Autonomous Vehicle

May 13, 2003

Students:

Ramona Cone

Erin Cundiff

Advisors:

Dr. Huggins

Dr. Irwin

ABSTRACT

A battery powered autonomously steered vehicle that follows a preset path along existing sidewalks is described. An onboard microcomputer (80515) controls the vehicle using three sensor systems and two drive systems. The sensor systems are: Edge detection of the sidewalk using dual ultrasonic transceivers; distance measurement using Hall Effect sensors and a toothed wheel; Orientation using a digital compass. The drive systems are: Multi-speed rear wheel drive motor with differential drive capability; Linear actuator controlled front wheel steering. The system design, simulation, and real world testing are discussed using schematics, flowcharts, scope pictures, and trial results.

TABLE OF CONTENTS

PROJECT GOAL

The goal of this project was to design a microcomputer-based system that would allow a vehicle to autonomously navigate along the sidewalks of the Bradley University quad. The vehicle would store the location of the sidewalk intersections in memory. A route for the vehicle was defined by a sequence of intersections. The vehicle navigated the route utilizing inputs from two ultrasonic sensors. A Hall Effect sensor and a digital compass were used to help negotiate through intersections. These inputs were used by the microcomputer to determine the relative position of the vehicle along its route. A system block diagram is shown in Figure 1.

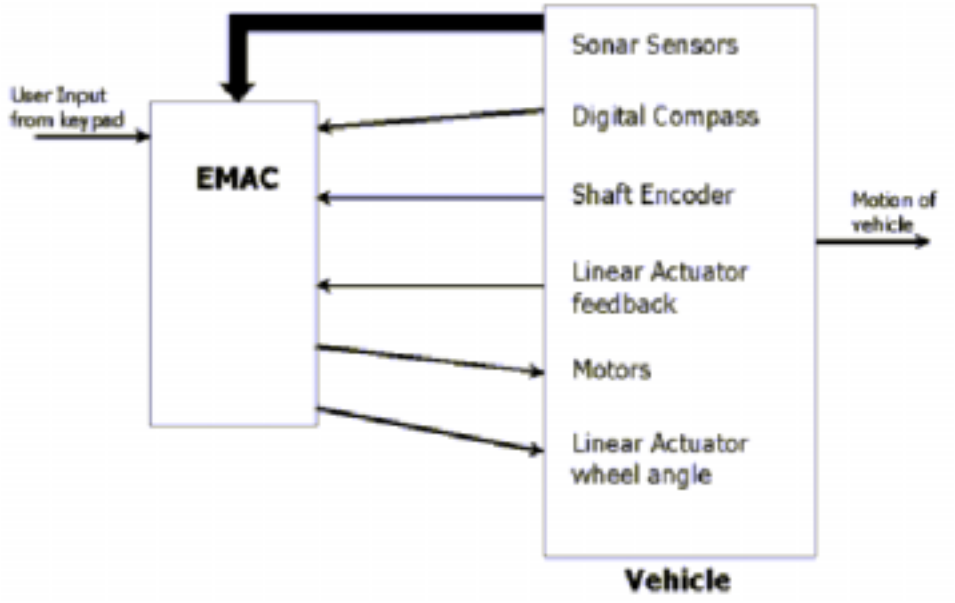

Figure 1 – System block diagram

BACKGROUND SIGNIFICANCE

As we enter the 21st century, many more tasks will be automated in our vehicles. Currently, speed and climate can be autonomously controlled, and many can use GPS to locate vehicles or help find their destinations. Today, research is occurring that will eventually enable vehicles to be completely autonomous. It is to this end that we took our design project. There were several problems to solve before this goal could be accomplished, namely: how to maneuver the vehicle, how to determine the vehicle's relative position, and how to control the vehicle.

DESIGN GOAL AND SPECIFICATIONS

The goal of this project was to design a microcomputer-based system that would allow a vehicle to autonomously navigate along the sidewalks of the Bradley University quad. To accomplish this, the vehicle had three modes of operation: query mode, straight mode, and intersection/lost edge mode. Specifications are described as they relate to each mode of operation.

Specifications for Query Mode:

- User Interface (Keypad/LCD display)
- The vehicle is stationary, and the user is prompted to enter how many intersections the vehicle should travel to. There is a maximum of 5 intersections the vehicle can travel to.
- The user can enter a number 1-5 to indicate the intersections the vehicle should include in its path.
- To start the vehicle along its assigned path, the user presses the letter A.
- In case of any problems, the user can press the letter F to stop the vehicle.
- The system ignores any invalid keys pressed by the user.

Specifications for Straight Mode:

- Relative position accuracy of +/- 5 ft
- Maximum speed of vehicle is 2.24 m/s
- Vehicle speed is proportional to the 10 Hz, PWM input to the drive circuit; it is currently set to 22% to just overcome static friction.
- Deviation from sidewalk edge is no more than 1 ft.

Specifications for Lost Edge/Intersection Mode:

- If the edge of the sidewalk is lost, but the relative position of the vehicle does not match the location of an intersection on the internal map, the vehicle will continue turning left until it re-acquires the edge of the sidewalk.
- The minimum turn radius is 10 ft.
- The vehicle returns to straight mode after it completes a turn at an intersection.

LEGACY WORK

Two previous groups have modified this vehicle. Both groups had the same objective, an autonomous vehicle. The first group tried to accomplish this task by using GPS. They were able to maneuver the vehicle, and determine the vehicle's relative position. But, their success was limited due to the large tolerance in the GPS measurements. They had the linear actuator installed on the front wheels for steering control. They also designed electronic hardware to drive the motors and linear actuator, which was reused by the second group. The second group of students to work on the vehicle chose to implement image processing to guide the vehicle. They were able to maneuver the vehicle. But, their success was limited due to the algorithm used to process the image.

PRESENT DESIGN

Given the information from the previous students' work, we wanted to approach the design from a different angle. During our junior year we had a laboratory project in which we designed and built hardware to operate an ultrasonic sensor. The sensor had an analog feedback that was proportional to the density and texture of the surface its signal echoed back from, as long as the angle and distance of it remained constant. It was our experience with the sensor in Junior Lab that led us to believe that we could detect the difference in the magnitude between the analog signals reflected from grass and concrete. This difference would allow us to guide the vehicle along the edge of the sidewalk. For this reason we chose the ultrasonic sensors to guide our autonomous vehicle.

How to Maneuver the Vehicle

One of our first goals was to be able to maneuver the vehicle. As mentioned earlier, there was electronic hardware designed by the first group of students to drive the motors and linear actuator. The second group of students also reused this hardware. We were unable to make this legacy hardware functional. Since there were no schematics to troubleshoot it, we needed to design our own electronic hardware.

Drive Motors

Requirements for this design were the following:

- Microcomputer needed to be isolated from the motors for protection from voltage spikes.
- Microcomputer needed to control the direction and speed of the motors.

When we first started working with the vehicle, the motors were still connected as per the factory installation, which switched between high-speed, low-speed, and reverse. The switch between high-speed and low-speed put the motors in either parallel or series, but the wheels operated in unison. We rewired the motors to be able to operate them separately thus allowing differential operation which could shorten the turn radius. Knowing that we were reusing the IRFP240 power MOSFET chosen by the first year's group, we found a suitable power MOSFET circuit in an application document on Fairchild Semiconductor's web site. (http://www.fairchildsemi.com/an/AN/AN-558.pdf) The circuit is shown in Figure 2. We then simulated this circuit in PSPICE with an RL circuit to model the motor. The simulation of the circuit showed inconsistent results such as large voltage spikes or slow switching. We decide that our motor model was not accurate enough. To remedy this problem we needed to determine the motor constant Kt, which has the units V/(rads/sec). Using a digital photo tachometer and a strobing light tachometer, we measured the RPM for a stalled motor and a motor with no load. We found the average rotational speed of the motor to be 17,570 RPM. Next, we measured the voltage across the motor to be 12.42 V. Then after converting V/RPM to V/(rads/sec), the resulting motor constant Kt was 6.76 x 10^-3 V/(rads/sec). Next, we measured the series resistance, and the inductance of the motor to be 0.7 Ohms and 97 uH, respectively. Using these measured values we modeled the motor in PSPICE as shown in the Figure 3. The Kt value was entered as the gain for the dependent sources. Then the design equations in Figure 4 were used to determine the resistance value to model the friction of the motor (1/B).

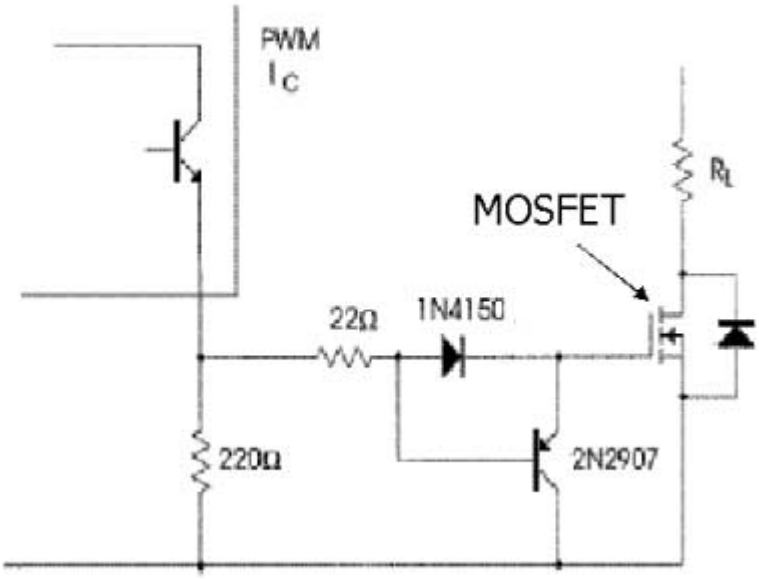

Figure 2 – Power MOSFET application circuit

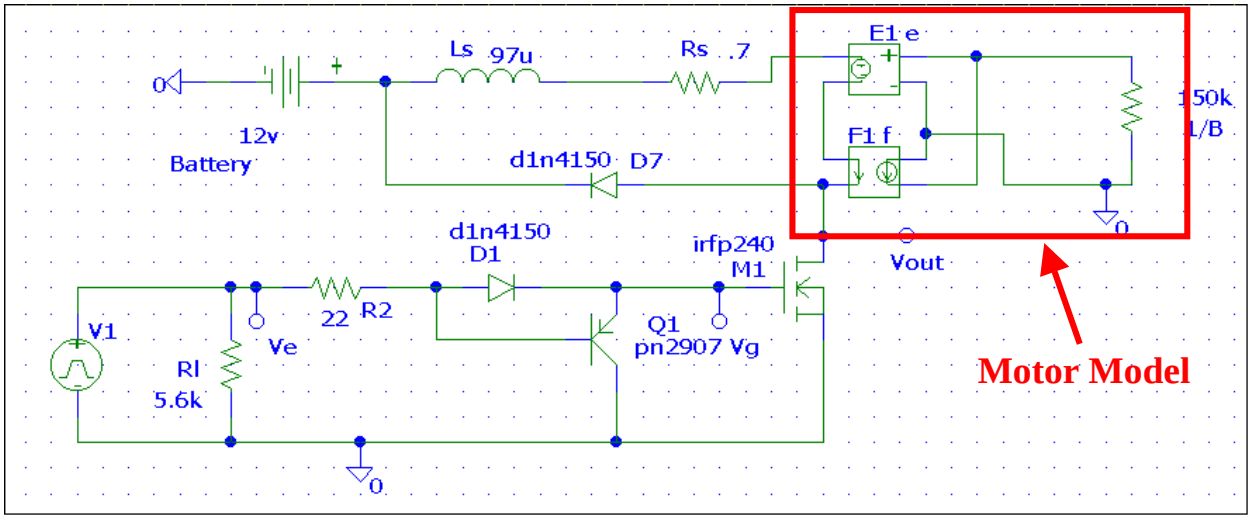

Figure 3 – PSPICE MOSFET circuit with motor model

$$V_a - V_o - V_{Ra} = V_B$$

$$\frac{V_B}{K} = W_d \leftarrow \text{ desired } \frac{\text{rads}}{\text{sec}}$$

$$V_{Ra} = I_a * R_a$$

$$\frac{1}{B} = \frac{W_d}{I_a * K_t}$$

Figure 4 – Design equations to determine motor constant

This motor model in simulation produced consistent results. These results correlated with measured values of current for stalled and no-load conditions. As shown in Figure 3, a fly-back diode was placed in parallel with the motor to protect the circuit from voltage and current spikes. Calculations were made to get an appropriate sized heat sink for the MOSFET transistors, see Figure 5.

$$P_{dMax} = 150W \qquad T_{jMax} = 150°C$$

$$T_A = 25°C \qquad \theta_{JC} = 0.83\frac{°C}{W}$$

$$(\theta_{CH} + \theta_{HA}) = \frac{P_{dMax} * \theta_{JC} + T_{jMax} - T_A}{P_{dMax}}$$

$*$ ignoring θ_{CH}(small value)

$$\theta_{HA} = 1.53\frac{°C}{W}$$

Figure 5 – Heat sink calculations for MOSFET

Our next step was to isolate the microcomputer, which provided the PWM signal to the drive circuit and control the speed of the motors. We used an optical isolator to accomplish this goal. Using the specification sheets, we determined the required resistor values to provide adequate saturation current for switching operation. The 4N25 was then introduced into the drive circuit, see figure 10. The optical isolator successfully drove the motor using a battery and frequency generator. Another design specification was to allow direction control of the motor. It was recommended that we use a double-pole, double -throw relay switch to implement this specification.

