

Written By:

Ben Hoover

Ryan Schmidt

Ben Silva

David Tu

## Abstract

For EE 382, Introduction to Design, we created a robot that would compete in the Trinity College Fire Fighting Home Robot Contest. The design phase organized the robot and insured that the modular components would interact. The systems of the robot are broken down into power, chassis, sensors, and software. Power is divided between the motor systems and the logic plus sensors. For the chassis, a differential drive configuration offered the best performance. To navigate the maze, we used three types of sensors: wall detection, flame sensing, and white line detection. The pulse width modulation is implemented in an HC12. Software uses closed loop control algorithms with proportional and integral. Wall following and an elimination method was used to run through the four rooms. Circuit boards were designed in MicroSim.



Keywords: Modular components, differential drive, sensors, pulse width modulation, closed loop control, microcontroller

## Table of Contents

# Table of Figures

## Introduction

The basis of this project is to design and build a robot that will be able to compete in the Trinity Fire Fighting Robot Contest. The robot must be able to navigate a maze find a candle and put it out. We will explain how we designed all of the major systems necessary to accomplish the task, and then cover the methods employed to construct each of these systems.

## Design

The design process was essential to get a robot that would be easy to construct and modify for different operating conditions. We decided to focus on coming up with a design that was as simple as possible and also easily adjustable. It was also important to make the design modular, and still keep the wire connections as short as possible to prevent any noise problems that might arise. We began the design process by coming up with a system block diagram that showed all of the major systems and how they would be interconnected. Next, we had to decide on a chassis that would allow us to build the modular design and still keep most of the signal connections short. Once a chassis layout was agreed upon, we were able to begin researching and designing each of the modular subsystems including the individual sensors, fire suppression system, motor feedback, and the motor driver board. Finally we had to get these systems to work together to implement closed loop motor control.

System Block Diagram

The system block diagram shows all of the major systems that we saw as being necessary to build a fully functional fire fighting robot. In creating the block diagram we were able to consider how each system would be connected and figure out how to supply power to each of the subsystems in the most efficient manner possible.

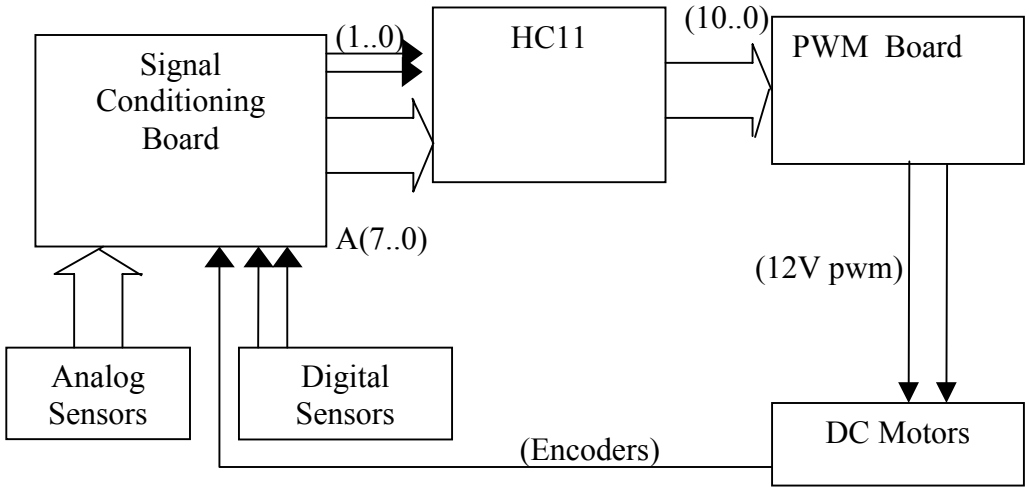


Figure 1. System Block Diagram

The first thing we had to decide upon was which microprocessor to use to take the sensor data and control the motors. This wasn't a difficult decision since we already had a great deal of experience working with the Motorola HC11 microprocessor.

The second consideration in designing a system block diagram was getting all of the sensor information to the microprocessor. The above block diagram shows how we planned to integrate all of the sensor information for use by the microprocessor. We decided to use one board to take all of the sensor input and route it through two ribbon cables to the HC11. One cable would carry all of the analog data directly to the input of the analog to digital converters on the HC11, while the other cable would carry only the digital signals. Also, this board would be able to supply the necessary power to each of

the sensors on the same cable that carried the information from the sensors to the signal conditioning board. All of the sensors would thus spider out from this board. By placing this board in the center of the chassis we could limit the length of the connections to each of the sensors, thereby limiting the noise on these lines.

The final consideration in designing the block diagram was motor control. We wanted a single board to do all of the pulse width modulation so that the processor wouldn't have to spend the time doing it. We decided to use an Altera chip that would take a number representing the percent duty cycle from the processor and pulse width modulate a motor control signal based on this input. The motor control board would also convert this signal into a high current, 12V signal to be used by the motors.

Power Distribution

The next thing that we needed to decide was how we were going to get power to all of the circuitry. At first we wanted to use the 12V camcorder battery to power everything. This meant that we would have to put 5V regulators on every board. We chose to use linear regulators even though they aren't the most efficient method of down converting voltage supplies. We stuck with this design until somehow one of our regulators burnt out and destroyed our pulse width modulation chip. Since the Altera chip we were using was pretty expensive, we decided to avoid the possibility of this occurring again by using a separate battery for all of the 5V circuitry. We also found that the chassis was still not balancing well enough so we decided to add another battery for ballast. The final power distribution design is shown in the following block diagram.

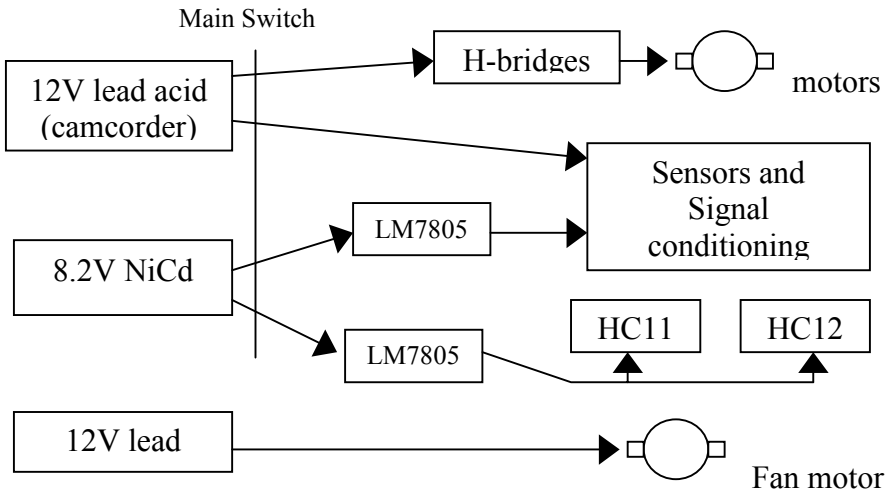


Figure 2. Power Distribution Block Diagram

We chose to use an 8.2V nickel cadmium battery for the 5V supply because Ryan already had one that we could use. This voltage could then be regulated using two LM7805 linear regulators. We used two separate regulators so that neither one would be dissipating too much power.

The ground wire from each battery was connected to a master ground, which was then attached to the chassis. We were also very careful not to create any ground loops by connecting supply grounds anywhere else on the robot. The two main power supplies, the 8.2V NiCa battery and the 12V lead acid camcorder battery, were connected to the rest of the robot with a master switch so that we could quickly shut off power to all of the driving circuitry if something started smoking. We didn't feel that this was necessary for the fan motor since once the control circuitry is shut off, the fan can no longer be turned on. The complete power budget is in the appendix.

**Chassis**

We decided that the simplest way to navigate the maze would be to use a differential drive design mounted on a round chassis. We decided to mount the wheels

on the center line of the chassis so that we could make turns in place. We were given a circular base (9.75 inches diameter) with two Pittman GM9000 motors mounted onto it. We decided to use a 3 wheel design, the third wheel was a simple caster wheel made out of a very rigid paper clip and wheels from a pine-wood derby car. This was mounted through the back of the base. We allowed the caster wheel to swivel through the use of a nylon screw. The screw was fitted through the hole and tightened. We then drilled a hole through the screw and fit the paper clip in the hole. The paper clip was crimped above and below the screw. This design was sturdy and very simple to make and use. The next step was to make the levels. The upper levels were made out of plexi-glass and had the same diameter as the base. They were mounted using 8-40 all-thread. This enabled us to quickly change the height of the levels. The base was threaded to fit the all-thread so that we wouldn't have any bolts sticking out of the bottom of the base which might disable the sophisticated Velcro battery mounting system. The levels were secured with nuts and lock washers above and below each level.

Each of the circuit boards was mounted onto the levels using non-conductive spacers and 6-32 screws. The H-bridge board and line sensor board were mounted onto the bottom of the base in the same way. The GP2D12s were mounted on top of the base with Velcro, and the sensor board was sitting directly over the sensor mounting block. Our three batteries (one on the bottom, two on top of the base) were mounted on the back of the robot so that it wouldn't tip over. The next level consisted of the HC11, HC12 and the fire sensors; this took up the whole level. The top level just had the fan motor and the motor driving circuitry. Figure 1 in the appendix shows the basic chassis layout of Jerry Rig.

**Sensors**

We took a long time researching and testing different types of sensors to get the most space efficient and simplistic design possible. The sensors that were necessary for the fire fighting competition were: distance sensors, white line sensor, short and long range fire sensors, and a tone decoder. When designing the sensor modules we had to make sure that the signal that was being sent to the signal conditioning board was pre-amplified enough that small noise induced on these connections would not corrupt the signal. We accomplished this by putting some amplifiers on the sensor boards themselves.

Wall Detection

A high priority for the robot was being able to navigate through the maze; specifically to follow walls. Our first plan was to use the Sharp GP1U52X modulated infrared sensor. The GP1U52X sensor uses the intensity of the reflection off the wall to determine distance. A flaw in the design was that the sensor needed modulated light to work. This meant extra complexity and parts. Also, the GP1U52X sensor was dependant on the reflective properties of the wall and its color. We heard about Sharp's GP2D12 sensors, which use triangulation from an infrared emitter/detector pair.

The GP2D12 sensors meant simplicity in circuit design and less dependence on wall conditions. The GP2D12 sensors were very easy to implement. They required 5V, ground and a third pin produced the analog output signal. This signal's maximum output was half the operating voltage. Since our A/D converter on the HC11 would be in

reference to 5V we felt it prudent to amplify the signal to 5V with an amplifier having a gain of two. Another problem with the GP2D12 was that it gave a peak value at 10cm; any closer and the voltage would drop (see figure 3). To fix this, the sensors were mounted 10cm inside the edge of the robot. The end result gave us a reliable wall detector with great resolution from the A/D converter of the HC11.

