4-28-1998

# Micro-Controlled Autonomous Vehicle Group

Matt Fillman
*Trinity University*

Matt Gardiner
*Trinity University*

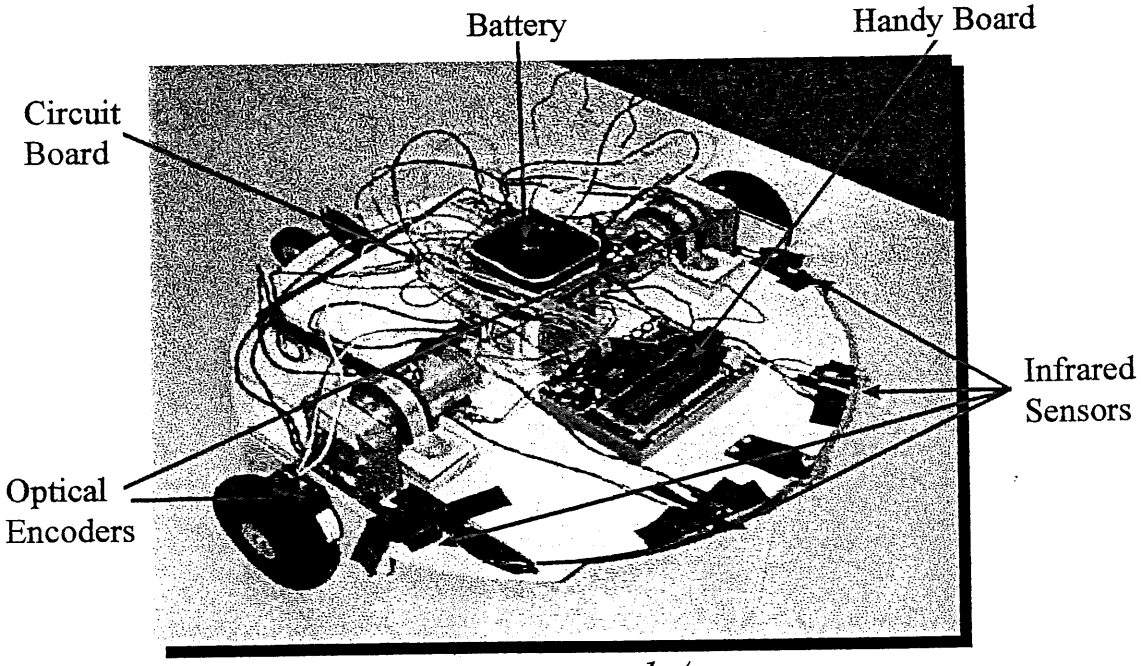
Justin Mackie
*Trinity University*

Kyle McNay
*Trinity University*

# Micro-Controlled Autonomous Vehicle Group

**Matt Fillman**
**Matt Gardiner**
**Justin Mackie**
**Kyle McNay**

**Advisors:** **Dr. Djaffer Ibaroudene**
**Dr. Farzan Aminian**




*www.engr.trinity.edu/~rover*

# Design of an Autonomous Microcontrolled Vehicle

Engineering Analysis and Design VIII

Trinity University

April 28, 1998

Matt Fillman

Matt Gardiner

Justin Mackie

Kyle McNay

Advisors: Dr. Farzan Aminian

Dr. Djaffer Ibaroudene

*Our purpose is to design and build an autonomous microcontrolled land vehicle. It will be able to navigate from a given starting point to a desired location avoiding obstacles in its way. The course we plan to operate in has a flat surface and no more than five static obstacles. Our design considerations are the vehicle platform, motors, type of microcontroller, obstacle avoidance sensors, navigational sensors, and basic program behavior. We chose the Handy Board to control our vehicle, which is based on the Motorola 68HC11 microcontroller. It is a universal board that will meet all our interfacing requirements. Obstacles will be detected using four infrared sensors placed around the front of the vehicle. Four algorithms for completing our goal are discussed. We chose to use a navigation method involving optical encoders to determine distance and direction traveled. This paper describes the design process for our rover, the reasons for the hardware we selected, and how these decisions successfully allowed us to achieve our objectives.*

# Table of Contents

## Table of Figures

## 1. Introduction

The objective of our project is to design and build a microcontrolled land vehicle which will navigate autonomously from an initial starting point to a desired final destination avoiding obstacles in its path. Our project is inspired by and modeled after the Mars Pathfinder Mission. The Mars Sojourner Rover traveled by means of several waypoints to a specific rock or area to be examined. The waypoints were used to provide hazard-free paths to its desired destination, but at times the Sojourner encountered an unexpected obstacle which it avoided on its own. Our vehicle simulates a journey to one of these waypoints.

The terrain we are going to use is much simpler than the surface on Mars. This is done to allow us to concentrate on the actual navigation and obstacle avoidance rather than the vehicle's ability to cross rough terrain. Our course and vehicle specifications are as follows:

1. It must be in a room that has a flat surface and is at least 30'x30', for example Moody 322 or an indoor basketball court.
2. There will be 0 to 5 static obstacles. Each will be a solid color having a length, width, and height no smaller than 2'. They will be spaced at least 3' apart from any other obstacle in the room.
3. The vehicle must complete the course in less than 15 minutes and be within a 3 foot radius of the intended target.
4. The vehicle must be able to fit in a 2' cubed box.

5. The vehicle must weight less than 20 lbs.
6. The total cost of the vehicle must not exceed $500

To accomplish these goals and requirements, we have researched different types of motors, obstacle avoidance sensors, navigation systems, and microcontrollers. Our decisions have been made based on what best meets our goals within our budget.

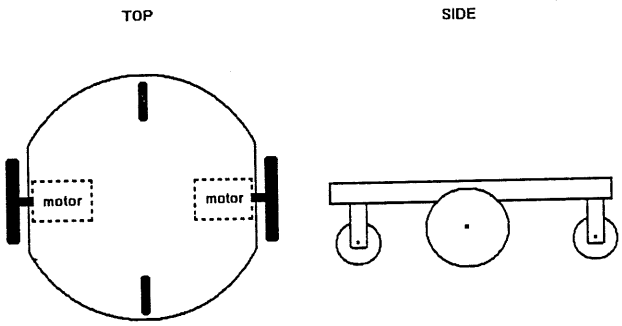
## 2. Research and Results

### 2.1 Platform

We looked at three types of materials to build the platform out of: metal, plastic, and wood. Without having to worry about the stresses caused by rough terrain, we were able to choose a material based on its weight and ease of use rather than its strength and durability. The availability, low cost, low weight and machinability of wood made it the superior choice because both metal and plastic are more expensive and harder to work with than wood. The wood we chose to use is model airplane wood. This type wood meets all our needs and is easily accessible. It is also lighter and easier to machine than the normal plywood.

The base of the platform is circular with a radius of 8 inches and a thickness of a ¼ inch (Figure 1). These dimensions are sufficient to hold and support all the electrical equipment needed to perform the variable tasks required by our vehicle. We chose a circular platform so that no corners could be caught on an obstact as the vehicle turns.

## Figure 1: Platform



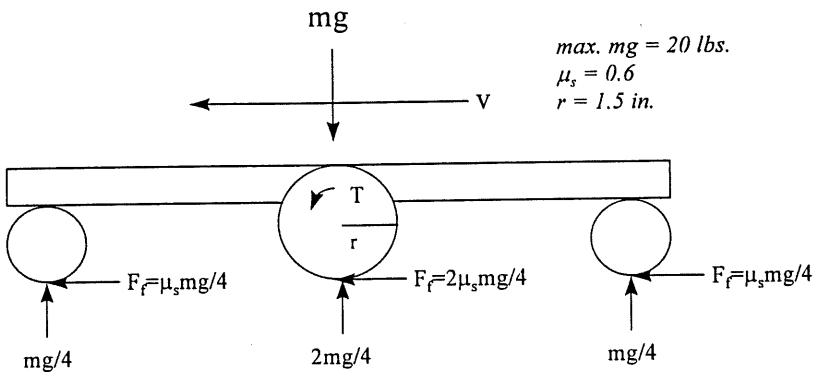

The vehicle is powered by two motors placed at opposite sides of the base. Each motor has independent speed control so that they can be used to steer the vehicle. The front and back wheels rotate freely similar to shopping cart wheels. These caster wheels allow the vehicle to turn sharply by rotating around its center axis. This configuration was selected because it allows the vehicle to make tight turns.

### 2.2 Motors

The first thing to consider when choosing a motor is the minimum torque required by the motor to move the vehicle. Since the vehicle is traveling on a flat surface without any obstacles to drive over, the static friction is the maximum force acting on the wheels at any given point. The motors must be able to overcome the moment caused by static friction. All the forces acting on the vehicle just before it starts to move are shown in Figure 2. The minimum torque required by the motors must be greater than the sum of the moments around the powered axial. This is shown below in Equation 1.

$$\text{Min. Torque} = \mu_s mgr \quad (1)$$

**Figure 2: Minimum Torque Diagram**

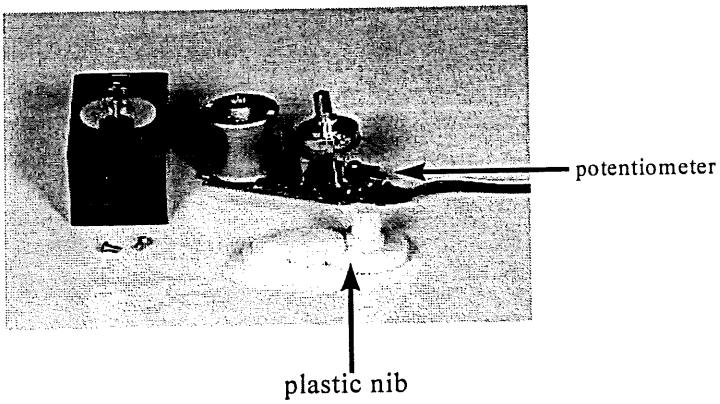



Using maximum values for the static coefficient of friction and weight of the vehicle and a minimum value for wheel radius, the maximum moment the motors have to overcome is calculated to be *288 oz.in.* With a dual motor design, each motor has to produce half the torque *(144 oz.in).* The static coefficient used ($\mu_s$= 0.6) is for rubber tire on cement. It is assumed that rubber on concrete will have a larger coefficient than the rubber on carpet. The upper limit of our weight specifications is used to make sure that the motors can power the heaviest vehicle within the specifications of our project. The two types of motors considered were servo and permanent magnet DC motors. Both are sufficient to power the vehicle, but they have different advantages and disadvantages.

