Maze Racer

Team 4

Jeff Ostrander (Team Leader)
David ‘Isaac’ Waters
James Bates
Beenu Pandey

ECEN 4023
Spring 2005
Oklahoma State University

Abstract

Our team designed an autonomous line following car that can read symbols, make decisions based on them, and complete a thirty to fifty foot track in less than one minute. Each team member, through hard work and good engineering, conquered the areas of computer modeling, controller coding, mechanical design, and electrical design to bring everything together and complete every given specification. The finished car can read a left turn, right turn, eject, or stop symbol and take appropriate action. It can also complete our forty-three foot track in thirty to thirty five seconds. We finished our project well within our budget of $400.

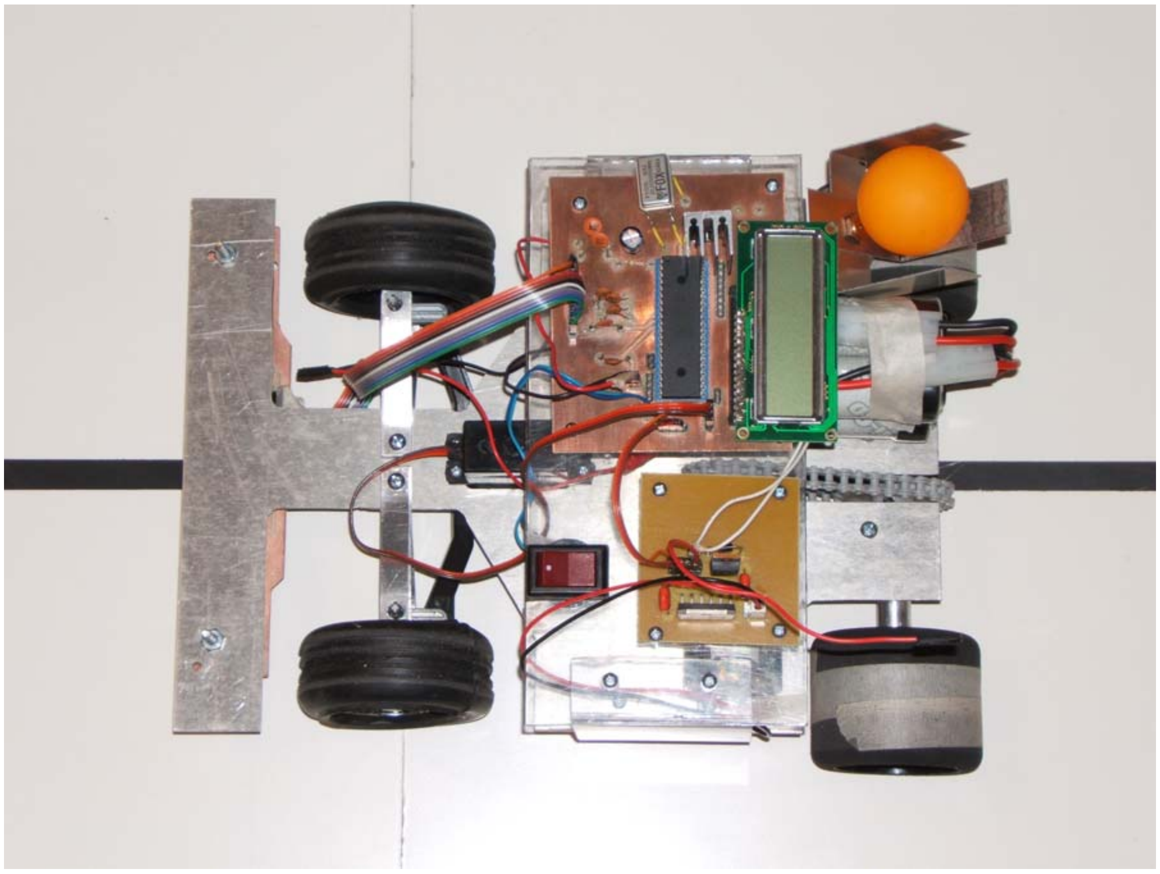


Table of Contents

Abstract i
Table of Contents ii
Table of Figures iii
Introduction 1
Project Specifications 2
Motor and Steering- Beenu Pandey 3
Reflective Object Sensors—James Bates 9
Track—James Bates 14
Ping Pong Ball Ejection—James Bates 16
Manufacturing—Jeff Ostrander 20
Modeling—Jeff Ostrander 28
Programming—David 'Isaac' Waters 35
Programming Results—David 'Isaac' Waters 43
Conclusion 45
References 46
Appendix A – Track Symbols 47
Appendix B – Miscellaneous Figures 49
Appendix C – Program Subfunctions 52
Appendix D – Itemized Cost 62
Appendix E – Advisor Meeting Minutes 63
Appendix F – Status Reports 66
Appendix G – Complete Code .... On Cd

Table of Figures

Table 1.1 - Difference between NiCd and NiMH........ 3
Figure 1.1- Pulse widths for servo motor [1]........ 4
Figure 1.1 - Pin Out for Servo Motor [1]........ 4
Table 1.2 - Characteristics of DC Motor [2]........ 5
Figure 1.2 - Test circuit for H-bridge [3]........ 6
Table 1.3 - Pin Description of LMD18200 [3]........ 7
Figure 1.3 - H-Bridge Schematic [3]........ 8
Figure 2.1-Internal schematic of a reflective object sensor [5]........ 10
Table 2.1-Sensor reading 3mm from surface........ 11
Figure 2.2 -Sensor layouts on printed circuit board........ 12
Figure 2.3-Sensor circuit board mount........ 13
Figure 2.4-Final track layout........ 16
Figure 2.5-Solenoid dimension diagram [6]........ 17
Figure 2.6-Solenoid circuit schematic........ 18
Figure 2.7-Final ping-pong ejection system........ 19
Figure 3.1 The final base cutout........ 20
Figure 3.2 The complete motor and axle setup........ 21
Figure 3.3 The two halves of the axle........ 22
Figure 3.4 The wheel mount........ 22
Figure 3.5 The wheels........ 22
Figure 3.6 The completed axle........ 23
Figure 3.7 The mounts used to attach the axle to the base........ 23
Figure 3.8 The front wheel........ 24
Figure 3.9 The front wheels and steering bar........ 24
Figure 3.10 The front view of the Indy Car style mounts........ 24
Figure 3.11 The mounting orientation of the servo........ 25
Figure 3.12 The complete steering system........ 25
Figure 3.13 The electronics platform........ 26
Figure 3.14 The complete design of FIO........ 27
Figure 3.15 A Simulink model of the kinematics equations........ 29
Figure 3.16 A Simulink model of the DC motor........ 30
Figure 3.17 A Simulink model of the servo........ 30
Figure 3.18 A Simulink model of the final system........ 30
Figure 3.19 The step response of the control in Sisotool........ 31
Figure 3.20 The final Simulink model, including the control........ 32
Figure 3.21 The Simulink model of the control........ 32
Figure 3.22 The turning response, psi, of a one-inch error........ 33
Figure 3.23 The response of the control to initial condition of one inch........ 33
Table 3.1 A comparison of simulated results vs. actual results........ 33
Figure 4.3 Symbol reading over time........ 38
Table 4.1 Criteria used to indentify symbols........ 38
Figure 4.4 Line sensing at various errors........ 40
Figure B.I Prototype sensor mount........ 49
Figure B.II Rear drive and steering mechanism........ 49

Figure B.III Steering Mechanisms 50
Figure B.IV Top view of final design 50
Figure B.V Pin-out diagram for the PIC18F452 [10] 51

Introduction

At the beginning of the semester, our group (Team 4) was given the task of building an autonomous, line following vehicle. First, we assigned each team member with a particular set of tasks. Beenu was in charge of the motor drive and steering. James was assigned the task of obtaining sensors and finding a way to eject the ping-pong ball. Jeff worked on simulating our project, and then he took over mechanical design. Isaac handled all of the software and code areas. We entered a planning stage that lasted approximately six weeks, in which we conducted research and preliminary design. After our planning, we all got to work on our respective parts. Several weeks later, we completed our project and presented the finished product to Dr. Yarlagadda and Dr. Chung.

We believe that the thinking and technology behind this line following robot could benefit many people in the future. Applications include: self-driving personal cars, toys for children, and automated exploration robots.

Project Specifications

- The racer will be able to follow a 30 to 50 foot track.
- The track will consist of a ½ inch wide black line on a hard white background.
- The racer will be able to read several shapes, which will appear at least 12” away from any turn junction. (See Appendix A)
- The racer will fit inside a 12” by 12” box with no parts overhanging
- The racer will complete the course within 60 seconds.
- The racer will carry a ping-pong ball and then eject it within 24” of reading the proper symbol.
- The racer will be able to navigate turns with radius of at least 2 feet.
- The racer will run off of two 7.2V batteries.
- The racer will be able to run continuously for at least 20 minutes on one charge.
- Recharge time is between 5 hours.
- If the racer does not detect the track, shut down will occur.

Motor and Steering- Beenu Pandey

Batteries

Since one of the specifications of our project was for the robot to be an autonomous vehicle, we chose batteries as the main source of power, since they are light and portable. Upon research, we discovered two popular types of batteries, Nickel Metal Hydride (NiMH) and Nickel Cadmium (NiCd). Some of the factors that were considered in determining the type of battery to use were: cost, recharge time and battery life (Table 1.1).

Table 1.1 - Difference between NiCd and NiMH

	NiCd	**NiMH**
Battery Life	Less	More
Recharge time	Less	More
Cost	Less	More

After taking these factors into consideration, we decided to use the NiCd batteries. Fortunately we found two reasonably priced 7.2 V batteries that had a life of 2000mAH and could supply 14.4V to the board. This also met the requirement of a minimum of 12V for the motor control circuit and solenoid.

Steering

Front wheel steering for our robot was achieved using a standard 5V servo motor and a steering rod. The servo had a pin connected to the microcontroller that made the rod turn at specific pulse widths. The angle that the servo turned is communicated using pulse widths (Figure 1.1).

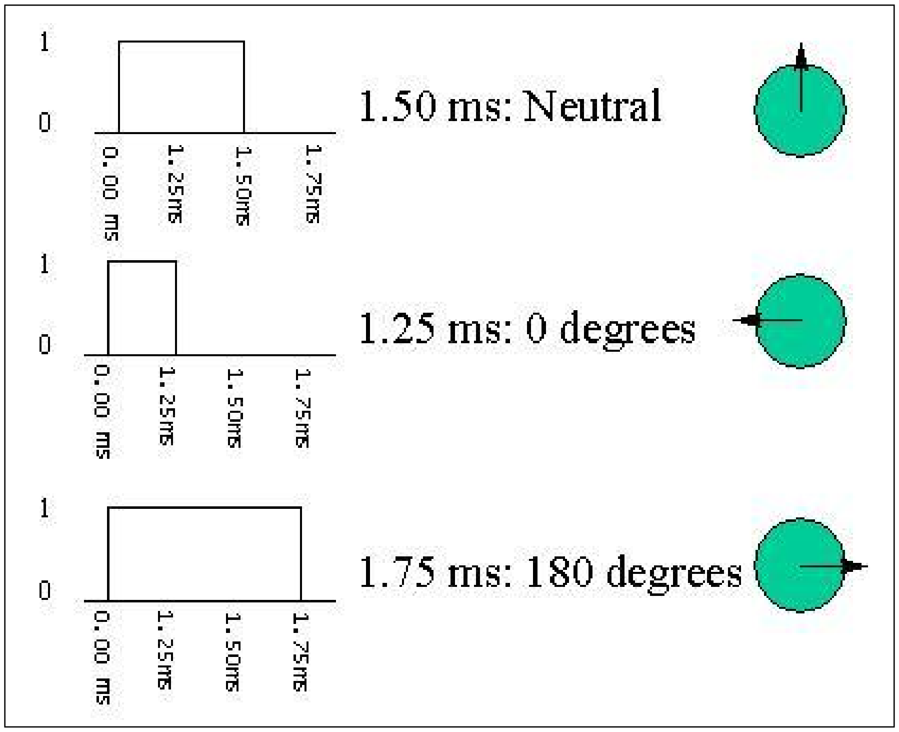


Figure 1.1- Pulse widths for servo motor [1].

As seen in the diagram above, the duration of the pulse width determines the angle of the output shaft. The figure below shows the pin out of the servo motor used.

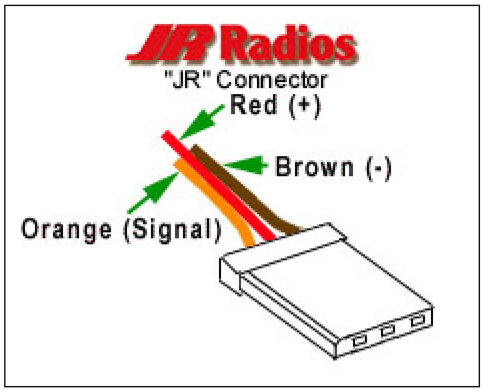


Figure 1.1 - Pin Out for Servo Motor [1]

Motor

In order to achieve rear wheel drive we decided to use a DC motor and a gear and chain assembly. However, before picking a motor, several factors were taken into consideration.

- Speed
- Flexibility of motion (forwards/backwards)
- Motor torque

After establishing these factors, we researched the motors available. Some of the popular ones were servos and stepper motors. Servos were eliminated because they did not rotate continuously. Stepper motors were discarded because they move in discrete steps and have no open loop feedback. Taking these factors into consideration, we decided to use a DC permanent motor. The DC motor had many useful characteristics (Table 1.2).

Table 1.2 - Characteristics of DC Motor [2]

	6V	**12V**	**24V**
No-load Final output Speed	250rpm	500rpm	1000rpm
Amps @ no load	0.2 A	0.5 A	0.7 A
Stall amps	2.1 A	4.7 A	6.0 A
Stall torque	61.89 oz-in	105.73 oz-in	132.73 oz-in
Gear Ratio	10:1		
Weight	7.76 oz (0.485 lb)		

Motor Control

The H-bridge was used to control the speed of the motor. The H-bridge used was from the National Semiconductor Company. It was rated at 12V, 3A. Our robot was pulling less than an amp of current continuously and more than 400mA when the solenoid was being used, so the H-bridge was adequate. We used a circuit to test the H-bridge (Figure 1.3).

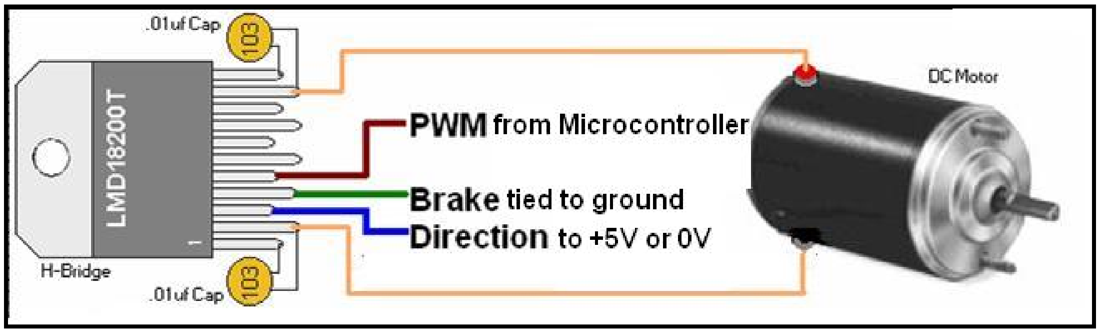


Figure 1.2 - Test circuit for H-bridge [3]

The table below describes the pins used to control the motor.

Table 1.3 - Pin Description of LMD18200 [3]

Pin #	**Pin Function**
Pin 1, BOOTSTRAP 1 Input	Bootstrap capacitor pin for half H Bridge 1. External 10nF capacitors connected from the outputs to the bootstrap pins of each high-side switch provide typically less than 100ns rise times allowing switching frequencies up to 500kHz.
Pin 2, OUTPUT 1	Half H-Bridge number 1 output. This pin is connected to the motor
Pin 3, DIRECTION Input:	This input controls the direction of the current flow between OUTPUT 1 and OUTPUT 2 (pins 2 and 10) and, therefore, the direction of rotation of a motor load. It can be connected to either to +5V or 0V for clockwise or anticlockwise rotation.
PIN 4, BRAKE, Input	This input is used to brake a motor by effectively shorting its terminals. When braking is desired, this input is taken to a logic high level and it is also necessary to apply logic high to PWM input, pin 5.
PIN 5, PWM Input	This pin accepts the PWM signal generated by the microcontroller.
PIN 6	*Power Supply*
PIN 7, GROUND Connection	This pin is the ground return and is internally connected to the mounting tab. This pin was not used.
PIN 8, CURRENT SENSE Output	This pin provides the sourcing current sensing output signal. For this application this pin was not used.
PIN 9, THERMAL FLAG Output	This pin provides the thermal warning flag output signal.
PIN 10, OUTPUT 2	Half H-Bridge number 2 output. This pin is connected to the motor
PIN 11, BOOTSTRAP 2 Input	Bootstrap capacitor pin for Half H-Bridge number 2.