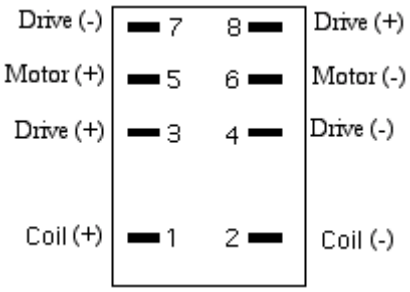

Figure 6 – DPDT relay pinout

Referring to the diagram in Figure 6, the motor would be connected to pins 5 and 6. Then the positive connections from the drive circuit would be connected to pins 3 and 8 and the negative connections from the drive circuit would be connected to pins 4 and 7. The relay operated as follows. When there was no signal on pins 1 and 2, pins 3 and 4 were connected to pins 5 and 6 and the motor turned in one direction. Then, when a signal was applied to pins 1 and 2, the relay switches and pins 7 and 8 were connected to pins 5 and 6 and the motor turned in the other direction. We needed to design the circuit that would enable the microcomputer to control the switching of the relay. To accomplish this we used a BJT as a switch, as shown in Figure 7.

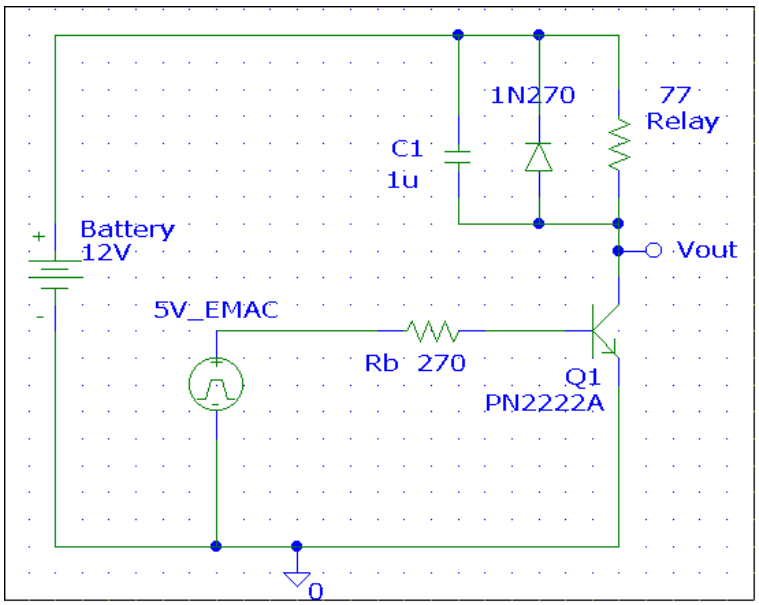

Figure 7 – BJT switch design in PSPICE

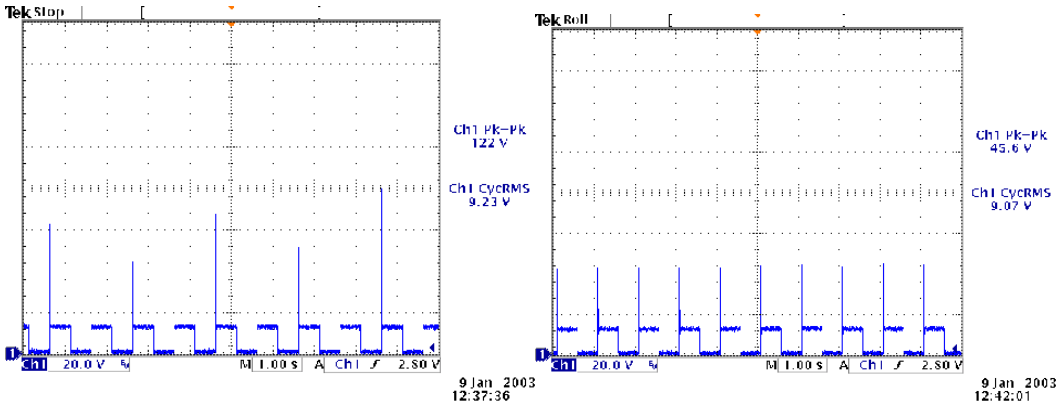

Figure 8 – BJT collector before diode and capacitor

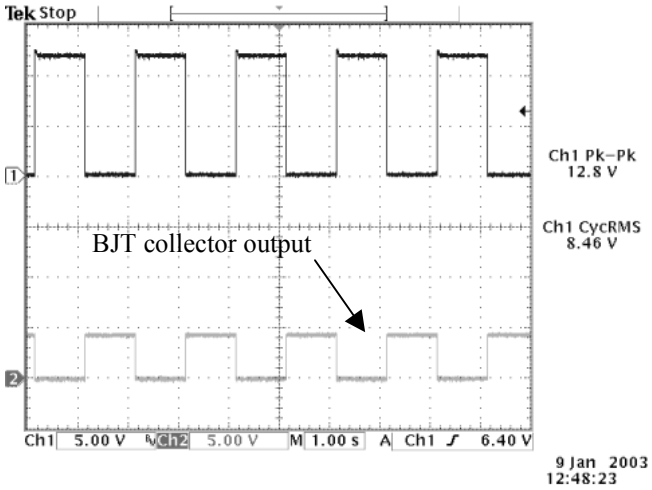

Figure 9 – BJT collector after diode and capacitor

The diode and capacitor shown in Figure 7 were added to eliminate the voltage spikes as seen in Figure 8. The result of adding these components is shown in Figure 9. The switching circuit and relay were then added to the drive circuit.

Figure 10 is the complete drive circuit. It met all the specifications: isolation of the microcomputer from the motors, direction and speed control of the motors.

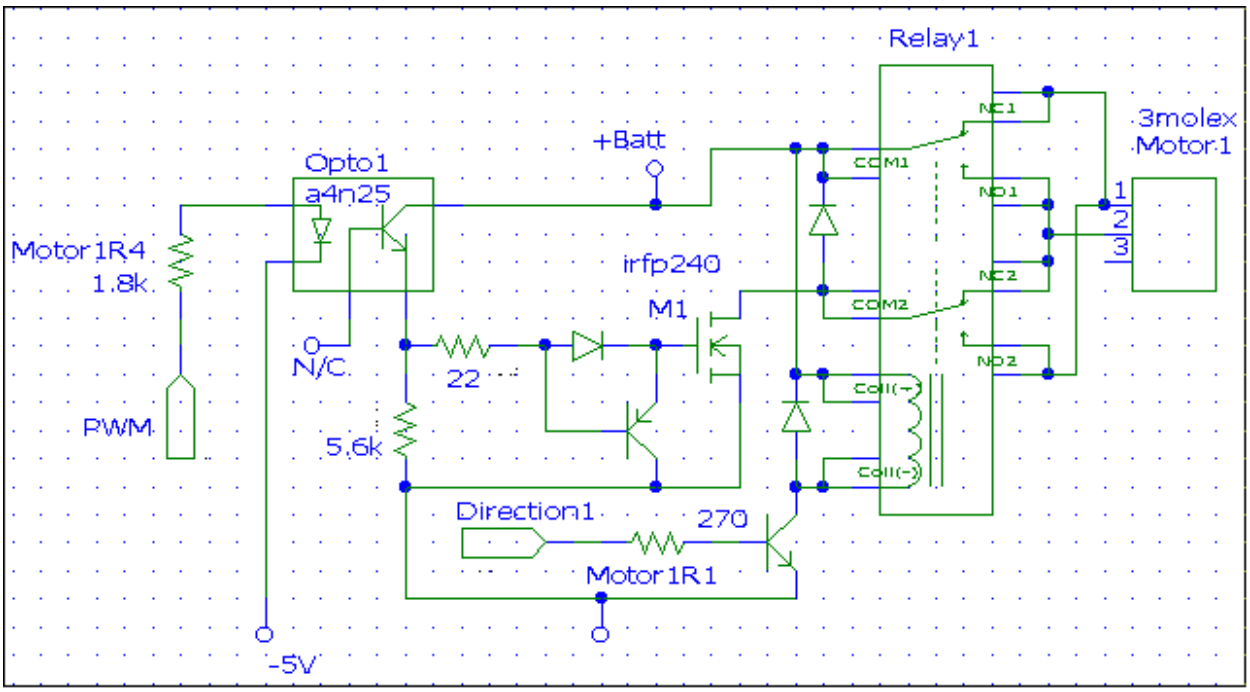

Figure 10 – Complete drive circuit

Control Steering

We wanted to be able to control the steering via the onboard microcomputer. This would involve being able to control the position of the linear actuator and determine the linear actuator's position. The first year's group had chosen a packaged h-bridge component, LMD18200, to control the position of the linear actuator. Since it could adequately handle 12 V from the battery and the maximum 6 A current needed by the linear actuator, we decided to use this component. A suitable circuit was found in the h-bridge's specifications sheet (see Figure 11). From this circuit we determined that we needed to design hardware to provide a 5 V square wave to the component and deliver a direction and brake signal from the microcomputer. We used a NE555 timer to deliver the 5 V square wave to the h-bridge. Experiments in lab showed that a 5 kHz frequency would be sufficient to drive the linear actuator. Using

the design equations provided on the specification sheets for the timer, we determined the required resistor values to achieve this frequency. Our final component values resulted in a 5 kHz signal with a 55.2% duty cycle. The linear actuator had a potentiometer feedback to determine its position. In lab, we set up the circuit, as shown in Figure 12, with the feedback (white wire) returning to the A/D converter for processing.

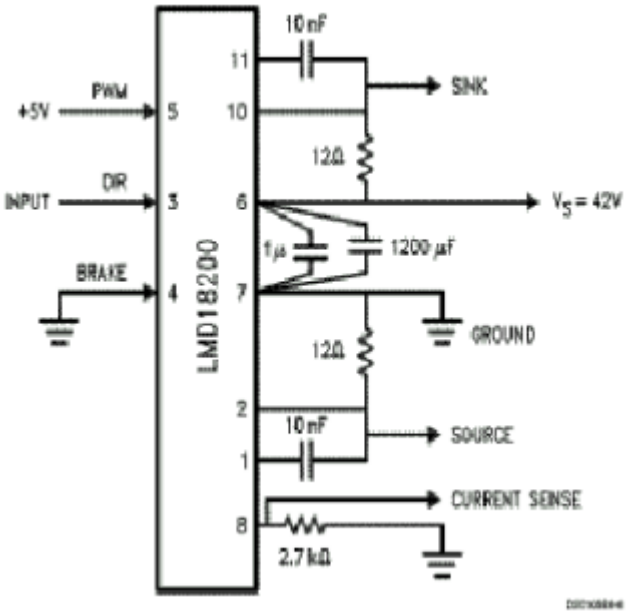

Figure 11 – Test circuit for h-bridge

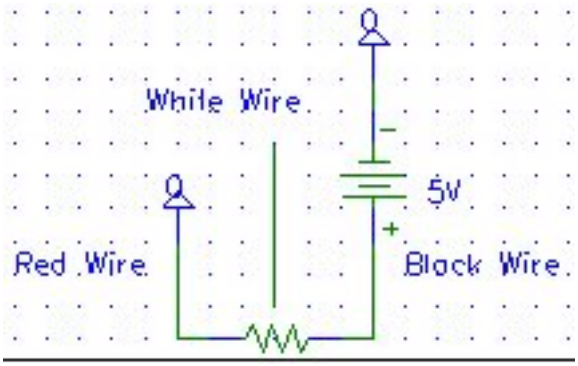

Figure 12 – Linear actuator feedback circuit

Voltage level	Decimal value	Hex value	
0.20	10	Ah	Full right
1.52	102	66h	Full center
3.80	236	C2h	Full left

Figure 13 – Table of A/D values and corresponding hex values

Figure 13 is a listing of recorded feedback voltage values according to the linear actuator's position. In bench testing, the linear actuator circuit shown in Figure 11 operated successfully. The circuit shown in Figure 14 controls the linear actuator's position with a voltage feedback value being processed by the microcomputer to determine its position.