The advantages of servo motors are they come already geared and are easier to attach wheels to. Servos come as a complete unit including a motor, gears, potentiometer, and a universal attachment on the axle—all at a relatively low price. Their disadvantage is that they do not have full rotation—they are usually designed to work within a 60° span. In order to obtain full rotation from a servo, it must be modified.

All the components must first be disassembled from the casing (Figure 3). Next, the plastic nib has to be removed from the gears. The nib prevents the gears from turning completely. Then, the potentiometer needs to be cut away from the motor allowing the motor to be interfaced with the microcontroller. Finally, the gears and motor are reassembled in the casing to produce a continuos geared motor (Jones 185).

**Figure 3: Servo Motor (Jones 187)**

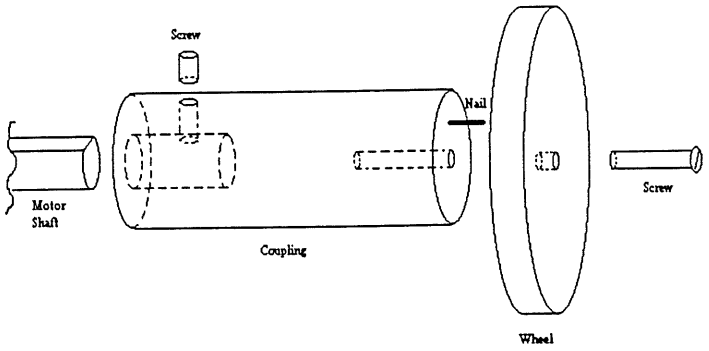



The major disadvantage of permanent magnet (PM) DC motors stems from the fact that most of them do not come with gears. To obtain the required torque, a gearbox must be purchased. In addition, attaching wheels or gears to the axle of a PM motor is more difficult because PM motors do not come with a universal attachment on their shaft. All they have is a flat side on the shaft to attach the gears or wheels to. Making solid contact is possible, only more difficult than with a servo motor. The advantage of a PM motor is its versatility. It does not come with unneeded components and it has continuous rotation, making it easier to use for various tasks. Gears, wheel attachments, and other add-ons can be purchase to meet specifications.

In searching for a motor, we came across PM motors with gearboxes included. These motor have the advantages from both types of motors we were looking. They have the full rotation of a universal PM motor and come with a complete gearbox included, like the servo motors. Purchasing these motors used lowered the cost to that of the servo motors, making the geared PM motor the better choice for our vehicle. The only problem with these motors is they do not come with a shaft interface.

To overcome this problem, we designed a costume coupling to interface the wheels to the motor shaft. It was constructed out of aluminum to insure it could support the weight of our vehicle (Figure 4).

**Figure 4: Wheel to Motor Shaft Interface**




A small screw is used to attached the coupling to the motor shaft. The screw is tightened on the flat portion of the shaft, preventing any slipping between the coupling and the shaft. On the other end of the coupling, a threaded hole was made so that the wheel could be attached by simply screwing it on. The problem with the wheel-coupling interface was that it did not prevent slippage between the wheel and the coupling. To solve this

problem, a nail was fastened to the coupling. The nail fit in the spokes of the wheel, preventing the slippage between the wheel and coupling.

## 2.3 Sensors

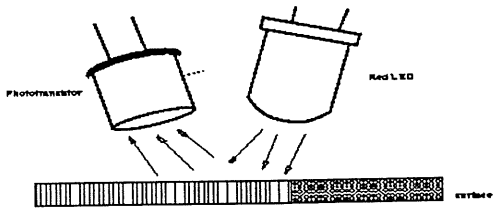
Sensors provide the rover with information about its surroundings. They can be used as feedback for control of the rover's movement and status. What follows is a description of the sensors considered for use on our rover.

### 2.3.1 Optical Encoders

The distance the rover travels will be determined by counting the number of revolution of the wheels. This can be achieved using reflectance sensors or a breakbeam sensors. They both utilize light to count the revolutions of the wheels.

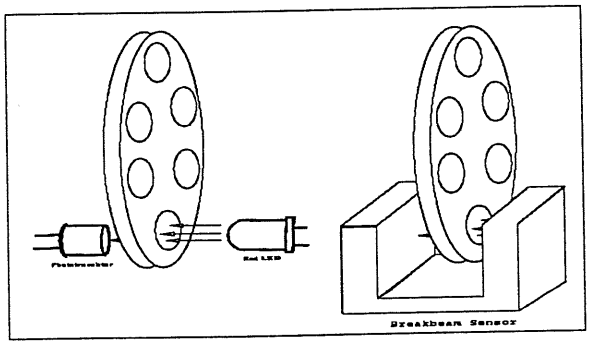
Reflectance sensors consist of an infrared or red LED and a phototransistor that is sensitive to the wavelength of the light emitted from the LED (Domsch 112). As light is bounced off different colored materials, the wavelength changes accordingly allowing the phototransistor to distinguish between colors (Figure 5). If a wheel is evenly divided into black and white segments, the phototransistor can pick up the contrast in color as the wheel rotates. The wheel revolutions can be then calculated from the number of times the phototransistor senses a change in color.

**Figure 5: Reflectance Sensor (Domsch 113)**



Another way to count wheel revolutions employs breakbeam sensors. Breakbeam sensors work by detecting interruptions in the light (Domsch 114). The photosensor searches for direct light instead of reflected light. When the beam is broken, it is detected by the photosensor, similar to the phototransitor detecting the change in wavelength. To determine wheel revolutions, the wheel must be divided into a known number of segments alternating from transparent to opaque. Each time the beam shines through the wheel, the sensor detects it. Given the number of times the beam passes through, the wheel revolutions can be calculated. This technique, called shaft encoding, is shown in Figure 6 below.

**Figure 6: Shaft Encoding with Breakbeam Sensor (Domsch 117)**



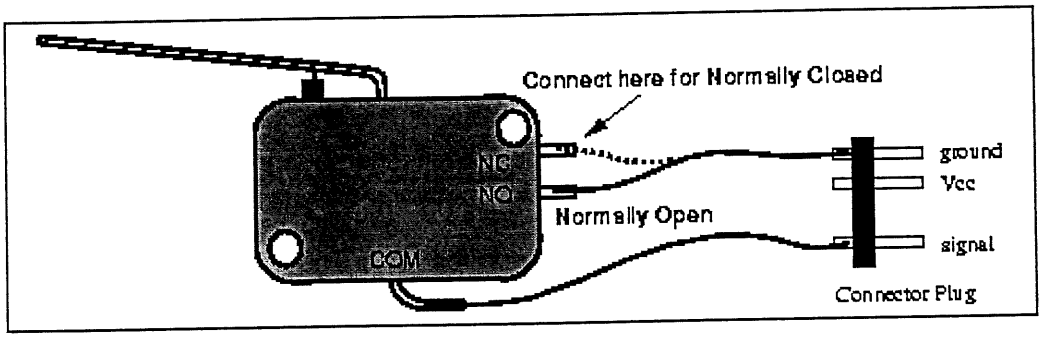
### 2.3.2 IR Sensors

Infrared sensors can be used for obstacle detection. They have a relatively short range, generally less than a foot. An infrared sensor emits IR light. If an object is near, some of the light will reflect off the object, back to the sensor. This will trigger the sensor output to go from high to low—allowing the microcontroller to detect if any object is in front of the sensor.

### 2.3.3 Microswitches

Microswitches are simple mechanical devices that produce a signal when they contact an object. If the switch comes in contact with an object, the switch output will go from high to low, or vice versa. By stopping the motor with a bump switch signal, some of the strain on the drive train caused by a collision will be averted. Although the switch is arguably one of the simplest components, it can cause glitches if polled rapidly. As the switch is depressed, the digital input may have a transient oscillation. We must debounce the switches or set a low polling rate to avoid a glitchy signal. Figure 7 shows a microswitch assembly.

**Figure 7: Microswitch Assembly (Domsch 99)**




### 2.3.4 Ultrasonic Transducer

An ultrasonic transducer can be used to determine the distance to the nearest object. To collect this distance data, the transducer emits an ultrasonic pulse. The pulse will travel to the nearest object and then reflect back to a collector. The pulse travels at the speed of sound, so by finding the time for the pulse to travel to the object and back, the distance can be calculated.

Polaroid Corp. manufactures an ultrasound transducer called the 6500 Series Sonar Ranging Module. This device has a range of six inches to 35 feet. The accuracy is stated to be within 1% of the actual value. The rangefinder comes as two components: a collector/emitter and a controller. The rangefinder does not perform the distance calculations. It sends an output when it emits a pulse and sends another output signal when it receives the pulse.