The H-bridge uses pulse width modulation (PWM) to control the speed of the motor. Varying the voltage can change the speed of the motor, however this is not a very efficient method of motor control.

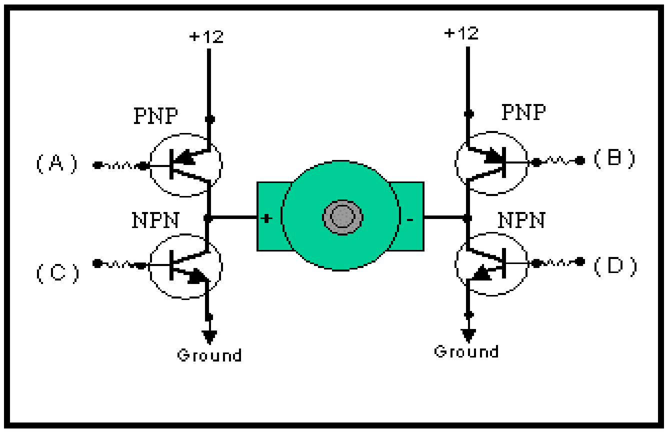


Figure 1.3 - H-Bridge Schematic [3]

H-bridges come in several pre-packaged chips that make motor control a less complicated task. The one used for this project is from National Semiconductor called LMD18200 [3]. The LMD18200 is a 3A H-bridge. It can handle up to 6A of peak output current. Its supply voltage ranges from +12V to +55V. The H-bridge only requires two inputs, a digital input for setting the direction and a pulse-modulated input for controlling speed. Each of the four switches in the LMD18200 has a built in protection diode that clamps any transient voltages exceeding the positive supply or ground to a safe diode voltage drop across the switch [4].

Reflective Object Sensors—James Bates

I began the search for the sensors with the knowledge that they would need to be able to tell the difference between black and white, sense a surface that was not very far away, and not be affected too much by ambient light. I looked on the internet and found photoresistors and emitter-detector pairs that would meet all of these needs. A CCD camera was also an option but the option would not have been cost effective, and CCD cameras are very memory intensive. A photoresistor changes its resistance based on the amount of reflection it senses, however you need another component, such as an LED, to provide the light source. The emitter-detector pairs contained an emitting diode and an NPN phototransistor within one casing. Therefore, it met all of our requirements for this project.

The actual component we used was the OPB742 Reflective Object Sensor (Figure 2.1). It had an infrared emitting diode paired with an NPN phototransistor in a plastic casing, and they were angled toward the center of the component. They were set so that the optimal focal length was three to four millimeters away from whatever reflective surface was to be used. The fact that the diode emitted infrared light made the sensor less susceptible to ambient light. The sensor phototransistor would measure the reflectivity of the surface in front of it and output a voltage according to the measurement. When the sensor was over a black, or less reflective, surface it resulted in a lower voltage than it did when it was over a white, or more reflective, surface. Because the optimal distance from the sensor to the surface was three to four millimeters, sensors could be placed on the front of the car between its base and the track. The only other components that the sensors needed to make them functional were two resistors, which made the sensor circuit very simple and clean. Another concern for the different parts that would be going on the car was how much current they pulled. We needed the batteries to last as long as possible and to supply enough total current for the entire car. The sensors I chose only pulled thirty-five milliamps with the resistors attached and they fell within budgetary constraints.

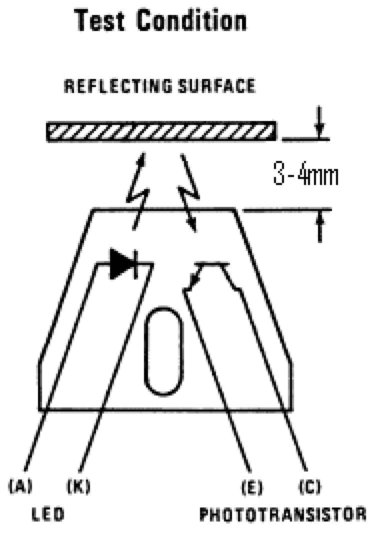


Figure 2.1-Internal schematic of a reflective object sensor [5]

I tested each of the sensors in pairs by setting them up on a breadboard next to each other, as they would be on the PCB and attaching the needed resistors. I found a piece of the track that was used the previous year for the same project and held it above the sensors, moving it side to side so that the sensors were sometimes under the black line and sometimes under the white background. I used a ruler to measure the distance between the track and the sensors and took the output voltage readings from the sensors using the DMM. Making sure that the current being pulled was correct, I took various readings (Table 2.1). Beginning at the center of the black line on the track, I made a mark every one sixteenth of an inch moving away from the center of the line until the marks were completely in the white section of the track. I placed the marked track over the sensors, took a voltage reading at the center of the black line, moved the track to the next mark, and then took another reading until the sensor was completely over the white surface.

Table 2.1-Sensor reading 3mm from surface

Distance from Center	Avg. Values
0"	115mV
1/16"	115mV
2/16"	135mV
3/16"	145mV
4/16"	520mV
5/16"	1.2V
6/16"	1.25V

The readings were taken again with the lights in the lab turned off and the voltages measured did not differ from those measured with the lights on. I therefore concluded that the ambient light did not have much affect on the sensor readings. The readings showed that the range of the sensors was about a decade which meant that there was easily enough of a difference between the low and high readings for the microcontroller to determine whether the sensors were over the black line or if they were on the white section. The readings also had a grayscale, not jumping from low to high instantaneously, which meant that we could send the outputs through the analog to digital pins of the microcontroller and obtain greater resolution. I made a vice from two pieces of wood and bolts that allowed us to place the sensors in different configurations and attach them to the car for testing (Appendix B-I).

We determined from testing that the best configuration of the sensors would be to use seven sensors in one row (Figure 2.2)

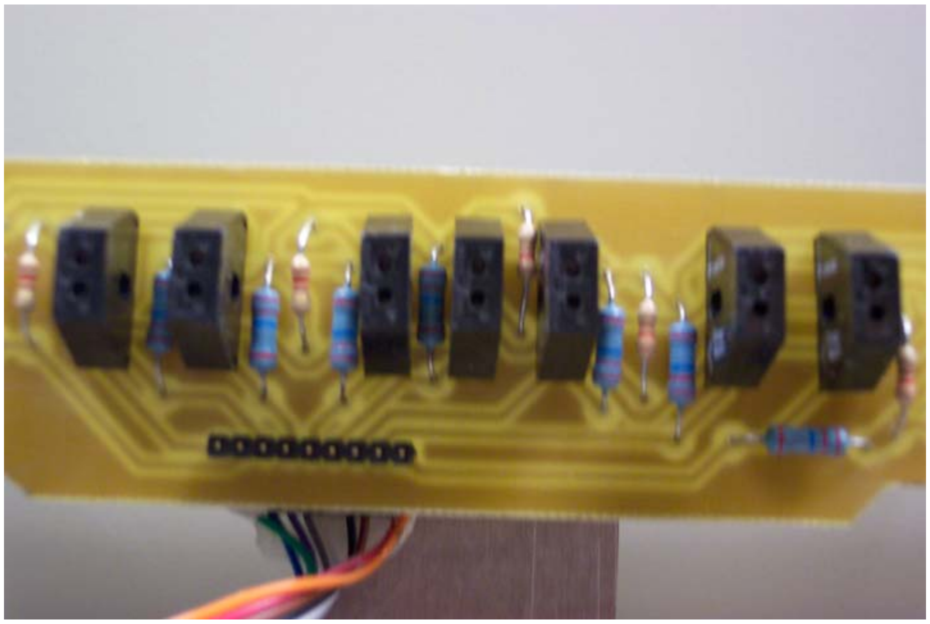


Figure 2.2 -Sensor layouts on printed circuit board

The center sensor was aligned with the center of the car and the two on either side of it were half an inch away, center to center. The next sensor on each side of the middle three were placed three quarters of an inch away from the outside sensors of the middle three. Then the last sensor on each end was half an inch away, center to center, from the one beside it. The middle three sensors were separated because they were designated as the primary line following sensors. The outer four sensors were used for symbol sensing and for detecting a fork in the track. With this setup, the total length of the sensor array is three and a half inches long. The symbols that needed to be read were four inches wide, so all of the sensors read black when the car went over a symbol. The line that had to be followed was a half-inch wide and the middle three sensors had a half-inch between each of them, so there would always be a sensor on the line. I used Allegro PCB Design and Cadence to design and mill the PCB for the sensors, and then I soldered the sensors and resistors on the board. I put nine headers on the back of the sensor circuit board so that a cable could be run from the front of the car to the back where the microcontroller circuit board was mounted. The headers were for power, ground, and the seven output signals from the sensors. I drilled two holes in the PCB and the front wing of our Indy car body and attached the sensor board to the body with screws and nuts so that the height of the board would be adjustable (Figure 2.3). This allowed us to get the sensor board at exactly the right height above the track surface.

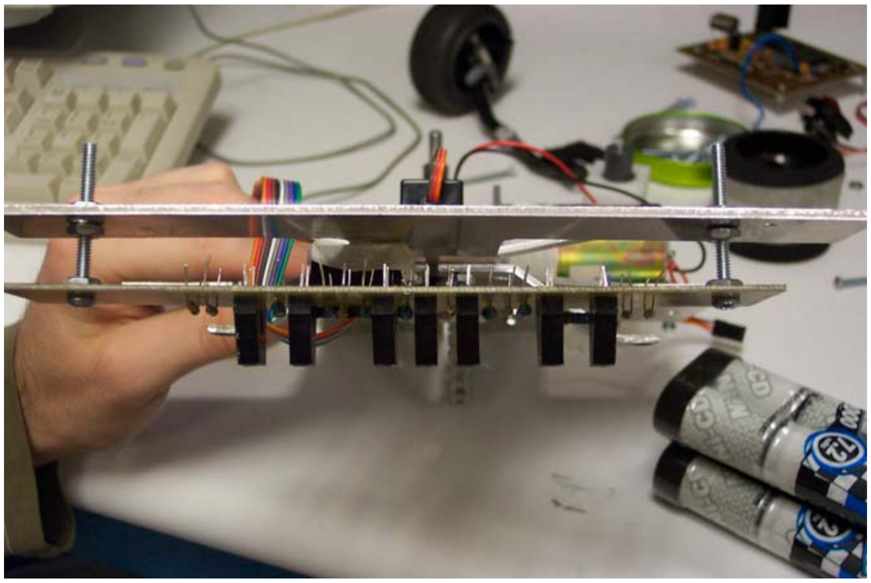


Figure 2.3-Sensor circuit board mount

The sensors worked very well and none of them ever went bad or stopped working. The sensors proved to be very sturdy, surviving many crashes into walls and other objects. Despite these blows the sensors performed well the entire time and were able to do everything needed to complete the project.

Track—James Bates

We needed to make a track that would be thirty to fifty feet in length when put together, and it had to have a black line painted on a white background. Last semester, the team doing the same project built a track made from sections of cardboard which were then starting to warp. We decided that we wanted something sturdier than cardboard and as flat as possible so that we could obtain the best reading with the sensors. Jeff's father, who works at Riverside Furniture, provided the wood for the track at no expense to us. Riverside Furniture also painted the wood glossy white on both sides. The wood was very rigid and flat and the glossy white was very reflective, two main characteristics that we desired for our track. Our decision to make the track out of two types of pieces also made the track very modular and we could set it up in many different patterns and transport it easily. The two-foot square pieces were for the straight sections and just needed one black line painted across the middle. The three-foot square pieces were for the turns and forks in the track, so every one of them at least had a curve painted on it and some also had a straight line through the middle. The turns needed to have a two-foot radius, which is why we ordered the three-foot square sections for the turns, so I made a template out of a long, thin piece of steel. I made a hole that could be nailed into the corner of the track piece and then two holes, one twenty three and three quarters inches away from the nail hole and one twenty four and a quarter inches away from the nail hole. When used this made a half-inch thick line centered at two feet away from the nail hole, giving us our two-foot radius. We drew straight lines on all of the two-foot square sections and curved ones on all of the three-foot square sections. I taped them off and then we covered them with a brown paper that would not let paint through. Beenu painted most of the black lines after they had been taped off and helped me in the process as well. We later painted a straight line on some of the curved pieces to make the forks in the track and we added the symbols to some of the two foot square pieces. There were four symbols that needed to be used and therefore tested. They were four inches wide and had to be centered on the black line, and the turn left and right symbols had to be placed at least two feet before a fork in the track (Appendix A). We made templates for the symbols and painted them on different pieces of the track. The track was used in the design lab for testing the car on all the different possibilities that it would encounter and many configurations were set up and used. I was put in charge of designing what the final track would look like (Figure 2.4). The only constraints were that I had to include a right and a left turn and the two other symbols as well, and it had to be anywhere from thirty to fifty feet long. So I just began to draw the pieces out on paper and tried to come up with a track that would meet the specifications and would also be somewhat compact, because we had to fit it into a room in the ATRC. I made the track so that if the car read the first turn symbol incorrectly it would enter into a loop and would continue in that loop until it read it correctly. If the car read the next turn incorrectly it would go straight into a dead end and run off the track. The eject ping-pong ball symbol was placed on the track near the end and the stop symbol was placed on the last piece. The track came together very nicely and worked very well. We did have some issues with the sensors reading the crack of the junction between the pieces of track and we fixed this by taping over them with white bandage tape.

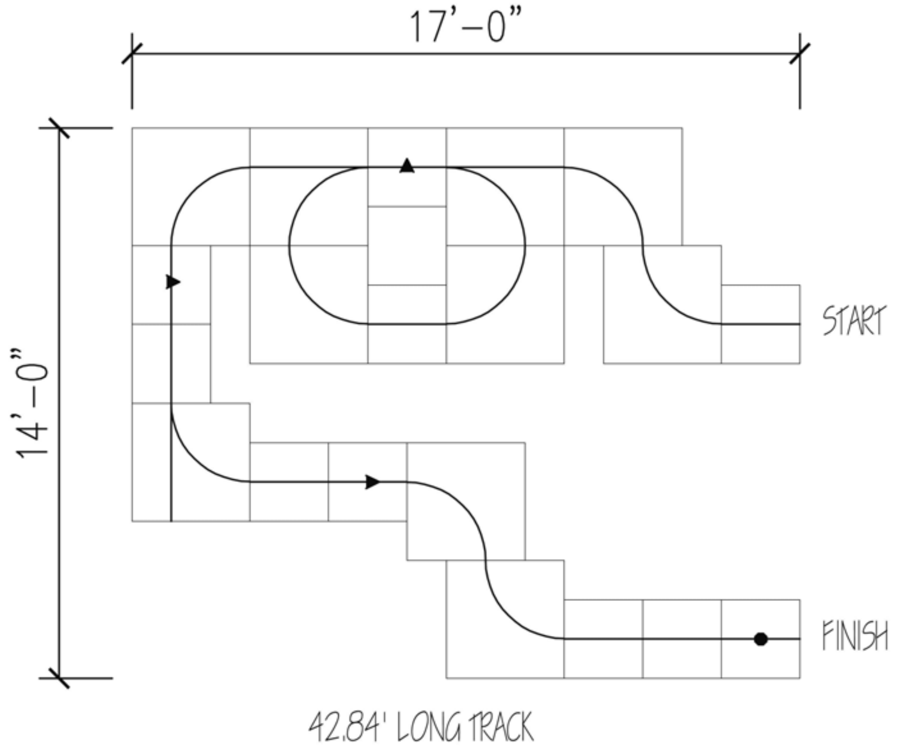


Figure 2.4-Final track layout

Ping Pong Ball Ejection—James Bates

I started looking for a component that would eject the ping-pong ball from the car at a certain point in time and that could be controlled by the microcontroller. What I came up with from the very beginning was a solenoid because I knew that it could be excited by just applying current to it. I searched until I found a small solenoid that pushed its rod outward when current was applied to it and that used the least amount of power possible.

I chose the SMT-1325S12A Tubular Push-Type Solenoid (Figure 2.5). There were many reasons that led me to choose this part. It ran off twelve volts, which is what the H-bridge on our motor drive circuit used, and the voltage could be applied continuously without a break. It pulled the least amount of current when it was turned on, three hundred and thirty milliamps. Even though it would not be on for very long, this helped save battery power. Priced at only $10.00, this option was very cost-efficient.