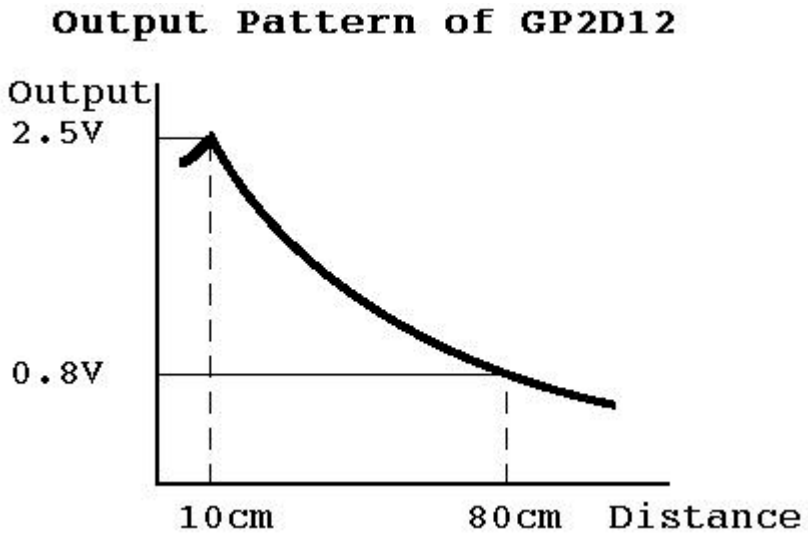

Figure 3. Graph of Voltage vs Distance for GP2D12

Line Sensor

The ability to realize whether or not the robot was in a room was a major point of the competition. The rooms were marked with white lines on the black floor. The lines span the doorway and were placed around the candle. Detecting these markings was essential for control. Our first design used a phototransistor with a resistor connected between the emitter and ground. The output was taken off of the emitter and run through an adjustable comparator. The comparator was given a variable reference voltage so that the sensor could be calibrated to the ambient lighting conditions. A problem with this design was that the amount of current the comparator drew from the phototransistor

changed depending on the level of the reference voltage. A buffer was added in between the transistor/resistor pair and the comparator (see figure 4). This configuration produced a 0V signal whenever a line was present and a 5V signal otherwise.

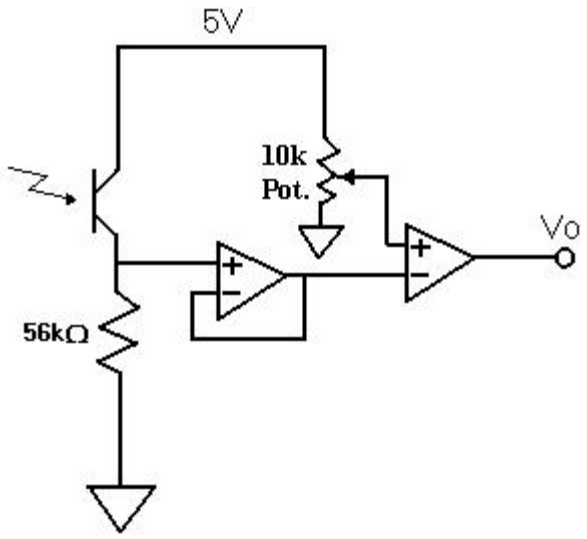


Figure 4. Line Sensor Schematic

Problems in the design were encountered when the robot was sent to the national competition. The ambient light was too much for the sensor, so a blind was constructed and placed around the sensor. The blind also eliminated the light that triggered the sensor. To address this issue, a red LED that lit the ground directly beneath the transistor was placed on the sensor board. We found that the red LED emitter combined with the phototransistor receiver provided us with a reliable, adjustable, and accurate white line detector.

**Fire Sensor**

A main objective of the competition was finding the flame. Sensing the candle's flame can be done with expensive ultraviolet equipment or with cheap infrared equipment. The Hamamatsu's UVTron was the highest quality flame detector that we could find, but the device would have cost 70% of our entire budget if it were purchased. We tried to get a sample, but since two other groups in our class had already ordered samples, they did not send us one. Since we could not get a sample, our next alternative was infrared. We attempted to use a phototransistor coupled with an infrared filter, made from a floppy disk, to detect the candle. Although we got very good response out of this device, it was rather large. We decided to place an order with Electronics Goldmine for several different types of infrared detectors. We ordered PIR infrared detectors and Honeywell's SDP8407 IR detectors. Of all the sensors that we tested, the Honeywell sensors worked the best.

In addition to being a very good sensor, the SDP8407 was also very easy to implement. The basic configuration was the same as our line sensor without the conditioning circuitry (see figure 5).

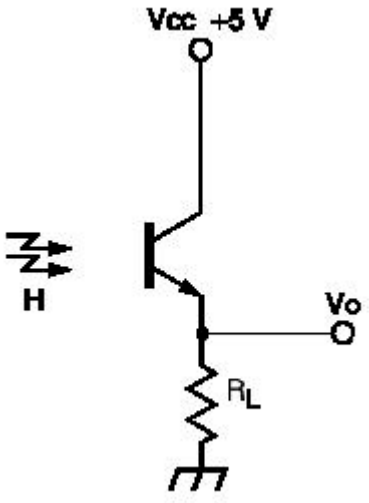


Figure 5. Fire Sensor Schematic

The detector gave us versatility with different resistor values. A resistance of 10kΩ gave us a good range of values for finding the location of the candle. A much larger resistance of 100kΩ gave us the ability of sensing a candle's presence from the doorway. Since these values ranged from 0V to 4V there was no need for amplification. Putting the signals directly to the A/D converter eliminated any requirements of a buffer. The SDP8407 had an observation angle of $135^{o}$. This is ideal for the detector that only had to look into the room, but for actually locating the candle, we needed a smaller aperture. To narrow the field of view, plastic blinds were put in front of the sensor and were adjustable to give a variable field of view. The adjustable blinds coupled with the SDP8407 sensors gave us a very accurate way of finding the flame. Except for their dependence on ambient light these sensors gave us excellent results.

Tone Decoder

The tone decoder was used as a start switch so that Jerry wouldn't move before we wanted him to. The national rules stated that the robot should start when a tone of 3.5kHz was sounded. At first we weren't going to use this optional start on Jerry; but we found out that National sampled a low-powered tone decoder chip (LMC567) that was an ideal solution for our problem. The chip's output is normally high until an input signal that contains a frequency within the programmed bandwidth is received. We decided to use the test circuit (figure 6) given on the data sheet without a switch off on pin 3.

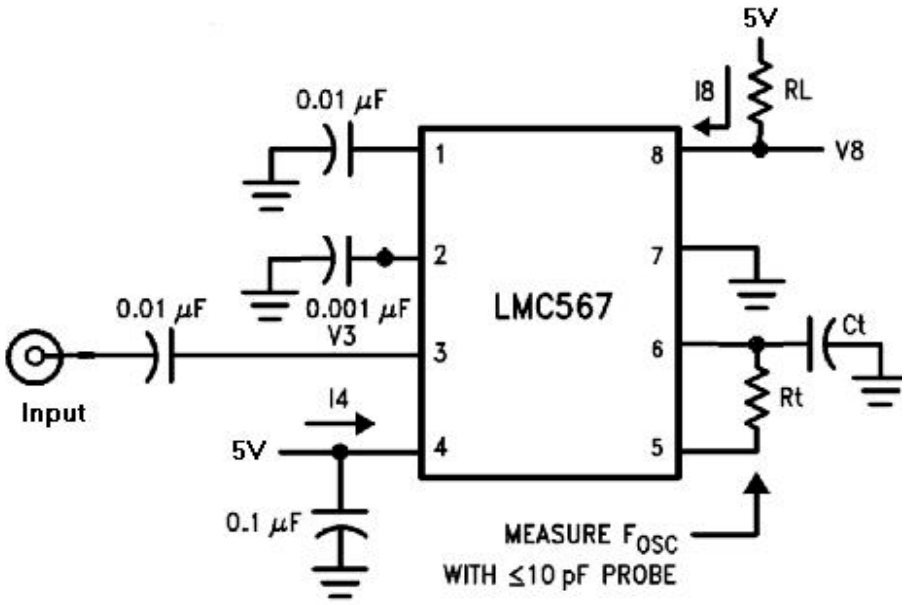


Figure 6. Tone Decoder Wiring Diagram

$F_{input}$= 1/(2.8 x $R_t$ x $C_t$)

$F_{input}$= 3.5kHz

$R_t$ = 9.8k

$C_t$ = .001uf

The bandwidth is determined by the capacitor connected to pin 2, but we just left that as it indicated on the data sheet. All the other capacitors are just filter capacitors so we used the values supplied by the data sheet.

After testing this circuit with a function generator, we interfaced the microphone with the chip. We used a condenser microphone that had a DC offset of 5V and a maximum output of 20m$V_{p-p}$. In order to interface with our chip, we needed to amplify the signal and null the offset. We used the simple inverting op-amp amplifier with a decoupling input capacitor to get rid of the DC offset (Fig. 7). However, since we used a single supply op-amp getting rid of the offset centers the signal at zero and only gives us the positive half of the signal. So to counteract this, we added another offset by giving

$V_+$ an input of 2.5V. This offset was easily acquired by using a voltage divider. We now had an output signal that ranged from 0-5V that was centered at 2.5V. The output of the op-amp was then connected to the input of the tone decoder chip and our starter was complete. The output of the tone decoder was then run through one of the adjustable op-amp comparator with the comparing voltage set at 2.5V. This basically just inverted the signal, and was just done because it was a convenient power supply. The signal was then run into PIA_B3.

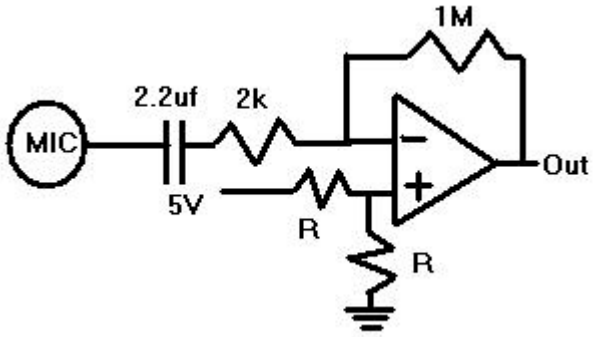


Figure 7. Input amplification for Tone Decoder

**Fire Suppression**

The choice for a fire suppression system was an easy one. We had considered using 3 types of fire suppression: inert gas, water, and a fan. The problem of using inert gas, such as $CO_2$, is controlling it. We had found some electronic valve regulators that could be used with 7-12oz $CO_2$ tanks that are used with paintball guns; however, since we had never used these before and questioned their reliability, we decided not to use them. There was also the question of finding enough space for the tank and getting it refilled. So, while $CO_2$ is a very efficient way of putting out the fire, it was not a practical decision for our robot. The same can be said of using water to put out the fire.

While finding and using a water pump (like a windshield water pump) is very simple, spraying water around our bare components didn't seem like a very good idea. The range of a water pump can also be questioned; if we had a narrow stream of water, then accuracy would become an issue. If the stream were wider, then we would have to get closer to the candle, possibly closer than the 12 inch white line surrounding the circle. In the end, we decided to go with a simple fan. There were many benefits of using a fan such as: the fan motor takes up very little space and can be very powerful, our motor goes up to 24,000rpm; also, implementing the control circuitry was extremely easy. The only drawback to the fan is that the propellers can block our fire sensors. The simple way to fix this problem was to choose a small fan blade, and to sit it high enough so it would not block the sensors.