Our group created an assembly program to interface with the sonar rangefinder. The program was written for the Motorola 68HC11 Evaluation Board (EVB) and loaded on to the board using the BUFFALO monitor. The program is designed to calculate the

distance to an object from the rangefinder's output. To do this, the EVB was programmed to wait for the signal indicating that a sonar pulse was fired. Once this is received, the EVB begins an internal clock. The clock is set to a count cycle of two microseconds. The EVB continues to count until it receives a signal verifying that the pulse has returned to the transducer. When this signal is received, the clock is stopped. Taking the elapsed time from the clock and multiplying it by the speed of sound will produce the distance to an object. The transducer assembly code used for the EVB is given in Appendix A.

The rangefinder can be used to accurately determine the distance to a wall. When the rangefinder is attached to the rover, this information can be used to calculate the vehicle's position in a room. The rangefinder can also be used to determine the distance to an obstacle. These functions can allow the rangefinder to have a role in the navigation system.

### 2.3.5 Compass

We considered the use of an electronic compass in order to determine absolute heading. The Vector-2X by Precision Navigation Inc. is a compass that we considered. It is based on a relatively new technology, the magnetoinductive compass (Everett 1995). It consumes ten times less power than traditional compasses and is cheaper due to fewer signal processing electronics. It has an accuracy of 2 degrees, runs off 5 volts with low current draw, and has digital inputs and outputs for easy interfacing with other electronics.

The Vector-2X measures magnetic north, assuming a uniform magnetic field surrounds it. However, we intend to test the rover in a classroom. Compasses can be

sensitive to metallic objects in the building walls, furniture, machinery, electrical wires, and even the rover itself. The worst case scenario is that the magnetic field is completely irregular and the readings will be different all over the room. According to Walter Stockwell, an Applications Engineer for Precision Navigation, less than a 4 degree error was observed in tests around a fairly crowded office, as long as the compass was a few feet away from large metallic objects (Stockwell 1997). Therefore, the compass accuracy falls within our design specifications.

# 3. Navigation Methods

An algorithm that will guide the rover from its initial position to its destination must be implemented. Any intermediate obstacles should be detected and avoided. The vehicle's initial position and the destination point are important parameters in the rover navigation algorithm. We considered four rover navigation methods. These methods include Dead Reckoning, Localization with Dead Reckoning, Sonar & Compass algorithm, and Sonar & Compass & Optical Encoder algorithm.

## 3.1 Dead Reckoning

Dead Reckoning is defined as finding position based on the starting point and the distance traveled. When the distance traveled relative to the starting point is known, the remaining distance to the destination can be computed. The distance traveled can be calculated by counting the number of wheel revolutions. Optical encoders can be used for this purpose. They also can be used to determine the direction in which the vehicle is heading. By keeping track of the number of revolutions the wheels make when moving in

opposite directions, the heading can be calculated. If an obstacle is detected, a waypoint will be calculated so that the vehicle will travel around the obstacle. The rover will then change its heading and go to the waypoint. At the waypoint, a new course will be determined so that the rover will reach its destination.

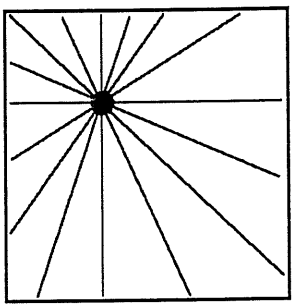
Dead Reckoning's main advantage is that the computations for this method are simple (Borenstein 1995). A disadvantage of dead reckoning is that there may be inaccuracies may be caused by non-ideal vehicle behavior. As the rover travels, slip between the wheels and the ground may occur causing inaccuracies in the distance traveled. Also, when the rover turns, it will be difficult to stop at the exact calculated angle. This is due to the lack of precision in the motors and in slight differences in the radius of each wheel from the middle of the rover. These two factors may result in errors in the heading and distance traveled. Since there is no way to reference or orient the rover's position, the errors could accumulate, growing without bound. The more the rover moves, the larger the errors may grow.

### 3.2 Localization with Dead Reckoning

Because of these inherent inaccuracies, dead reckoning alone may not be an effective means to navigate our rover. However, by combining this method with an ultrasonic rangefinder, cumulative errors can be eliminated. This type of navigation is called Localization with Dead Reckoning. With a sonar rangefinder, the distance from the vehicle to each wall can be determined. To implement sonar rangefinding, a sonar is attached to a servo to allow it to rotate. While the sonar transducer is rotated, ultrasonic

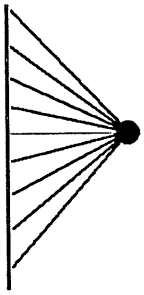
pulses are emitted and collected. From the collected sonar pulses, we can compute the distance to the nearest barrier (see Figure 8).

**Figure 8: Sonar Scan of a Room**



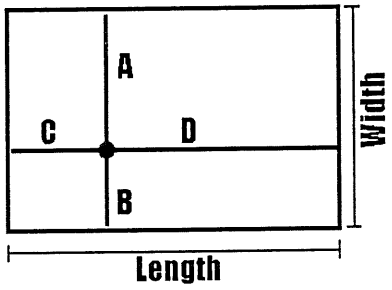
The circle represents the rangefinder and the lines represent the sonar rays. To determine the distance from a wall, it is necessary to determine where the sonar ray is perpendicular to the wall. To find this, a pattern must be found in the distances. The sonar rays values must decrease towards a minimum value and then begin to increase. The minimum value represents the distance when the sonar ray is perpendicular to the wall. This is illustrated in Figure 9. The shortest line, shown in gray, represents the sonar ray that is perpendicular to the wall. This process is applied to all four walls.

**Figure 9: Finding a Perpendicular Sonar Ray**



To confirm the distance to a wall, the rover will calculate the distance between two opposing walls and compare that distance to the known room dimensions. If the distances match within a reasonable tolerance, the rover will know its exact position in the room. This is illustrated below in Figure 10.

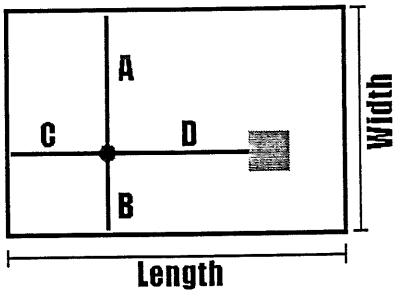
**Figure 10: Confirmation of Position**




**Width = A + B**
**Length = C + D**

If the calculated distance between walls does not match the room dimensions, then the rover will know that an obstacle is obstructing the path to the wall. This case is illustrated in Figure 11. If an obstacle is in the way, the rover will move and rescan the room until no obstacle obstructs the perpendicular distance to all four walls.

**Figure 11: Detecting an Obstacle**




**Width = A + B**
**Length > C + D**

The sonar positioning method can be combined with dead reckoning to produce an accurate navigation system. When the rover is traveling to a point, it can stop periodically to check its position. If there is any error, the rover can recalculate it course, thus eliminating compounding errors.

The localization with dead reckoning method has several advantages. The accuracy is good because cumulative errors are eliminated. The sensors for this method are reasonably priced at around $80. A disadvantage of this method is that it is complex. Data collected must be analyzed before it can be in a useable form. This navigation method is also slow. The entire room must be scanned in order to determine the position. The rover must also stop periodically stop to confirm its position.

### 3.3 Sonar and Compass Algorithm

In the Sonar and Compass (S & C) algorithm, feedback from the electrical compass is used to set rover bearing. Data from experimental runs is used to determine what distance to travel instead of dead reckoning the distance. Rover average velocity is

determined by experimentation. We can then attempt to estimate how long to run the motors to travel a given distance. Once the destination is reached, sonar will find the new coordinates. The rover will decide whether it is close enough to the destination, or repeat the process. Advantages of the Sonar and Compass algorithm include: (1) It is independent of vehicle history, (2) Navigation error is bounded. Disadvantages are the following: since we are estimating how long to run the motor, the algorithm will converge more slowly to the destination. Furthermore, the motor/battery combination has poor repeatability. As the rover's batteries weaken from use, the motors will receive less power. As the motors wear they will produce more heat and less mechanical energy. Therefore, the rover speed will not be constant. Consequently, we can not accurately estimate the distance traveled from the motor run time.

### 3.4 Sonar, Compass and Optical Encoder Algorithm

The Sonar, Compass and Optical Encoder Algorithm is the same as the S & C algorithm except for how the rover travels a given distance. The difference is that optical encoders allow for precise computation of the distance traveled. This algorithm is more accurate than the S & C algorithm because distance is accurately measured with optical encoders. The primary disadvantage is its higher costs and complexity. This is because it requires three sensors. All other algorithms employ two sensors.

## 4. Algorithm Selection

The four algorithms we considered were evaluated based on the following design criteria: accuracy, cost, speed, and ease of implementation. Accuracy means we are

confident the rover will reach the destination using the given algorithm. Dead reckoning received "fair" rating because error may occur due to inaccurate encoders and imprecise construction of the vehicle's wheel axis. As shown in Figure 12, the three algorithms that fit within our budget received "good" for cost; the sonar & compass algorithm received "fair" because it would require a larger portion of our budget. For the speed criterion, we estimated the time each algorithm would take to reach the destination. We judged dead reckoning to be the fastest. "Ease of implementation" criterion was based on the number of sensors and complexity of the program needed to run the algorithm. The Sonar, Dead Reckoning and Compass algorithm requires more than two sensors so it received a rating of "fair," also any algorithm involving sonar was given a "fair" rating due to the complexity of the program needed to run the sonar.