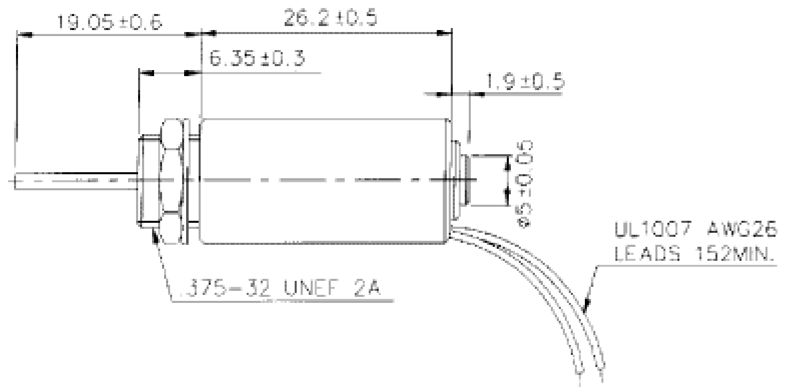


Figure 2.5-Solenoid dimension diagram [6]

The solenoid had to be controlled by the microcontroller because the sensors would read the eject symbol and let the microcontroller know to eject the ball. I needed a way to apply the twelve volts to the solenoid but not let any current through it until the microcontroller said to. The microcontroller cannot do this directly as it only outputs five volts. The answer I came up with was to use an N-Channel FET. If the solenoid was placed between the drain of the FET and twelve volts, the twelve volts would be across the solenoid but no current would flow through it as long as the gate of the FET was held low. When the microcontroller would apply five volts to the gate of the FET, the switch would be opened and current would flow from the drain to the source of the FET. This caused current to flow through the solenoid. I gathered the necessary parts and built the circuit on a breadboard and it worked perfectly, and it was pulling the correct current (Figure 2.6). I added a diode in parallel with the solenoid to prevent damage to the solenoid from voltage spikes.

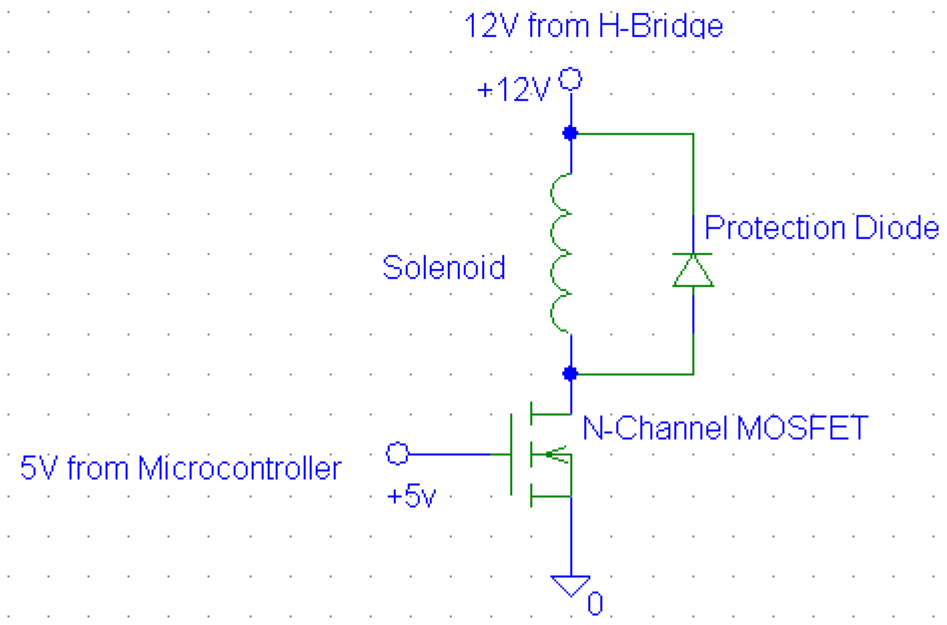


Figure 2.6-Solenoid circuit schematic

The ball had to be ejected within two feet of reading the eject symbol on the track. This was not a problem because the response time of the solenoid was between thirty and forty milliseconds. The ball needed to be carried by the car around the track without falling out so I needed something that would hold the ping-pong ball securely and directly in front of the solenoid. The end of the solenoid was treaded and had a nut on it. I used this to secure two bent parts made of Lexan to the solenoid that formed a holding area for the ping-pong ball directly in front of the solenoid's rod. This threaded area was also used to attach the solenoid to the Lexan mount that I made and attached to the base of the car. The solenoid was suspended in air so that the rod had room to extend out the back of the solenoid when it was not pulling current. I needed something to stop the rod from falling completely out of the solenoid so I measured how far I would allow the rod to be out and attached a screw there in the middle of the mount to catch the rod when it fell back after current was taken away. The solenoid worked great and there was plenty of force to push the ping pong ball out of the holder even after I had angled it forty degrees upward for better security while maneuvering the track (Figure 2.7).

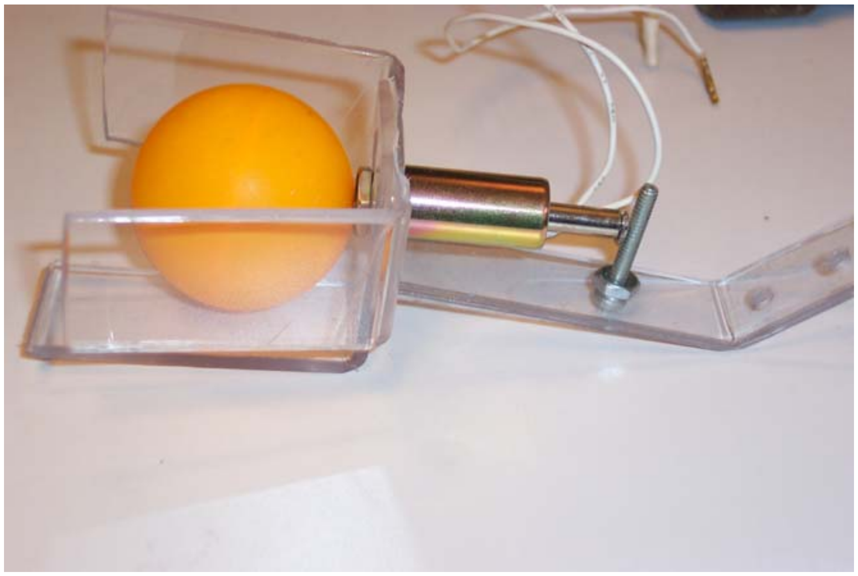


Figure 2.7-Final ping-pong ejection system

Manufacturing—Jeff Ostrander

The Base

When designing the overall look of our robot, we decided upon an Indy Car style frame. We felt that the front wing of the base would serve as a good position for mounting sensors and that the shape would provide plenty of room for other components (Figure 3.1).

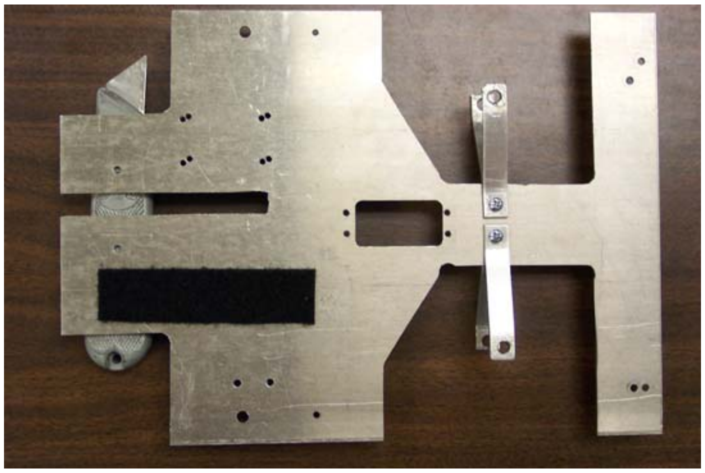


Figure 3.1 The final base cutout.

Once we completed the design of the base, I started to look at materials that we could use to manufacture it. The two materials that were readily available to us were aluminum and Lexan. We choose a 3/32" thick aluminum sheet because, in order to achieve the same rigidity of the aluminum, the Lexan had to be much thicker. After choosing the material, I started from a one-foot by one-foot square and using a band saw I roughly cut out the major sections that were to be removed. To clean up the cuts and to make them more precise, I used a belt sander to take down the rough edges.

Rear End

After the base was finished, we were able to design the rear end drive system. This was made up of the propulsion motor, gears, a chain, rear axle, and the rear wheels. I first worked with the motor by determining its best location on the base. Because the motor ran clockwise in normal operation mode, I decided to put it on the left side of the car facing the right. Next, I determined a spot where the motor would not interfere with the rear wheels and a location where the motor took up the least amount of space (Figure 3.2).

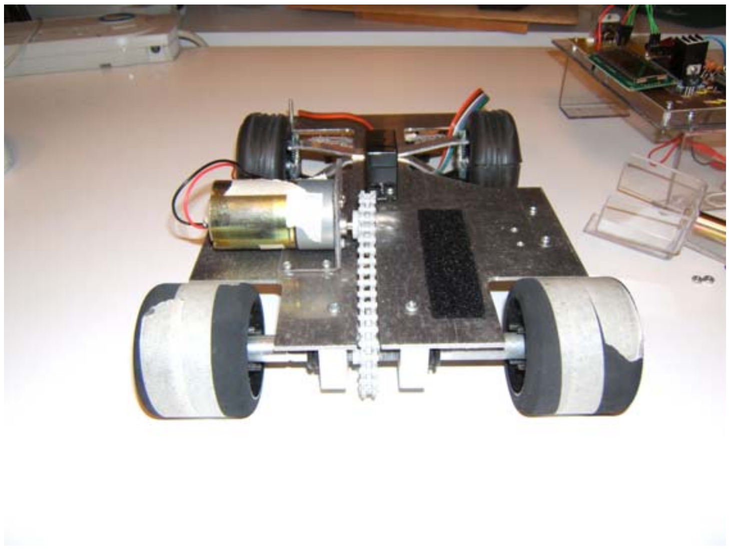


Figure 3.2 The complete motor and axle setup.

To mount the motor, I made a simple L-bracket out of Lexan that would be attached to the face of the motor and to the base of the car. Because the motor was mounted on the top of the base and the axle was going to be mounted on the bottom, I needed to cut a slot for the chain to connect the gear attached to the motor to the gear attached to the axle (Figure 3.1). To do this, I marked a slot a little wider than the chain at the position of the gear attached to the motor.

With the motor and chain in the correct position, I needed to determine what I was going to use for an axle that would work with the gears we purchased. Because we could not find a usable axle that was cheap enough or one that we could get in a reasonable amount of time, I decided to make it. Since the inner diameter of the gear was 6.5mm, I choose to use two pieces of ½" diameter aluminum rod. The ½" diameter allowed me to work with longer pieces on the lathe. The right side of the left half of the axle is taken down to 6.5mm so the gear can slide on. Then by threading the outside of narrower section along with drilling and threading a hole in the other half, I was able to screw the two shafts together to create a single solid axle (Figure 3.3).

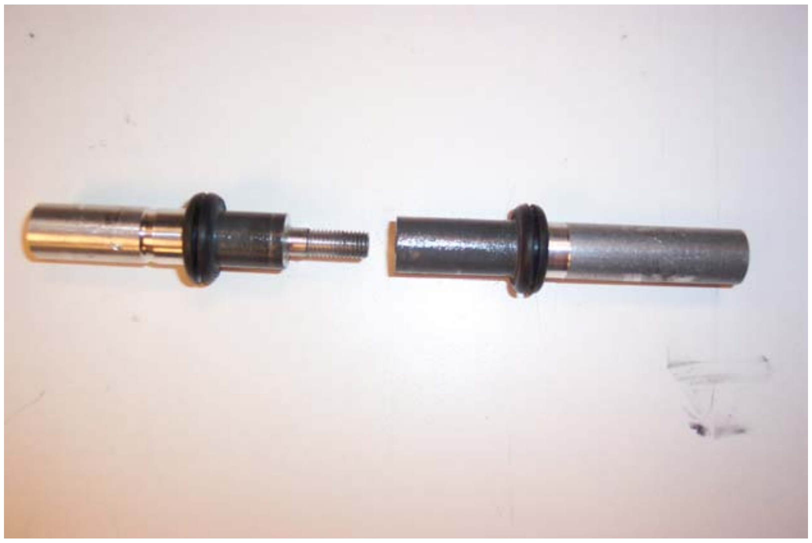


Figure 3.3 The two halves of the axle.

Then by drilling a hole in each end of the axle, the wheels could be mounted to the shaft. To do this, we milled two disks with four holes to attach the wheel to the disk and another hole in the middle to attach the wheel to the axle (Figure 3.4-3.5).

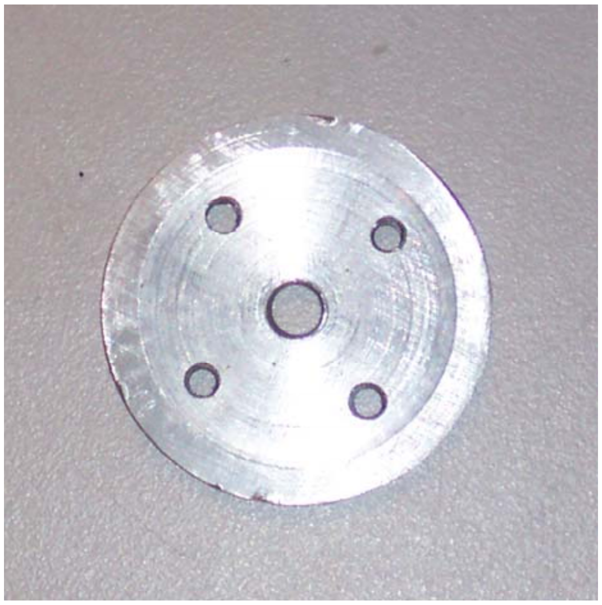


Figure 3.4 The wheel mount.

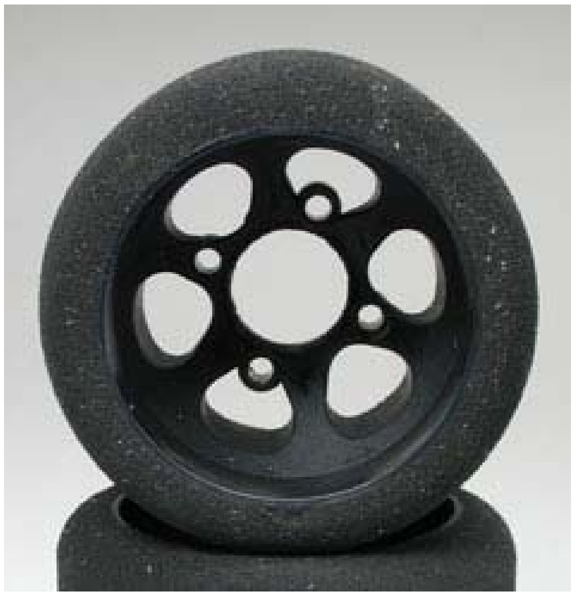


Figure 3.5 The wheels.

With the components complete, I put the axle together (Figure 3.6). To finish the drive system, I need to find a way to mount the axle to the base. After some thought, I decided upon a piece of nylon because it is very smooth which allowed easy rotation of the shaft. Again, I used the mill to cut the piece out and to drill the hole in the middle (Figure 3.7).

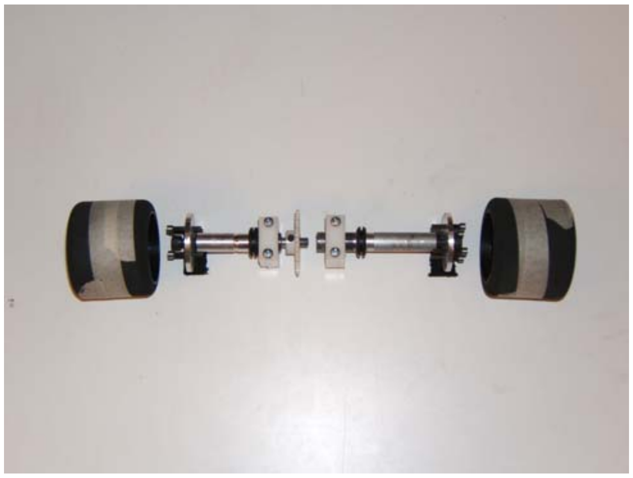


Figure 3.6 The completed axle.

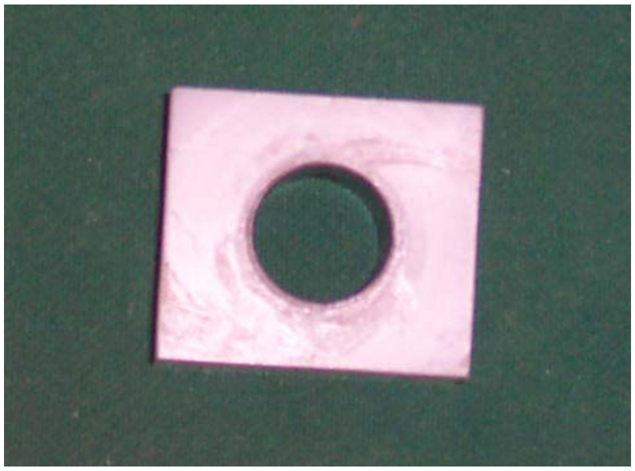


Figure 3.7 The mounts used to attach the axle to the base.

Drilling the hole with the mill was a very important step because if the center axis of the holes were not exactly perpendicular to the face of the mount, the axle would bind in the hole. After making the mounts the third time, along with a little grease, I had a complete, working rear-end drive system (Figure 3.2). Later in testing, we realized that the axle was sliding left and right inside the mounts. To fix this problem, we pushed two rubber grommets against the axle mounts.