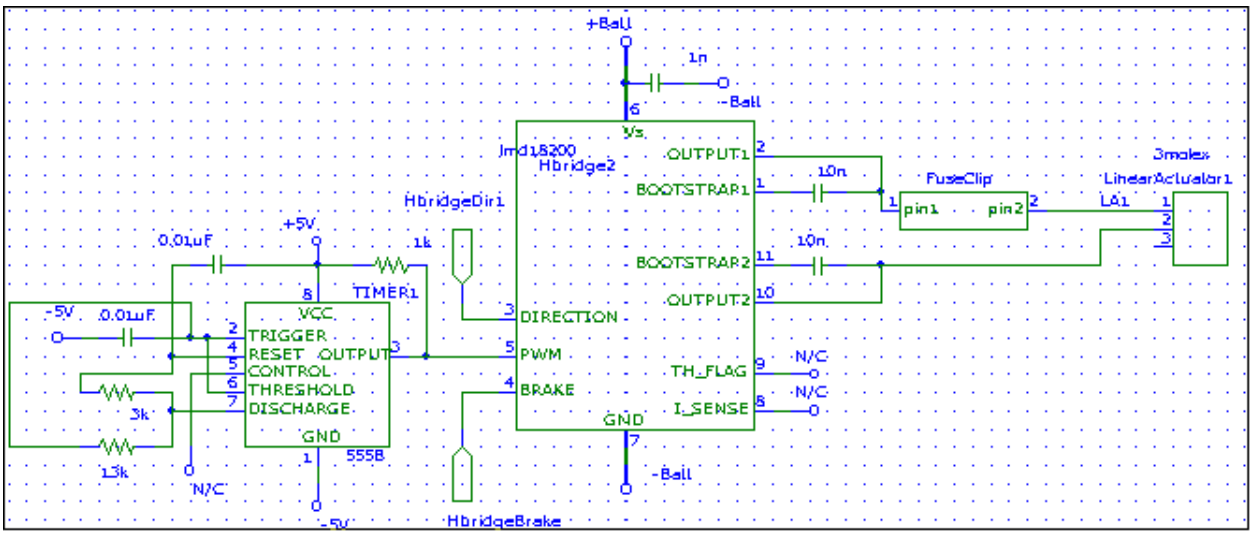

Figure 14 – Complete linear actuator circuit

Calculations to get an appropriate sized heat sink for the linear actuator are shown in Figure 15.

$$P_{dMax} = 25W \qquad T_{jMax} = 110°C$$

$$T_A = 25°C \qquad \theta_{JC} = 5\frac{°C}{W}$$

$$(\theta_{CH} + \theta_{HA}) = \frac{P_{dMax} * \theta_{JC} + T_{jMax} - T_A}{P_{dMax}}$$

$*$ ignoring θ_{CH} (small value)

$$\theta_{HA} = 2.5\frac{°C}{W}$$

Figure 15 – H-bridge heat sink calculations

How to Determine the Vehicle's Relative Position

Relative vehicle position was determined using two ultrasonic sensors, a Hall Effect sensor, and a digital compass. The ultrasonic sensors helped the vehicle follow the sidewalk. The Hall Effect sensor helped determine the distance traveled by the vehicle, and the digital compass kept track of the vehicle heading.

Follow the Sidewalks

In order to keep track of relative vehicle position, the sidewalks in the quad were used as a map where each intersection was a waypoint. The vehicle could travel to five possible intersections or waypoints. It operated under the assumption that it would always start from the same point, and would travel in only one direction. A map of the Jobst-Baker quad was stored in the microcomputer's memory. The internal map contained the distance to each intersection and what the vehicle should do at each intersection: turn left, turn right, or go straight. Figure 16 shows the five possible intersections the vehicle could travel to in the Jobst-Baker quad map. The intersections are labeled 1-5. The vehicle used the sidewalks in the quad as a map, but in order to do this, the vehicle needed to be able to stay on the sidewalk and to identify the intersections. To accomplish this, we used ultrasonic sensors. We chose the ultrasonic sensor from previous experience.

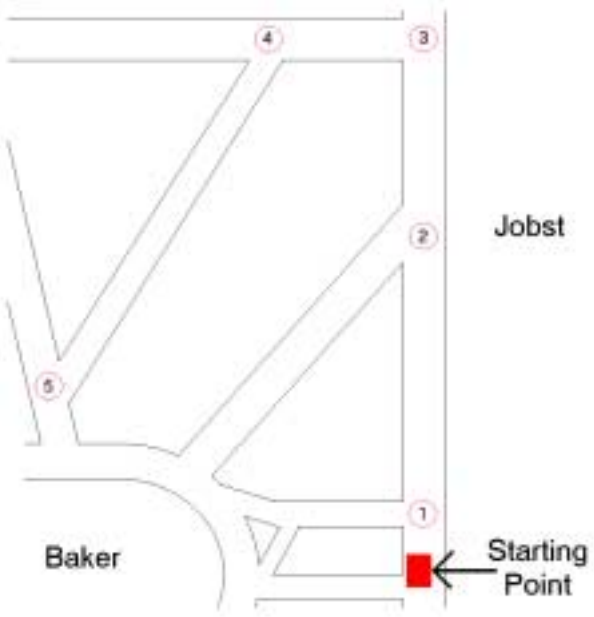

Figure 16 – Jobst-Baker quad map

Ultrasonic Sensor Operation

A block diagram of the ultrasonic sensor can be seen in Figure 17.

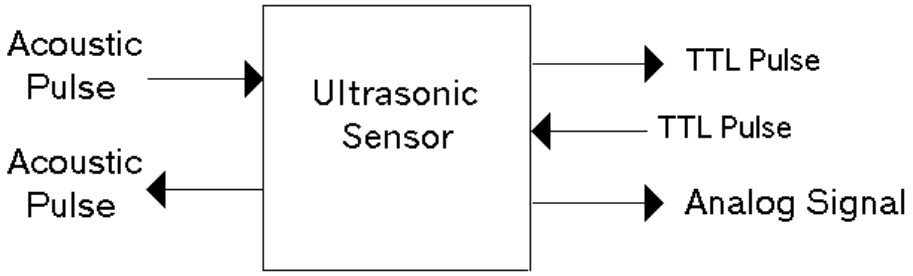

Figure 17 – Block Diagram of ultrasonic sensor with inputs and outputs

TTL Pulse Input:

- This input caused the sensor to transmit an acoustic pulse. This TTL pulse came from the microcomputer.

Acoustic Pulse Input and Output:

- If the transmitted acoustic pulse would hit an object, a portion of the reflected energy would propagate back to the sensor where it was detected. If a return acoustic pulse was detected, the sensor generated two outputs: a TTL output, and an analog output.

TTL Signal Output:

- The time delay of the TTL signal relative to transmission time was proportional to the distance of the object to the sensor.

Analog Signal Output:

- The magnitude of this output depended on the reflected energy received by the sensor. This magnitude depended on the angle of the surface to the sensor, the distance of the surface to the sensor, the density of the surface, and the texture of the surface.

We mounted the two ultrasonic sensors on the vehicle, so that the angle of the surface to the sensor and the distance of the surface to the sensor were the same. Therefore, when comparing output analog signals from the two ultrasonic sensors, the differences in the magnitude of the analog output signal could be attributed to different surface densities or different surface textures. Figure 18 shows two different analog signal outputs from an ultrasonic sensor. The analog output in response to grass had a magnitude of 50 mV, and the analog output in response to concrete had a magnitude of 270 mV. This experimental data showed us that the ultrasonic sensors could be used effectively to distinguish between grass and concrete.

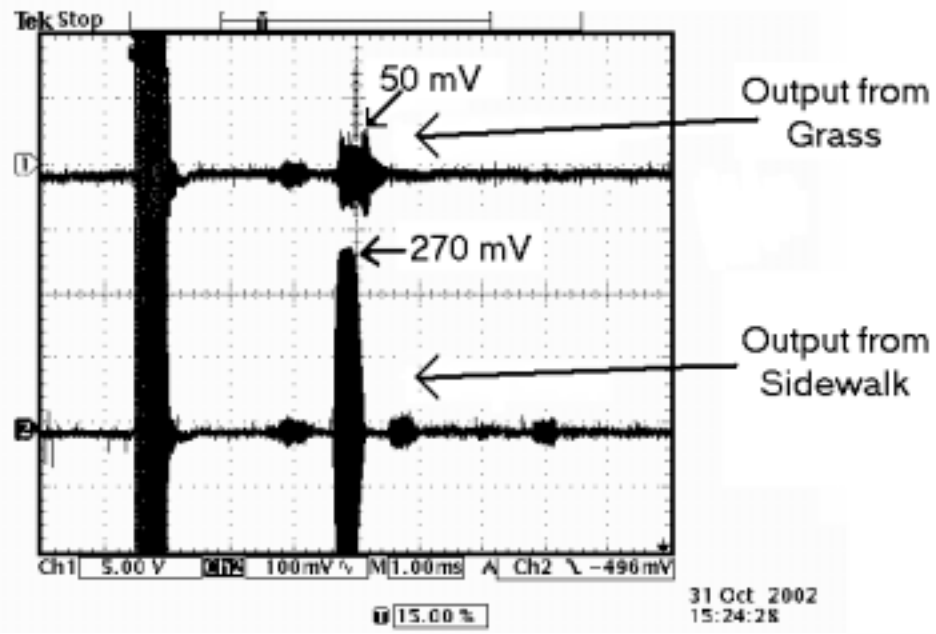

Figure 18 – Ultrasonic sensor analog outputs in response to grass and concrete

In order to process this information, the analog signal had to be sent to the microcomputer. However, within the pulse of interest, there were 16 pulses at 50 kHz. Trying to sample a magnitude of 50 mV or 270 mV from this type of signal would be very difficult.

To make sampling the data easier, we designed a hardware circuit to manipulate the analog signal into a signal that would be easier to sample (see Figure 19 for a block diagram of the hardware circuit). First, we sent the analog signal through an amplifier. We did this because 50 mV was a fairly small voltage for the microcomputer to sample. Then we sent the signal through a high-pass filter to eliminate the DC offset that was present in the analog signal. (Note: The DC offset cannot be seen in Figure 18 because the data was taken on the oscilloscope using AC coupling.) Next, we sent the signal through a rectifier to get rid of any negative voltages because the microcomputer cannot handle negative voltages. After that, we sent the signal through a low-pass filter. This filtered out the 50 kHz, and changed the 16 pulses into one solid pulse. Finally, we sent the signal through a buffer to prevent any reflection problems when the signal was sent to the A/D converter on the microcomputer. See Appendix A for a detailed schematic of this circuit and more detail on the design process for this circuit.

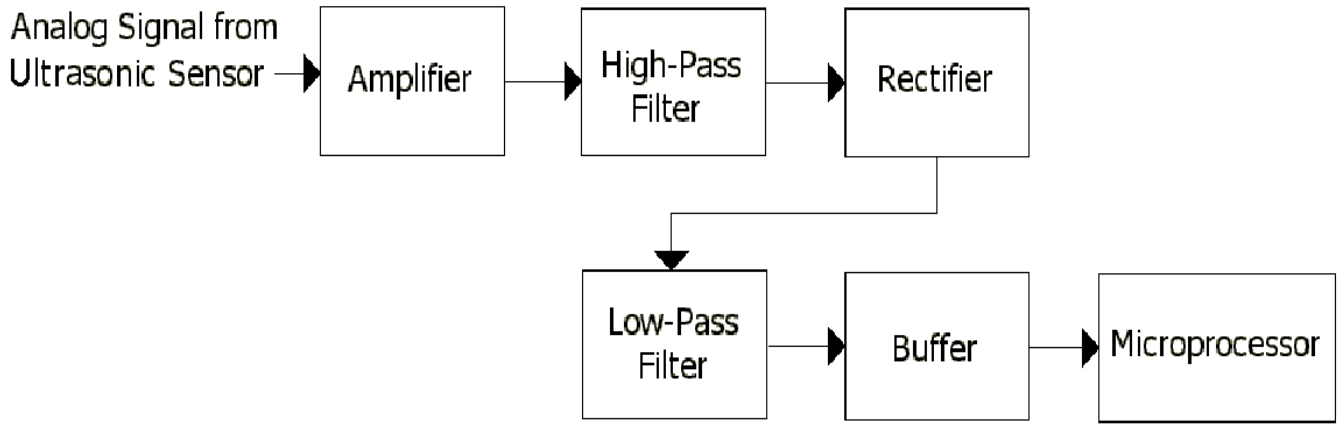

Figure 19 – Block Diagram of ultrasonic sensor analog signal amplifier and filter

Figure 20 shows the input to the hardware circuit seen in Figure 19 compared to the output. The input to this circuit was the analog output from the ultrasonic sensor. As can be seen in Figure 20, the analog output from the sensor was amplified and filtered into a signal that was much easier to sample with the microcomputer. Also, notice that the rising edge of the TTL signal output from the ultrasonic sensor coincided with the analog pulse of interest. Note that the information before and after this pulse was insignificant. We utilized the fact that the TTL signal rising edge coincided with the analog signal pulse of interest to help sample the data. First, we sent the TTL signal to an external interrupt on the microcomputer. Then we set up the software so that the external interrupt would be triggered on the rising edge of this TTL signal, and at the time of interrupt, the software would sample the analog signal through the A/D converter.

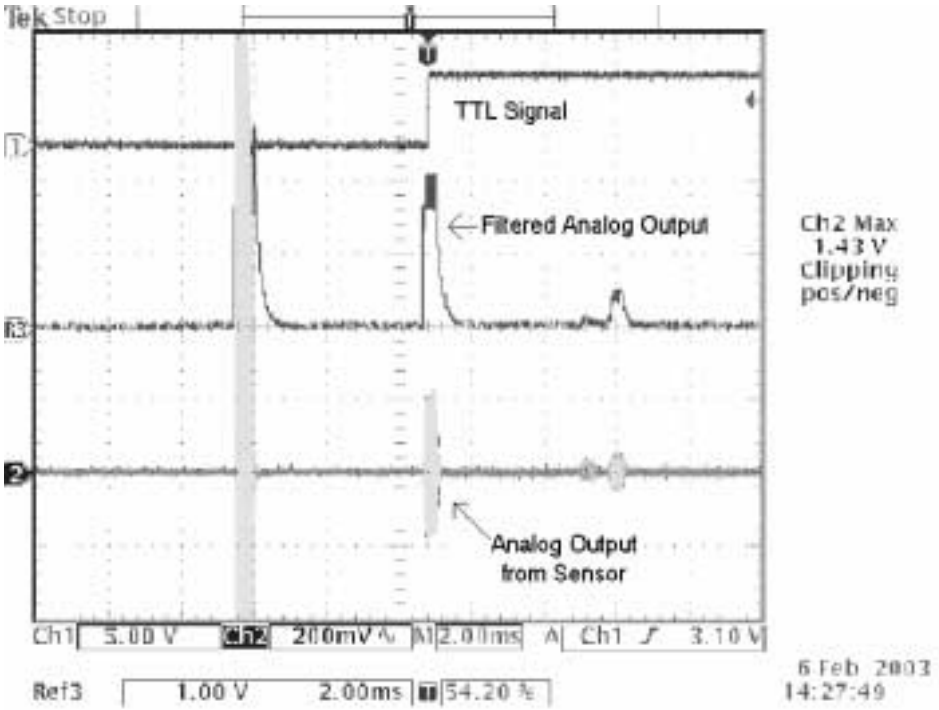

Figure 20 – Ultrasonic sensor analog output vs. filtered analog output

One problem encountered when using the TTL signal as an external interrupt was that the TTL signal had noise during the acoustic pulse transmission. This noise had a peak voltage that was at a TTL high level. This caused an external interrupt to occur during transmission time, and therefore, the software was sampling the analog signal during transmission, which led to false data. Figure 21 shows the noise on the TTL signal during transmission. To correct this problem, we sent the TTL signal through an inverter before sending it to the external interrupt. As can be seen in Figure 22, this solved the noise problem. The software was then adjusted to sample the analog signal on the falling edge of the TTL signal rather than on the rising edge.