**Figure 12: Algorithm Choice Based on Four Criteria**

| | Dead Reckoning | Sonar & D. R. | Sonar & Compass | Sonar & D. R. & Compass |
|---|---|---|---|---|
| **Accuracy:** | Fair | Good | Good | Good |
| **Cost:** | Good | Good | Fair | Poor |
| **Speed:** | Good | Fair | Fair | Fair |
| **Ease of Implementation:** | Good | Fair | Fair | Fair |

### 4.1 Navigation Method

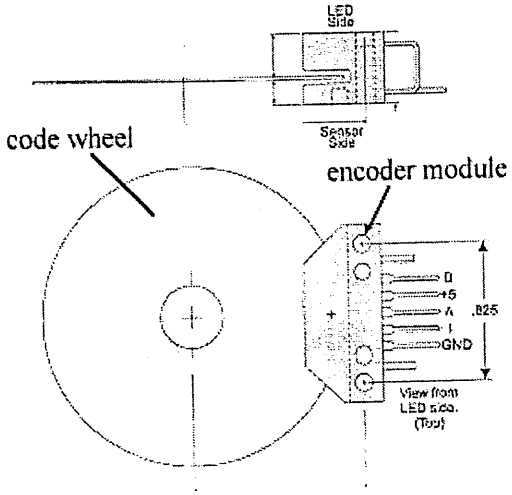
After examining the different navigation methods, we decided to use the dead reckoning algorithm. We made this selection because it had a low cost and was the least

difficult to implement. The main disadvantage with dead reckoning is inferior accuracy. However, we felt that for the distances required in this project, dead reckoning would be reasonably accurate. With the purchase of high resolution encoders, we hoped to lower the error enough, such that our vehicle would be able to perform all its tasks and reach its final destination within our specifications. Because of the dead reckoning attributes, we felt it would be the most effective algorithm. If this project needed to be expanded to greater distances, such as more than 100 feet, then dead reckoning would probably not be a reliable navigation system.

### 4.1.1 Dead Reckoning Setup

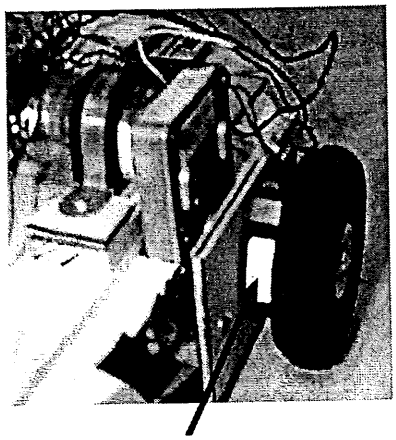
For dead reckoning, we need two shaft encoders, one for each of the two powered wheels. The encoders we used were model E3 ordered from US Digital. The encoder consists of two main parts, the code wheel and the encoder module. The code wheel is a transparent plastic disk that has 500 black marks around the edge. The code wheel is mounted directly to the motor shaft, so that the code wheel will turn as the motor shaft turns. The encoder module is attached to the rover and is placed so that the outer edge of the code wheel passes through the module. This assembly is shown in Figure 13.

**Figure 13: Encoder Assembly**




When a mark on the code wheel passes through the encoder module, the module changes its output signal from high to low. As the motor shaft rotates, the encoder module's output is a square wave. This output was attached to one of the Handy Board's analog input ports. The mounted shaft encoder is shown below in Figure 14.

**Figure 14: Mounted Shaft Encoder**



Encoder

Using an encoder driver, we detected when the encoder module made a transition in its output. With this setting and the code wheel with 500 marks, we had a resolution of 1000 counts per revolution. With the vehicle's 3.5 inch wheels, we calculated that 3.58 counts will move the rover forward one millimeter. Through experimentation, we determined that 3.71 counts per millimeter produced more accurate distance readings.

### 4.1.2 Shaft Encoder Feedback

While the rover is moving and the encoders are counting, the encoder driver can determine the rover's velocity. This information was used to create a feedback loop. When traveling to a point, we programmed the rover's left wheel to remain at a constant speed. Then by checking the two wheel velocities, our program would tell the right wheel to either slow down our speed up. The left wheel velocity was held at a constant to prevent an unstable feedback loop.

### 4.1.3 Turning the Rover

To turn the rover we use a differential steering method by spinning the two motors in opposite directions. To calculate the angle to turn to, we use the following formula:

$$\Theta_{abs} = \frac{\Pi}{2} - \arctan\frac{Y - Y_{dest}}{X - X_{dest}} \quad (2)$$

In this equation $\theta$ is the current angle of the rover with respect to the y-axis, $\theta_{abs}$ is the new angle, $X_{dest}$ and $Y_{dest}$ are the destination coordinates, and X and Y are the current rover coordinates. The sign of $\theta_{abs}$ must be adjusted depending on the direction the rover is facing. The angle $(\theta - \theta_{abs})$ gives the value for the rover to rotate to. Our program then converts this angle into a number of encoder counts. The motors will then turn in opposite directions until the encoder counts matches the calculated value. Because of the velocity feedback, the wheels will spin at the same speed and so the rover will pivot about its center point. This has a slight error, so that the more the vehicle turns, the more the uncertainty in rover heading. Refer to Appendix E for more information about this function.

### 4.1.4 Shaft Encoder Difficulties:

When testing our dead reckoning navigation, we found that the encoders were not sending any output. We later determined that this was due to an encoder alignment problem. When the shaft encoders were mounted, we placed the rover on a platform so that there was no force on the wheels and motor. However, when the vehicle is traveling on the ground the weight of the rover causes the motors and wheels to shift very slightly. The encoders are very sensitive to their alignment, and this shift caused the encoders to no longer work properly. To correct this problem, additional wooden supports were added to stabilize the motors. After adding thesupports, the encoders worked properly.

## 4.2 Obstacle Avoidance

For our vehicle's obstacle avoidance sensors, we selected infrared (IR) sensors. These sensors have a low cost, are relatively easy to implement, and can detect an obstacle before the rover collides into the object. Bump sensors were not used because they required the rover to run into the obstacle, which was undesirable. Sonar was not used because of its high cost and complexity to implement.

### 4.2.1 Infrared Sensor Setup

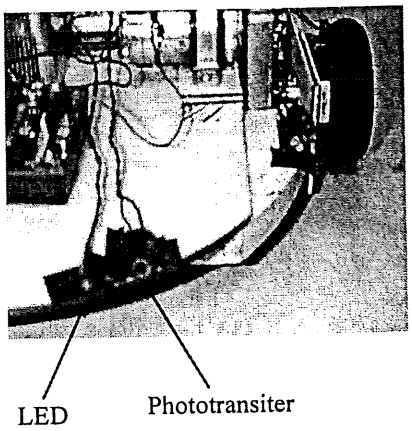
We ordered infrared sensors from Optek. These IR sensors have two components: an IR emitter and a phototransistor. To setup the sensor, a resistor was placed between the five-volt power supply and the emitter. This was done to reduce the amount of current flowing through the emitter and prevent a burnout. Another resistor was added between the phototransistor and the power supply in order to adjust the sensitivity of the phototransistor.

The phototransistor is capable of receiving IR signals up to six feet away, but a one-foot range is desired. When an obstacle is detected, a one-foot range will allow the rover enough room to come to a stop and turn to avoid the object. To adjust the range of the detected infrared light, a potentiometer was placed in series with the phototransistor. By adjusting the potentiometer value, we found the resistor value that yielded a one-foot range. We then replaced the potentiometer with a resistor that had the same resistance. For our specifications, we found that the resistor in series with the phototransistor should

have a value of 610 kΩ and the resister in series with the emitter should have a value of 50 Ω.

Once the proper resister values were known, we attached the infrared sensors to the vehicle. A total of four sensors were used; one sensor was in front of each of the two powered wheels and two sensors were placed on the front of the rover. Figure 15 shows the two IR sensors on the left half of the vehicle.

**Figure 15: Placement of IR Sensors**



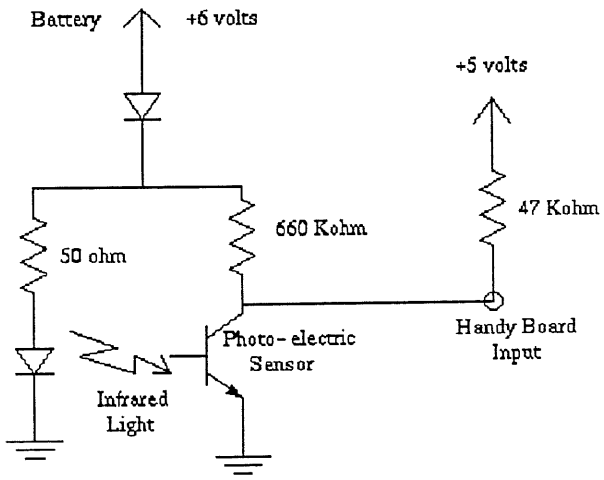

Since the rover never moves in reverse, it was not necessary to attach any sensors to the back.