Front End – Steering

Since the front wheels and the steering bar came from a purchased RC car, not as much machining was necessary as in the rear axle. The front wheels and the steering bar were the two main components taken from the RC car (Figure 3.8-3.9).

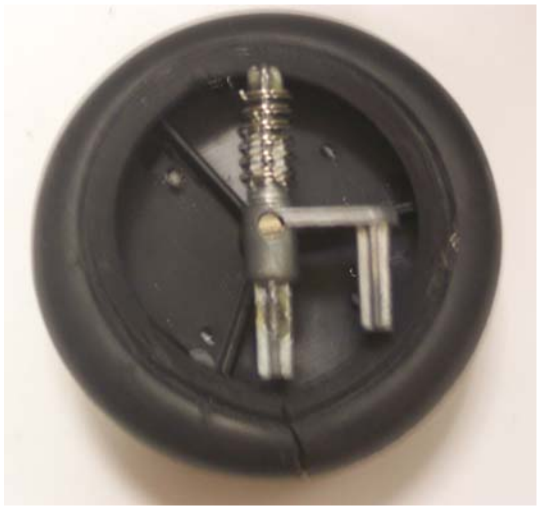


Figure 3.8 The front wheel.

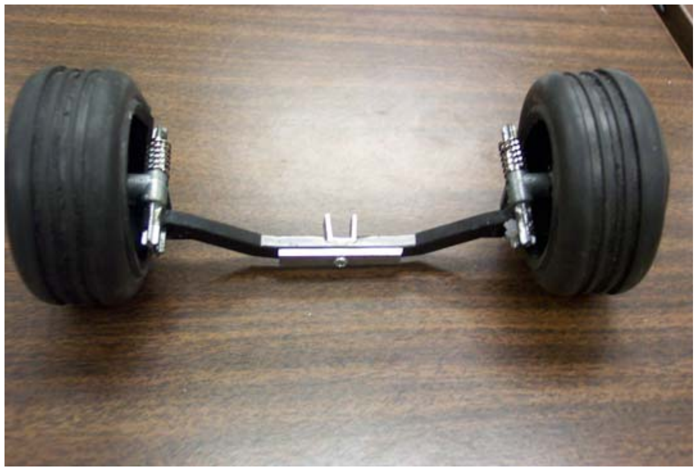


Figure 3.9 The front wheels and steering bar.

The steering bar was modified because the initial piece broke so the U-shaped piece of aluminum in the middle of the bar was added as a replacement. Because we wanted to keep the style of an Indy Car, we used similar mounts for the front wheels (Figure 3.10).

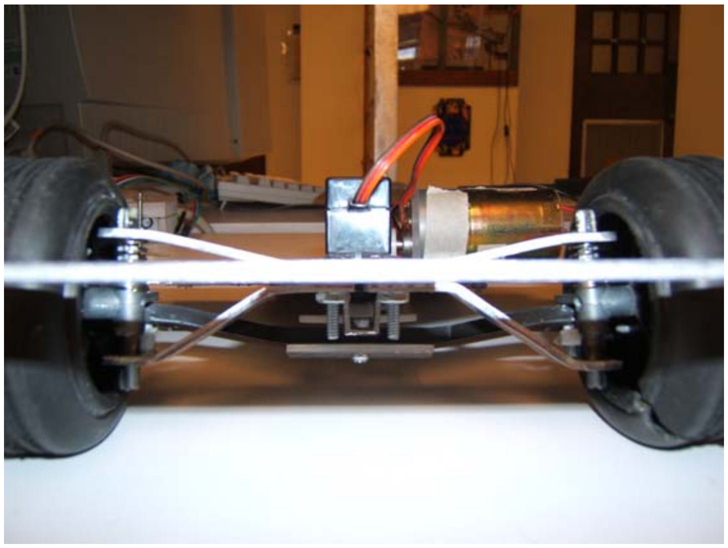


Figure 3.10 The front view of the Indy Car style mounts.

The next task was to find a location in the base to mount the servo. Looking at the base (Figure 3.1) the rectangular hole is where the servo is mounted face down (Figure 3.11).

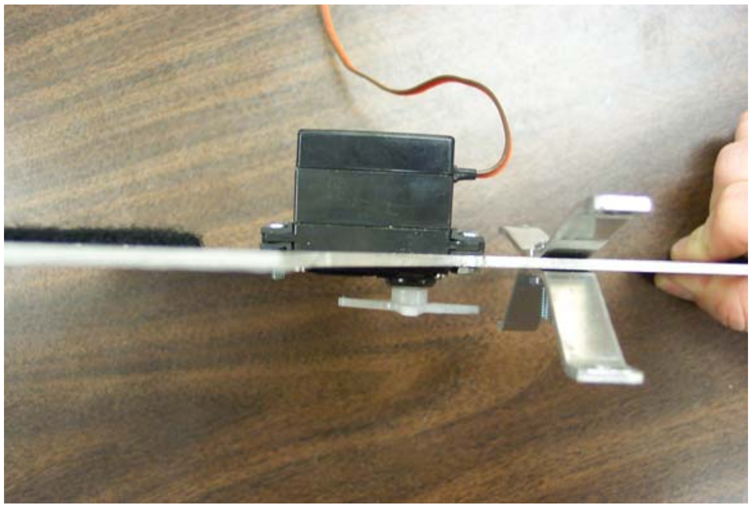


Figure 3.11 The mounting orientation of the servo.

I wanted to make sure a prong from the star-shaped piece on the servo fit into the U-shaped piece on the steering bar. After careful measurements, the final steering mechanism came together very nicely (Figure 3.12).

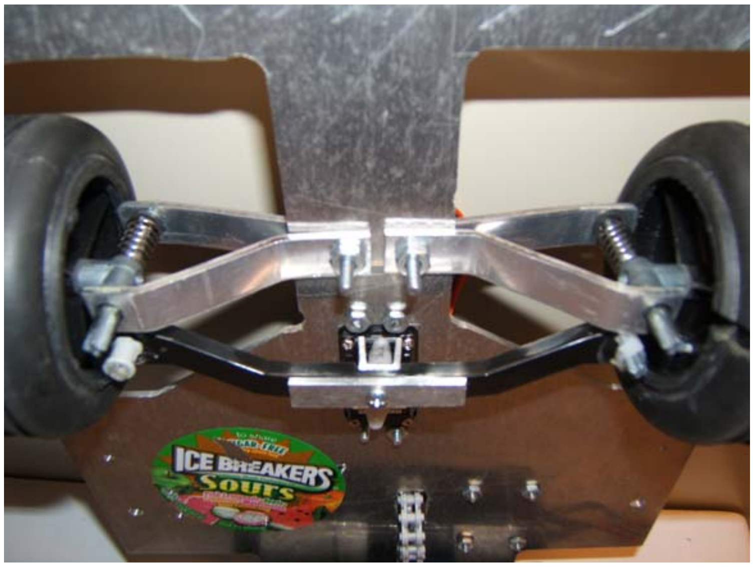


Figure 3.12 The complete steering system.

The last modifications involved altering the cutouts for the front wheels to make sure they had free rotation without running into the base.

Electronics

The main issue with the location of the electronics was making sure they were away from the aluminum base. We decided to create an additional platform to elevate the boards. Another consideration was to allow other parts of the car to be mounted underneath the platform. Lexan was used to create the platform to and it consisted of two C-brackets elevating the flat raised area (Figure 3.13).

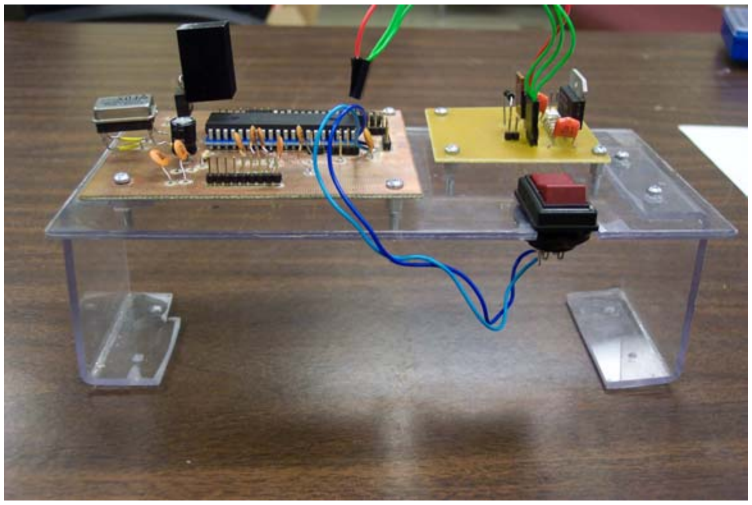


Figure 3.13 The electronics platform.

Once the C-brackets were attached to the base, the main electronics mounting was complete.

Final Mechanical Design

The rest of the mounting included the sensors, the batteries, and the ping-pong ejector (Figure 3.14). We knew from the initial design that we wanted to mount the sensors under the front wing. This was done with two machine screws with three nuts on each one to allow us to alter the height of the sensor board. For mounting the batteries, we decided to use Velcro attach them underneath the platform. That way the weight of the batteries is close to the center of gravity of the car. Since we mounted the ping-pong ejector last, we just found some room that was available and it turned out to be a good location. Having it under the platform, next to the batteries allowed us to face it towards the rear so when the ping-pong ball left the car, it would not get in the way during the course.

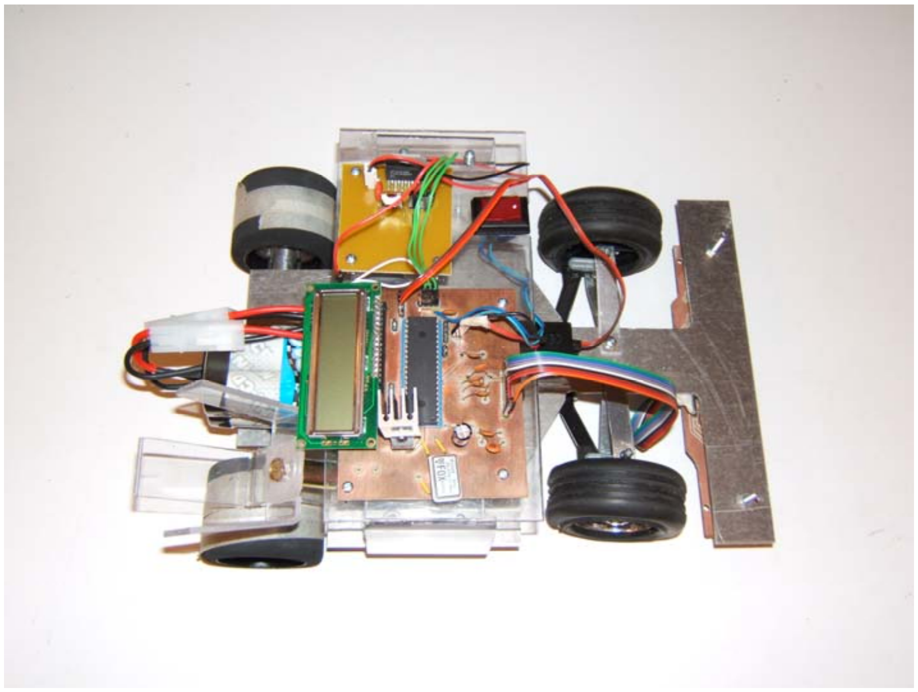


Figure 3.14 The complete design of FIO.

For a group of electrical engineers, we felt that we completed a successful mechanical vehicle. During the design of the robot, durability and reliability were major considerations. After running into walls, cabinets, and feet, the car continued to hold up very well. The structural section was very strong and connections stayed together after several jolts and some vibration. However, some major problems occurred with the front wheels. We had trouble getting the wheels to grip during a sharp turn. To fix the problem, we added a softer rubber tire around the existing wheel. This helped immensely, but in the future, new wheels would be a better option.

Modeling—Jeff Ostrander

Mathematics

The control algorithm was a very large part of this project. For the robot to complete the track correctly, the control needed to work well. Symbol reading worked best while going straight and movement stability was needed throughout the course. We felt that breaking down the problem to its basics and modeling each part of the system would provide insight on the performance of our control. So, for our robot, three dynamic systems are present: the kinematics of the car, the propulsion motor, and the steering servo. Each system is modeled by the following equations:

1. Kinematics

$$\begin{aligned}\dot{x} &= v(\cos\zeta\cos\theta)\\ \dot{y} &= v(\cos\zeta\sin\theta)\\ \dot{\theta} &= v(\sin\zeta)\end{aligned}$$

   Where x and y are the distances in the x and y direction, ζ is the angle of the front wheels in relation to the direction of the car, θ is the angle orientation of the entire car, and v is the velocity of the car [7].

2. DC Motor

$$\begin{aligned}L_a\frac{d\theta}{dt} + R_a i_a + e_b &= e_a\\ J\frac{d^2\theta}{dt^2} + b\frac{d\theta}{dt} &= T = Ki_a\end{aligned}$$

   Where i_a is armature current, R_a is armature resistance, L_a is armature inductance, e_a is applied armature voltage, e_b is back emf, J is moment of inertia, b is the friction coefficient, θ is shaft angle, T is torque, and K is the motor-torque constant [8].

3. Servo

The servo contains its own feedback so the response is basically a time delay. Once the servo receives an input angle, it takes a certain amount of time, 0.23 sec/60^o, for the output angle to occur [9].

Simulink

Modeling in Simulink consists of converting the mathematical models into Simulink blocks. Taking the Laplace transform of each of the above equations and solving for their transfer functions allowed us to use representative blocks from the Simulink library (Figures 3.15-3.17).

1. Kinematics

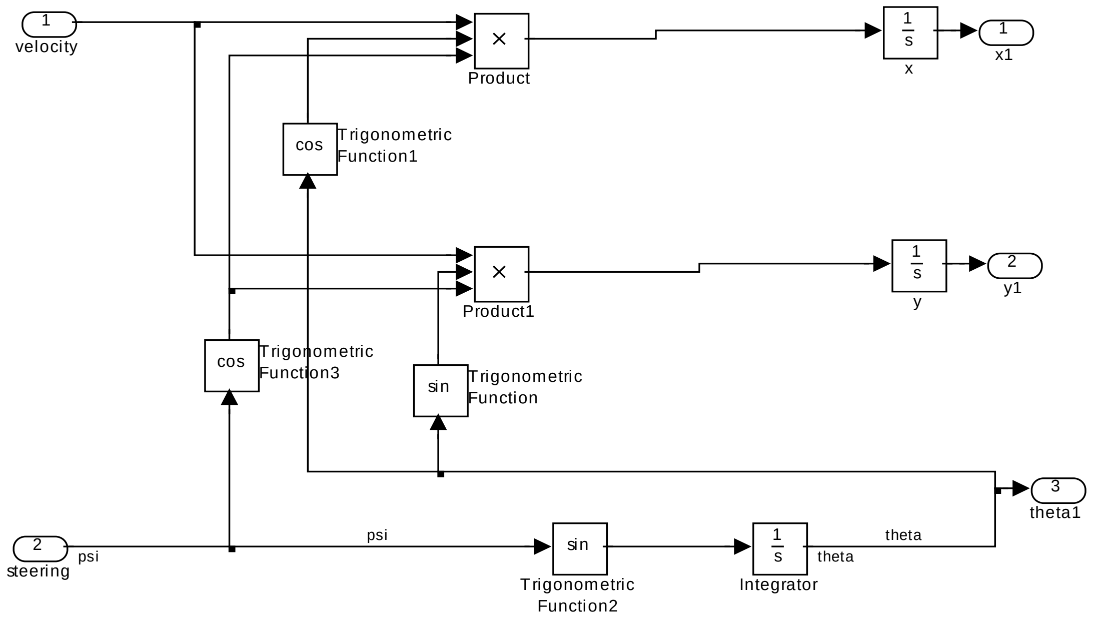


Figure 3.15 A Simulink model of the kinematics equations.

2. DC Motor

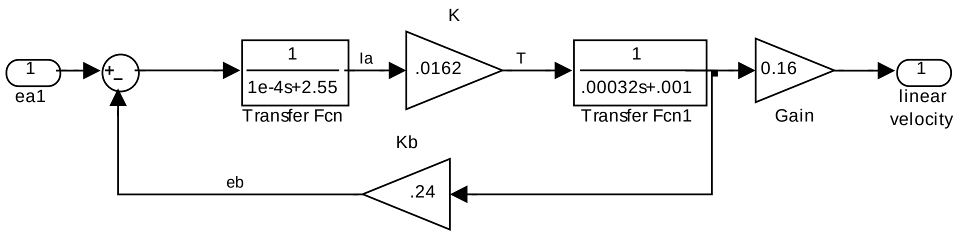


Figure 3.16 A Simulink model of the DC motor.