In order to drive our fan, we decided to use a simple MOSFET switch. We connected one end of the motor to 12 volts and the other end to the drain of an IRF711 N-channel MOSFET. We also put in a 1N4007 rectifier diode in parallel with the motor prevent problems caused by flyback voltage. The gate of the MOSFET was connected to the HC11 through a buffer, which gives a signal of 0 or 5 volts. The source of the MOSFET was connected to ground. In this configuration, whenever a voltage of 3V or greater is put to the gate, the drain and source are connected turning on the motor. However, we found out that at a drain voltage of 12V the IRF711 only sources about 600mA; our fan motor runs at about 1.9A nominal. In order to source more current to the motor, we put three MOSFETs in parallel. This should have provided us with enough current. However, we felt that the fan was moving kind of sluggishly, and so put on three additional MOSFETs to the board, giving us a grand total of six MOSFETs. This

configuration worked decently but it would occasionally stop working. We finally decided to use a relay. The coil on the relay was connected to the output of the MOSFET switch and the connected ground to the motor. The schematic is shown in figure 8.

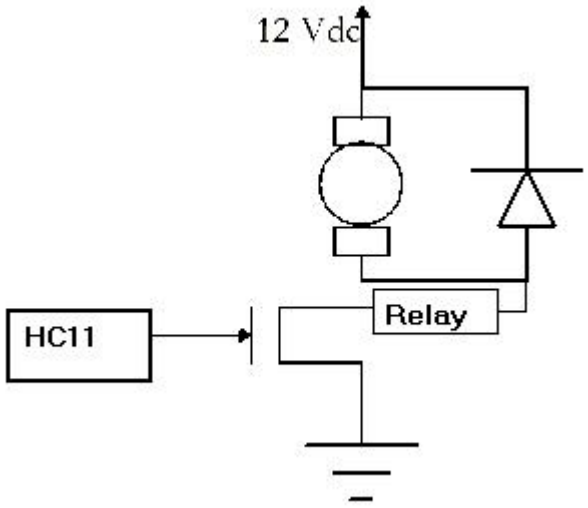


Figure 8. Motor Driving Circuit

**Motor Feedback**

In order to have closed loop control over our motors, we had to have some motor feedback. The Pittman motors that were supplied to us have built in motor encoders. We considered two methods of converting the encoder data into a usable signal. These options were pulse accumulators or frequency to voltage (f-v) converters. A pulse accumulator counts the number of pulses generated by the motor encoders. In order to get velocity information, these distance values would have to be divided by a time interval in code. The other option was a f-v converter. This was the simplest option because f-v converter puts out a voltage that the HC11 can convert into a value representing speed without any extra calculations. We chose the f-v converters because

of their simple interfacing. National's LM2917 was ideal for our setup; the chip was easy to implement and the output voltage range was tunable to exactly what we needed. Figure 9 shows the basic wiring diagram. We chose the component values according to the following equations.

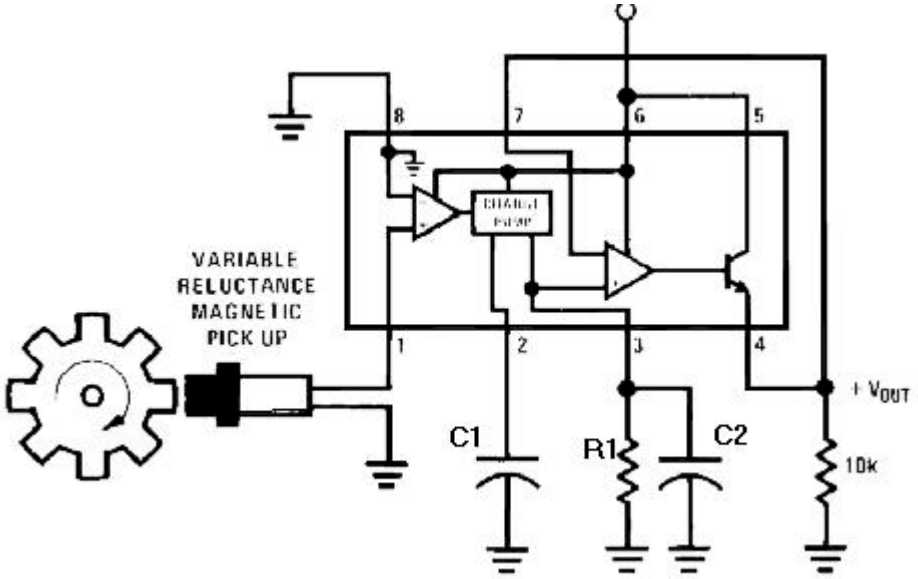


Figure 9. Wiring diagram for Frequency to Voltage Converter

$V_o = V_{cc} \times F_{in} \times C_1 \times R_1$

$V_{ripple,\ p\text{-}p} = (V_{cc})/2 \times C_1/C_2 \times (1 - (V_{cc} \times F_{in} \times C_1)/2)$

$F_{max} = I_2/(C_1 \times V_{cc})$

After calculating the values, we tested them on a proto-board. It turned out that the values we used were not as accurate as we wanted so, we fiddled around with the values for a while. We found out that $C_2$ determines the ripple voltage, $R_1$ and $C_1$ determine the

frequency to voltage range. After testing for a while, we chose $C_1$ to be 500pf, $C_2$ was .33uf, and $R_1$ was chosen to be 27k.

Our max frequency generated by the motor encoders was approximately 33 kHz. The $V_{ref}$ was 12 volts and the desired output voltage ranged from 0 to 5 volts. After tuning the ripple voltage we were able to get some very desirable outputs. The frequency to voltage relationship was linear from 100 Hz to 34kHz where the voltage railed at 4.95V. The ripple on the output voltage was approximately 200m$V_{p-p}$. The output from the f-v converter was then buffered with an op-amp in a voltage follower configuration. This signal was then converted to an 8-bit number via the A/D converter on the HC11.

**Pulse Width Modulation**

Perhaps the most important aspect of our robot is the motor control. The motor control consists of PWM (pulse width modulation), h-bridge motor controllers, some type of motor feedback, and the software necessary to interface these three components.

Initially, we decided to implement our PWM in Altera. After writing the software and simulating it using the Altera compiler, we were ready to order our device and print up a circuit board. Since our design used about 45 logic cells, we decided to use the EPM7064LC44-15T, which is a 64 logic cell variation of the Altera device that we have used in the past.

Since we were not going to need very many digital inputs or outputs on the HC11, we decided to interface the Altera PWM device to the expansion ports on the HC11. Expansion port A was going to be used to provide eight bits of resolution to the PWM controller. By giving our PWM controller eight bits of resolution, we could effectively

vary our duty cycle by as little as .39%. In addition to expansion port A, we also used three bits off of expansion port B. We used the least significant bit of port B as a motor select bit, which would tell the Altera device which motor to output the PWM signal to. We also used a bit of port B as a direction bit, which would tell the motor which way to spin. Lastly, we had a latch enable bit. This latch enable bit made it so that the Altera device would never change the output PWM signal unless we specifically told it to. By using a counter, a comparator, and several internal nodes, the Altera device would simply count up until the counter value matched the value written to the resolution register. While it was counting, one of the PWM outputs was driven high. After the count value reached the value written to the resolution register, the PWM output was brought low and stayed low for the remainder of the count. Our PWM controller had two completely independent sets of outputs. Each output set consisted of a PWM output and a direction bit.

Once we got our device, we wired it all up on a proto-board. At first, we used DIP switches and a de-bounced push button to simulate the data that would come off of the HC11. Our controller seemed to work fine at first, but we saw some very flaky behavior that happened when we tried to go over 50% duty cycle. After a bit of trouble shooting, we found out that our problem was a floating reset. After we grounded the reset, the device worked flawlessly.

We ran into major problems when we tried to drive the motors with our Altera controller. We could drive each motor independently, but when we tried to run them at the same time, nothing worked. As soon as both motors were attached, our direction bit on one of them would start pulsing. We tried everything from Schmitt triggers to sawing

the board in half, and nothing worked. After many hours of work, we discovered that internal coupling of the Altera device was the most likely culprit. Since we had to go to nationals in less than 2 weeks, we needed a solution fast. We scrapped the entire Altera idea, and decided to go with an HC12.

Since we were short on time, we could not memory expand the HC12 and use it exclusively. Instead, we decided to make the HC12 work exactly as our Altera device should have. We had the same eight bits of resolution, motor select, and direction bits. The latch enable on the HC11 would, when pulsed, generate an interrupt on the HC12. In the interrupt service routine, the HC12 would check the data on its port A and port B, modify the outputted PWM signals, and modify the outputted direction bit as well. Figure 10 shows how the HC11 and HC12 were interfaced. This design worked beautifully for us. It only took a couple of hours, and we had our robot rolling. We put the program into the HC12 flash EEPROM and set the jumpers so that we would never have to touch the HC12 again. The HC12 did everything we asked of it, and never gave us a single problem.

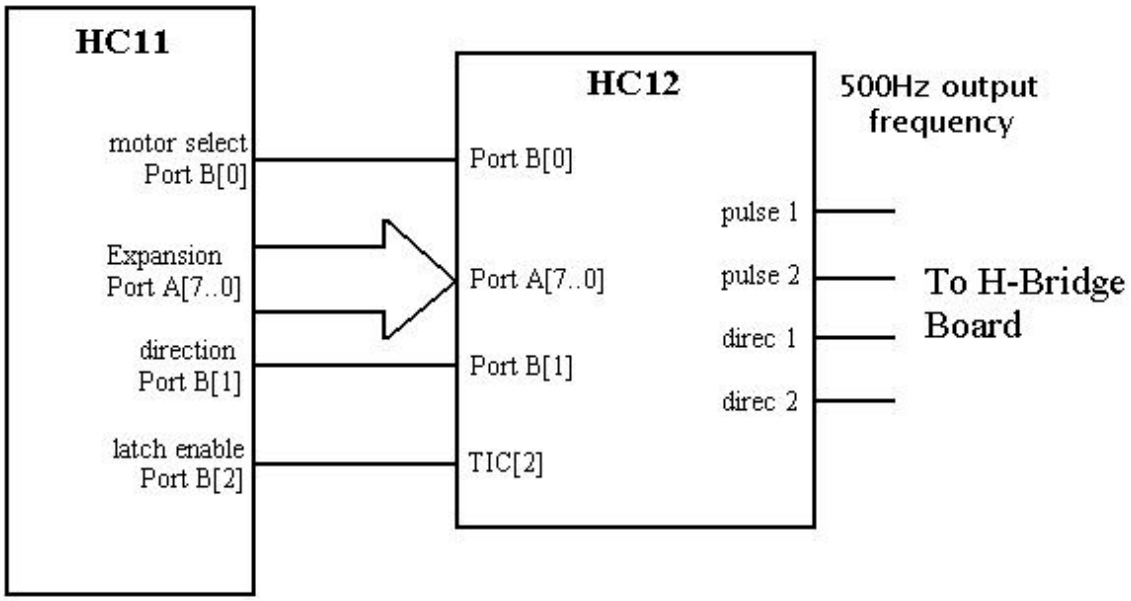


Figure 10. Interface between the HC11 and HC12

The whole purpose of the PWM described above is to drive an h-bridge circuit, which will in turn drive the motors. We decided to use the LMD18200-T h-bridges that were supplied to us at the beginning of the semester. We heard horror stories about these h-bridges, so we all ordered tons of samples from National Semiconductor. It turned out that we did not damage any h-bridges, so we have tons of extras now. These devices were very easy and straightforward to use. The schematic that we used is shown below (figure 11). On our design, we decided not to use the thermal flag output, brake bit, or current sense feature. Even though we ran our motors at 500Hz, we included the bootstrap capacitors just to be safe.