### 4.2.2 Infrared Sensor Difficulties

When testing the IR sensors, we discovered that the phototransistors were very sensitive to ambient lighting. To help reduce false readings, shielding was placed around the sensors. The shielding consisted of black poster board wrapped around the sensor and held together with electrical tape. Although this is not a very sophisticated shield, it was

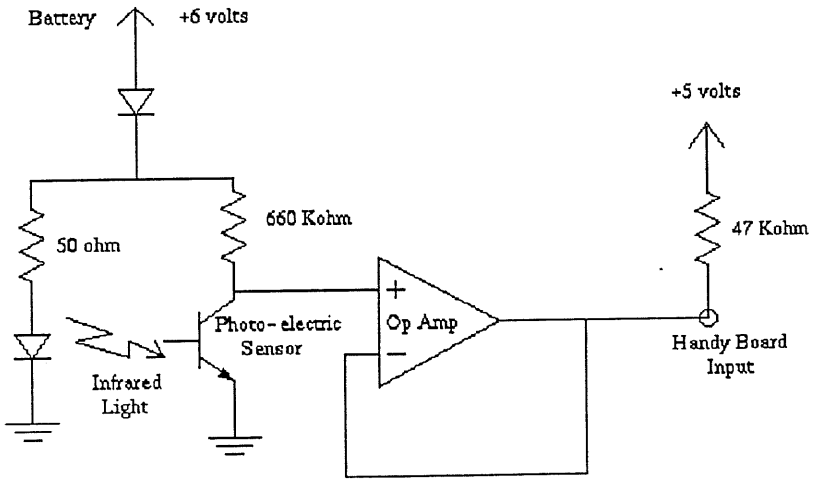
adequate for our purposes. We then attached the IR sensors to the Handy Board, which has an internal 47 kΩ pull up resistor. Our circuit diagram for this is shown below in Figure 16.

**Figure 16: IR Circuit Diagram**




When the Handy Board-interfaced circuitry was tested, the IR sensors no longer functioned properly. This was because the 610 kΩ resister for the phototransistor was now in parallel with the 47 kΩ internal resister for the Handy Board, lowering the total resistance to 44 kΩ. To correct this, an operational amplifier was placed between the sensor's output and the Handy Board's input. This is shown below in Figure 17.

**Figure 17: IR Circuit Diagram with Op Amp**




The op amp was the 324 model purchased at Radio Shack. The op amp acted as an interface buffer, so the voltage from the IR sensor was passed directly to the Handy Board. Once this was implemented, the IR sensors accurately.

# 5. Controllers

## 5.1 Microcontrollers and Robot Controllers

The microcontroller is the brains of the rover. It can be programmed to handle inputs, make decisions and drive outputs in the manner desired by the designer. Microcontrollers have 4 main parts: the control unit, the arithmetic logic unit (ALU), memory, and input/output (I/O) ports. Some of the more popular controllers available on the market are the Motorola 68XX series, Intel 80C186, 8051, Microchip PICs and Parallax BASIC Stamps. The main differences are in the power of the processor and control unit, the size of the onboard memory, and the number and type of I/O ports.

These microcontrollers are not application specific. Additional circuitry must be added so that it is suitable for our purpose. For the rover, we must have motor drivers and at least 32K memory. There are a number of controllers available with this circuitry all on one board, hereafter referred to as robot controllers. In addition to the requirements mentioned, these boards facilitate access to the I/O ports, include an RS-232 connector to the PC and often bundle a high-level programming language such as C. Some of the programming will be done in Assembly for specific routines in order to ensure efficient processor usage. However the majority will be in C—a higher-level language is needed to code complex algorithms. The programs will be written on a PC and then downloaded to the rover for testing during the building phase.

Some of the more popular robot controllers are the Mini Board, Handy Board and Finger Board, all derived from the well known MIT Robot Competitions. Others include the F1 Board, BOTBoard and ModCon. Appendix B contains a comparison of the robot controllers under consideration. They cover a range of capabilities and prices, and they are all based around the Motorola 68HC11 microcontroller.

### 5.2 Robot Controller Design Requirements

Our design requirements of the robot controller are as follows. It must have a means of driving two separate DC motors. It must have at least 4 digital inputs for the IR sensors and 2 analog inputs for the optical encoders, but more are desirable. The processor should be fast enough to handle our algorithms and sensor requirements, at least 1MHz. It also must have 32K RAM, preferably battery-backed and with the ability to expand to 64K. It should have low power consumption, be small and lightweight, and

must cost under $300. It must have an RS-232 port for easy interfacing with an IBM-compatible computer, and bundled C programming software that is guaranteed to work with the board. It must also be a well supported product and preferably have a large user base.

The robot controller we have chosen to purchase is the Handy Board by the Media Laboratory at MIT (Martin 1997). It satisfies all our hardware requirements and also leaves room for expansion under subsequent projects. It is well supported and has a large number of users we may contact in case we have questions. This will aid in troubleshooting both hardware and software problems. The software that is included is Interactive C, which is specifically designed for robotic use. It includes the useful ability to execute code on the microcontroller line by line to help in debugging. The Handy Board also has an LCD that can display error messages or status during operation, and a NiCd power supply is included. It is more expensive than some of our other options, but it is important that we know the hardware is likely to function as specified.

## 6. Software Implementation

The rover functions in an unknown environment, and the only information it is given is its starting coordinates and heading, and the coordinates of the destination. In the absence of obstacles in its path it takes the most direct route to the destination. When the rover does detect an obstacle it carries out a preset sequence of maneuvers designed to most easily get the rover around the obstacle, and then it continues. The sequence we created was based on the course criteria, in particular that obstacles must be a minimum of three feet apart.

The programming was done in Interactive C, a multitasking version of C written expressly for use with the Motorola 6811 and geared towards robotics projects. Assembly language routines for reading the shaft encoder values[1] and improved pulse width modulation routines by Julian Skidmore were used. This allowed 100 motor power increments as opposed to the original eight, which was inadequate for motor feedback control.

There are five initial parameters given to the rover: the (x,y) coordinates of the rover and destination, and the rover's heading. The origin and axes of the coordinate system can be set up at an arbitrary position, maximizing flexibility. For instance the x and y axes can be the walls of a room or centered on the rover. The heading is the angle of the forward direction of the rover with respect to the y-axis. These parameters are defined in the program code and must be downloaded to the rover whenever they need to be changed.

[1] http://el.www.media.mit.edu/groups/el/projects/handy-board/software/encoders.html

The following is a simplified description of the important components of the program. The first thing the program does is check if it is at the destination (refer to Fig. 18). If the rover's coordinates are within a set distance of the target then it considers itself close enough. If it is not there yet then it calculates the straight line distance and angle needed to reach the destination. The current $(x,y,\theta)$ position parameters of the rover are stored in global variables, along with the angle and distance needed to turn and move, respectively. The rover turns towards the target by spinning around its central axis. The crucial element of the spin function is code that, using feedback, keeps both wheels spinning in opposite directions at the same speed. This is necessary to ensure the rover spins as close about its center as possible. The rover then proceeds to move towards the target (refer to Fig. 19). While it is moving the program continuously ensures that the wheels are at the same velocity, and it also polls the IR sensors for obstacles. If the rover detects an obstacle, a flag is set indicating the obstacle position (either front-left or front-right). The rover will stop moving and update its coordinates if either an obstacle is detected or it reaches its desired travel distance. If an obstacle is detected then the program executes the avoid-obstacle routine, which consists of a sequence of spin and move operations. If an obstacle is detected on the front-right, the rover will turn left 90 degrees, move forward three feet, turn right 90 degrees and move forwards three feet. The mirror image movement occurs if an obstacle is detected on the front-left. Note that the rover does not currently poll for obstacles as it executes the obstacle avoidance maneuver: it assumes none will be encountered due to the course setup. Once the rover is done moving, it checks again if it is at the destination. If it is not then it repeats this whole process again. Refer to Appendix E for our C code.