3. Servo

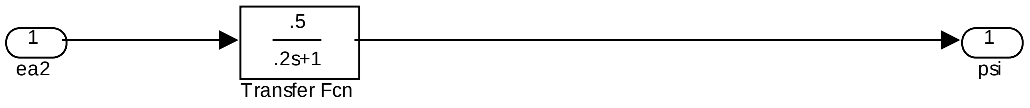


Figure 3.17 A Simulink model of the servo.

Each of these systems is combined to form a complete model that will enable us to do various types of tests to observe the response of the system. The systems above are condensed down to single blocks so the final model looks a little cleaner (Figure 3.18).

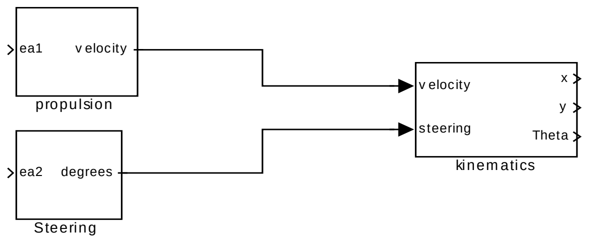


Figure 3.18 A Simulink model of the final system.

Sisotool

After modeling the systems in Simulink, we developed the control. Sisotool is a tool within Matlab that allows a user to test different types of controls for systems that have a single input and a single output. For our model, the input was ea2 and the output was y in the final Simulink system (Figure 3.18). After importing the model into Sisotool, the control was determined by looking at the rise time, overshoot, and settling time of a step response (Figure 3.19).

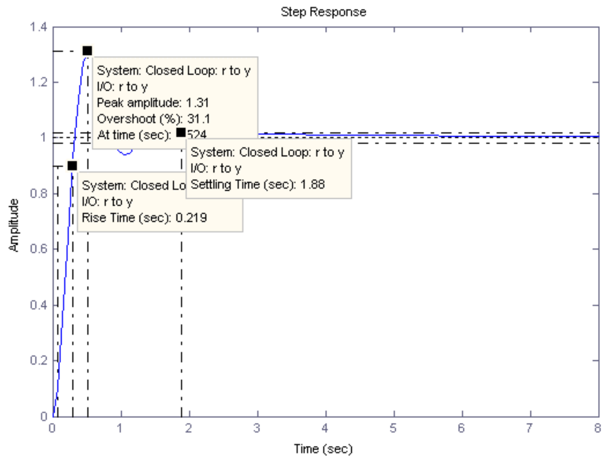


Figure 3.19 The step response of the control in Sisotool.

The control only involved a single real zero. That meant that the control involved proportional and derivative (PD) parts. We did not require an integral factor because the steady state error was zero without it. The proportional factor reduced the rise time and steady state error, while the derivative reduced overshoot and settling time. By using Sisotool, it showed us what type of control algorithm to use, along with the gains, before the robot was even constructed.

Control

After the Simulink open-loop model and the control gains were solved, a closed-loop model was created. The final model included control and quantization of the feedback, which was a result from the sensor values being non-linear (Figure 3.20).

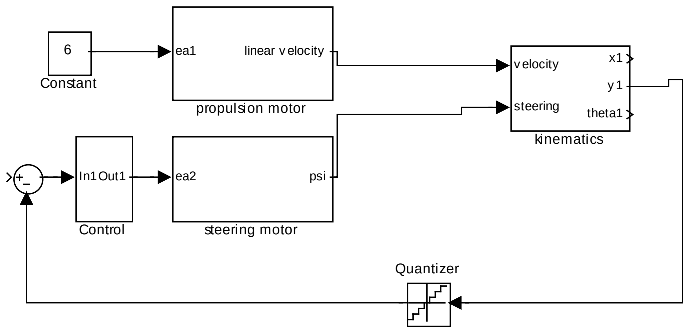


Figure 3.20 The final Simulink model, including the control.

The control was placed between the servo input and the output of y, which was summed with the desired value. In our case, the desired value was zero so no input was necessary (Figure 3.21).

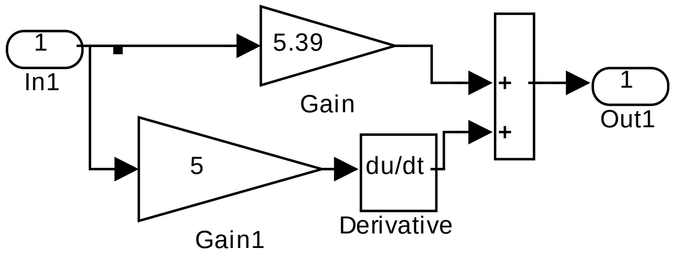


Figure 3.21 The Simulink model of the control.

Results

When simulating the closed-loop system, I could graph several different parameters over a ten second period. I felt that the important data w the steering angle and the error. I used an initial condition of one inch for the error to see the response of the control (Figure 3.22-3.23).

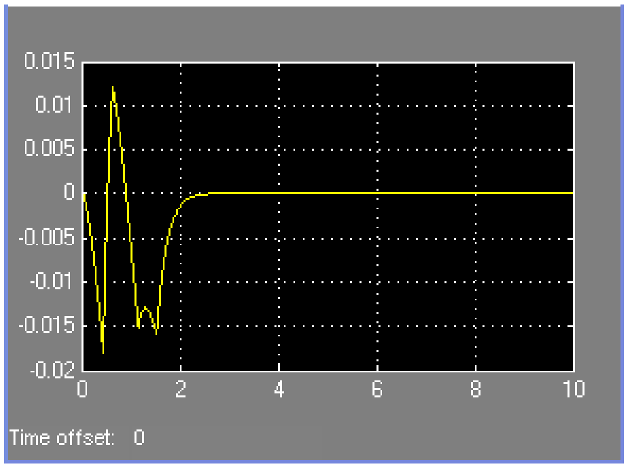


Figure 3.22 The turning response, psi, of a one-inch error.

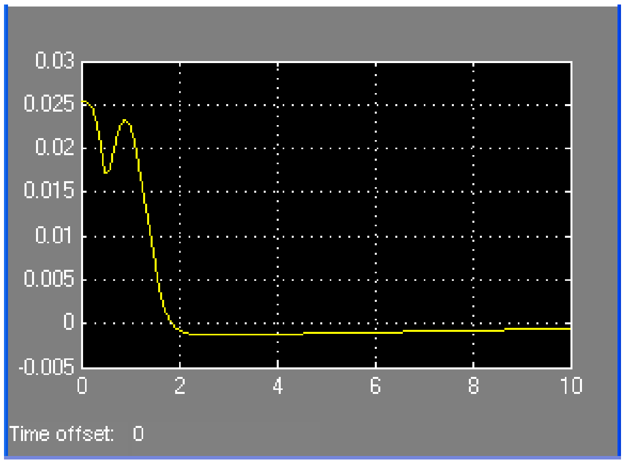


Figure 3.23 The response of the control to initial condition of one inch.

The y-axis is in meters so for an initial condition of one inch, y starts at 0.0254m and the x-axis is in seconds. From the simulation results, the overshoot was less than 1/10" while the settling time was about 2 seconds.

When testing the final design of our car, the overshoot was quite a bit larger than the simulated values, while the settling times were very comparable (Table 3.1).

Table 3.1 A comparison of simulated results vs. actual results.

	Overshoot (Max)	Settling Time
Simulation	1/10"	2 seconds
Actual Testing	8"	2-3 seconds

We found that the 8" overshoot occurred occasionally during the first s-shape curve at the beginning of the track. We have yet to figure out the reason for it, but we feel that it may have been a result in the orientation of the car when turning it on. Also, the 2-3 second settling time was when the 8” overshoot occurs. During normal operation, the car oscillated less than 1” at a time and the settling time was negligible.

If I had more time to work on this, the main goal would be to improve the overall control of the car. The random deviations from the line could be eliminated with some fine-tuning along with a method that provides compensation for power loss. We felt that a decrease in battery power caused deterioration in the operation of our robot. Another change that I feel would improve the performance would be adding more sensors around the middle. By having more in a closer proximity, it would give us more resolution at the center of the car, which would increase the accuracy of the control.

Programming—David ‘Isaac’ Waters

Hardware & Programming Tools

For this project, we used a PIC18F452 microprocessor to control FIO. We chose to use a PIC because we already had several on hand, and we were comfortable with them. While we realize that these reasons are not always practical, for this project it suited both our time and budgetary constraints. We did not have to purchase many new microprocessors, and we did not have to spend time getting over the learning curve involved with using a new chip. Also, we knew that the 18F452 was capable of doing everything we needed for this project. It had enough memory and eight A/D ports, which were our main concerns.

We downloaded the free MPLab software from microchip.com. It had a 60 day full-feature evaluation. It also came with the C18 compiler, which we used for our program. We had a demo copy of a different compiler, but it was not able to compile the chip we had chosen, so we opted to use MPLab instead. This presented a bit of a challenge for us, since we had not used the C18/MPLab combination before. However, we found a compiler library for C18 that helped us set up our code with the correct functions and syntax.

In order to move the code from our computer to our microprocessor, we needed to use a programmer. Three of our group members had an ICD-U40, which is a programmer obtainable from www.ccsinfo.com. The ICD is very simple to use, and it is very efficient.

Executive Program

The first task I had to handle was setting up a framework of code. I used an executive program to initialize variables and call the sub functions (Appendix A-I). We wanted our main program to run at a rate of 200 Hz, so I added a while loop that regulated the program speed. Using a timer, this loop held the program in a wait mode after it ran through the sub functions. The program sat and 'idled' until the timer hit a certain value, and then it ran the sub functions again (Figure 4.1). We found this to be a very effective way to keep our program from running too quickly, which would cause the sensors to read too frequently and be too sensitive.

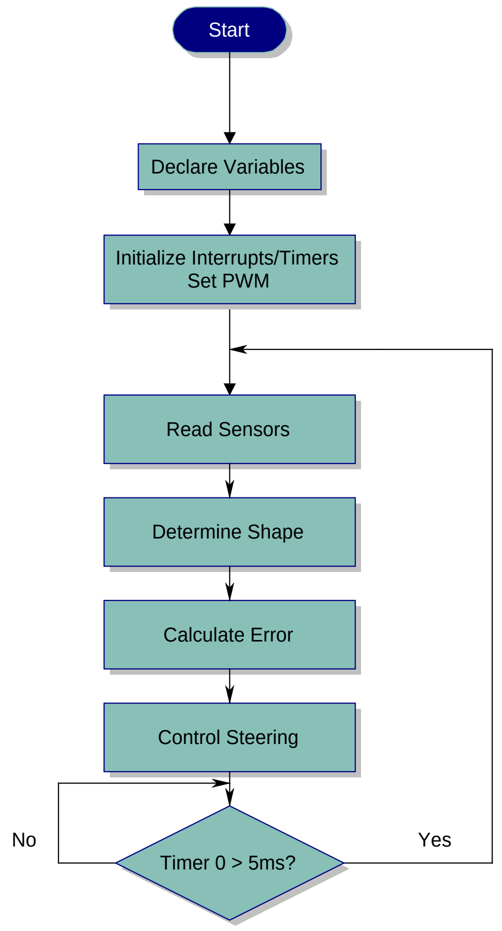


Figure 4.1 Executive Summary.

Read Sensors

In order to calculate where FIO was with respect to the line, we needed to read in the sensors. We used seven A/D channels on our PIC (Appendix B.V) [10]. Each sensor went to one A/D pin, and the PIC read in a voltage value. Our A/D range was from 0-1024, where 0V would be read as a zero and 5V would be read as 1024. I used a sub function to read in each sensor and stored the converted A/D value in an integer type variable (Appendix A – II). We kept our program very modular, so the 'Check Position' function only read in the sensors and stored values (Figure 4.2).

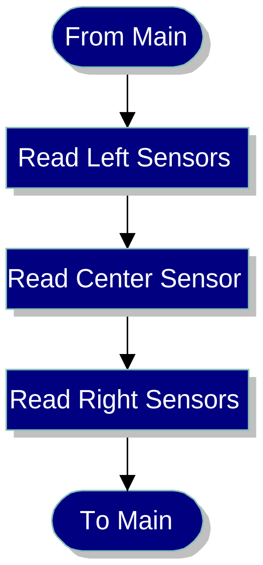


Figure 4.2 Sensor reading flowchart

Detect Shape

After reading in the sensor values, I used them to determine whether or not FIO was on a shape. If the center three sensors read black, we knew we were over a shape, so the program set 'on_shape' equal to one (Appendix A – III). We used 'on_shape' as a flag that we set to one when on a shape and set to zero when we were not. I created a few variables that we used as 'counters'. Each variable represented one of the outer four sensors. When FIO ran over a shape, each counter incremented once for each time its sensor read 'black' (Figure 4.3).

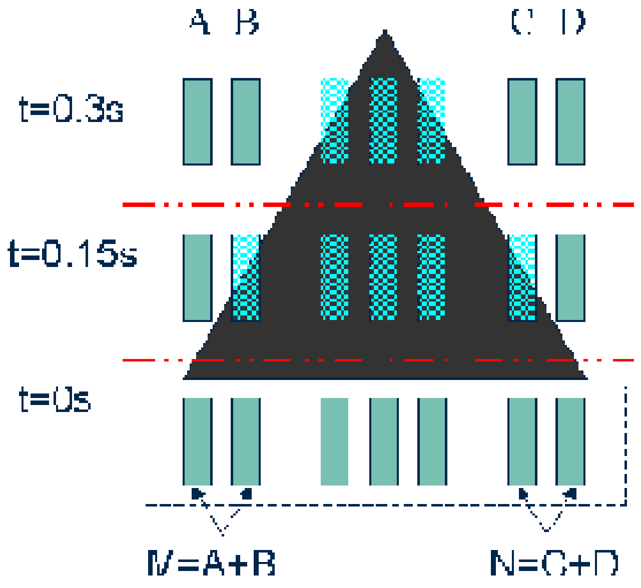


Figure 4.3 Symbol reading over time

After FIO was no longer on a shape, the program reset 'on_shape' to zero and then determined what shape had just passed. We added the counters for the two leftmost columns together and set it equal to 'M', and we added the counters for the two rightmost columns together and set it equal to 'N'. We then used M and N to determine the shape (Table 4.1).

Table 4.1 Criteria used to indentify symbols.

M>N	M<N	M+N<75	M+N>400
Turn Right	Turn Left	Shoot Ball	Stop

If ‘M’ was greater than ‘N’, we knew it was the ‘Turn Right’ symbol. If ‘M’ was less than ‘N’, FIO had read the ‘Turn Left’ symbol. However, we had two other options that took precedence over the two turns. If ‘M+N’ was less than 75, we knew it was the eject symbol. If ‘M+N’ was greater than 400, we knew it was the stop symbol. In each case, we ran the correct code according to the symbol FIO crossed.

Calculate Error

Our program calculated error based on its current position with respect to the line.   We calculated several error functions that we used in our code. Essentially, we had discrete sections that calculated continuous error (Appendix A – IV). We calculated these formulas by taking readings from the sensors at different distances from the line. We then assumed that the readings were linear between certain points and wrote several equations in the form of y = m*x + b. The equations were all based on the readings from the center sensor, but it did take into account the sensors to either side (Figure 4.4). If all seven sensors were off the line at the same time, we assumed it was at maximum error.

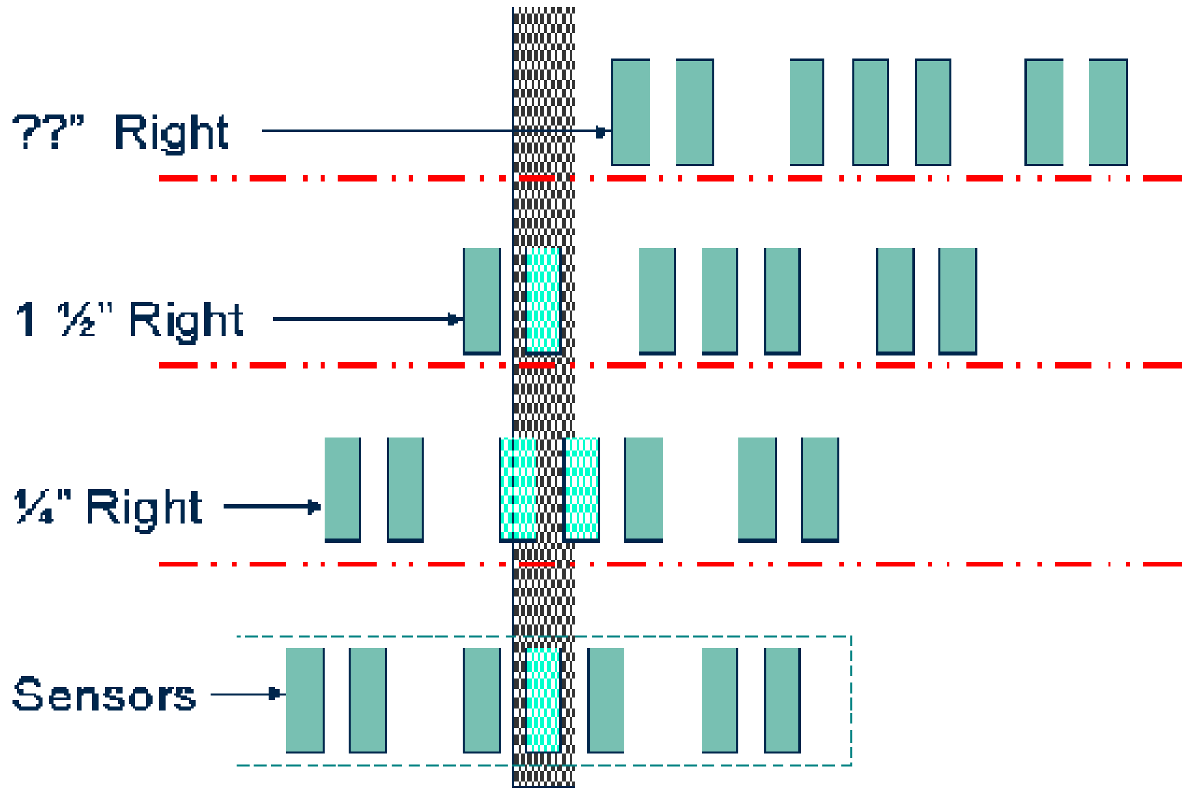


Figure 4.4 Line sensing at various errors

Control Algorithm

We included three main components in our control algorithm. First, we started with proportional control (Appendix A – V). We multiplied the calculated error (actual distance from the line minus the desired distance from the line) by a proportional constant, and we set our output equal to that. However, proportional control alone was not enough to make FIO run correctly. We added derivative control, where we multiplied the difference between the previous and current errors by a derivative constant. After adding derivative control, we subtracted the derivative value from the proportional value and set our output equal to the result. The third main component in our control was our digital filter. We used it to smooth out our turning and to get rid of large, quick, error spikes.