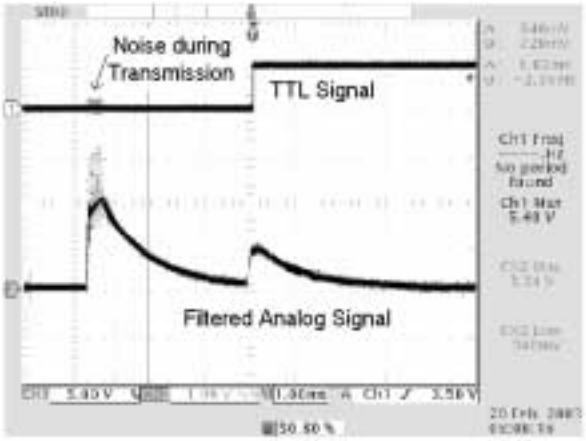

Figure 21 – Ultrasonic TTL signal noise during transmission

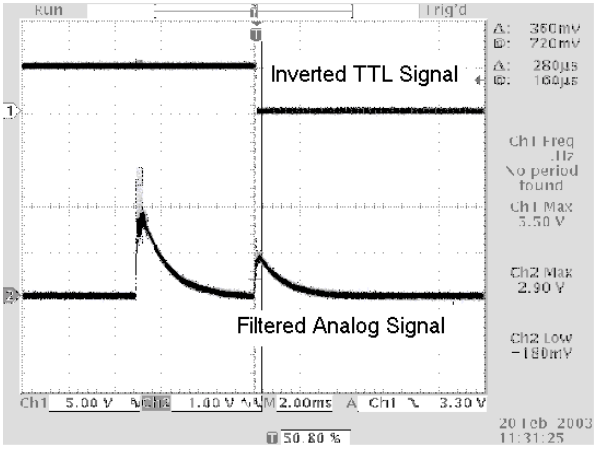

Figure 22 – Ultrasonic TTL signal inverted to eliminate noise

Figure 23 shows how the two ultrasonic sensors were mounted on the vehicle. Both of the sensors pointed towards the ground so that the transmitted acoustic pulse was perpendicular to the ground. As described before, this configuration kept the angle and distance of the ground to the sensor constant.

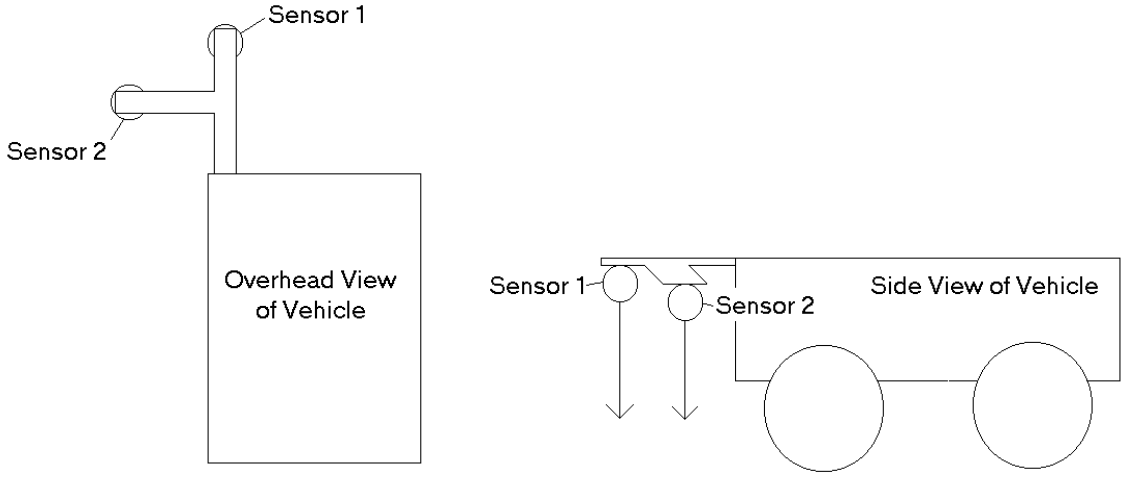

Figure 23 – Ultrasonic sensor configuration on vehicle

Determine Distance Traveled

We kept track of the relative vehicle position by measuring the distance traveled by the vehicle. We did this using a Hall Effect sensor. Figure 24 shows a picture of a Hall Effect sensor mounted on the inner part of the vehicle's wheel. We chose this sensor for this application because it was a durable sensor. Since it was mounted on the inner part of the wheel, it would experience a lot of vibrations. Our technician mounted two Hall Effect sensors on the vehicle. One was mounted on the front right wheel, and the other was mounted on the front left wheel. Since the accuracy of the distance measurement did not need to be exact for this application, only the front right wheel Hall Effect sensor was used. The right wheel sensor was used because it was more likely to be on concrete than the left wheel. When a wheel was in the grass, there was a better chance of the wheel slipping, which would cause inaccurate readings. In the future, both Hall Effect sensors could be used for more accuracy.

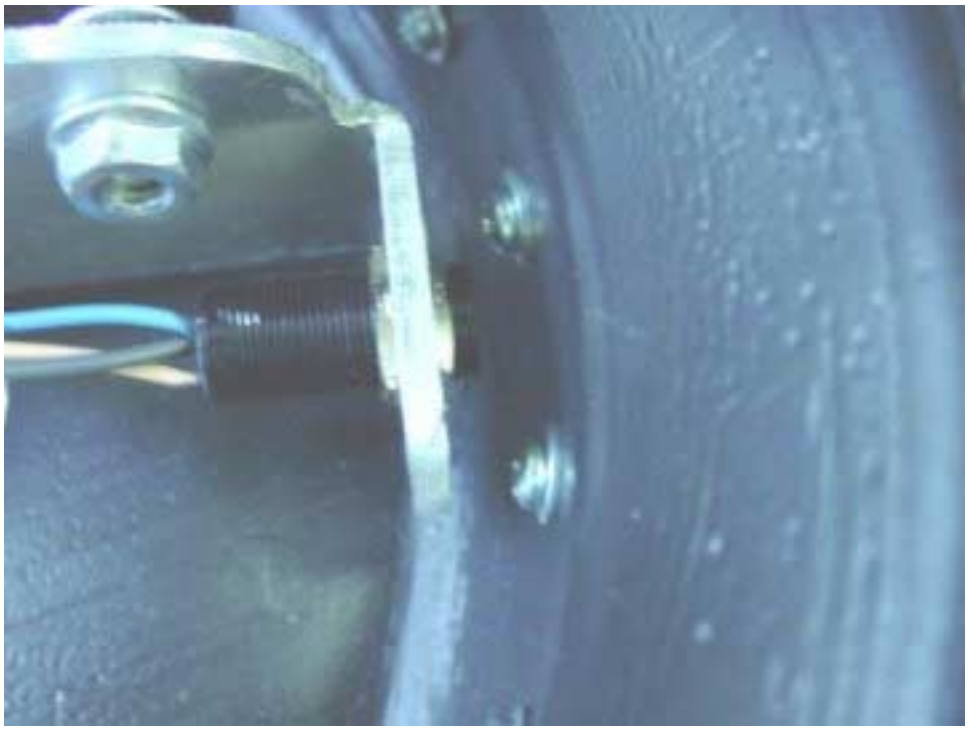

Figure 24 – Hall Effect sensor mounted on inner part of vehicle wheel

The Hall Effect sensor detected the motion of a target containing iron. We had 25 screws mounted on the inner part of the vehicle's wheel, which can be seen in Figure 24. These screws served as the iron targets.

Each time one of these screws passed the Hall Effect sensor; the Hall Effect sensor produced a TTL pulse. Figure 25 shows the output of the Hall Effect sensor in response to these screws. We sent the output TTL signal from the Hall Effect sensor to an external interrupt on the microcomputer, and after every external interrupt, the distance count was incremented. The circumference of the wheel was measured to be 4 feet, so after the Hall Effect sensor produced its 26th pulse, the wheel had gone through one full rotation. In other words, the vehicle had traveled 4 feet. By keeping track of the number of pulses from the Hall Effect sensor through the external interrupt, the microcomputer software kept track of the distance traversed by the vehicle. In the future, these sensors could also be used to measure the speed of the vehicle, since the frequency of the TTL signal output from the Hall Effect sensor was proportional to the speed of the vehicle.

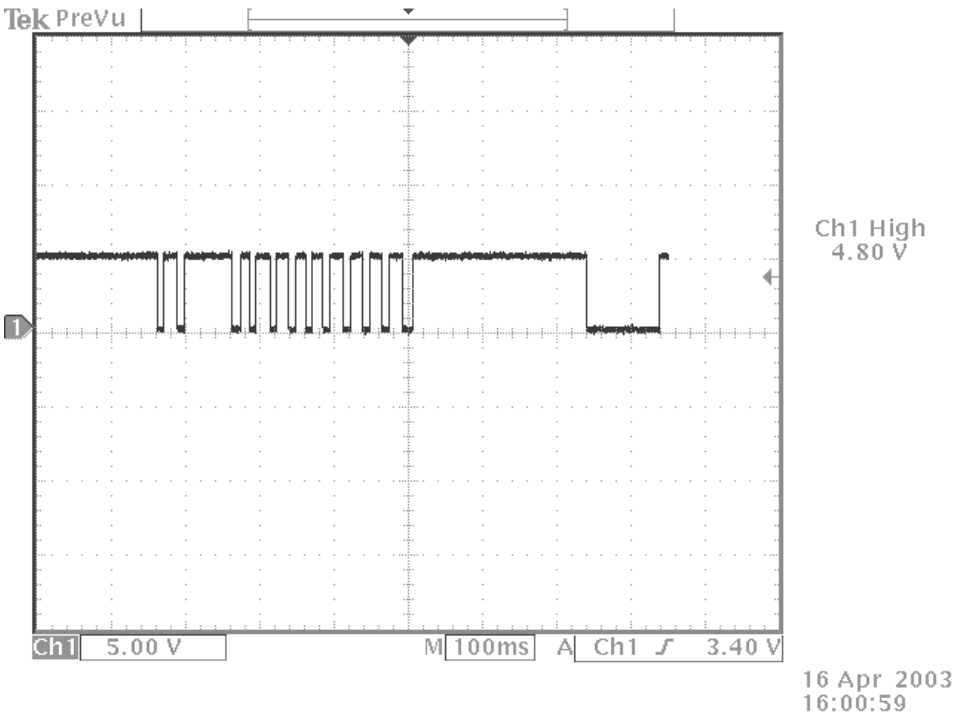

Figure 25 – Output from Hall Effect sensor in response to ferrous targets

Determine Vehicle Heading

Another way we kept track of the relative vehicle position was to determine the vehicle heading. The digital compass used for this application had a 9-bit serial word output. The output was from 0˚-360˚, where North equals 0˚. Figure 26 shows how each direction corresponded to the compass reading. For this application, 9-bit accuracy was not needed. Plus, the microcomputer being used was an 8-bit microcomputer. So, we ignored the least significant bit from the compass output. The vehicle heading information was used while the vehicle was turning. This information indicated when the vehicle had made a complete turn at one of the intersections.

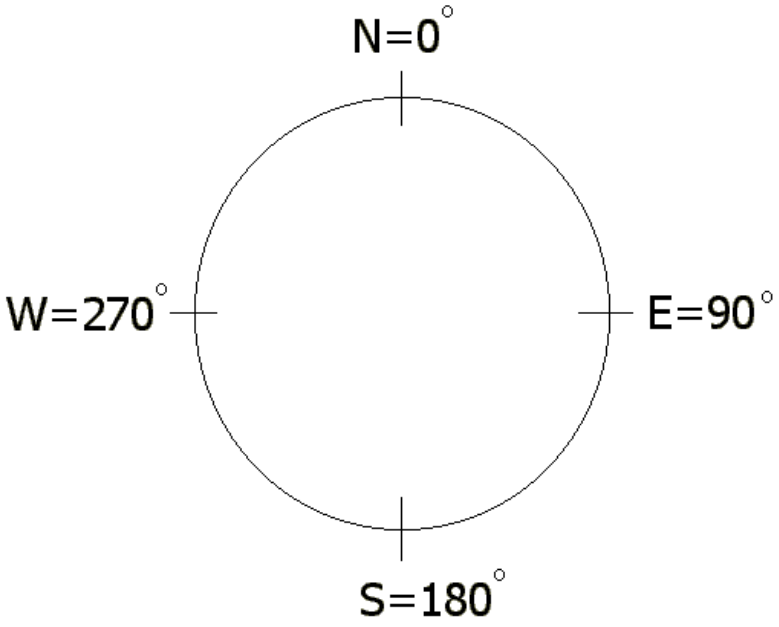

Figure 26 – Digital Compass output relative to direction

How to Control the Vehicle

An 80515 microcomputer was used to control the vehicle through modular programming. As an interface between the hardware and the software the vehicle had integration modules. As mentioned earlier, there were three possible modes of operation for the vehicle: Query Mode, Straight Mode, and Intersection/Lost Edge Mode. From these three modes of operation the vehicle determined where to go and how to get there.