Figure 18: Flowchart of the overall program architecture

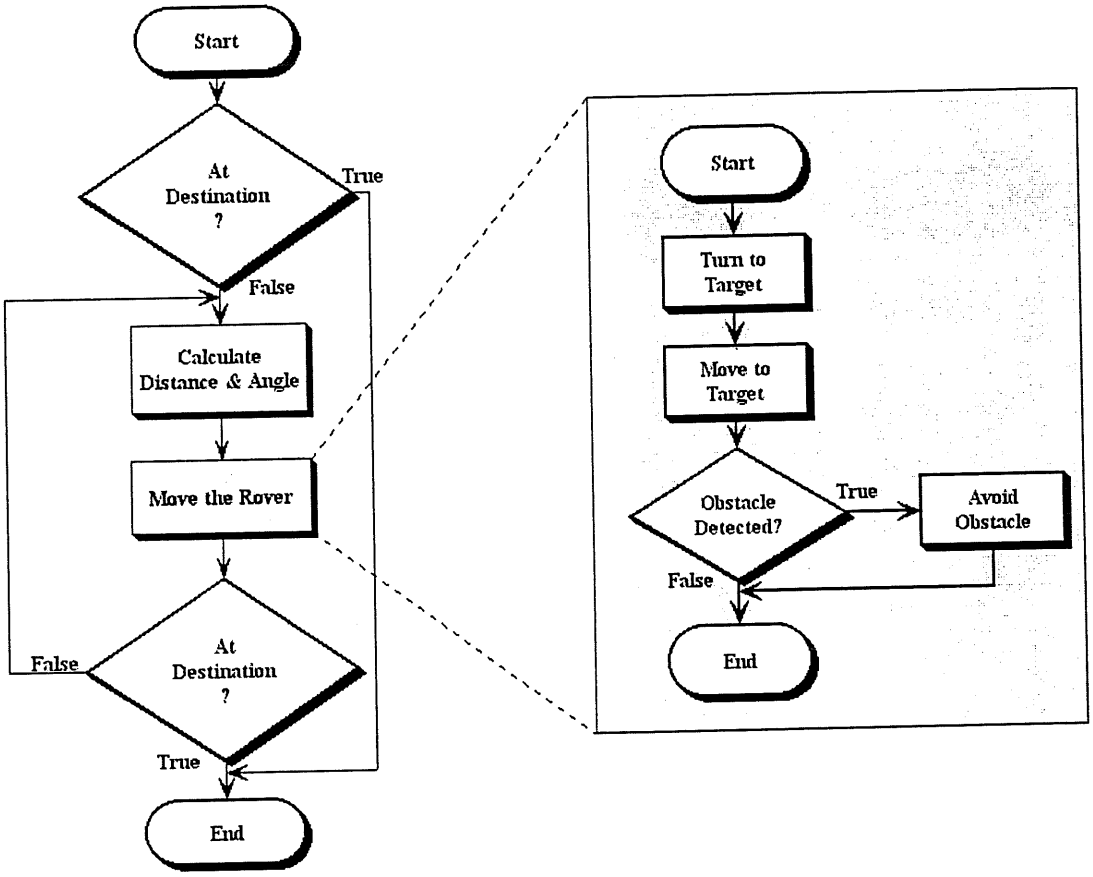
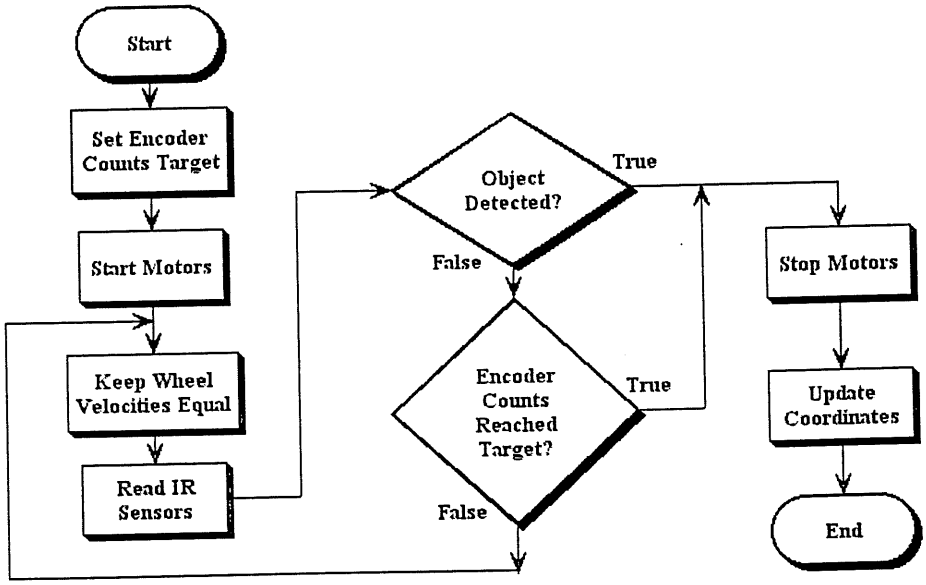



Figure 19: Simplified Flowchart of the "Move to Target" module



We implemented an hierarchical program structure. This inherently top-down approach allows us to break down the needs and operations of the rover into individual components. However it was also necessary to take a slightly bottom-up perspective. This is because the sensors and motors are the most critical and restrictive elements of the system, and their demands needed to be met. This hierarchical structure is usually best at fast calculations at the lowest levels, such as determining wheel velocity, but it is often slower at complex, higher level, decision making tasks. In addition the use of abstraction often removes too much information about the environment to be useful for some higher level decisions (Payton 1990). For instance, it was necessary in our program to pass the location of the obstacle detected by the IR routine to the highest level; with more complex programs more information such as this will need to be abstracted. In dynamic, unstructured environments, this program structure often proves to be too cumbersome and inflexible.

Another approach that was considered was a distributed or behavior based architecture (Rosenblatt 1997). This consists of multiple modules that operate at the same time and share control of the rover. An arbitrator periodically combines the votes of the modules, selects the behavior with the highest priority, and executes their commands. The primary benefit of this approach is that the behaviors are *independent* so they can be changed and new ones added easily. Also it allows multiple goals and constraints to be accomplished at the same time while avoiding bottlenecks that are a problem with the hierarchical approach. However, our course criteria were simple enough that the hierarchical approach to programming was the most direct and efficient

method. Moreover, we already had experience in the top-down approach, whereas the behavior-based one was completely new.

To expand upon the rover's flexibility, there are a number of changes we can make to the program. Currently the rover is limited to one method of obstacle avoidance. This could be expanded, for instance, with additional IR sensors the rover could follow the sides of the obstacle until it has passed it. The rover currently only reacts to its immediate environment. A sensor such as a movable sonar could be used to extend the range of obstacle detection, and then some form of path planning could be implemented. If the environment is even less structured and the rover has more than one set of goals, then we believe the distributed architecture approach would prove more efficient and flexible.

# 7. Results

## 7.1 Rover Specifications

We attained our goal of avoiding five obstacles and getting within a 3 foot radius of the destination:

**Figure 20: Three rover runs with varying numbers of obstacles**

| # of Obstacles | Distance to Target | Error |
|---|---|---|
| none | 6 ft. | 7 in. |
| 1 | 12 ft. | 2 in. |
| 5 | 20 ft. | 16 in. |

The rover is specified to fill less than 2 cubic feet. It is 19 in. wheel to wheel, 17 in. front to back, and 7 inches high, which is 1.3 cubic feet. Thus it fills less than two cubic feet. The rover weighs in at 9 lbs, below our maximum specification of 20 lbs. It reached

the destination well within our time limit of 15 minutes per run for all runs. The rover operates with no tether, and thus is completely autonomous, as intended.

### 7.2 Budget

Initially the budget for our project was $500. A large portion of this was spent on our microcontroller, the Handy Board, which cost $284. Because of the large cost of the microcontroller, we received permission to increase our budget to $600. Figure 21 shows a complete listing of all of our expenses.

**Figure 21**

| Project Expenses | |
|---|---|
| Handy Board | 284.00 |
| 2 Motors | 20.00 |
| Wheels | 9.00 |
| Caster Wheels | 16.00 |
| Platform | 11.00 |
| 2 Shaft Encoders | 150.00 |
| 14 IR Sensors | 25.00 |
| Amplifiers | 5.00 |
| **Total Cost** | **$ 520.00** |

Figure 21 shows that are total expenses were within our $600 dollar budget.

### 7.3 Rover Problems and Limitations

Our project has a number of limitations. These can be subdivided into three categories: (1) Hardware Limitations; (2) Inherent Limitations; and (3) Software Limitations. A hardware limitation is the HB power supply: it limits the rover top speed to 5 in/sec. In order to get a higher top speed we must have a separate power supply for

the motors that provides more current such as the 6 V lantern battery on our rover. Switching to the lantern battery or another power supply would involve severing a part of the HB circuitry. The IR sensors present another difficulty: on occasion, they are triggered by ambient light. Better shielding would reduce the frequency of this problem.

An inherent limitation to the rover is its navigation method: dead reckoning navigation error is unbounded. Thus, our navigation strategy is not feasible for large distances. Tied to unbounded dead reckoning error is systematic dead reckoning error. Systematic error is caused by uncertainty in the effective wheelbase of the rover. It is also caused by slightly unequal wheel diameters. These hard-to-control variables reduce the accuracy of rover turning and straight line motion.

Systematic error is addressed in software to the extent that empirical constants were determined. For example, we changed the number of shaft encoder counts to execute a 90 degree turn until the rover turned exactly 90 degrees. The literature says that systematic error may be reduced by *at least* one order of magnitude by experimentation (Borenstein 1995). This requires a series of experiments in which the rover moves in a square. From experimental data, "correction equations" are derived that guarantee the optimal reduction of systematic error. This would enable our rover to complete its mission in larger areas.

## 8. Conclusion

The rover project has potential for expansion in future Senior Projects. It is a platform for software experimentation in feedback control systems. Since dead reckoning is unsatisfactory for large distances, the next logical step is to give the rover the ability to globally position itself. Coupled with the appropriate sensors, the rover will be able to execute complex tasks, such as landmine or chemical detection.

We successfully met all project specifications. Our rover is able to avoid up to five obstacles, arriving within a three foot radius of the destination. We encountered many unexpected software and hardware problems during the implementation process, including floating point underflows and shaft encoder misalignment. Shrewd decision-making, firm scheduling, and hard work contributed to successful completion of the project.

## References


Borenstein, Johann and Liqiang Feng. "Correction of Systematic Odometry Errors in Mobile Robots." Proc. 1995 Int. Conf. Intelligent Robots and Systems, Pittsburgh, Pennsylvania, p569-574, 1995.

Domsch, M., P. Oberoi and K. Ulland. "MIT 6.270 Course Notes." EECS Department, Massachusetts Institute of Technology.

Dowling, Kevin. "Robotics: comp.robotics Frequently Asked Questions." 23 Oct. 1996. <http://www.frc.ri.cmu.edu/robotics-faq/> (8 Sep. 1997).

Drumheller, Michael. "Mobile Robot Localization Using Sonar." IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 9, No. 2, p325-332, March 1987.

Everett, H.R. *Sensors for Mobile Robots: Theory and Application*, A K Peters, Ltd., Wellesley, MA, 1995.

Harrelson, Kevin. "Everything You Always Wanted To Know About Programming Behaviors But Were Afraid To Ask." Machine Intelligence Laboratory, University of Florida, Spring 1995.

Hwang, Santai and Brian P. Kintigh. "Implementation of an intelligent roving robot using multiple sensors." Proc. IEEE Int. Conf. Multisensor Fusion and Integration for Intelligent Systems, Las Vegas, NV, p763-770, 1994.