We included two special cases in our control algorithm. If the center sensor was on the line, we set the steering angle to zero. If all of the sensors were off the line, we set the steering to maximum left or maximum right, depending on which side read the line last.

The servo we used relied on pulse widths sent from the PIC to control its turning angle. The Pulse Width Modulation (PWM) on our microprocessor could not generate long enough pulses to control the servo, so we used a timer driven interrupt to do what our PWM could not. The code continually compared a timer to a constant. When the numbers matched, it went into the interrupt routine and generated a pulse to turn the servo.

The control algorithm took in the result from our error calculation and turned it into the number we used in our timer driven interrupt. Every time the control sub function ran, it set the number that the timer was compared to (Figure 4.5).

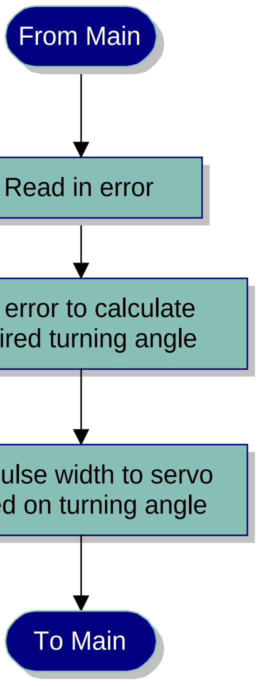

From Main
Read in error
Use error to calculate desired turning angle
Set pulse width to servo based on turning angle
To Main

Figure 4.5 Control flowchart

Programming Results—David ‘Isaac’ Waters

For this project, our approach to coding was more than adequate. We began with the main loop, and we worked our way through the main sub functions. Our microprocessor did everything we needed it to do, and once we learned how to use the C18 compiler, the only problems we ran into were control related.

We began testing the control while only using a proportional term. This turned out to be a mistake, as the robot never reached steady state. The oscillation was not within a tolerable range, and symbol reading would have been nearly impossible. After we added the derivative term, our system started to behave much better. FIO followed the line more closely, and our oscillation was brought well within acceptable levels. This allowed FIO to cross the symbols going fairly straight, which led to more accurate readings. PD control turned out to be all we needed to succeed on this project.

Our method of calculating error was adequate (after all, the project was a success), but given more time, we probably would have refined our calculations. In fact, if we had placed our sensors closer together, we could have had more resolution around the line. This would have led to more accurate error readings and would have improved the control. Improving the control might have allowed us to run the course a little faster.

The symbol-sensing algorithm worked fairly well, and if we had more time we could have worked the last bug out of it. FIO read ‘left’ on the ‘eject’ symbol quite a bit, which caused some problems. During our presentation, Dr. Chung suggested that we use a time difference algorithm instead of the one we had been running. Instead of incrementing counters when a sensor read black, we could have kept track of which sensor hit black first, and that would have given us an accurate symbol reading. If we had this project to do over, we would definitely take that route. Towards the project deadline, our control code was working well enough that we would not have had to worry about oscillation over a symbol, and so using time difference would have worked very well.

We chose to read in the sensor values through A/D ports instead of digitally, and that turned out to be a very good idea. If we had only used on and off settings for our sensors, our control would not have worked nearly as well.

Our method of running the executive program worked flawlessly. If we wanted to run the program faster, all we would have to do is change our timer cycle. It would work the same way if we wanted to run the program slower. This gave us the freedom to experiment and see how well our control ran at different speeds.

Overall, our program was a success. FIO ran the track well based on the control we created.

Conclusion

Our goal was to design and construct an autonomous, line-following robot. We feel that our approach was the best with the knowledge we possessed, and our design met our time and budgetary constraints. While working on this project, we practiced good engineering skills, such as documenting and creating planning documents (such as flow charts and mechanical drawings). Our robot worked according to the specifications during our demonstration. If we had to do this project again, we would start on the construction sooner, spend less time on the modeling, and look for an alternative to batteries as a power supply.

References

[1] Seattle Robotics Society. (n.d.). [online]. Whats a Servo?. Available: http://www.seattlerobotics.org/guide/servos.html May 1, 2005 [date accessed]

[2] National Semiconductor. (n.d.). [online]. LMD18200 3A, 55V H-Bridge. Available: http://cache.national.com/ds/LM/LMD18200.pdf April 30, 2005 [date accessed]

[3] The Robot Market Place. (n.d.). [online]. DC Motors. Available: www.therobotmarketplace.com May 1, 2005 [date accessed]

[4] S. Kazuma, A. Pandey, T. Smith, and S. Toh, "Hayai, Maze Racer," Oklahoma State University, Stillwater, OK, Dec. 2004.

[5] Mouser Electronics. (n.d.). [online]. OPB-742 Data Sheet. Available: http://www.optekinc.com/pdf/OPB740.pdf January 30, 2005 [date accessed]

[6] Jameco Electronics. (n.d.). [online]. Part #: 14973 Data Sheet. Available: http://jameco.com/wcsstore/Jameco/Products/ProdDS/149753.pdf . April 5, 2005 [date accessed]

[7] F. Lamiraux and J.-P. Laumond, "Smooth Motion Planning for Car-Like Vehicles," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 4, pp. 498-502, Aug. 2001.

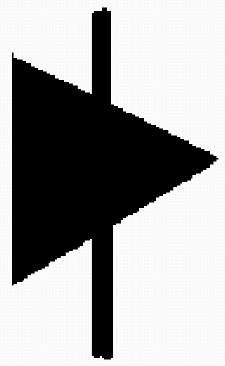
[8] K. Ogata, "Electrical Systems," in *System Dynamics*, 3rd ed., B. Stenquist, Ed. New Jersey: Prentice Hall, Inc., 1998, pp. 134-175.

[9] Mark III Robot Store. (n.d.). [online]. Standard-torque Ball-bearing Servo Motor. Available: http://www.junun.org/MarkIII/Info.jsp?item=18 March 13, 2005 [date accessed]

[10] Microchip.com. (n.d.). [online]. PIC18FXX2 Datasheet. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/39564b.pdf

Appendix A – Track Symbols

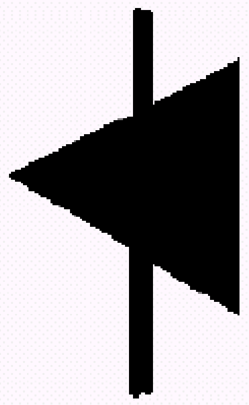
<u>Right Path Symbol</u> –



Symbol for "take the right most path" at a track juncture. Triangle is equilateral and 4" on each side. Stripe is 1/2 " and centered within triangle.

NOT TO SCALE

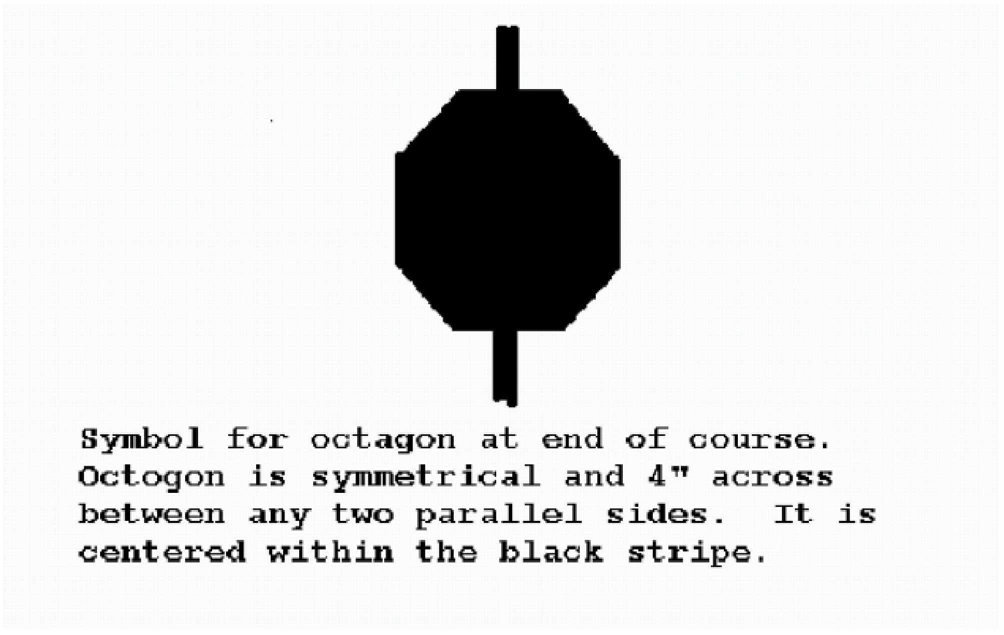
<u>Left Path Symbol</u> –



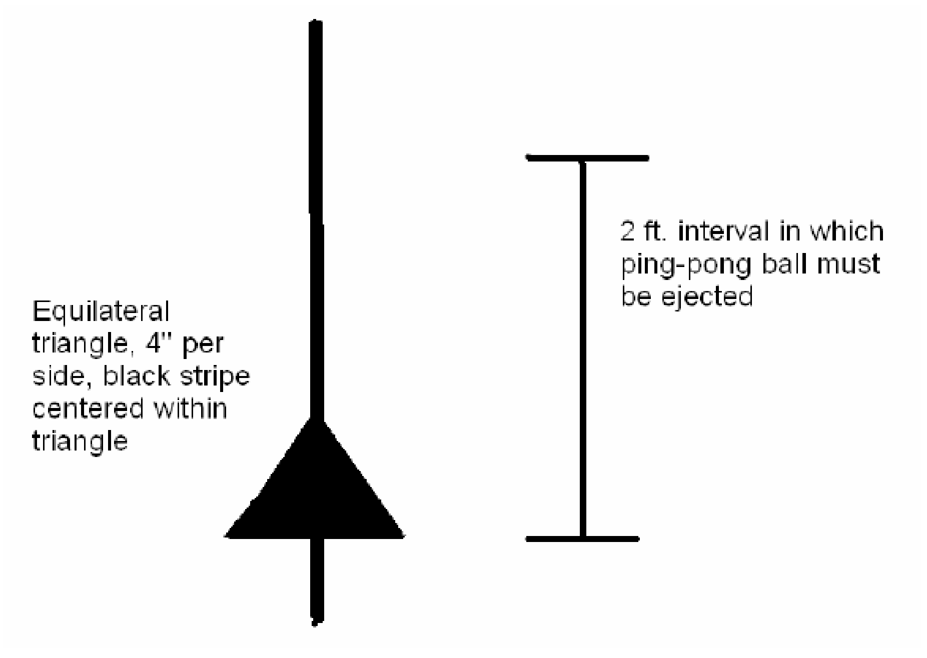
Symbol for "take the left most path"

NOT TO SCALE

End of Course Symbol –


Symbol for octagon at end of course. Octogon is symmetrical and 4" across between any two parallel sides. It is centered within the black stripe.

Eject Ball Symbol –


Equilateral triangle, 4" per side, black stripe centered within triangle
2 ft. interval in which ping-pong ball must be ejected

Appendix B – Miscellaneous Figures

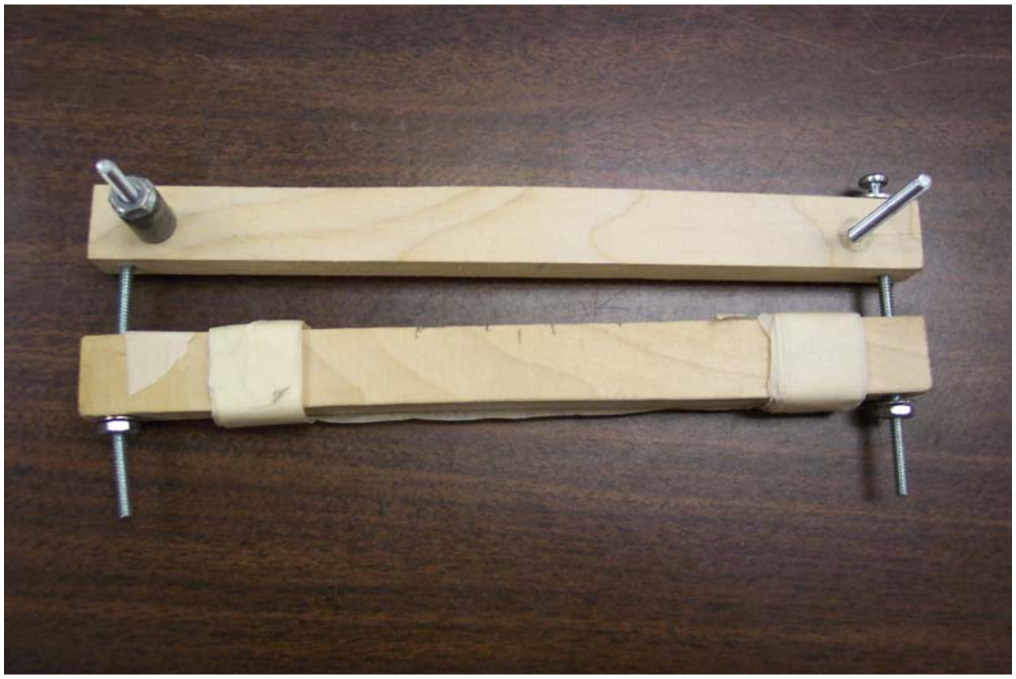


Figure B.I Prototype sensor mount.