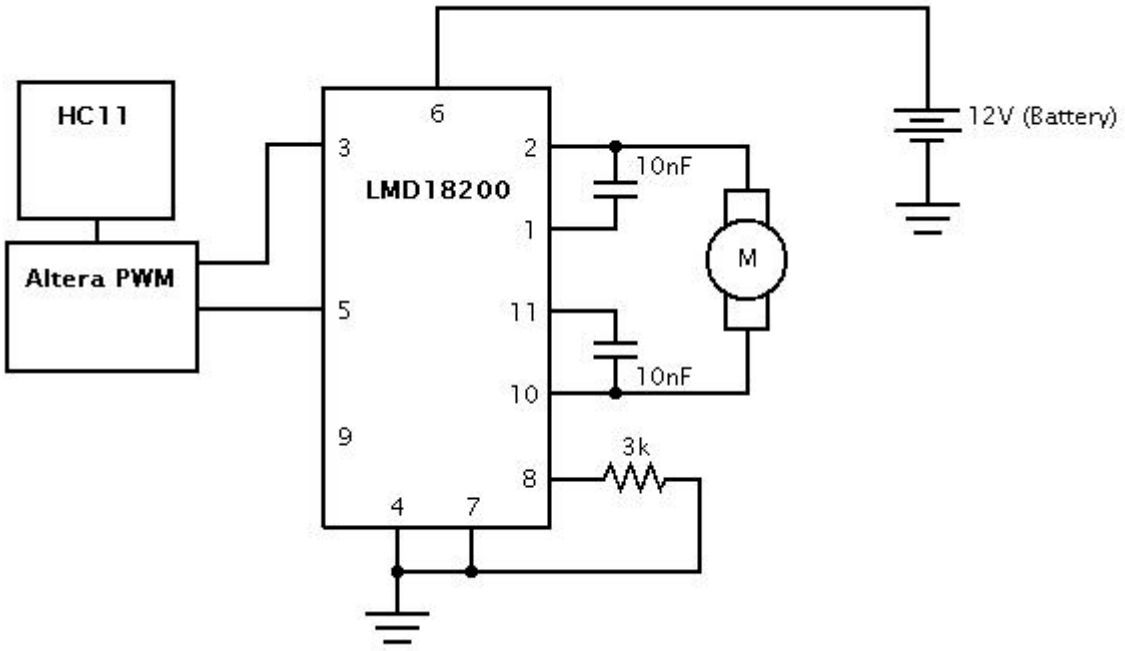


Figure 11. Basic h-bridge schematic

**Design and Construction of Circuit Boards**

All of the circuit boards were designed by hand and then converted into a MicroSim PC board drawing that could be printed onto the iron-on-masking for etching circuit boards. There was some difficulty generating a file for each board since the trial version of MicroSim that is in the labs limits the user to 50 parts per design. This would not have been a problem except that we had to make the footprints for most of our chips by hand using vias in the program. To get around this, we just made drawings that fit together and then ironed them onto the same PC board. The advantage of using MicroSim instead of Protel was that it was much easier to learn in a short amount of time. By hand drawing the boards and then converting these drawings into a MicroSim file, we were able to make some very complicated boards in a very short amount of time.

Signal Conditioning Board

Once we had designed all of the major sensor modules we were able to design and build the signal conditioning board. This board converts the signals coming from all of the sensors into either a 0-5V analog voltage so that we can get the best resolution possible from the A/D converters or a digital signal that goes straight to the digital input ports on the HC11. Figure 13 shows how the board modifies each of the signals being received from the sensor modules.

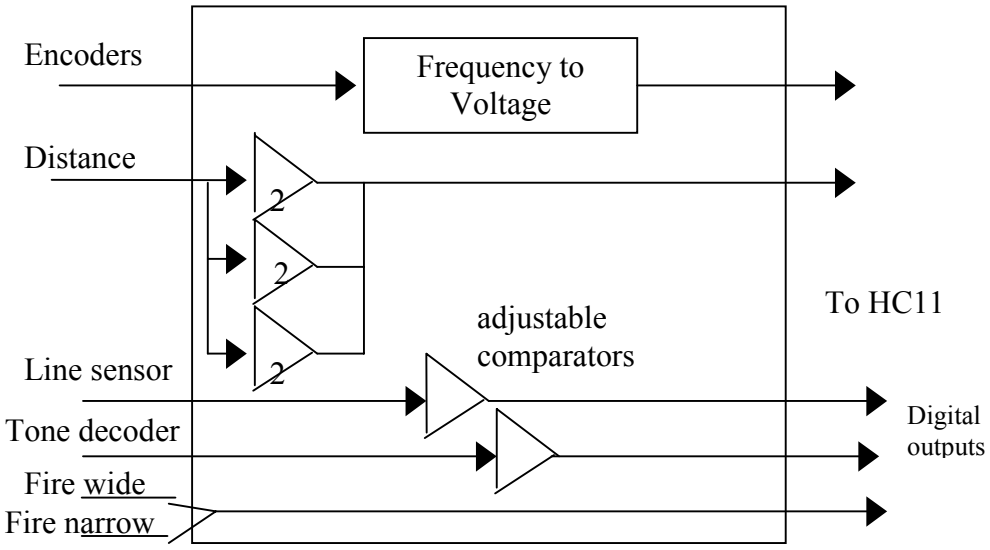


Figure 12. Block Diagram of the Signal Conditioning Board

The encoder lines are converted into an analog voltage between 0 and 5V using the frequency to voltage converters. The distance sensors output a voltage between 0 and 2.5V. This signal is amplified by a factor of 2 to get the 0 to 5V range that the A/D converters will be setup for. All of the op-amps used on this board were the Maxim MAX494 quad single supply op-amps. The line sensor is run through an adjustable comparator so that the line detection threshold can be adjusted based on the ambient

conditions. The tone decoder was run through the signal conditioning board because this was the most convenient way to supply power to it. The comparator voltage was set to 2.5V and thus acts basically as an inverter. The wide and narrow flame detectors didn't need any amplification and were just connected through the signal conditioning board to supply power to them. All of the connections to the signal conditioning board were made using the Molex friction lock header connectors and ribbon cables. The MicroSim drawing of the signal conditioning board is figure 2 of the appendix.

Pulse Width Modulation Board

This board was designed to take the motor control signals from the HC11 and convert them into two pulse width modulated signals and two direction signals to be used by the H-bridges which were also on this board. As we have already said the H-bridge portion of the board was eventually cut off and used by itself because of the eventual demise of the Altera PWM implementation. The entire board drawing can be found in figure 3 of the appendix. The motor driver wires were soldered onto the H-bridge part of the board using headers soldered to wires and then wrapped in shrink tubing. The connector that eventually connected the HC12 to this board was a friction lock header connector from Molex. The connection to the 12V supply was one of the Amp connectors supplied to us by the EE department.

**Closed Loop Control Design Issues**

The model that we used as the basis for our closed loop control design is outlined in the following block diagram.

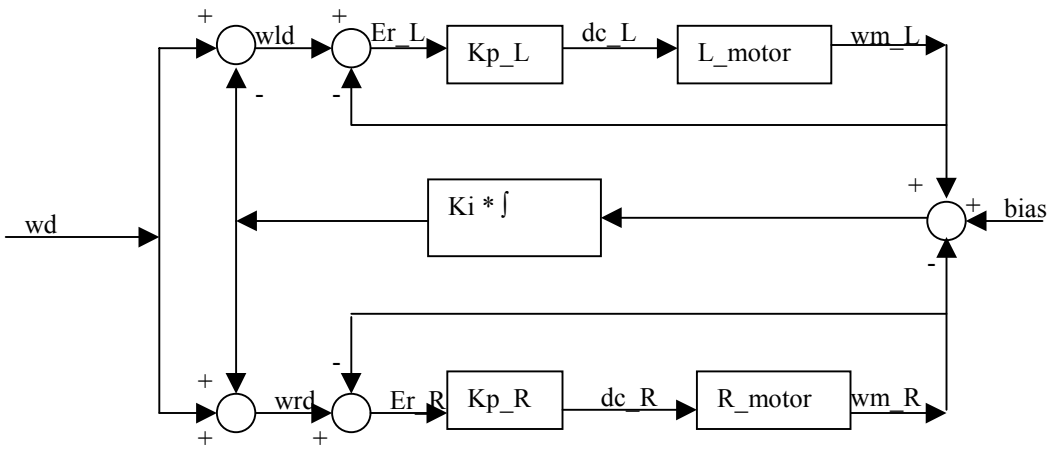


Figure 13. Block diagram used for closed loop control

The equations derived from this diagram show how closed loop control can be implemented in code.

$$bias = Kd(d_d - d_a)$$

$$wld = wd - Ki* \int[(wm\_L - wm\_R) - bias]dt$$

$$wrd = w_d + Ki* \int[(wm\_L - wm\_R) - bias]dt$$

$$dc\_l = Kp*(wld - (wld - wm\_L))$$

$$dc\_r = Kp*(wrd-(wrd - wm\_R))$$

The desired speed for each motor (left or right) is adjusted based on a set desired speed and the integral of the bias term and the difference between motor speeds. The bias term is based on the difference between the actual distance from a wall and the desired distance. The difference term tries to keep the two motors going the same speed. The closed loop part of the control is based on a proportional controller. The duty cycle for each side is proportional to the difference between the actual speed and the desired speed

plus a term that is proportional to the desired speed. This type of control allows us to wall follow, go in straight lines, and turn in place.

Another major issue in motor control was the angling of the side distance sensors. We started with the sensors perpendicular to the front sensor. To get the closed loop control to work properly the distance from the wall needs to get shorter as soon as the robot begins to turn towards the wall. With the sensors perpendicular to the wall the distance gets larger even though you are turning towards the wall. To fix this problem the sensors were angled at 30 degrees. A diagram showing the distance sensor configuration is in Figure 1 of the appendix.

**Software Design**

Throughout the design of our robot, we probably gave the most amount of thought to the software. We were always trying to come up with faster, more efficient algorithms for our robot. The software was probably the most difficult aspect of this project. We had a meeting near the beginning of the semester where we all agreed that the easiest way to navigate the maze would be to left wall follow the first three rooms, and then worry about getting into the island room.

The way that we did our code was somewhat haphazard, but it seemed to work for us. What we would do was get one part of our robot working well, then would make a new version of code. Each new version built upon the old one, adding features as we went along. At the end of the semester, we ended up with over forty different versions of code.