Integration Modules

The microcomputer we used required some initialization code to set it up for operation. The method for initialization was found in the manual for the 8051 microcomputer. We also had software provide the inputs and outputs to control the electronic hardware. A PWM signal was created by timer 2 to drive the motors. Figure 27 shows the equations used to determine the reload values for the frequency and duty cycle of the PWM signal, which were put into the compare and capture registers to produce the desired signal.

To calculate period reload value:

$$desired\_cnt = \frac{1}{2 * desired\_frequency}$$

$$timer\_count = desired\_cnt \div \frac{12}{11.06MHz}$$

$$reload\_value = 2\text{^}16 - timer\_count$$

To calculate duty cycle reload value:

$$timer\_count * \%duty\_cycle = new\_timer\_count$$

$$reload\_value = 2\text{^}16 - timer\_count$$

Figure 27 – Timer 2 reload value equations

When the PWM signal was sent into the optical isolator, it successfully drove the motors. Next, a module was created to read the A/D converter, and then control the position of the linear actuator appropriately. The flowcharts shown in Figures 28, 29, and 30 show the logic we were using to control the steering of the vehicle: right, center, and left.

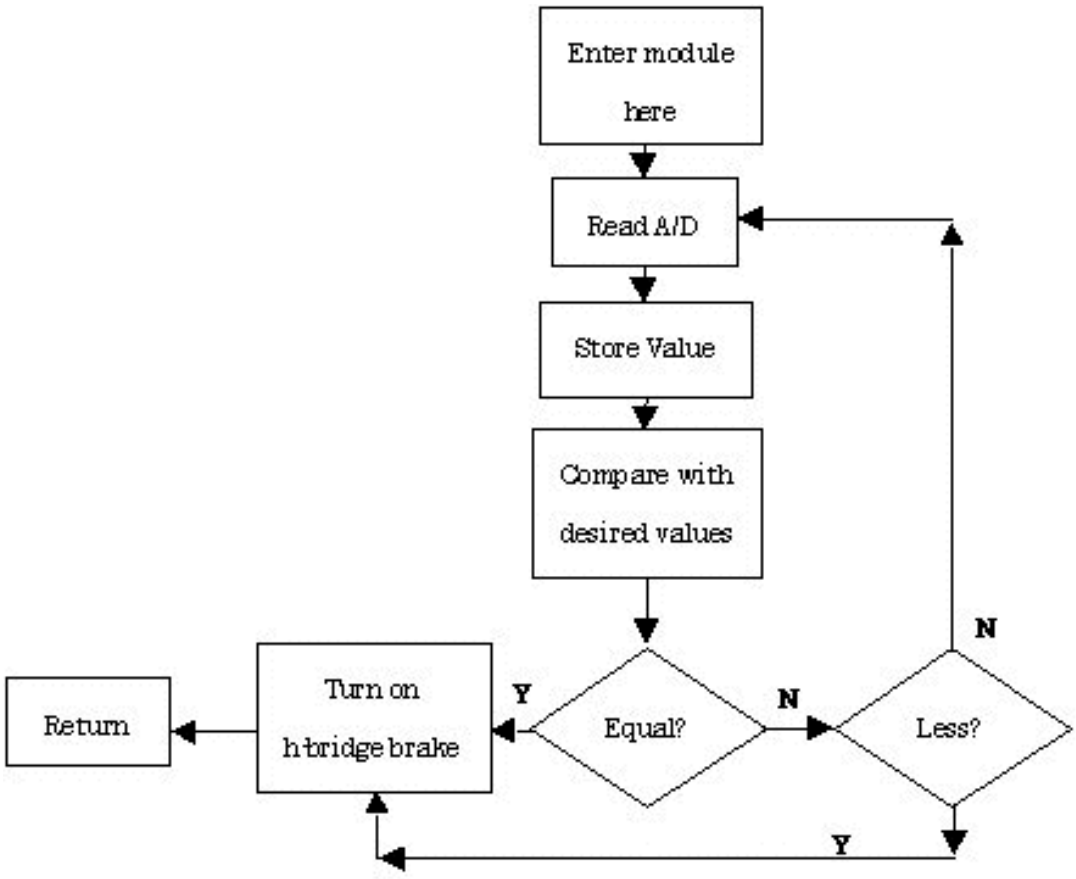

Figure 28 – Turn wheels right flowchart

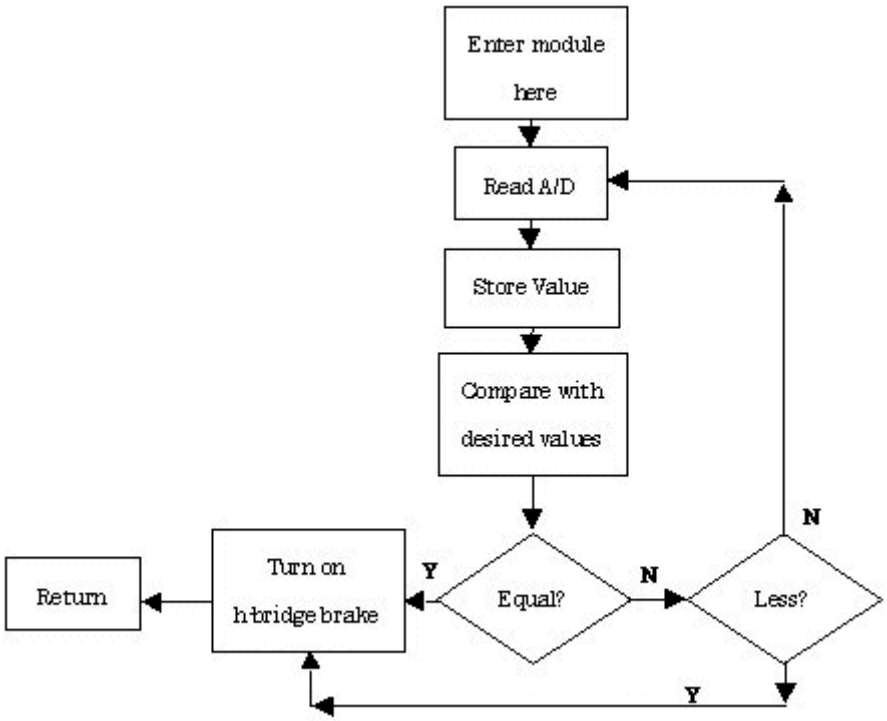

Figure 29 – Position wheels in center flowchart

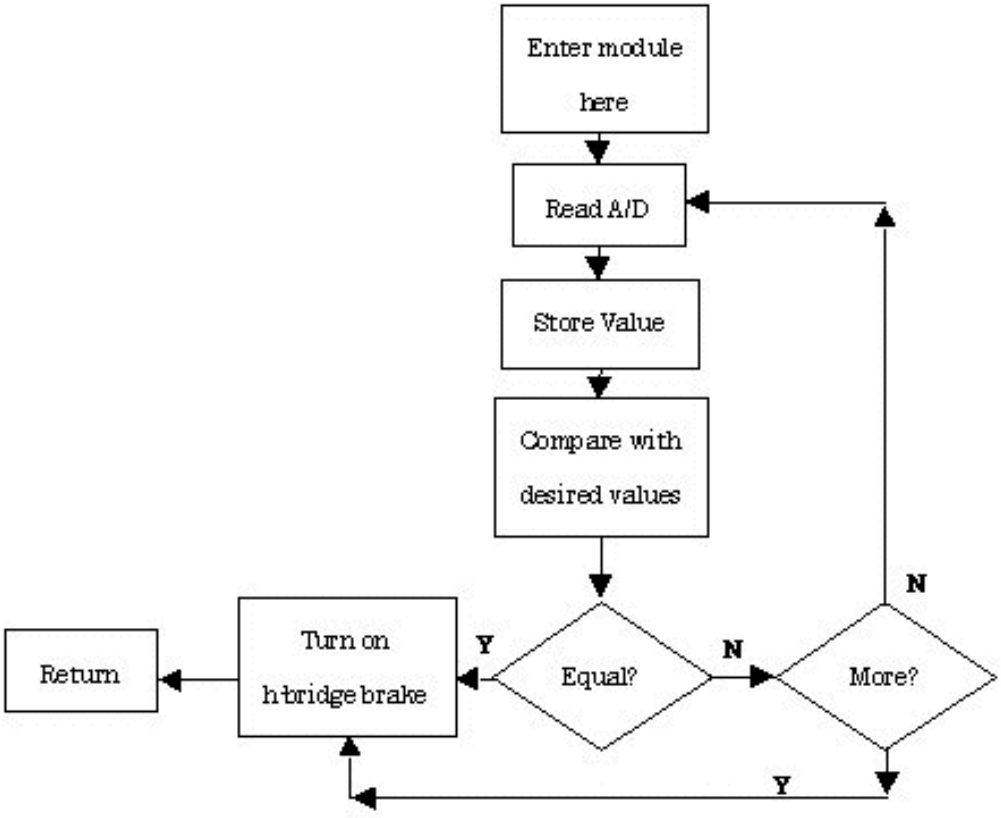

Figure 30 – Turn wheels left flowchart

We also had a software module to read the digital compass. The flowchart for its operation is shown in Figure 31. Lab results from testing this module are shown in Figure 32, which have the microcomputer's reading printed in the upper right-hand corner and the binary value output by the compass labeled. Calculations of the binary value showed that the software was reading the digital compass correctly.

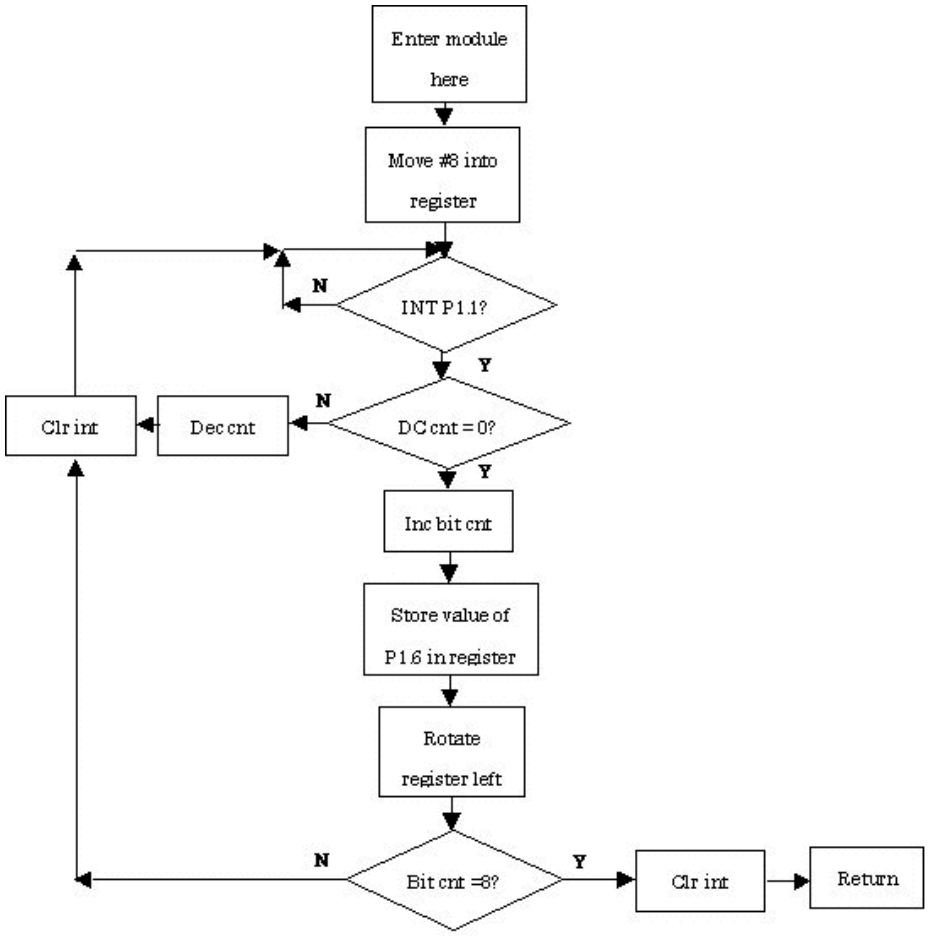

Figure 31 – Read compass flowchart

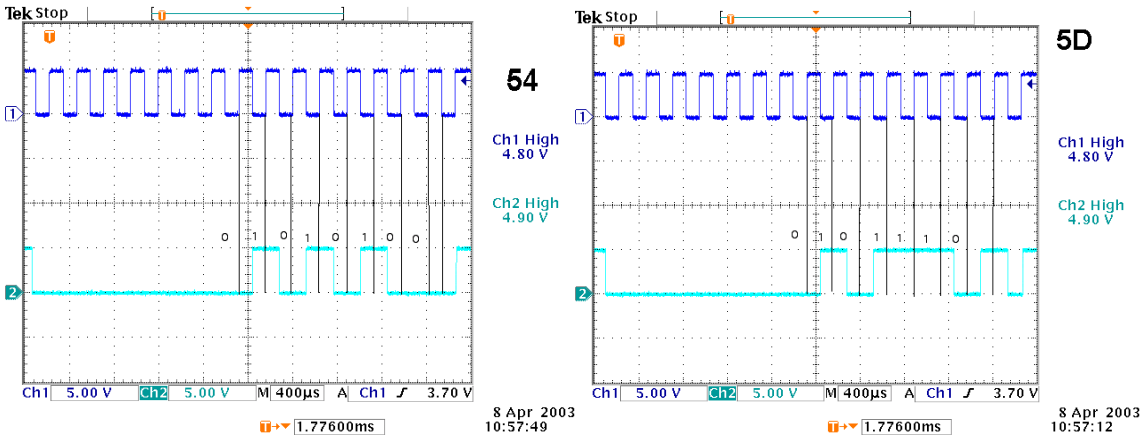

Figure 32 – Results from compass test

Determine Where to Go – Query Mode

In order to control the vehicle, it needed to know where to go, and this was the purpose for Query Mode. A software flowchart for Query Mode can be seen in Figure 33. First, Query Mode started up and initialized the microcomputer, the LCD, and the keypad. Then the LCD prompted the user with a question: "How many intersections?" The user would specify how many intersections the vehicle should go to by entering a number 1-5. (Refer back to Figure 17 to see the 5 possible intersections in the Jobst-Baker quad.) The program waited for a keypad input from the user. After the user entered a number 1-5, this information was stored in the microcomputer's memory. Then the LCD prompted the user to press the letter 'A' to start the vehicle. Once the letter 'A' was pressed, the vehicle would then start to travel to the intersections entered by the user and would stop when it reached the last intersection. For example, if the user entered the number 3, the vehicle would travel to the third intersection and stop.