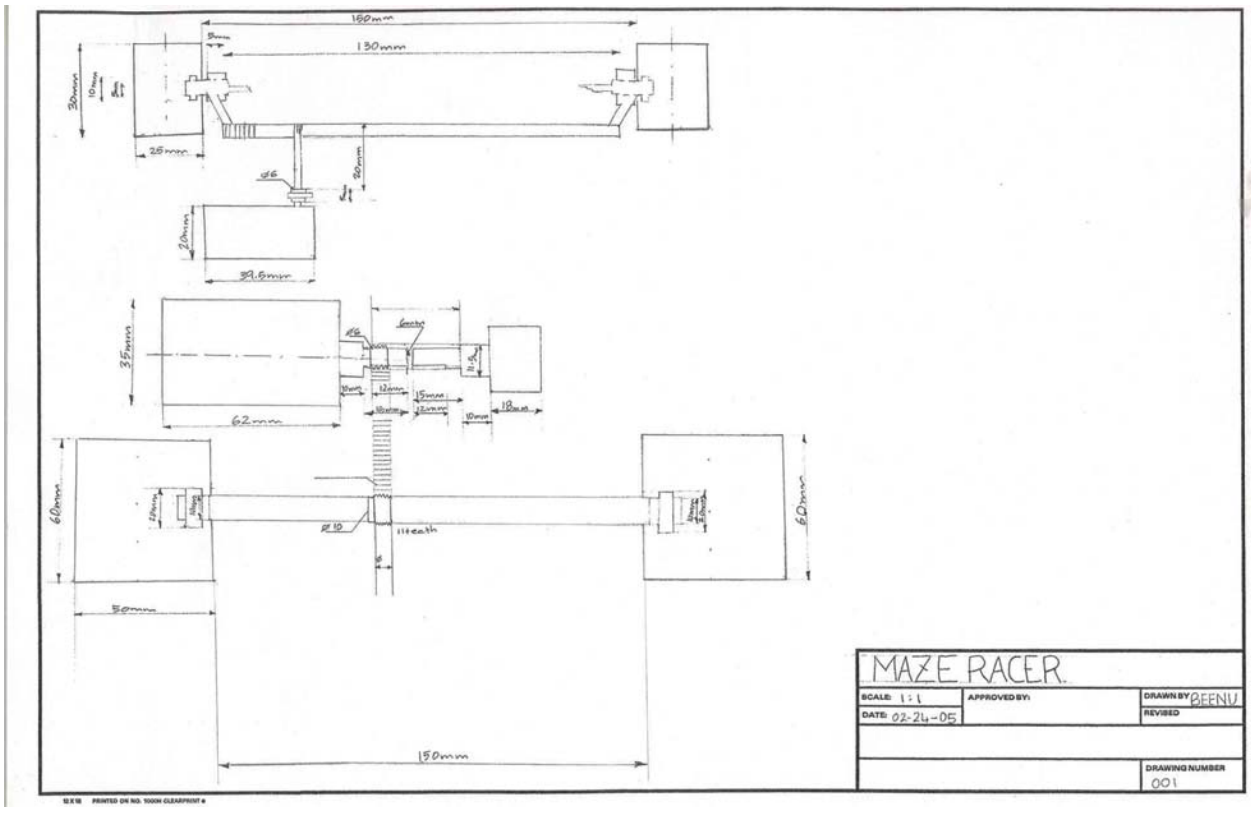


Figure B.II Rear drive and steering mechanism

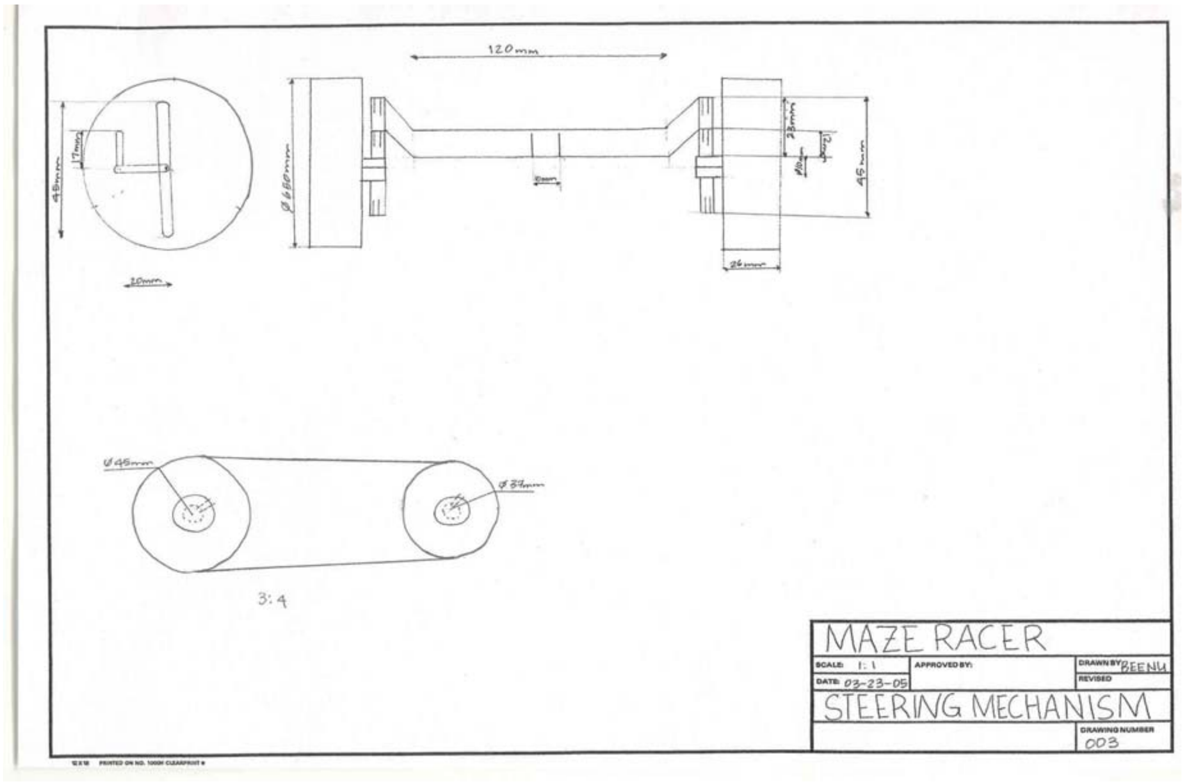


Figure B.III Steering Mechanisms

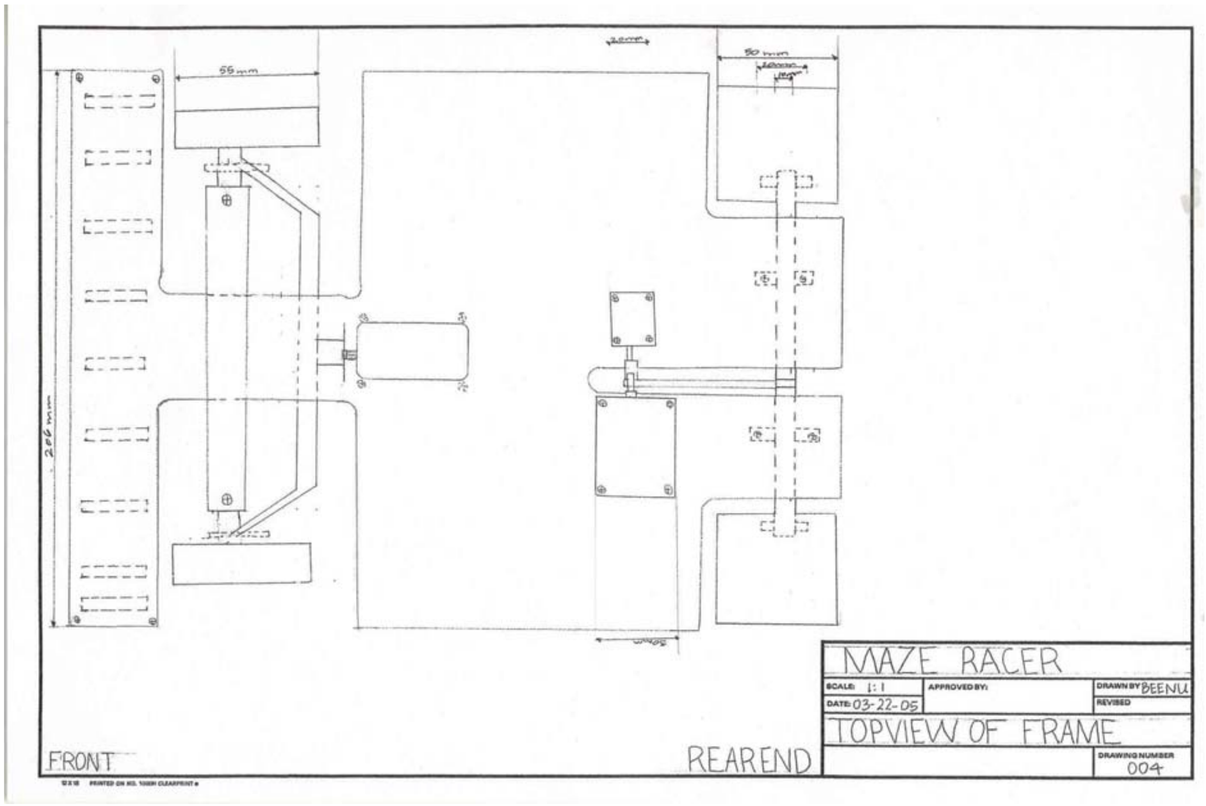


Figure B.IV Top view of final design

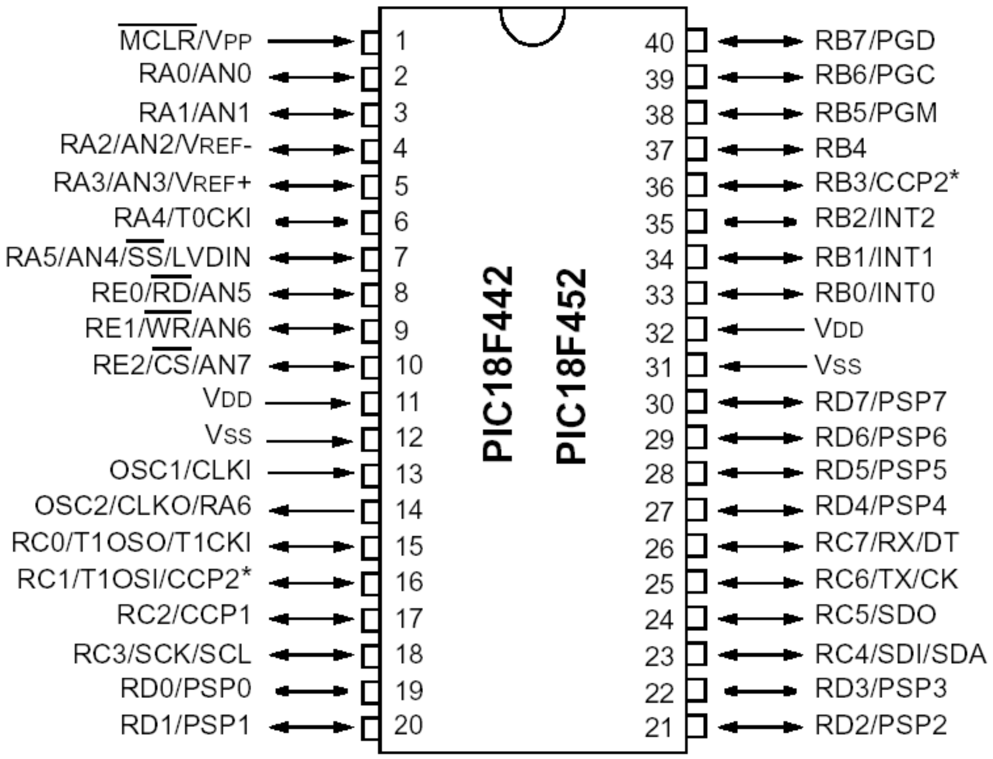


Figure B.V Pin-out diagram for the PIC18F452 [10]

Appendix C – Program Subfunctions

I. Main

```
void main (void)
{
        TRISD=0;     /*Set PORTD to output*/
        TRISB=0;     /*Set PORTB to output*/
        TRISC=0;     /*Set PORTC to output*/
        PORTDbits.RD3=0;
        XLCDInit();
        /* Initialize SPI, CCPs, timers, interrupts, inputs, and outputs used */
OpenTimer0(TIMER_INT_OFF & T0_16BIT & T0_SOURCE_INT & T0_PS_1_16);
OpenTimer1(TIMER_INT_OFF & T1_16BIT_RW & T1_SOURCE_INT & T1_PS_1_8 &
T1_OSC1EN_OFF & T1_SYNC_EXT_OFF);
        OpenTimer2(TIMER_INT_OFF & T2_PS_1_16 & T2_POST_1_1);
        OpenPWM2(0xff);
        SetDCPWM2(500);
        RCONbits.IPEN=1;       /*Disables priority levels on interrupts*/
        INTCONbits.GIEH=1;   /*When IPEN=1, enables all high priority interrupts*/
        CCP1CON=0b1010;
        PIE1bits.CCP1IE=1;
        IPR1bits.CCP1IP=1;
        CCPR1 = 937;
        n=0;
        m=0;
        i=0;
        XLCDClear();

        while(1)
        {
                if(ReadTimer0()>1500)
                {
                        WriteTimer0(0);  //Controls the rate at which the program runs
                        check_position(); //Checks position
                        detect_shape();    //Detects shape
                        calculate_error(); //Calculates error
                        control_alg();     //Steers the car
                }
        }
}
```

II. Check Position

```
int check_position (void)
{
        /*Read in extreme left sensor and store value*/
        OpenADC(ADC_FOSC_32 & ADC_RIGHT_JUST & ADC_7ANA_1REF, ADC_CH0
& ADC_INT_OFF);
        ConvertADC();
        while(BusyADC());
        {
                ex_left=ReadADC();
        }
        CloseADC();

        /*Read in far left sensor and store value*/
        OpenADC(ADC_FOSC_32 & ADC_RIGHT_JUST &  ADC_7ANA_1REF, ADC_CH1
& ADC_INT_OFF);
        ConvertADC();
        while(BusyADC());
        {
                result_far_left=ReadADC();
        }
        CloseADC();

        /*Read in left sensor and store value*/
        OpenADC(ADC_FOSC_32 & ADC_RIGHT_JUST & ADC_7ANA_1REF, ADC_CH2
& ADC_INT_OFF);
        ConvertADC();
        while(BusyADC());
        {
                result_left=ReadADC();
        }
        CloseADC();

        /*Read in center sensor and store value*/
        OpenADC(ADC_FOSC_32 & ADC_RIGHT_JUST & ADC_7ANA_1REF, ADC_CH4
& ADC_INT_OFF);
        ConvertADC();
        while(BusyADC());
        {
                result_center=ReadADC();
        }
        CloseADC();

        /*Read in right sensor and store value*/
```

```
	OpenADC(ADC_FOSC_32 & ADC_RIGHT_JUST & ADC_7ANA_1REF, ADC_CH5
& ADC_INT_OFF);
	ConvertADC();
	while(BusyADC());
	{
		result_right=ReadADC();
	}
	CloseADC();

	/*Read in far right sensor and store value*/
	OpenADC(ADC_FOSC_32 & ADC_RIGHT_JUST & ADC_7ANA_1REF, ADC_CH6
& ADC_INT_OFF);
	ConvertADC();
	while(BusyADC());
	{
		result_far_right=ReadADC();
	}
	CloseADC();

	/*Read in extreme right sensor and store value*/
	OpenADC(ADC_FOSC_32 & ADC_RIGHT_JUST & ADC_7ANA_1REF, ADC_CH7
& ADC_INT_OFF);
	ConvertADC();
	while(BusyADC());
	{
		ex_right=ReadADC();
	}
	CloseADC();
}
```

III. Detect Shape

```
void detect_shape(void)
{

if(on_shape==1) //If on shape, we increment each sensor that is on black
{
                if(ex_left<400)
                {
                        a++;
                }

                else if(ex_left>400 && result_far_left<400)
                {
                        a++;
                }
                if(result_far_left<500)
                {
                        b++;
                }
                if(result_far_right<400)
                {
                        c++;
                }
                if(ex_right<400)
                {
                        d++;
                }
                else if(ex_right>400 && result_far_right<400)
                {
                        d++;
                }

}
                m=a+b;//Sum left sensors
                n=c+d;//Sum right sensors


        if(on_shape==0)
                {

                        if(m<n)
                        {
                                Turn_Right=0; // Don't need to turn right
                                a=0;        // Reset Counters
                                b=0;
                        c=0;
                        d=0;
                }

                if(n<m)
                {
                   Turn_Right=1; // Need to turn right
                        a=0;        // Reset Counters
                        b=0;
                        c=0;
                        d=0;
                }

                if(n+m>400)
                {
                        Stop();
                }

                if(n+m<75)
                {
                    Shoot_Ball();
        }

            }



    }
```

IV. Error Calculation

```
int calculate_error (void) //Error calculation function
{
        if(result_center<250)  //If center sensor is reading black, set error=0
        {
                PS_Error=0;
        }

        else if(result_center>=250 && result_center<600)
        {
                PS_Error=-9635+2109*log(result_center);
        }

        else if(result_center>=600)
        {
                PS_Error=3*result_center+1900;
        }

        if(result_center<400 && result_right<400 && result_left<400) //If middle three sensors
are low, we're on a shape
        {
                on_shape=1;
                i=0;
        }

        else //Otherwise we're not
        {
                on_shape=0;
                i++;
        }

        if(on_shape==0) //If we're not on a shape, but sensors are reading dark, we're too far right
or too far left
        {
                if(result_right<450 || result_far_right<400)
                {
                        Too_Far_Left=1;
                }

                if(result_left<450 || result_far_left<400)
                {
                        Too_Far_Left=0;
                }
        }
        else if(on_shape==1)
        {
                Too_Far_Left=0;
                PS_Error=PS_Error;
        }

}
```