Jones, Joseph L., and Anita M. Flynn. *Mobile Robots: Inspiration to Implementation*, A K Peters, Ltd., Wellesley, MA, 1993.

Kelly, Alonzo. "Modern Inertial and Satellite Navigation Systems." Carnegie Mellon University, 1994.

Liu, K., and F.L. Lewis. "Fuzzy Logic-based Navigation Controller for an Autonomous Mobile Robot." Proc. 1994 IEEE Int. Conf. Systems, Man and Cybernetics. Part 2(of 3), San Antonio, p1782-1789, 1994.

Martin, Fred. "The Handy Board." MIT Media Laboratory, 6 Dec. 1997 <http://el.www.media.mit.edu/groups/el/projects/handy-board/>

Martin, Fred. *The Handy Board Technical Reference*, MIT Media Laboratory, 3 Dec. 1997.

Motorola Inc., *M68HC11 Reference Manual*, Phoenix, AZ, 1991.

Oriolo, Giuseppe, M. Vendittelli and G. Ulivi. "On-line map building and navigation for autonomous mobile robots." Proc. 1995 IEEE Int. Conf. on Robotics and Automation. Part 3 (of 3), Nagoya, Japan, p2900-2906, 1995.

Payton, D., Rosenblatt, J. and D. Keirsey. "Plan Guided Reaction." IEEE Transactions on Systems Man and Cybernetics , 20(6), 1990, p1370-1382.

Polaroid Corporation, *Ultrasonic Ranging System*, Cambridge, MA.

Rosenblatt, J. "DAMN: A Distributed Architecture for Mobile Navigation." Journal of Experimental and Theoretical Artificial Intelligence, Vol. 9, No. 2/3, p.339-360, April-September, 1997.

Rosenblatt, J. and C. Thorpe. "Combining Multiple Goals in a Behavior-Based Architecture." Proceedings of 1995 International Conference on Intelligent Robots and Systems (IROS), Pittsburgh, PA, August 7-9, 1995.

Stockwell, Walter. "Vector-2X Question: Indoor Accuracy." Personal e-mail, (24 Nov 1997).

Stone, H. W. "Mars Pathfinder Microrover A Low-Cost, Low-Power Spacecraft." Proc. 1996 AIAA Forum on Advanced Developments in Space Robotics, Madison, WI, August 1996.

Webb, Jeff. "Clyde The Exploring House Robot." Intelligent Machines Design Laboratory, University of Florida, December 1996.

# Appendix A.

```
*******************************************
* THIS PROGRAM MEASURES THE ELAPSED TIME BETWEEN INIT
* AND ECHO FOR THE ULTRASONIC TRANSDUCER.  THE TIME IS MEASURED
* IN CLOCK CYCLES AND STORED AT $0012 AS A TWO BYTE NUMBER.
*
*******************************************
**SYMBOL DEFINITIONS
*******************************************
PORTC       EQU    $1003          I/O PORT C REGISTER
DDRC        EQU    $1007          DATA DIRECTION FOR PORTC
TCNT        EQU    $100E          TIMER COUNTER REGISTER
TIC3        EQU    $1014          TIMER INPUT CAPTURE 3 REGISTER
TCTL2       EQU    $1021          TIMER CONTROL REGISTER
TMSK1       EQU    $1022          TIMER MASK REGISTER
TFLG1       EQU    $1023          TIMER FLAG REGISTETER
TMSK2       EQU    $1024          TIMER MASK REGISTER 2
IOPAT       EQU    $01
* MASKS
BIT0        EQU    %00000001
BIT1        EQU    %00000010
IC3F        EQU    BIT0
*******************************************
*DATA SECTION
*******************************************
    ORG     $0010
INITIME     RMB    2              COUNTER AT FIRST IC3 CAPTURE
ELAPSED     RMB    2              ELAPSED TIME SINCE LAST CAPTURE
DONE        RMB    1              END CONDITIONS
*   ORG     $FFEA                 DISABLED FOR BUFFALO-C3 INTERRUPT VECTOR
*   FDB     IC3ISR
    ORG     $00E2                 JUMP ADDRESS FOR TIC3
    JMP     IC3ISR
*******************************************
*MAIN PROGRAM
*******************************************
    ORG     $C100
*INITIALIZATION
START       LDS    #$CFFF         INITIALIZE STACK
*INITIALIZING PORT C
    CLR     PORTC
    LDAA    #IOPAT
    STAA    DDRC
*INIT TIMER
    LDAA    #$01
    STAA    TMSK2                 SET COUNT PERIOD TO 2 MICROSECONDS
*INIT INPUT CAPTURE 3
    LDAA    #$01
    STAA    TCTL2
*ENABLE IC3 INTERRUPT
    LDAA    #BIT0
    STAA    TMSK1
*INIT ELAPSED TIME
    LDD     #0
    STD     ELAPSED
    STAA    DONE
*SEND INIT
    LDAA    #$01
    STAA    PORTC
*INIT INIT TIME
    LDD     TCNT
    STD     INITIME
*TURN ON INTERRUPT SYSTEM
    CLI
```

## Appendix A. (continued)

```
*****************************************
*MAIN PROGRAM LOOP
*****************************************
HERE        LDAA   DONE          LOOP...
    BEQ     HERE
    SWI                          STOP PROGRAM
*****************************************
*INTERRUPT SERVICE ROUTINE
*****************************************
IC3ISR      LDX    #TFLG1
    BRCLR  0,X,IC3F,RTIC3
*CLEAR IC3 FLAG
    LDAA   #BIT0                 STORE 1 TO CLEAR FLAG
    STAA   0,X
*CALCULATE ELAPSED TIME
    LDD    TIC3
    SUBD   INITIME
    STD    ELAPSED
*SET FLAG DONE TO $FF
    COM    DONE
*RETURN
RTIC3       RTI
    END
```

| Robot Controller | Mini Board | Handy Board* | F1 Board w/Motor | BOTBoard 2 | ModCon |
|---|---|---|---|---|---|
| Dimensions | 84X47mm | **108X80mm** | 118X80mm | 50X750mm | 76mmX? |
| Power Supply Connectors | 5v Regulator (5.6-36v) | **internal 9.6v NiCd** | 6.5-35v | 5v. | 5v |
| Motor Drivers | 4 | **4 DC 2 Servo** | 4 DC 2 Stepper 2 R/C Servos | 4 R/C Servo | 4 R/C Servo. |
| Digital Input | 8 | **9** | 8 | 8 | 8 |
| Analog Input | 8 | **7** | 8 | 8 | 8 |
| Connection to PC | RS-232 | **RS-232** | RS-232 | RS-232 | RS-232 |
| Development Software | C | **C** | C | C | C, Forth |
| EEPROM (Bytes) | 2K | **NA** | 32K | 512 | Unknown |
| RAM (Bytes) | 256 | **32K** | 32K | 256 | 32K |
| Microcontroller | M68HC11e2 | **M68HC11 2MHz** | M68HC11f1 4MHz | M68HC11 | M68HC11 |
| Price (Assembled) | $110 | **$284** | $125 | $25 Unassembled | $157 (+ S&H) |

*** Handy Board also includes: 16x2 LCD, rechargeable NiCd, 2 buttons, 1 knob, 1 beeper, IR I/O**

## Appendix C. Draft Sonar & Dead Reckoning Algorithm (not implemented)

### Alpha 1.15

```
#include<stdio.h>
#include<math.h>

initialize Handy Board
initialize IR polling for dead reckoning
initialize Interrupt Vector(s)
    Obstacle Interrupt:  when IR transducer=1 for n sec, point to function Avoid
                         Obstacle

Main Module
    Call Destination Check
    For Rover not at destination yet
    {
        Call Calculate Distance Vector
        Call Move Rover
        Call Scan Room
        Call Destination Check
    }
    End

Function Destination Check
    Load initial starting position
    Load distance traveled
    Load destination position
    Calculate if rover is at destination
        Set flag True/False
    Return

Function Calculate Distance Vector
    Use trig to calculate rover heading with respect to destination
    Use trig to calculate rover distance with respect to destination
    Return

Function Move Rover
    Initialize shaft encoder
    Enable IR obstacle interrupt
    Turn to New Bearing
    Travel to New Distance
    Return
```

## Appendix C. (Continued)

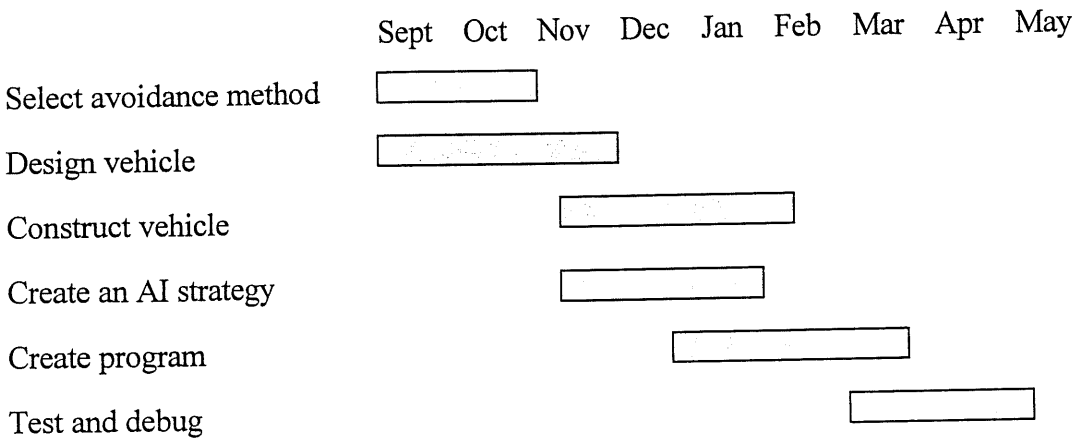
Function **Avoid Obstacle**

Disable IR interrupt vector
Stop Motor
Back up 10 wheel revolutions
Enable IR interrupt vector
Turn Left 90 degrees
Move forward 20 wheel revolutions
Return