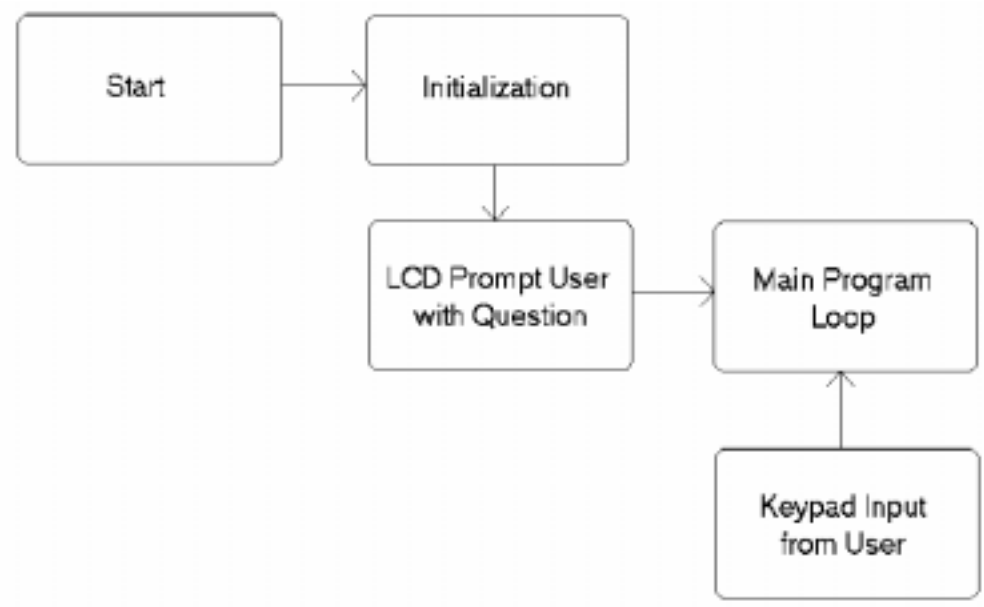

Figure 33 – Query Mode software flow chart

Determine How to Get There – Straight Mode

In order to control the vehicle, it needed to know how to travel to where it needed to go. This was the purpose for Straight Mode. Straight Mode was the default mode of operation. This mode utilized the two ultrasonic sensors mounted on the vehicle. The sensors could be in three possible situations. Figure 34 shows situation 1, where sensor 1 detected sidewalk and sensor 2 detected grass. In this situation, the vehicle would turn slightly left. Figure 35 shows situation 2, where both sensors 1 and 2 detected grass. In this situation, the vehicle would turn slightly right. The result, which can be seen in Figure 36, was the vehicle would zigzag along the edge of the sidewalk. This zigzag pattern would keep the vehicle on the edge of the sidewalk.

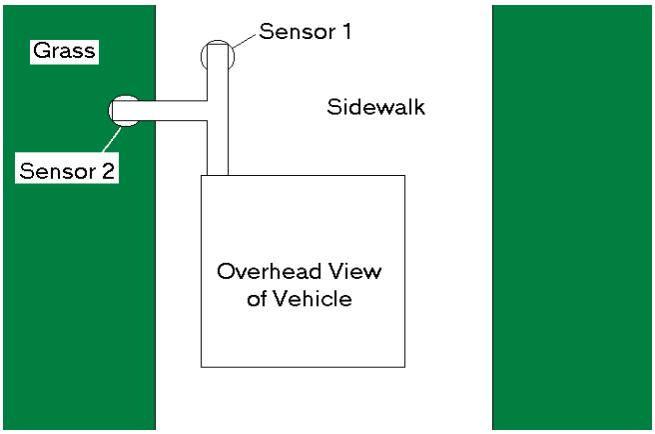

Figure 34 –
- Situation 1:
  - Sensor 1 = Sidewalk
  - Sensor 2 = Grass

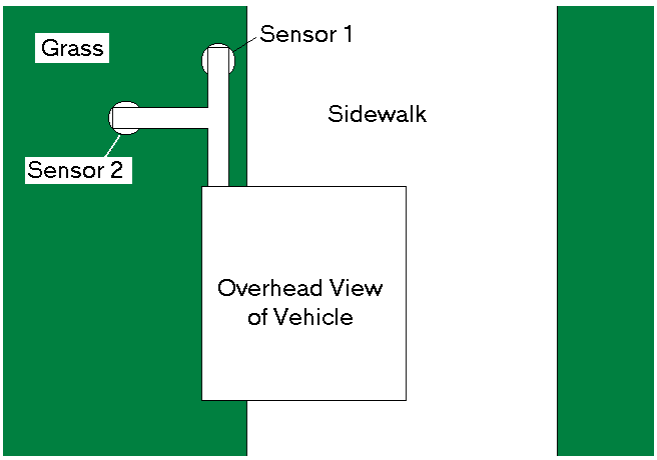

Figure 35 –
- Situation 2:
  - Sensor 1 = Grass
  - Sensor 2 = Grass

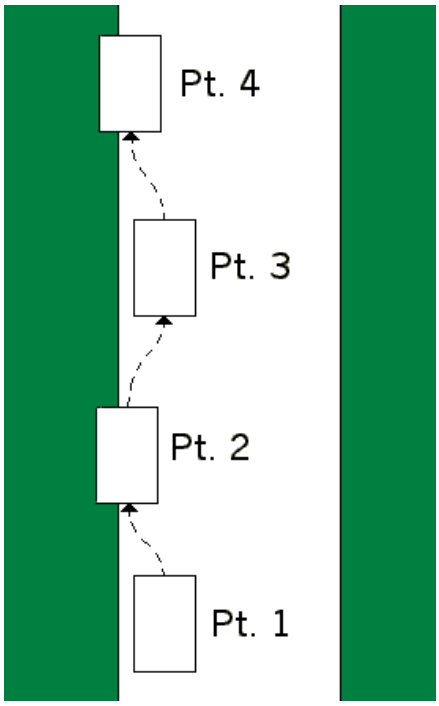

Figure 36 –
- Straight Mode

Figure 37 shows situation 3, where both sensors 1 and 2 detected concrete. In this situation, the vehicle's software would check the actual distance traveled to see if the vehicle was actually at an intersection. This process is discussed in more detail in the intersection/lost edge mode section.

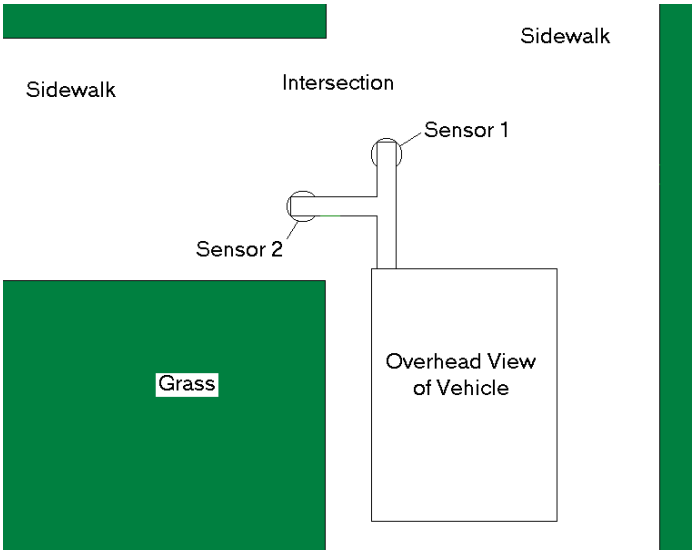

Figure 37 –
- Situation 3:
  - Sensor 1 = Sidewalk
  - Sensor 2 = Sidewalk

A problem encountered during straight mode was that while the wheels would turn slightly right or slightly left, the linear actuator would cause noise on the Hall Effect sensor signal, which can be seen in Figure 38. Since we sent the Hall Effect signal to an external interrupt, the noise from the linear actuator caused the external interrupt to trigger falsely. This was causing the distance reading to be too high. In the future, this problem could be fixed by having two separate power supplies and by isolating the linear actuator from the microcomputer. Due to a lack of time, this problem was avoided by ignoring any interrupts that would occur while the wheels turned. Since the distance measurement had a +/- 5 ft tolerance, avoiding the noise problem was not an issue.

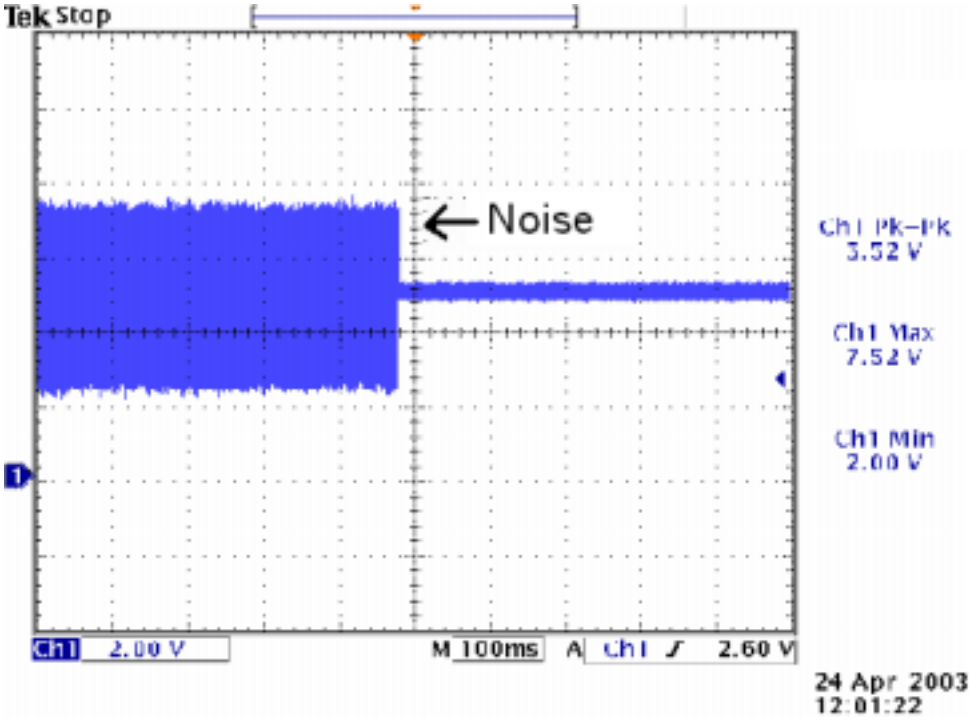

Figure 38 – Hall Effect signal with linear actuator noise

Straight mode was very successful. The vehicle successfully traveled to the first three intersections, and indicated when it reached each intersection. The software flow chart for straight mode can be seen in Figure 39. The program started up and initialized the microcomputer, the LCD, and the sensors. Then the program entered the main loop. From the loop, it continuously sent signals to the two ultrasonic sensors to transmit an acoustic pulse. Meanwhile, it received an external interrupt (external interrupt 2) indicating it was time to sample one of the two sensors. After it sampled the analog signals from the ultrasonic sensors, it compared the data to determine the situation of the sensors. Depending on the situation, the vehicle would turn slightly left, turn slightly right, or enter intersection/lost edge mode. The program also received an external interrupt (external interrupt 3) from the Hall Effect sensor, and each time this interrupt occurred, the distance count was incremented.