The very first thing that we got working was our left wall follow function. Since it was closed loop, we didn't have to worry about navigating around corners, because the closed loop algorithm did it for us. We also made it so that if our robot saw a wall in front, it would spin until the front sensor dropped below a threshold value. This came in very handy since it turned the same amount, regardless of battery voltage. The combination of these two functions made our robot very consistent in entering the first three rooms.

Our robot would start on the home circle and wait for a tone. After hearing the tone, it would dead reckon forward for a small amount just so that it would not get white line interrupts from the home circle. The robot would then left wall follow until it saw a white line. Each time it saw a white line, the robot would stop and do a quick scan to determine whether or not the candle was in the room. Our fire sensors were awesome, which made our robot capable of scanning even the largest room in a matter of milliseconds. If the candle was not in the room, then our robot would spin to the right enough to make it clear the doorway. It would then begin left wall following again. If the candle was not in any of the first three rooms, then we knew that it had to be in the fourth. After turning enough to clear the doorway of the third room, our robot would left wall follow just enough to get it pretty straight. We then had it dead reckon straight forward, using the small_increment_forward() function, until it saw a front wall. Upon seeing the front wall, our robot would begin right wall following, which enabled the robot to get into the fourth room.

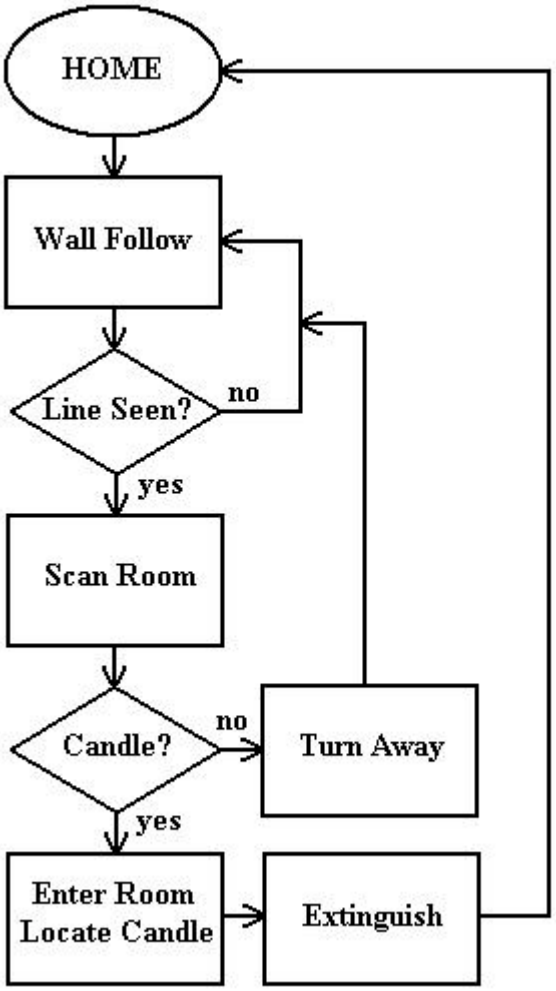


Figure 14. Simplified flowchart for navigation algorithm

When the robot saw the candle, it would do several things. First, it would go into the room a certain amount, or until it saw the line around the candle, whichever came first. If it had not yet seen the line, it would scan the room to find the peak value coming off of the narrow fire sensor. After finding the peak, it would scan again to return to the position where it saw the peak. It would then go straight forward until it saw a white line. If it got too close to the left or right walls, then it would start to wall follow until it saw a white line. When it saw the line around the candle, it would go straight about an inch,

and do the scan routine again. Once it found the peak this time, it would turn on the fan. It would leave the fan on about a second and then poll the fire sensor to see if the candle was out. It would keep turning the fan on until the candle was out. When the candle finally went out, our robot would return home.

We had our robot set up so that it would count the number of rooms it had seen. Depending on the room number where it put the candle out, it would return home in a different way. If it extinguished the candle in the first room, then the robot would turn left, go straight to clear the candle white line, and then go straight until it picked up a wall on any of its sensors. It would then spin in place until the right sensor picked up the wall. When the right sensor saw the wall, it started to right wall follow and did not stop until it saw two white lines, which made it stop on the home circle. For rooms three and four, it did the same as in room one, but it spun to the right and left wall followed home. When the candle

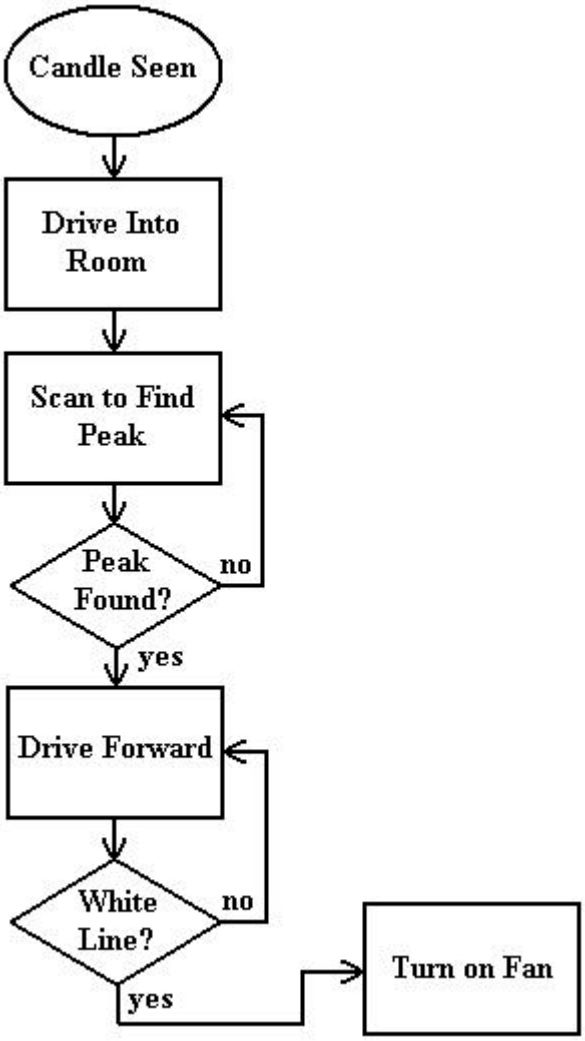


Figure 15. Simplified flowchart for zeroing in on candle location

**Conclusion**

By basing our design on simplicity we were able to many of the problems that slowed down other groups. Also, by doing a complete design before building the robot, we were able to design and build each component with a clear idea of how it would fit into the system as a whole. We feel that our focus on modularity and simplicity in our design was what allowed us to complete the project in the amount of time given.

In the future we plan to make some modifications to our design that will enable us to be more effective at next year's competition. The first modification that we would like to make is to use the HC12 to do all of the control as well as pulse width modulation. We plan to use a central power distribution board that will provide 5V, 12V and ground to each board through one cable. The modification that would probably benefit us the most would be to get a UV based fire sensor like the Hamamatsu to do our initial fire check of each room. This would allow us to quickly check each room in the maze without having any dependency on the ambient lighting conditions. We also have some improvements for future line and fire sensors that we are planning to implement for next year. Finally we would like to make some small improvements on the chassis design to make it more stable and robust.

We feel that this project has given us confidence in our capabilities as engineers that we did not get from our other classes at New Mexico Tech. This was the first time that we have been able to see a very complicated project through to the end. We enjoyed working on this project and feel that it was an invaluable part of our education here.

## Appendix

### Power Budget

**12V Lead Acid (Camcorder);**

Motors(2): ~ 550mA each

Frequency to
Voltage converters(2):
~10mA each

Total: 1.12A
Battery was rated at 2.3 Amp hours

**12V Lead Acid**

Fan Motor: 1.2A

Relay: 100mA

Total: 1.3A
Battery was rated at 1.5 Amp hours

**8.2V NiCd**

| | |
|---|---|
| Distance Sensors(3): | 10mA each |
| Fire/ Line Sensors(3): | 0.25mA each |
| HC11: | 150mA |
| HC12: | 150mA |
| Amplifiers(8): | ~6mA each |
| Total: | 380mA |