V. Control Algorithm

```
void control_alg(void)
{

Kd=58;    //Derivative Constant
Kp=1.15;  //Proportional Constant
alpha=0.5;//Digital filtering constant

vel_err=alpha*vel_err+(1-alpha)*(PS_Error_P-PS_Error);
output=(Kp*(PS_Error/10)-Kd*(vel_err));
PS_Error_P=PS_Error;

        if(output<0 || result_center<350)
        {
                output=0;
        }

        if(result_center>400 && result_left>400 && result_right>400) //If middle sensors read
high
        {

                        if(Too_Far_Left==1) //Make right turn back towards line
                        {
                                pulse_width=15*(10-(output/45))+730;
                        }

                        if(Too_Far_Left==0) //Make left turn back towards line
                        {
                                pulse_width=10*(output/45)+932;
                        }

        }


        else //Edit this to allow for smoother driving on the line
        {
                if(Too_Far_Left==1) //If we're too far left or right, but a sensor is still on the
line...
                {
                        pulse_width=22*(10-(output/45))+730;
                }
                else if(Too_Far_Left==0)
                {
                        pulse_width=10*(output/45)+920;
                }
```

```
}

if(pulse_width>1060) // If pulse width is high, set to max
{
        pulse_width=1095;
}

if(pulse_width<780) // If pulse width is low, set to min
{
        pulse_width=690;
}

//This section of code determines if the car is on a split and controls it if it is
if((ex_left<400 || ex_right<400 || result_far_left<400 || result_far_right<400))
{
        if(Turn_Right==1 && (ex_right<500))
        {
                pulse_width=705;
                XLCDPut(' ');
                XLCDPut('R');
                Too_Far_Left=1;
                Delay10KTCYx(125);
                Turn_Right=2;
        }

        else if(Turn_Right==1 && (result_far_left<500 || ex_left<500))
        {
                pulse_width=1010;
                XLCDPut(' ');
                XLCDPut('R');
                Turn_Right=2;
        }

        if(Turn_Right==0 && (ex_left<600 || result_far_left<420))
        {
                pulse_width=1095;
                XLCDPut(' ');
                XLCDPut('L');
                Too_Far_Left=0;
                Delay10KTCYx(125);
                Turn_Right=2;
        }
        else if(Turn_Right==0 && (ex_right<475) && result_center<400)
        {
                pulse_width=800;
                XLCDPut(' ');
```

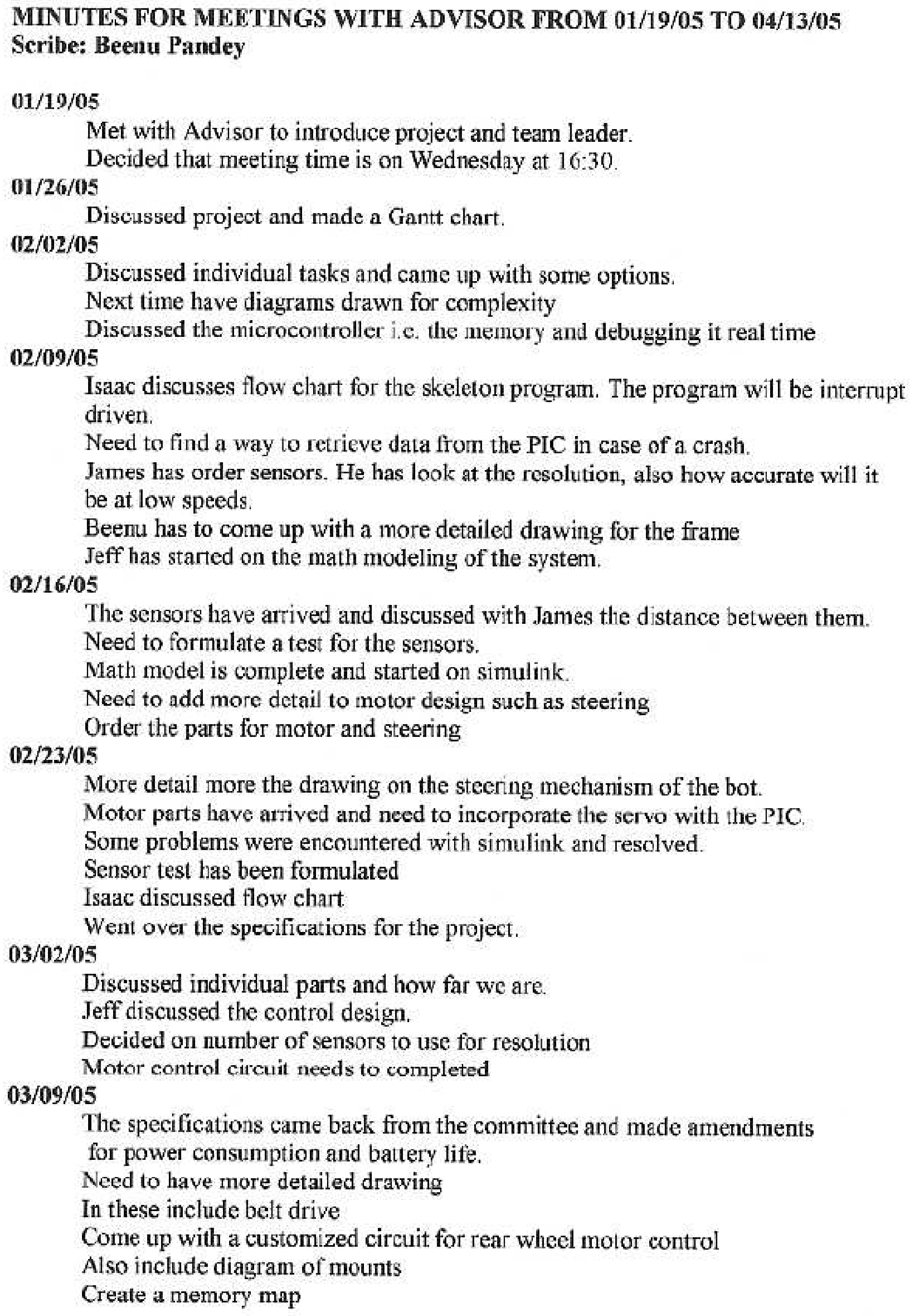
```
                        XLCDPut('L');
                        Delay10KTCYx(150);
                        Turn_Right=2;
                    }

            }
    }
```

Appendix D – Itemized Cost

Dc Motor	**$18.00**
Servo Motor	**$10.00**
Wheels	**$9.00**
Steering rack	**$20.00**
Batteries	**$54.17**
Microcontroller	**$6.00**
H-bridge	**$11.20**
Sensors	**$20.23**
22M Resistors	**$13.86**
Solenoid	**$16.90**
Misc	**$20**
Total	**$199.36**

Appendix E – Advisor Meeting Minutes



MINUTES FOR MEETINGS WITH ADVISOR FROM 01/19/05 TO 04/13/05
Scribe: Beenu Pandey

01/19/05

Met with Advisor to introduce project and team leader.
Decided that meeting time is on Wednesday at 16:30.

01/26/05

Discussed project and made a Gantt chart.

02/02/05

Discussed individual tasks and came up with some options.
Next time have diagrams drawn for complexity
Discussed the microcontroller i.e. the memory and debugging it real time

02/09/05

Isaac discusses flow chart for the skeleton program. The program will be interrupt driven.
Need to find a way to retrieve data from the PIC in case of a crash.
James has order sensors. He has look at the resolution, also how accurate will it be at low speeds.
Beenu has to come up with a more detailed drawing for the frame
Jeff has started on the math modeling of the system.

02/16/05

The sensors have arrived and discussed with James the distance between them.
Need to formulate a test for the sensors.
Math model is complete and started on simulink.
Need to add more detail to motor design such as steering
Order the parts for motor and steering

02/23/05

More detail more the drawing on the steering mechanism of the bot.
Motor parts have arrived and need to incorporate the servo with the PIC.
Some problems were encountered with simulink and resolved.
Sensor test has been formulated
Isaac discussed flow chart
Went over the specifications for the project.

03/02/05

Discussed individual parts and how far we are.
Jeff discussed the control design.
Decided on number of sensors to use for resolution
Motor control circuit needs to completed

03/09/05

The specifications came back from the committee and made amendments for power consumption and battery life.
Need to have more detailed drawing
In these include belt drive
Come up with a customized circuit for rear wheel motor control
Also include diagram of mounts
Create a memory map

Car will be modeled after an Indi car
Animation is complete.

03/16/05

Spring Break – no meeting

03/23/05

Pulse width done for servo motor to control the front wheel steering
Discussed control loop
Code control algorithms for next time
Sensor board circuitry complete need to put it on a PCB.
Solenoid has been ordered for ping pong ball ejection
Structure of executive program complete.
Waiting for frame to be complete to run and check code
Need to connect encounter and counter chip to determine velocity
Rear wheel motor circuit is complete on PCB.
For next time need to put everything together to make the bot move
Detail drawing completed

03/30/05

Beenu

- Completed drawing and worked on frame

For next week

- Complete PCB mounts
- Design a test procedure with different values for PWM and different duty cycle

Isaac

- Symbol & line sensing

For next week

- Sense symbols & be modular
- Separate for all 9 sensors
- Also describe what the program is doing maybe some equation or flow chart

James

- Looking at solenoids and trying to find a fast reaction one.

For next time

- Order solenoid and write a test procedure for encounter and counter chip.

Jeff

- Worked on construction of frame with Beenu

For next week

- Work on control code
- Apply servo & check steering

04/06/03

Beenu

- completed pcb mounts

For next week

- Design the test procedure for motor
- Draw lines on track

- Sensor board
- Make mounts

Isaac

- Sense symbols
- Error algorithms
- Get servo to work with sensors

For next week

- Get LCD to work
- Integrate code with mechanical design

James

- Solenoid has been ordered

For next week

- Bring solenoid test circuit
- Test the encoder and counter

Jeff

- Worked on mounts for car and hooked up battery

For next week

- Increase frequency to see if it works
- Finish control
- Calculated error

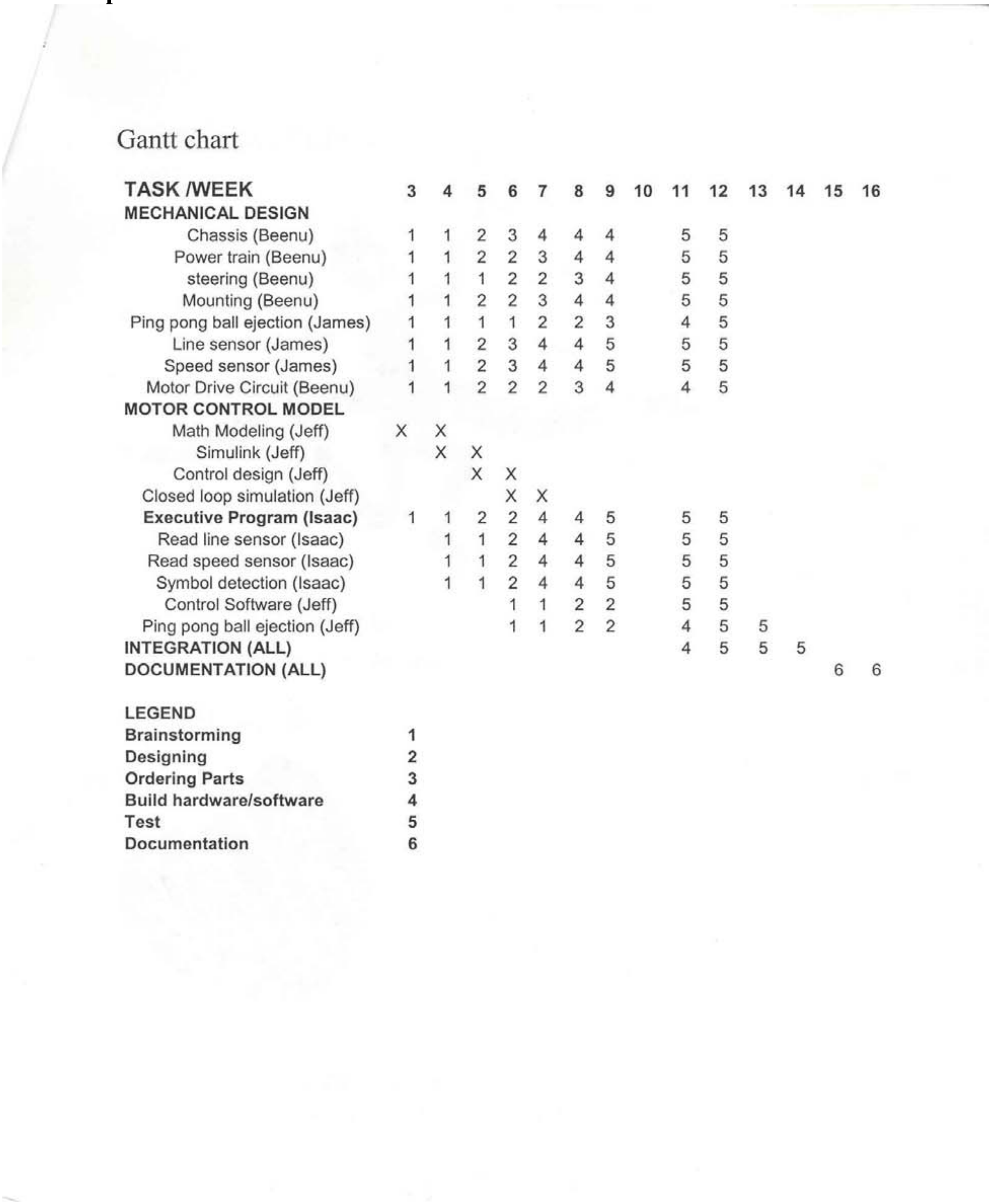
04/13/03

Complete Project

Mark T Hagen

Appendix F – Status Reports

Status Report #1



Gantt chart

TASK /WEEK	3	4	5	6	7	8	9	10	11	12	13	14	15	16
MECHANICAL DESIGN														
Chassis (Beenu)	1	1	2	3	4	4	4		5	5				
Power train (Beenu)	1	1	2	2	3	4	4		5	5				
steering (Beenu)	1	1	1	2	2	3	4		5	5				
Mounting (Beenu)	1	1	2	2	3	4	4		5	5				
Ping pong ball ejection (James)	1	1	1	1	2	2	3		4	5				
Line sensor (James)	1	1	2	3	4	4	5		5	5				
Speed sensor (James)	1	1	2	3	4	4	5		5	5				
Motor Drive Circuit (Beenu)	1	1	2	2	2	3	4		4	5				
MOTOR CONTROL MODEL														
Math Modeling (Jeff)	X	X												
Simulink (Jeff)		X	X											
Control design (Jeff)			X	X										
Closed loop simulation (Jeff)				X	X									
Executive Program (Isaac)	1	1	2	2	4	4	5		5	5				
Read line sensor (Isaac)		1	1	2	4	4	5		5	5				
Read speed sensor (Isaac)		1	1	2	4	4	5		5	5				
Symbol detection (Isaac)		1	1	2	4	4	5		5	5				
Control Software (Jeff)				1	1	2	2		5	5				
Ping pong ball ejection (Jeff)				1	1	2	2		4	5	5			
INTEGRATION (ALL)									4	5	5	5		
DOCUMENTATION (ALL)													6	6

LEGEND

Brainstorming	**1**
Designing	**2**
Ordering Parts	**3**
Build hardware/software	**4**
Test	**5**
Documentation	**6**

TASKS COMPLETED

- Identified sub tasks
- Assigned individual tasks

FUTURE TASKS

- Design of individual task
- Model of control system in Matlab

Status Report #3
Senior Design II

Team 4
Line Following Robot

James Bates
Jeff Ostrander
Isaac Waters
Beenu Pandey

James Bates: Sensor hardware, ping pong ejection, speedometer

Since last report:

- Found a solenoid to use for ping pong ball ejection
- Helped to put car together
- Put together a test apparatus for the line reading sensors
- Found usable steering components for car

To do before presentation:

- Test solenoid for ping pong ball ejection
- Mill sensor circuit on PCB
- Test speed sensing hardware
- Help make the black line on the track
- Look into serial interface to computer for testing and debugging

Jeff Ostrander: Simulink modeling and control

Since last report:

- Most of the machining of car body and axles
- Testing and integration of hardware and software
- Programming
- Finished closed loop and animation and control code
- Decided and bought batteries to be used

To do before presentation:

- Finish integration and testing
- Debug control algorithm

Isaac Waters: Programming all code other than control algorithm

Since last report:

- Helped with building of car
- Testing and integration of hardware and software
- Programming

To do before presentation:

- Finish integration and testing
- Debug programming
- Finalize line and symbol sensing code

Beenu Pandey: Chassis and mounts, steering, motor drive circuit

Since last report:

- Detailed drawings of car and components
- Milling of motor drive circuitry
- Helped with building of car
- Keeps minutes for meetings with advisors

To do before presentation:

- Design and milling of circuitry on PCB
- Help with putting black line on track
- Test and record motor speed values

Gantt Chart

TASK /WEEK	**13**	**14**	**15**	**16**
MECHANICAL DESIGN				
Chassis (beenu)				
Power train (beenu)				
steering (beenu)				
Mounting (beenu)	5	5	5	
Ping pong ball ejection (James)	5	5		
Line sensor (James)				
Speed sensor (James)	5	5		
Motor Drive Circuit (Beenu)				
MOTOR CONTROL MODEL				
Control design (Jeff)				
Closed loop simulation (Jeff)				
Executive Program (Isaac)				
Read line sensor (Isaac)	5	5	5	
Read speed sensor (Isaac)	5	5	5	
Symbol detection (Isaac)	5	5	5	
Control Software (Jeff)	5	5		
Ping pong ball ejection (Jeff)	5	5	5	
INTEGRATION (ALL)	5	5	5	
DOCUMENTATION (ALL)				6
LEGEND				
Brainstorming	1			
Designing	2			
Ordering Parts	3			
Build hardware/software	4			
Test	5			
Documentation	6			

Robot in Testing