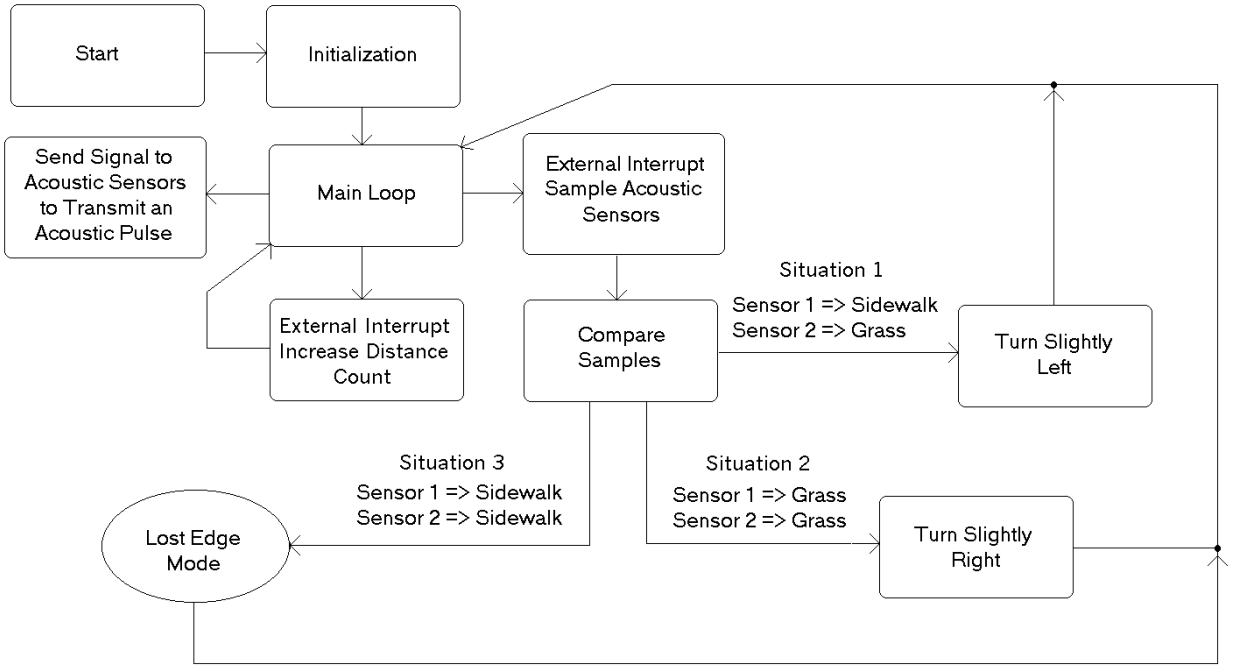

Figure 39 – Straight Mode software flow chart

Determine How to Get There – Lost Edge/Intersection Mode

Our last module involved controlling the vehicle when both sensors were reading concrete. A flowchart for its operation is shown in Figure 40. Once called from the straight mode, we first needed to determine if the vehicle was at an intersection.

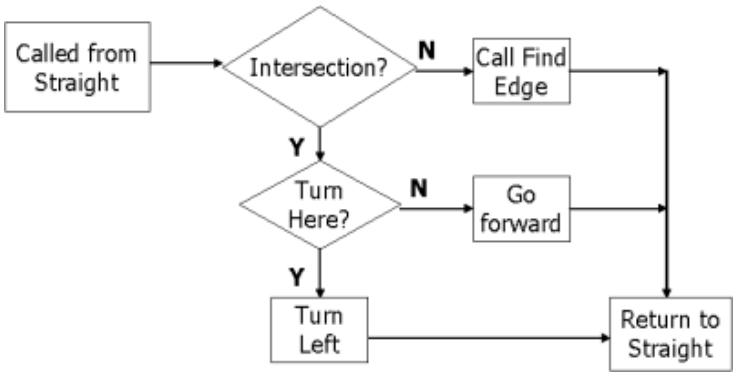

Figure 40 – Lost edge flowchart

Each intersection in the quad map had 2 bytes of information stored in the microcomputer's memory. One byte, shown in Figure 41, stored the direction of travel at an intersection (bits 0 – bit 2) and the desired distance the vehicle should travel to this intersection (bit 3 – bit 7).

Figure 41 – Intersection information byte (L = left, S = straight, R = right)

First, we compared the desired distance to the actual distance measured by the Hall Effect sensors. If these two values were not within a specified tolerance, then the vehicle was not at an intersection, and we needed to return to straight mode and continue to find the grass edge. If these two values were within a specified tolerance, the vehicle was at an intersection, and it needed to determine which direction to should travel. Referring to Figure 41, if bit 1 was set the vehicle should go forward and find the grass edge again.

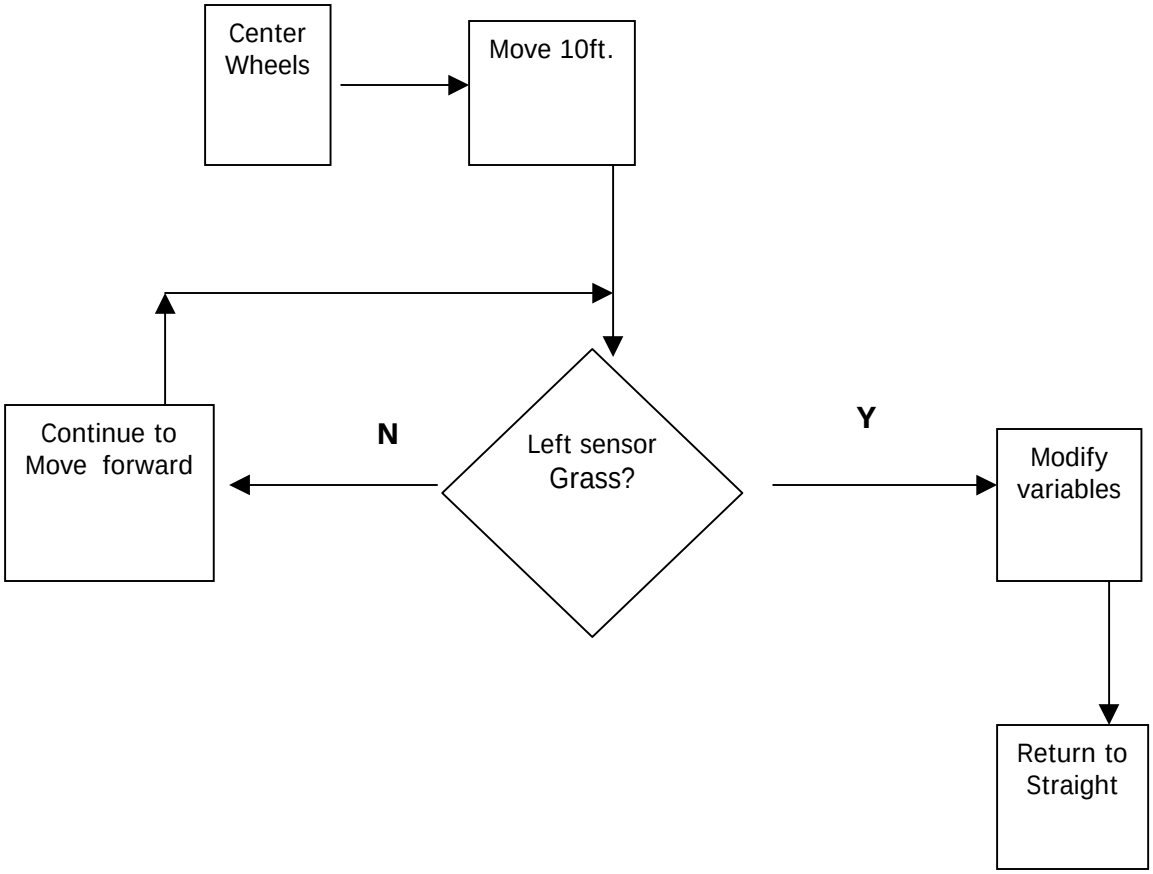

Figure 42 – Go Forward flowchart

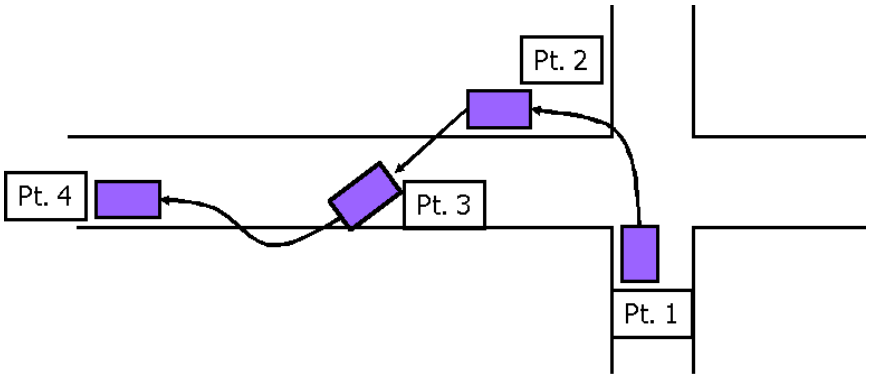

Figure 43 – Model of Vehicle Turn

The flowchart for this operation is shown in Figure 42. If bit 2 was set, the vehicle would turn left, which is illustrated in Figure 43. Referring to Figure 43, at point one the vehicle's wheels were turned full-left and the compass was continuously polled. The actual compass heading was compared to the desired compass (stored in second information byte) heading until they were equal. The vehicle would then be at point 2. From point 2, the

sensors were polled until both the sensors were reading concrete. Then the wheels would be adjusted to center and the vehicle would continue to travel forward until grass was detected under one or both of the sensors, point 3. Next, the wheels were turned slightly right and the compass was polled until the desired and actual headings were equal, point 4. At this point the vehicle was ready to return to straight mode. The code was written for this module, but we did not have enough time to test in the field.

CONSTRUCTION AND TESTING

Construction

Our next step involved integrating all the individual hardware components onto the vehicle, using a combination of breadboards and wire wrap circuits. It was during this stage that we discovered a problem with the linear actuator design. When we connected the circuit as designed, the actuator did not operate. During troubleshooting we discovered that the h-bridge was not switching as desired. The problem was determined to be a floating ground between the microcomputer and its drive circuit. A temporary solution was to connect the 5 V and 12 V GND wires together. Another problem we encountered involved the initial state of the microcomputer. Upon power up or reset all of the microcomputer's I/Os were set high, which would drive the motors until the software was initialized. As a temporary fix we disconnected the PWM input to the optical isolators until the microcomputer was initialized.

Testing

Initial system testing occurred indoors with the vehicle elevated off the floor. This allowed stationary testing of the overall system. First, we confirmed that the microcomputer was controlling all the components as expected: drive motors, control wheel position, read compass, and trigger and read ultrasonic sensors. Next, we tested the operation of straight mode. During this testing we discovered that noise from the h-bridge was causing false readings of the return analog signal from the ultrasonic sensors: low amplitude and incorrect sampling time. As a solution we used a second battery to supply the sensor’s power, which corrected the amplitude of the signal. Then, we ignored the external interrupt from the sensors while the h-bridge was operating. Once this mode was operating successfully in lab we took the vehicle out to the quad for field trials. After some software integration problems were solved, the

vehicle was able to follow the edge of the sidewalk. The vehicle control was not as smooth as we had hoped. The vehicle would travel off of the edge of the concrete and into the grass, which hindered the steering and required more power to drive the motors. This at times caused a large amount of overshoot. There are two solutions that should be implemented to solve this problem. One is to have the sensors mounted farther away from the vehicle. This would minimize the travel off of the concrete. The second would be to implement in software a proportional control, which would count the frequency from the shaft encoder. If the count was less than a specified amount, then the duty cycle to the motors would increase and vice versa. Our next testing goal was to get the vehicle to determine intersections and then turn. During this phase, we discovered that once again noise from the linear actuator was causing false readings. This time we were getting false Hall Effect sensor readings, which distorted our distance measurements. Accounting for this we allowed a 5 foot tolerance in software when checking the actual with the desired distance. Implementing this solution allowed us to find the first three intersections. We attained this result on the last day of lab, so we were unable to test the turn mode software.

FUTURE TASKS

- Isolate Linear Actuator circuit and use two batteries, one for motor and linear actuator and the other for all other components. This will eliminate the noise problems discussed.
- Extend ultrasonic sensors left to keep the vehicle out of the grass.
- Add obstacle detection, so the vehicle will stop if someone walks in front of it. This could be implemented using an ultrasonic sensor.
- Expand the quad map to include a larger variety of intersections.

ACCOMPLISHMENTS

Overall our project was successful. We designed and implemented all the electrical hardware necessary to control the vehicle's mechanical hardware and the ultrasonic sensors. Using the ultrasonic sensors, the shaft encoder, and the digital compass, we were able to determine the vehicle's relative position. We were able to control the vehicle through straight mode and lost edge/intersection mode via software modules. In the end, the vehicle was able to follow the edge of the sidewalk and detect the intersections.