Battery was rated at 1.2Amp Hours

**Chassis**

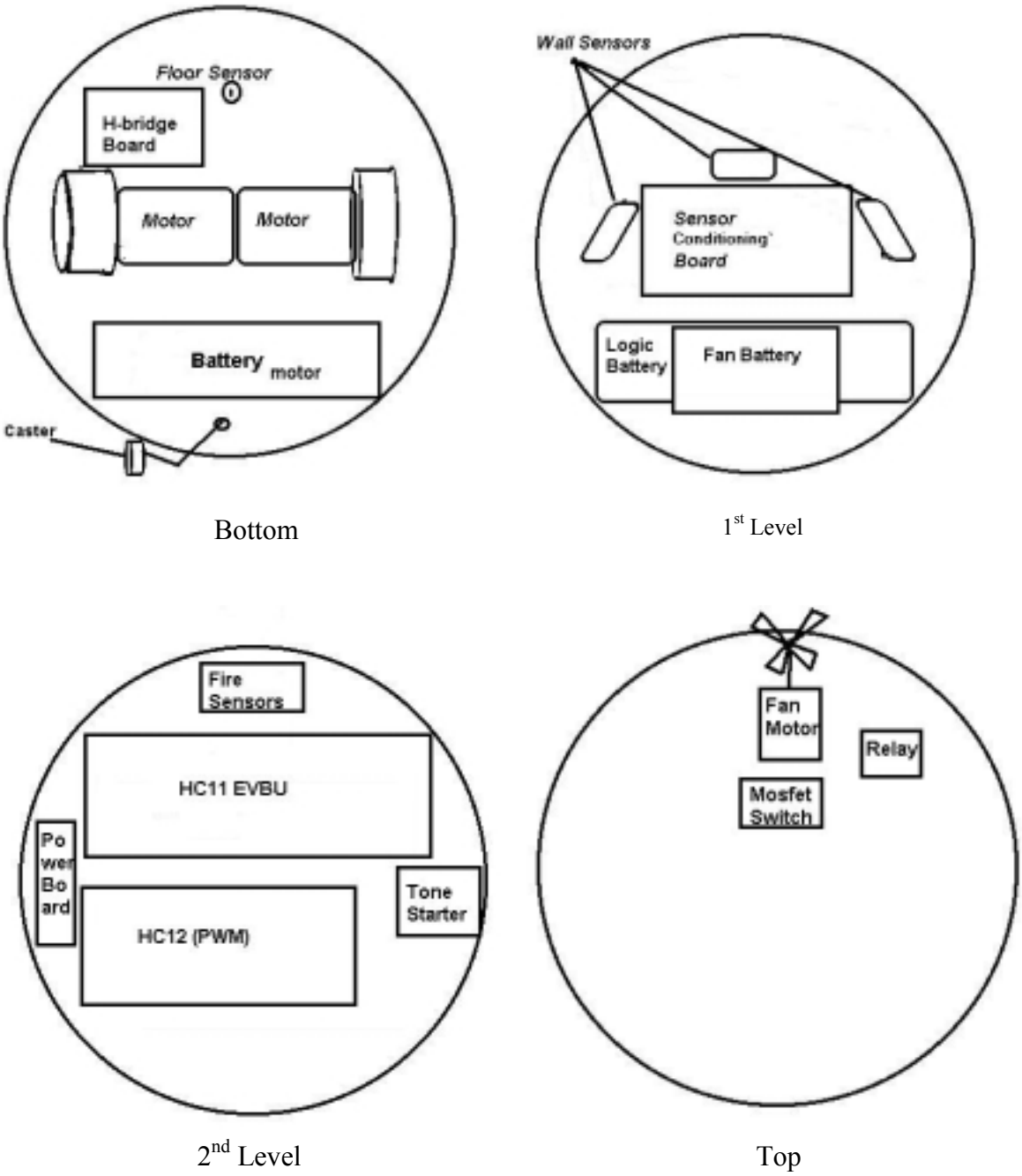


Figure 1. Chassis layout diagrams

**Circuit Boards**

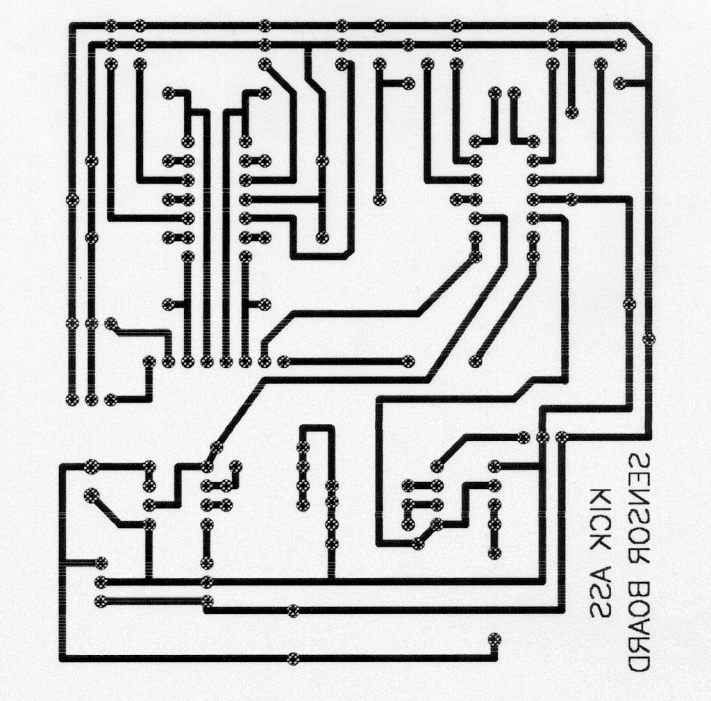


Figure 2. Signal conditioning board

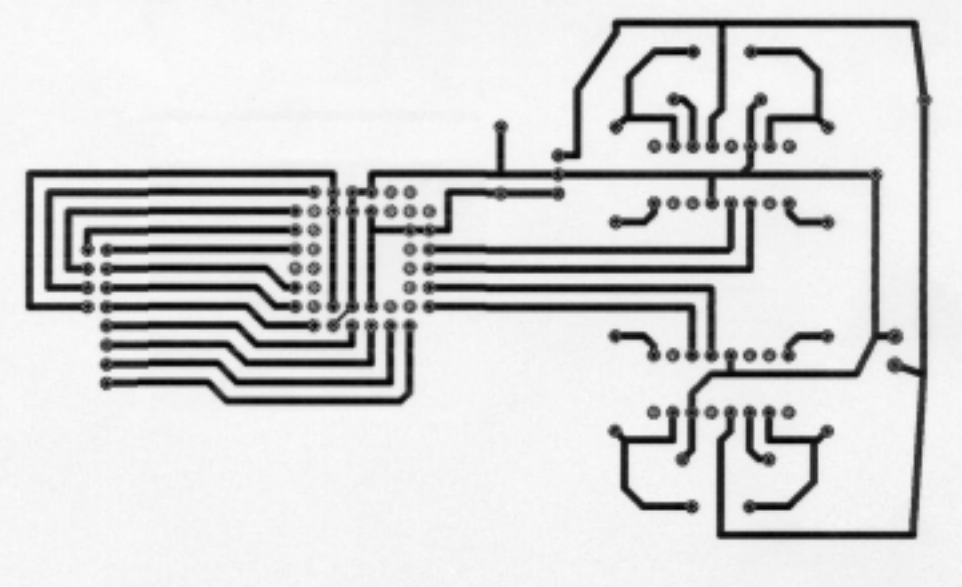

Figure 3. Pulse width modulation board

**Software**

**Final Budget**

| | |
|---|---|
| HC11, HC12 | Free |
| Distance Sensors: | $23.56 |
| Chassis Components: | $17.00 |
| Connectors: | $25.17 |
| Frequency to Voltage Converters: | $11.90 |
| Battery: | $10.00 |
| Altera Chip: | $12.60 |
| Total: | $100.23 |

## Software

```
#include <stdio.h>
#include <hc11.h>
#include "pia.h"

/* Below are the constants that our robot will use to gauge the distance
 * that it should stay away from the walls.  The FIRE constant is a value
 * which the robot uses to determine whether or not the candle is in a
 * room.  RIGHT and LEFT are just defined so the turn functions are more
 * intuitive as to what they do.  The MEM_LENGTH variable is the amount
 * of integral errors we are taking into account.
 */

#define FRONT_THRESHOLD 55
#define WALL_THRESHOLD 62
#define FIRE_THRESHOLD 70
#define RIGHT 0
#define LEFT 1
#define TRUE 1
#define FALSE 0
#define MEM_LENGTH 10


#define K 1
#define Kd 1
#define Ki 1
#define Kp 1
#define Kw .6


/* The following variables are all used to poll the A/D converter */
unsigned char front_sensor;                     /* GP2D12 sensors */
unsigned char left_sensor, right_sensor;
unsigned char left_motor, right_motor;          /* Motor encoders */
unsigned char flame_wide;                       /* Flame sensors */
unsigned char flame_narrow;


/* The following variables are flags */
unsigned char line;                             /* How many lines have been seen?
*/
unsigned char post_candle_line = 0;             /* Lines seen after candle is put out
*/
unsigned char candle_put_out;                   /* Set when candle extinguished */
unsigned char room = 0;                         /* Are we in a room? */
unsigned char flame_seen;                       /* Have we seen a flame? */
unsigned char wall_follow;                      /* 0 = RWF, 1 = LWF */
unsigned char flame_flag = 0;                   /* Have we seen a flame? */
unsigned char number_rooms = 0;                 /* The number of rooms we have entered */
unsigned char lined_up_for_island = 0;          /* This helps us get into room 4 */
unsigned char home = 0;                         /* Stops the robot once it gets home */
unsigned char dright, dleft;                    /* Direction bits */
unsigned char put_the_candle_out = 0;           /* Tells robot to put out candle */
```

```
/* Here are the closed loop motor control variables */
char wdiff, eint, wld, wrd;
char dc_r, dc_l, wd, wld_prev, wrd_prev;
int dist_diffl, int_error;
int_error = 0;

unsigned int counter, degrees;                  /* Used in turn_stepper() */
unsigned int initial_delay = 0;                 /* Used so that we don't get
multiple
                                                * interrupts on white lines.
                                                */
unsigned char val1 = 0;
unsigned char number_scans = 0;                 /* Used to determine scan angle when
                                                * we are looking for candle.  The initial
                                                * scan is wider than the secondary scan
                                                */
unsigned char starter = 0;                      /* Enables interrupts once the robot
                                                * leaves the home circle.
                                                */



/* This function will setup the A/D converter and poll every sensor.  It
 * will first poll the second set of A/D pins, and then it will poll the
 * first set of A/D pins.  These values will all be stored in variables
 */
void
poll_sensors(void){
        /* ADCTL = X X 0 1 0 1 0 0 = 0x14
         *       | | | | | | | |
         *       | | | | | | When MULT = 1 then CB/CA don't matter
         *       | | | | | CC determines which 4 to convert
         *       | | | | CD always 0 for our purposes
         *       | | | Convert 4 channels in selected group
         *       | | Do 4 conversions and stop
         *       | Not implemented
         *       Conversions complete flag
         */
        ADCTL = 0x14;
        while(!(ADCTL & 0x80)){}                /* Wait until conversion is complete */
        left_sensor = ADR4;                     /* Take sensor data */
        front_sensor = ADR3;
        right_sensor = ADR2;
        flame_narrow = ADR1;
        ADCTL = 0x10;                   /* Convert other 4 channels */
        while(!(ADCTL & 0x80)){}                /* Wait until conversion is complete */
        left_motor = ADR3;              /* Take sensor data */
        right_motor = ADR4;
        flame_wide = ADR2;
        left_motor += 25;               /* Our frequency -> voltage converters were
                                         * slightly mismatched, so we have to add a
                                         * constant to one of the variables
                                         */
```

```
	left_sensor = left_sensor >> 1;		/* Shift these to get rid
						 * of sign bit
						 */
	right_sensor = right_sensor >> 1;
	front_sensor = front_sensor >> 1;
}


/* This function will take the duty cycles that have been calculated by
 * another function and it will pass those values to the motors.
 */
void
go_motor(void){
	unsigned char temp = 0;
	dright = dc_r & 0x80;			/* AND the duty cycles with a mask to
						 * determine whether or not the direction
						 * bit is set.
						 */
	dleft = dc_l & 0x80;

	dc_r = dc_r << 1;			/* Shift the duty cycles up one to get a
						 * higher resolution.  We had to shift them
						 * down before to do the signed calculations,
						 * but now we mask off the direction bit
						 * and shift them up.  This way, we only lose
						 * the least significant bit of resolution
						 * rather than the most significant.
						 */
	dc_l = dc_l << 1;

	if(dright == 0x80){				/* If right motor, direction bit is set */
		PIA_A = dc_l;
		PIA_B |= 0x03;			/* Set right motor and direction bit */
		while(temp <= 4){			/* Small delay to satisfy hold times */
			temp++;
		}
		temp = 0;
		PIA_B |= 0x04;			/* Pulse the latch enable signal */
		PIA_B &= ~0x04;
	}
	if(!(dright == 0x80)){			/* If right motor, direction bit is not set */
		PIA_A = dc_l;
		PIA_B &= 0xFD;			/* These must be ANDed and then ORed so that
						 * the siren will still work.
						 */
		PIA_B |= 0x01;
		while(temp <= 4){
			temp++;
		}
		temp = 0;
		PIA_B |= 0x04;
		PIA_B &= ~0x04;
	}
	if(dleft == 0x80){				/* If left motor, direction bit */
```

```
                PIA_A = dc_r;
                PIA_B &= 0xFE;
                PIA_B |= 0x02;
                while(temp <= 4){
                        temp++;
                }
                temp = 0;
                PIA_B |= 0x04;
                PIA_B &= ~0x04;
        }
        if(!(dleft == 0x80)){                   /* If left motor, direction bit is not set */
                PIA_A = dc_r;
                PIA_B &= 0xFC;
                while(temp <= 4){
                        temp++;
                }
                temp = 0;
                PIA_B |= 0x04;
                PIA_B &= ~0x04;
        }
}



/* This turn_stepper function will be used to find the precise position of
 * the candle.  This function must be passed a variable, either RIGHT or
 * LEFT based on which direction you want to turn.  It will turn in small
 * increments, the angle of which is based on the value to which counter
 * is allowed to run.  This function is also used to turn away from a room
 * once it has been determined that the candle is not in there.
 */
void
turn_stepper(unsigned char l_or_r)
{
        unsigned char fload_left;                       /* Closed loop variables */
        unsigned char fload_right;
        wd = 0x25;                                      /* Desired speed of turn */
        counter = 0;
        while(counter <= (9*degrees)){
                left_motor = left_motor >> 1;           /* Shift the left and right
                                                         * motor encoder variables
                                                         * so that the sign bit goes
                                                         * away.
                                                         */
                right_motor = right_motor >> 1;

                wdiff = left_motor - right_motor;       /* All our closed loop math */
                wdiff = wdiff >> 2;

                wrd = wd + wdiff;
                wld = wd - wdiff;

                fload_right = ((wrd - right_motor) >> 1);
                fload_left = ((wld - left_motor) >> 1);

                if(!flame_seen){
```

```
                dc_r = wrd + fload_right + 10;
                dc_l = wld + fload_left;
        }else {
                dc_r = wrd + fload_right;
                dc_l = wld + fload_left - 17;
        }

        if(l_or_r == RIGHT)             /* Based on the direction the robot is to turn,
                                         * the duty cycles need to be in opposite
                                         * directions.
                                         */
                dc_l |= 0x80;
        else
                dc_r |= 0x80;

        wld_prev = wld;
        wrd_prev = wrd;
        go_motor();

        counter++;
    }
    counter = 0;
    room = FALSE;
    dc_r = 0;
    dc_l = 0;
    go_motor();
}


/* This is one of our most dynamic and cool functions.  When the robot sees
 * a wall in front of it, it needs to turn either right or left.  You pass
 * this function a variable that will tell it which way to turn.  This
 * function is cool because it will turn in the specified direction until
 * the front sensor no longer picks up a wall.  The amount it turns is
 * roughly governed by the FRONT_THRESHOLD variable.  We were very proud
 * of this function because it turns the same amount, regardless of battery
 * voltage or other constantly changing hardware variables.
 */
void
turn90(unsigned char which_way){
    unsigned char fload_left;
    unsigned char fload_right;
    wd = 0x35;                          /* Desired speed of turn */

    /* The loop below will make it turn until it looses the front wall.
     * This function implements closed loop control, as well.
     */
    while(front_sensor >= FRONT_THRESHOLD){

        left_motor = left_motor >> 1;
        right_motor = right_motor >> 1;
        wdiff = left_motor - right_motor;
        wdiff = wdiff >> 2;

        wrd = wd + wdiff;
```

```
                wld = wd - wdiff;

                fload_right = ((wrd - right_motor) >> 1);
                fload_left = ((wld - left_motor) >> 1);

                dc_r = wrd + fload_right;
                dc_l = wld + fload_left;

                if(which_way == RIGHT)
                        dc_l |= 0x80;
                else
                        dc_r |= 0x80;

                wld_prev = 0;
                wrd_prev = 0;
                go_motor();
                poll_sensors();
        }

}



/* This function is similar in construction to our left_wall_follow
 * function.  It simply uses closed loop control to make the robot go
 * straight for however long you want.
 */
void
small_increment_forward(void){
        unsigned char fload_left;
        unsigned char fload_right;
        poll_sensors();                         /* Keep variables up to date for
                                                 * proper closed loop control!!
                                                 */
        wd = 0x35;                              /* Desired speed of movement */

        left_motor = left_motor >> 1;
        right_motor = right_motor >> 1;
        wdiff = left_motor - right_motor;
        wdiff = wdiff >> 2;

        wrd = wd + wdiff;
        wld = wd - wdiff;

        fload_right = ((wrd - right_motor) >> 1);
        fload_left = ((wld - left_motor) >> 1);

        dc_r = wrd + fload_right;
        dc_l = wld + fload_left;
        wld_prev = wld;
        wrd_prev = wrd;
        go_motor();
}
```

```
/* Despite the fact that this function is called left_wall_follow, it
 * actually does both left and right wall follow.  There is a global
 * variable called wall_follow.  First, you set this variable to either
 * LEFT or RIGHT, depending on the method of wall following you want.  Then
 * you simply call this function, it does the rest.
 */

void
left_wall_follow(void){
        unsigned char fload_left;
        unsigned char fload_right;
        wd = 0x2A;                          /* Desired speed of wall follow */

        /* The if block below simply determines which wall following is going to
         * be implemented, and whether or not a front wall is present.  It will
         * vary the method of calculating dist_diffl based on the type of wall
         * following.
         */

        if(wall_follow){                                        /* If wall_follow == LEFT */
                if(front_sensor >= FRONT_THRESHOLD){
                        turn90(RIGHT);                  /* Turn if too close to front wall */
                }else {
                        dist_diffl = WALL_THRESHOLD - left_sensor;
                }
        }else {                                                 /* If wall_follow == RIGHT */
                if(front_sensor >= FRONT_THRESHOLD){
                        turn90(LEFT);
                }else {
                        dist_diffl = right_sensor - WALL_THRESHOLD - 5;
                        /* When battery voltage is > 12.7, then subtract 5 from above
                         * At 12.6V, the robot works with this code
                         */
                }
        }


        /* Closed loop math is below */
        int_error = ((MEM_LENGTH - 1) * int_error / MEM_LENGTH) + dist_diffl;
        left_motor = left_motor >> 1;
        right_motor = right_motor >> 1;

        wdiff = left_motor - right_motor;
        wdiff = wdiff >> 2;
        wrd = wd + wdiff + (char)((6*int_error)/(10*MEM_LENGTH));       /* Ki */
        wld = wd - wdiff - (char)((6*int_error)/(10*MEM_LENGTH));       /* Ki */


        fload_right = ((wrd_prev - right_motor) >> 1);
        fload_left = ((wld_prev - left_motor) >> 1);

        dc_r = wrd + fload_right;
        dc_l = wld + fload_left;
        wld_prev = wld;
```

```
        wrd_prev = wrd;
        go_motor();
}


/* Once it is determined that a candle is in the room the robot has just
 * entered, the robot needs to scan to find the candle.  This function
 * will do just that.  The robot does two scans, one to initially find the
 * general vicinity of the candle, and a second to find the exact location
 * of the candle once it has moved forward enough to see the white line.
 */
void
scan_room(void){
        unsigned int val2;
        unsigned char scan360 = 0;
        unsigned char scan_right = 0;
        unsigned char scan_left = 0;
        unsigned char flame_diff = 0;
        starter = FALSE;                /* Disable interrupts */
        if(number_scans == 0){          /* Depending on which scan it is doing, the
                                         * robot will scan different angles.  The
                                         * first scan is very wide, so that the robot
                                         * can see the entire room.  The second scan
                                         * is much smaller, because by the time it
                                         * does this, the robot should be fairly
                                         * centered on the candle.
                                         */
                scan_left = 30;         /* Scan left 30 degrees */
                scan_right = 100;       /* Scan right 100 degrees */
                flame_diff = 5;         /* This little variable helps us to find the
                                         * peak of the candle more realiably.  After
                                         * doing the initial scan where the robot is
                                         * taking data from all the points of the
                                         * room, the robot will scan again to refind
                                         * the highest peak that it saw.  Instead of
                                         * trying to find the exact peak, the robot
                                         * will try to find a peak that is 5 less than
                                         * the greatest peak.  This helps us to find
                                         * the candle even if it is flickering
                                         * slightly.
                                         */
        }else {
                scan_right = 70;
                scan_left = 25;
                flame_diff = 10;
        }

        val1 = 0;
        val2 = 0;
        degrees = 3;
        /* This is the initial scan left loop */