APPENDIX A – ULTRASONIC SENSOR CIRCUIT DESIGN

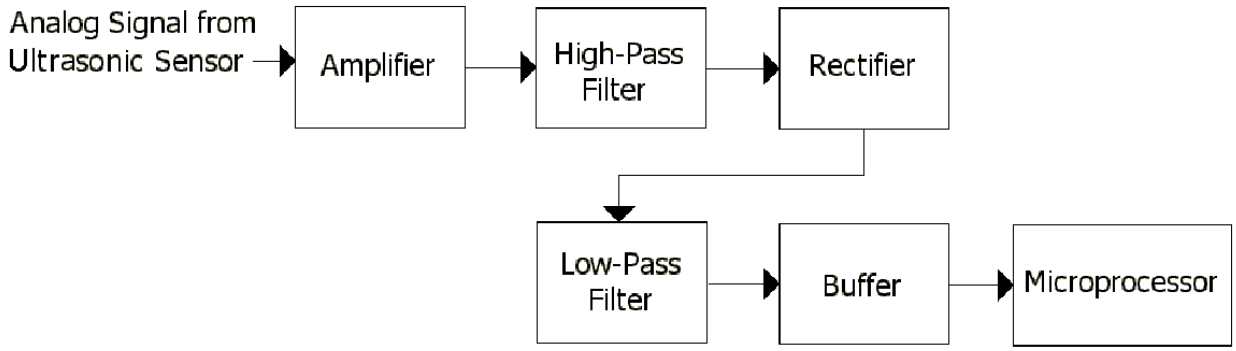

Figure A1 – Block Diagram of ultrasonic sensor analog signal amplifier and filter

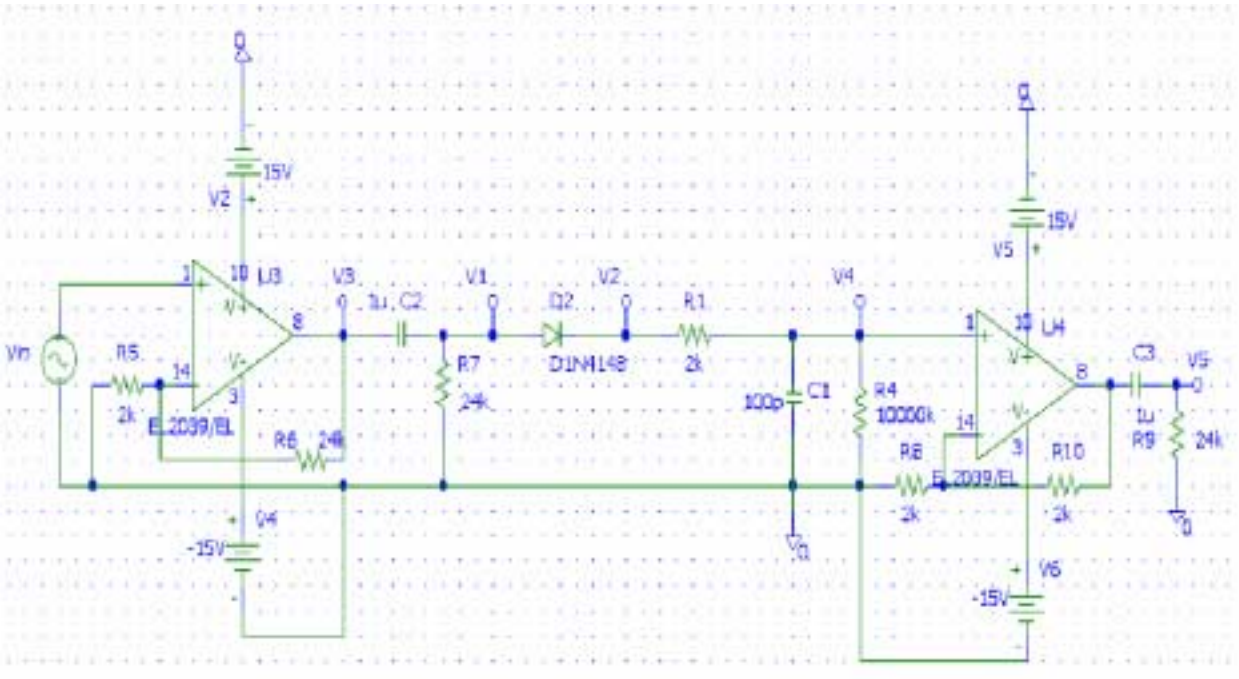

Figure A2 – PSPICE Schematic of ultrasonic sensor analog signal amplifier and filter

Note: PSPICE did not have a model for the actual op-amps used in this design. Instead of 2039 op-amps, we actually used TL3472 op-amps. The TL3472 op-amp was uni-polar, so the actual supply voltage used was 0 to +12 V, not -15 V to +15 V.

Non-Inverting Amplifier Gain = 24/2 + 1 = 13

There needed to be a buffer stage before the signal was sent to the microcomputer, and there was a 0.7 V loss in the signal from the diode. To "make-up" for the 0.7 V loss in voltage, we actually designed the buffer stage to be a non-inverting amplifier with a gain = 2/2 + 1 = 2. The real need for this amplifier was not for the gain, but to prevent any reflection problems before the signal was sent to the microcomputer.

High-pass and low-pass filter cut-off frequency equation:

$$\omega_c = 1/(R*C)$$

First, we implemented the design using a 1 nF capacitor for C1 seen in Figure A2. The simulation result in response to a 50 kHz sine wave input can be seen below in Figure A3. As can be seen in this figure, the output is fairly smooth. However, when this was tested experimentally, we noticed that the capacitor C1 was not discharging fast enough, so the output result was getting distorted. To correct this, we replaced the 1 nF capacitor for C1 with a 100 pF capacitor, which can be seen in Figure A2. The final PSPICE simulation results can be seen in Figure A4.

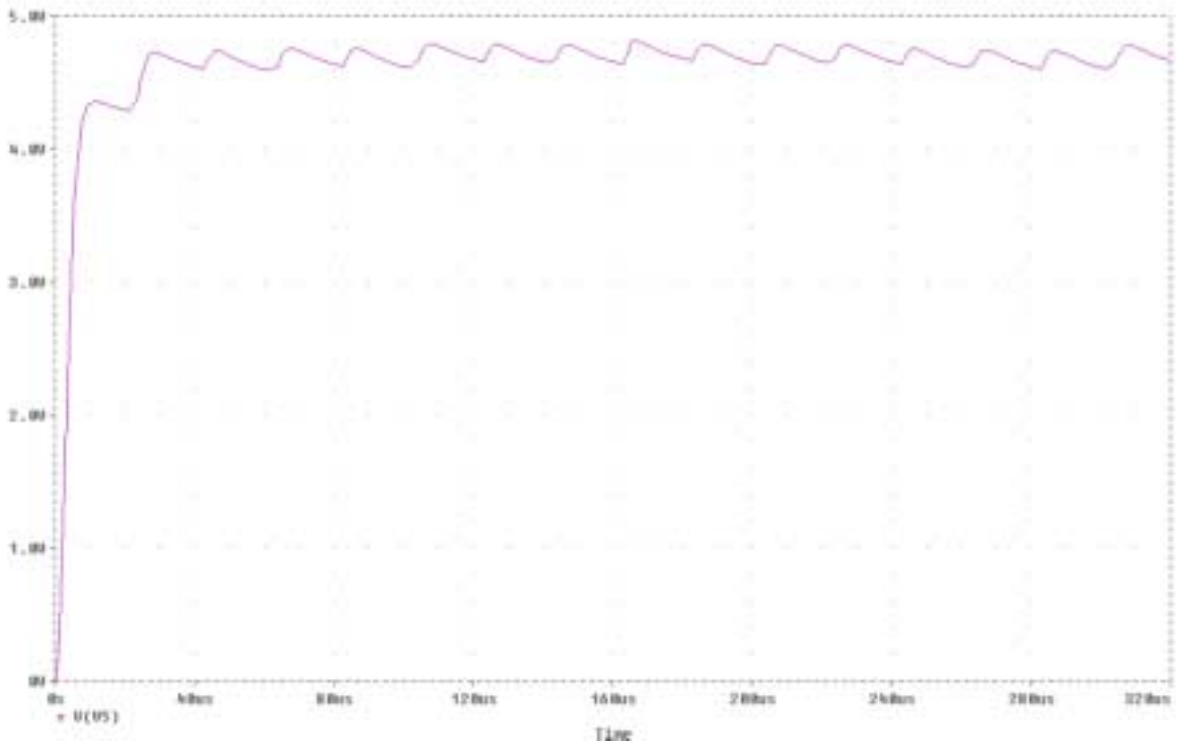

Figure A3 – PSPICE simulation result output from circuit seen in Figure A2 where C1 = 1 nF

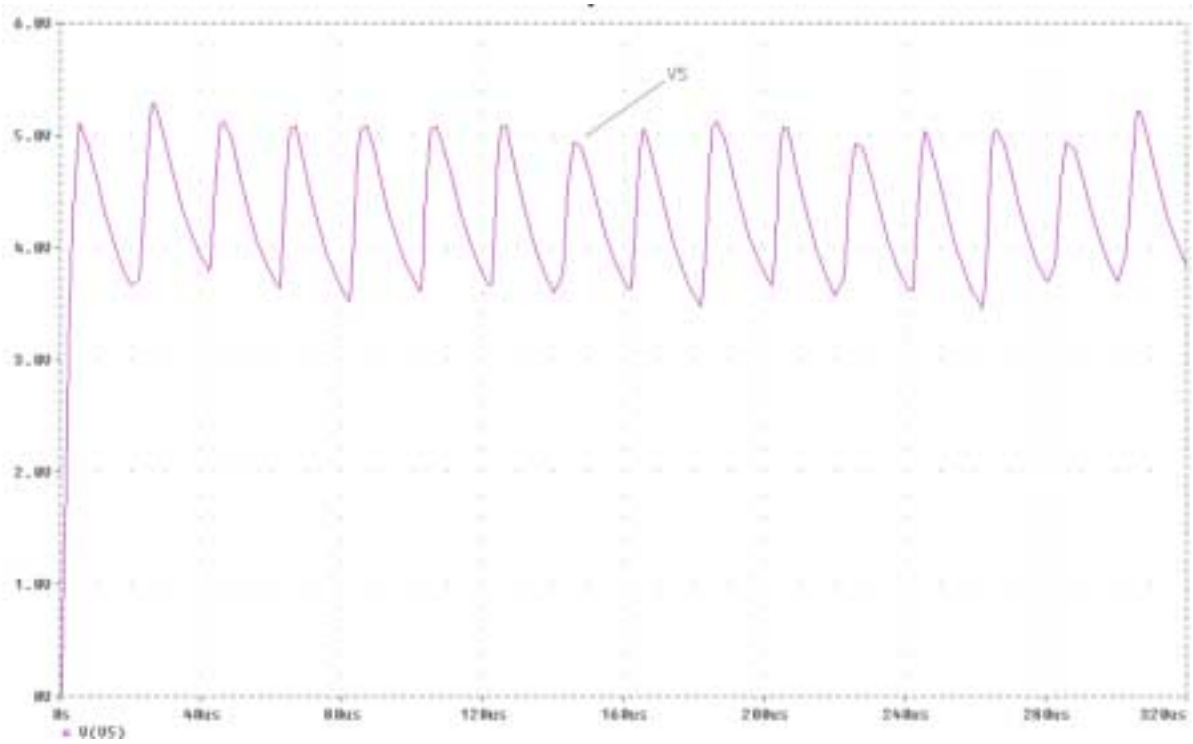

Figure A4 – PSPICE simulation result output from circuit seen in Figure A2 where C1 = 100 pF

This circuit worked very well. It converted the analog signal from the ultrasonic sensor into a signal that the microcomputer could sample easily. The final experimental results for this circuit can be seen in Figure A5. The output from the circuit was the filtered analog output, and the input to the circuit was the analog output from the ultrasonic sensor. We sent the filtered analog output from this circuit to the A/D converter on the microcomputer.

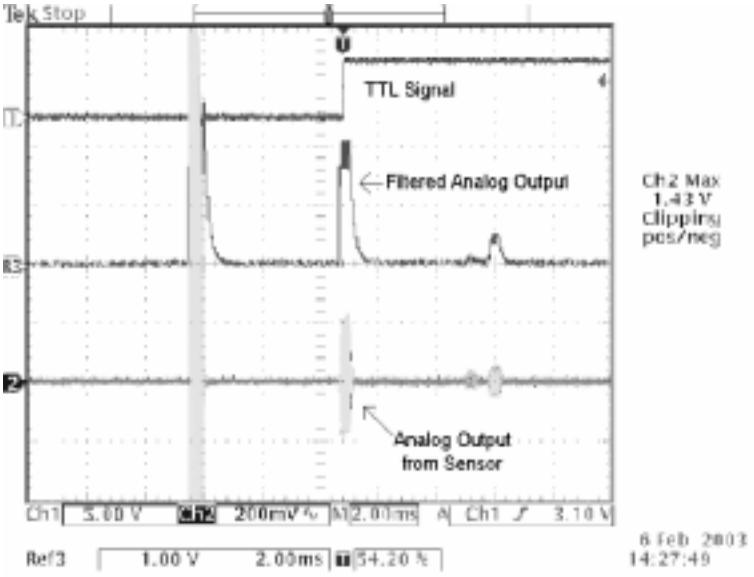

Figure A5 – Filtered analog output vs. analog output from ultrasonic sensor

APPENDIX B – RELEVANT PATENTS

We did some research before starting our project and here are some of the relevant patents that we found:

- System and method for causing an autonomous vehicle to track a path
    - U.S. Patent 5,657,226 August 12,1997
    - Integrated vehicle positioning and navigation
        - Combination of apparatus and methods
        - Included GPS and an IRU (inertial reference unit)
        - Position calculations and vehicle control with obstacle detection
- Autonomous vehicle arrangement and method for controlling an autonomous vehicle
    - U.S. Patent 6,151,539 November 21,2000
    - System includes
        - Input of travel orders
        - Digital map
        - Path generating unit
        - Laser and radar sensors for detecting obstacles and condition of path
- Autonomous vehicle capable of traveling/stopping in parallel to wall
    - U.S Patent 6,038,501 March 14,2000
    - Wheel encoder for distance and speed
    - Gyro for detecting direction
- System and a method for enabling a vehicle to track a present path
    - U.S. Patent 5,838,562 November 17,1998
    - GPS & IRU
    - Remote Control
- System and method for enabling an autonomous vehicle to track a desired path
    - U.S. Patent 5,684,696 November 4, 1997
    - Plans a continuous path and return path if vehicle deviates from desired path
    - GPS & IRU