        while(scan360 <= scan_left){            /* Turn the set amount of degrees */
                poll_sensors();
                turn_stepper(LEFT);
                for(val2 = 0; val2 <= 300; val2++);
                scan360++;
```

```
                poll_sensors();
                if(flame_narrow >= val1){/* Get the highest peak */
                        val1 = flame_narrow;
                        printf("val = %d\n\r", val1);           /* This printf is necessary, it
                                                                * provides us with the necessary
                                                                * delay to ensure proper functioning.
                                                                */
                }

        }

        /* This is the scan right loop */
        while(scan360 <= scan_right){
                poll_sensors();
                turn_stepper(RIGHT);
                for(val2 = 0; val2 <= 300; val2++);
                scan360++;
                poll_sensors();
                if(flame_narrow >= val1){
                        val1 = flame_narrow;
                        printf("val = %d\n\r", val1);           /* We need this, too */
                }
        }
        dc_r = 0;                                               /* Stop the robot for a bit */
        dc_l = 0;
        go_motor();
        for(val2 = 0; val2 <= 32400; val2++);
        flame_narrow = 0;

        /* After scanning slightly left and a lot right, this loop will make
         * the robot scan back to the left until it sees a value that is
         * slightly lower than the peak it saw in the initial scan.
         */
        while(flame_narrow <= (val1 - flame_diff)){
                poll_sensors();
                turn_stepper(LEFT);
                printf("val = %d\n\r", flame_narrow);           /* Delay */
                for(val2 = 0; val2 <= 150; val2++);
        }
        dc_r = 0;
        dc_l = 0;
        go_motor();
        for(val2 = 0; val2 <= 10000; val2++);
        starter = TRUE;                         /* Enable interrupts */
}



/* This function will turn on the fan for a little bit, and then turn it
 * off and poll the wide sensor to make sure the candle has been put out.
 * It will repeat this process until the candle is extinguished
 */
void
put_out_candle(void){
        unsigned int candle_count;
```

```
	unsigned char looper;
	dc_r = 0;
	dc_l = 0;
	go_motor();
	for(candle_count = 0; candle_count <= 8000; candle_count++);
	candle_put_out = FALSE;
	while(!candle_put_out){
		PIA_B |= 0x20;

		for(candle_count = 0; candle_count <= 22000; candle_count++){
			for(looper = 0; looper <= 2; looper++);
		}
		for(candle_count = 0; candle_count <= 32400; candle_count++);
		for(candle_count = 0; candle_count <= 32400; candle_count++);
		PIA_B &= ~0x20;
		poll_sensors();
		if(flame_wide < FIRE_THRESHOLD)
			candle_put_out = TRUE;
		else
			candle_put_out = FALSE;
	}
	for(candle_count = 0; candle_count <= 32400; candle_count++);
	for(candle_count = 0; candle_count <= 32400; candle_count++);
}


/* This function will be called when the robot sees a white line that means
 * it is looking into a room.  This function will scan for a flame and then
 * either go through the routine to put out the flame or turn around to
 * continue on through the maze.
 */
void
stop(void)
{
	unsigned int temp_count = 0;
	unsigned int distance = 0;
	unsigned int ben;
	dc_r = 0;
	dc_l = 0;
	go_motor();
	poll_sensors();
	number_rooms++;
	if(flame_wide >= FIRE_THRESHOLD){
		flame_seen = TRUE;

		if(number_rooms == 1)			/* These distances are just a number
							 * of counts that the robot will use
							 * to determine how far to go into the
							 * room in order to find the candle
							 * once it has been determined that the
							 * candle is in there.  The robot will
							 * travel forward the number of counts,
							 * or until it sees the white line for
							 * the candle, whichever comes first.
							 */
			distance = 700;
```

```
else if(number_rooms == 2)
        distance = 800;
else if(number_rooms == 4)
        distance = 500;
else
        distance = 600;

initial_delay = 0;
for(temp_count = 0; temp_count <= distance; temp_count++){
        poll_sensors();
        if(number_rooms == 4)           /* Get into room differently for room 4 */
                wall_follow = RIGHT;
        else
                wall_follow = LEFT;

        left_wall_follow();
        if(initial_delay <= 120)
                initial_delay++;
        if(put_the_candle_out){
                temp_count = 600;
                goto Scan2;             /* If we see a white line before the
                                         * number of counts has been incremented,
                                         * then break out of this loop and start
                                         * the secondary scan immediately.
                                         */
        }
}

scan_room();
number_scans++;
for(ben = 0; ben <= 1000; ben++){
        dc_r = 0;
        dc_l = 0;
        go_motor();
}

PIA_B |= 0x80;                          /* Turns on siren */
while(!put_the_candle_out){
        initial_delay = 120;
        poll_sensors();
        /* We are very proud of this little block of code below.  What it
         * does is as follows:  once it does the preliminary scan and
         * points itself in the general direction of the peak, it will
         * start to move straight forward.  If it gets too close to either
         * the left or right wall, it will start to wall follow until it
         * sees the white line.  This saved us at the competition, because
         * one time it found a false peak and then went straight for the
         * wall.  It sensed this and started wall following and it found
         * the candle white line.  It then did the secondary scan and we
         * had a good run.
         */
        if(left_sensor >= WALL_THRESHOLD){
                wall_follow = LEFT;
                left_wall_follow();
        }else   if(right_sensor >= WALL_THRESHOLD){
                wall_follow = RIGHT;
```

```
                        left_wall_follow();
                }else {
                        small_increment_forward();
                }
        }

        dc_r = 0;
        dc_l = 0;
        go_motor();
        for(ben = 0; ben <= 32000; ben++);
        for(ben = 0; ben <= 32000; ben++);

Scan2:
        for(ben = 0; ben <= 200; ben++){
                small_increment_forward();
        }
        scan_room();
        number_scans++;
        dc_r = 0;
        dc_l = 0;
        go_motor();
        for(ben = 0; ben <= 8000; ben++);
        put_out_candle();
}else {                                         /* If the candle is not in the room, then
                                                 * the robot will turn a different amount,
                                                 * depending on which room it is in
                                                 */
        if(number_rooms == 0)
                degrees = 90;
        if(number_rooms == 1)
                degrees = 92;
        if(number_rooms == 2)
                degrees = 100;

        for(ben = 0; ben <= 10000; ben++);
        turn_stepper(RIGHT);
        for(ben = 0; ben <= 10000; ben++);



        for(ben = 0; ben <= 450; ben++){        /* Go forward a bit to clear
                                                 * the doorway
                                                 */
                small_increment_forward();
        }
}
}



/* This is our interrupt handler.  The only thing we have on an interrupt
 * is the white line sensor.  When we get an interrupt, this ISR will do
 * different things, depending on which flags have previously been set.
 */
#pragma interrupt_handler irq_isr
void
```

```
irq_isr(void)
{
	unsigned int temp = 0;
	unsigned char temp1 = 0;
	temp1 = PIA_A;				/* Clear the interrupt */
	if(starter){				/* We used this flag to get us off home */
		if(initial_delay <= 100){
			temp1 = PIA_A;
		}else if(!flame_seen) {
			/* a line has been detected */
			line++;
			room = TRUE;
			initial_delay = 0;
		}else if(flame_seen && !candle_put_out){   /* This means candle line */
			put_the_candle_out = TRUE;
			initial_delay = 0;
		}
		if(initial_delay >= 120){		/* Used to help us return home */
			if(candle_put_out){
				post_candle_line++;
				initial_delay = 0;
			}
		}
	}
}




void
main(void){
	unsigned int fart;
	unsigned int tone;

	OPTION |= 0x80;				/* Turn on the A/D converter */
	PIA_CRA = 0x00;				/* Select the DDRA register */
	PIA_DDRA = 0xFF;			/* Set PIA_A to be all outputs */
	/* PIA_CRA = X X 0 0 0 1 1 1 = 0x07
	 *	| | | | | | | |
	 *	| | | | | | | Interrupt on CA1 enabled (line sensor)
	 *	| | | | | | Rising edge interrupt
	 *	| | | | | Select PIA_B data register
	 *	| | CA2 is not used, these are don't cares
	 *	Read only bits
	 */
	PIA_CRA = 0x07;

	PIA_CRB = 0x00;				/* Select the DDRB register */
	PIA_DDRB = 0xF7;			/* Set PIA_B to be all outputs */
	/* PIA_CRB = X X 0 0 0 1 0 0 = 0x04
	 *	| | | | | | | |
	 *	| | | | | | | Interrupt on CB1 disabled
	 *	| | | | | | Falling edge interrupt
	 *	| | | | | Select PIA_B data register
```

```
 *          | | CB2 is not used, these are don't cares
 *          Read only bits
 */
PIA_CRB = 0x04;

IRQ_JMP = JMP_OP_CODE;
IRQ_VEC = irq_isr;
PIA_A;                                          /* Clear the CA1 interrupt */
enable();

/* This infinite loop is so that our tone decoder doesn't false trigger.
 * It will wait in this loop until the tone is applied for approximately
 * 1ms
 */
while(1){
        if (PIA_B & 0x08)
                tone++;
        else
                tone = 0;
        if(tone >= 550)
                break;
}


for(fart = 0; fart <= 600; fart++){             /* Get us off the home circle */
        small_increment_forward();
}
starter = TRUE;             /* Enable interrupts */

counter = 0;                            /* Initialize variables */
room = FALSE;
flame_flag = FALSE;
initial_delay = 120;
number_rooms = 0;
lined_up_for_island = FALSE;
put_the_candle_out = 0;
flame_seen = FALSE;
while(1){
        poll_sensors();                 /* This will get all A/D sensor data */
        if(initial_delay <= 120){       /* We use these little blocks so that we
                                         * don't get multiple interrupts on the
                                         * white lines.  We were getting multiple
                                         * triggers before, so we added this little
                                         * delay that does not allow interrupts
                                         * to occur one right after the other.
                                         */
                initial_delay++;
        }

        starter = TRUE;

        if(room){                       /* If we're in a room, stop and scan */
                stop();
        }
```

```
/* This mess of code below is to get the robot to determine which
 * room it's in, and if it has to get into room 4, this will tell it
 * how.
 */
if(!candle_put_out){                                    /* Not returning home */
        if((number_rooms == 3) && (!lined_up_for_island)){
                for(fart = 0; fart <= 1000; fart++){
                        wall_follow = LEFT;
                        poll_sensors();
                        left_wall_follow();             /* Line up robot to get into rm 4 */
                }
                while(left_sensor >= 30){
                        wall_follow = LEFT;
                        poll_sensors();
                        left_wall_follow();
                }

                dc_r = 0;
                dc_l = 0;
                go_motor();

                /* Go forwardf until robot sees front wall. */
                while(front_sensor <= (FRONT_THRESHOLD - 10)){
                        small_increment_forward();
                }
                lined_up_for_island = TRUE;

                dc_r = 0;
                dc_l = 0;
                go_motor();
                for(fart = 0; fart <= 32400; fart++);

        }else if((number_rooms == 3) && (lined_up_for_island)){
                wall_follow = RIGHT;      /* RWF into room 4 */
                left_wall_follow();
        }else {
                wall_follow = LEFT;       /* If not room 4, then LWF */
                left_wall_follow();
        }
}else {  /* After candle has been extinguished */

        /* The way our go home algorithm works is quite unique.  Depending
         * on which room it's in, we tell the robot to turn a certain
         * direction and then go straight a little to clear the candle
         * white line.  We then tell the robot to go straight until one of
         * the sensors picks up a value greater than or equal to the
         * threshold.  Once it sees this value, the robot will spin in
         * place until the correct (right or left) sensor picks up that
         * value and then the robot will start to wall follow.
         */
        if(number_rooms == 1){
                degrees = 110;
                turn_stepper(LEFT);                     /* Get away from candle line */
                while((front_sensor <= FRONT_THRESHOLD) &&
                                (left_sensor <= WALL_THRESHOLD) &&
```

```
                        (right_sensor <= WALL_THRESHOLD)){
                small_increment_forward();
        }

        dc_r = 0;
        dc_l = 0;
        go_motor();
        for(fart = 0; fart <= 10000; fart++);


        degrees = 5;
        while(right_sensor <= WALL_THRESHOLD - 10){
                turn_stepper(LEFT);
                poll_sensors();
        }
        wall_follow = RIGHT;

        for(fart = 0; fart <= 1000; fart++){
                left_wall_follow();
                initial_delay = 0;  /* Disable interrupts */
                poll_sensors();
        }
}

/* We don't return home from room 2 */
if(number_rooms == 2){
        while(1){
                dc_r = 0;
                dc_l = 0;
                go_motor();
        }
}

/* This is just like if rooms = 1 above */
if(number_rooms >= 3){
        degrees = 90;
        turn_stepper(RIGHT);
        while((front_sensor <= FRONT_THRESHOLD) &&
                        (left_sensor <= WALL_THRESHOLD) &&
                        (right_sensor <= WALL_THRESHOLD)){
                small_increment_forward();
        }

        dc_r = 0;
        dc_l = 0;
        go_motor();
        for(fart = 0; fart <= 10000; fart++);


        degrees = 5;
        while(left_sensor <= WALL_THRESHOLD - 10){
                turn_stepper(RIGHT);
                poll_sensors();
        }
        wall_follow = LEFT;
```

```
                    for(fart = 0; fart <= 600; fart++){
                        left_wall_follow();
                        initial_delay = 0;
                        poll_sensors();
                    }
                }


                dc_r = 0;
                dc_l = 0;
                go_motor();


                home = 0;
                post_candle_line = 0;
                initial_delay = 0;
                starter = TRUE;
                candle_put_out = TRUE;

                /* This infinite loop will make the robot wall follow until it is
                 * home.  The home flag is set once the robot sees two white lines
                 * after it has put out the candle and left the candle vicinity.
                 */
                while(!home){
                    poll_sensors();
                    left_wall_follow();
                    if(initial_delay <= 120)
                        initial_delay++;
                    if(post_candle_line >= 2){
                        home = TRUE;
                    }else {
                        home = FALSE;
                    }
                }

                /* Just for looks...this gets our robot to put a big portion of
                 * itself on the white line by going forward a smidge.  This is
                 * because our white line sensor is in the front and our robot
                 * comes close to stopping short without it.
                 */
                for(fart = 0; fart <= 300; fart++){
                    small_increment_forward();
                }

                /* You're home, Jerry!!!  Stop forever */
                while(1){
                    dc_r = 0;
                    dc_l = 0;
                    go_motor();
                }

            }
        }
}
```