Function **Scan Room**

Initialize Sonar Transducer
Set servo to zero degrees with respect to rover
Scan room
Calculate distance to each wall
Align coordinates to room dimensions
Return

## Appendix D. Gantt Chart



# Appendix E. IC Program Code

```
/*****************************************************************************/
/*                                                                           */
/*  Omain.c    ; IC main() function                                          */
/*                                                                           */
/*  by                                                                       */
/*                                                                           */
/*  Matt Gardiner, Justin Mackie, Kyle McNay                                 */
/*  Mobile Autonomous Vehicle Group                                          */
/* (Senior) Engineering Design VIII, ENGR 4382                               */
/*  Trinity University                                                       */
/*                                                                           */
/*****************************************************************************/
/*                                                                           */
/*  VERSION HISTORY:                                                         */
/*                                                                           */
/*  2.0     14 April 1998        ;Final version used in videotape            */
/*                                                                           */
/*  1.X                          ;Improved obstacle avoidance, move and      */
/*                               ;spin functions                             */
/*                                                                           */
/*  1.0     30 March 1998        ;Functioning program that allows Rover to   */
/*                               ;reach specified coordinates given its      */
/*                               ;initial coordinates and heading            */
/*                                                                           */
/*  0.X                          ;Previous versions undocumented             */
/*                                                                           */
/*****************************************************************************/
/*                                                                           */
/*  EXTERNAL FILE DEPENDENCIES:                                              */
/*                                                                           */
/*                                                                           */
/*    Omain1.lis                 ;Contains list of C files needed            */
/*    OMain4.c                                                               */
/*    Omove7.c                                                               */
/*    OCalcDistVect2.c                                                       */
/*    Ospin4.c                                                               */
/*    OAvoidObstacle4.c                                                      */
/*    ODestinationCheck1.c                                                   */
/*    Ocheck_IR2.c                                                           */
/*    ORovabs.c                                                              */
/*    fencdr0.icb                                                            */
/*    fencdr1.icb                                                            */
/*                                                                           */
/*****************************************************************************/

/*****************************************************************************/
/*                                                                           */
/*                            #define declarations                           */
/*                                                                           */
/*****************************************************************************/

#define PI 3.1415926535898
#define X_DESTINATION  0.0            /* Destination coordinates in mm */
#define Y_DESTINATION  6096.0         /* 6ft=1828.8mm, 12ft=3657.6          */
#define X_ROVER_START  0.0            /* Rover initial coordinates in mm    */
#define Y_ROVER_START  0.0
#define THETA_ROVER_START 0.0         /* Rover initial heading in radians   */
#define AVOID_OBSTACLE_DISTANCE 914.4 /* 3 feet in mm                  */
#define COUNTS_PER_MM 3.71            /* Experimentally determined          */
#define VELOCITY_TOLERANCE 1
#define COUNTS_PER_RADIAN_CLK 838.5   /* calibrated for 90 degrees          */
#define COUNTS_PER_RADIAN_ANTICLK 857.0 /* calibrated for 90 degrees        */
#define TOLERANCE  250.0              /* Rover must be within ## mm of target */
#define WHEELBASE 450.85              /* Rover wheelbase of 17.75 inches in mm*/
/*****************************************************************************/
```

## void main()

```
{
	int move_power = 90, check_IR_flag = 1, at_end=0, object_detected=0;
	float x_rov, y_rov, theta_rov, x_dest, y_dest, move_distance, turn_angle;
	float *x_rovptr, *y_rovptr, *theta_rovptr, *move_distanceptr, *turn_angleptr;

	x_rovptr = &x_rov;
	y_rovptr = &y_rov;
	theta_rovptr = &theta_rov;
	move_distanceptr = &move_distance;
	turn_angleptr = &turn_angle;

	x_rov  = X_ROVER_START;		/*Initialize global variables to coordinates*/
	y_rov  = Y_ROVER_START;
	theta_rov = THETA_ROVER_START;
	x_dest = X_DESTINATION;
	y_dest = Y_DESTINATION;


	if (x_dest == 0.0)	x_dest = 0.01;	/*to prevent overflow in CalcDistVec*/
	if (y_dest == 0.0)	y_dest = 0.01;	/*to prevent overflow in CalcDistVec*/

	sleep(6.0);
	at_end = DestinationCheck(x_rov, y_rov, x_dest, y_dest);
	while(at_end != 1)
	{
		CalcDistVec(&x_rov,  &y_rov,  &theta_rov,  &move_distance,  &turn_angle,  x_dest,
y_dest);
		spin(turn_angle);
		object_detected  =  move(move_distance,  move_power,  &x_rov,  &y_rov,  &theta_rov,
check_IR_flag);
		if(object_detected != 0)
		{
			AvoidObstacle(&x_rov, &y_rov, &theta_rov, object_detected);
			check_IR_flag == 1;
		}
		at_end = DestinationCheck(x_rov, y_rov, x_dest, y_dest);
	}
	tone(750.0, 0.6);		/*Play different beep sound*/
	printf(" x=%f, y=%f\n", (x_rov), (y_rov));
	sleep(2.0);
	printf("counts0&1: %d, %d\n", encoder0_counts, encoder1_counts);
}
```

## int DestinationCheck(float x_rov, float y_rov, float x_dest, float y_dest)

```
{
	if ((Rovabs(x_dest  - (x_rov)) < TOLERANCE) && (Rovabs(y_dest - (y_rov)) < TOLERANCE))
		return 1;
	else
		return 0;
}

/*#define PI 3.1415926535898*/

/* CalcDistVect calculates two numbers:  (1)Linear distance from rover to	*/
/* destination; (2)angle theta the rover must turn with respect to the		*/
/* positive y-axis.  It returns the numbers to to the global variables		*/
/* dist and theta.  It also updates global rover coordinates x_rov and y_rov.	*/
/*
	*/
/* Pointers are used to modify the global variables (call by reference).		*/
```

## void CalcDistVec(float *x_rovptr, float *y_rovptr, float *theta_rovptr, float *move_distanceptr, float *turn_angleptr, float x_dest, float y_dest)

```
{
     float x_comp, y_comp, theta_absolute;
     printf("CalcDistVec\n");
/*   sleep(1.0);*/

     x_comp=x_dest-(*x_rovptr);       /* x_comp is x-comp from rover to destination     */
     y_comp=y_dest-(*y_rovptr);       /* y_comp is y-comp from rover to destination     */

     *move_distanceptr=sqrt(((Rovabs(x_comp))^2.0)+((Rovabs(y_comp))^2.0));
     /* move_distance is the linear distance from the rover to destination.     */
     printf("1) %f %f %f\n", x_comp, y_comp, *theta_rovptr);
     sleep(0.5);
     if(x_comp>0.0)
     {
            /* Calculates theta w.r.t +y axis if x_comp is positive           */
                   theta_absolute = (PI/2.0)-atan(y_comp/x_comp);
     }
     else
     {      /* Calculates theta w.r.t +y axis if x_comp is negative           */
                   theta_absolute = -((PI/2.0)+atan(y_comp/x_comp));
     }
     *turn_angleptr = theta_absolute - (*theta_rovptr);
     if ((*turn_angleptr) > PI)
            *turn_angleptr = (*turn_angleptr)-(2.0 * PI);
     if ((*turn_angleptr) < (- PI))
            *turn_angleptr = (*turn_angleptr)+(2.0 * PI);
/*   printf("2) %f %f %f\n",theta_absolute, *turn_angleptr, *theta_rovptr);
     sleep(10.0);*/
     *theta_rovptr = theta_absolute;
/*   printf("3) %f %f %f\n",theta_absolute, *turn_angleptr, *theta_rovptr);
     sleep(10.0);*/
     printf("D = %f Angle = %f\n", *move_distanceptr, *turn_angleptr);
}

/*#define COUNTS_PER_MM 3.73
#define VELOCITY_TOLERANCE 1
#define COUNTS_PER_RADIAN_CLK 838.5
#define COUNTS_PER_RADIAN_ANTICLK 857.0
*/
```

## void spin(float spin_angle)

```
{
/* note angle is in radians, -ve      turns counter clockwise  */
/* -ve spin_power will rotate counter clockwise                */
     int counts_target, velocity_delta;
     int spin_power = 90, spin_sign = 1, mot0=0, mot1=0;
     float counts_per_radian = COUNTS_PER_RADIAN_CLK;
     encoder0_counts = 0;
     encoder1_counts = 0;
     if (spin_angle < 0.0)
     {
            spin_angle = - spin_angle;
            spin_sign  = - 1;
            counts_per_radian = COUNTS_PER_RADIAN_ANTICLK;
     }
      motor(0, (spin_sign * spin_power));
      motor(1, (-1 * spin_sign * spin_power));
     counts_target = (int)(counts_per_radian * spin_angle);
     while((mot0 || mot1) == 0)
     {
/*          if (encoder0_counts % 200 == 0)
                   printf("%d\n", encoder0_counts);          */
            velocity_delta = (int)Rovabs((float)encoder0_velocity - (float)encoder1_velocity);
            if (((mot0 && mot1) == 0)&&(velocity_delta > VELOCITY_TOLERANCE)&& (velocity_delta
< 12))
            {
                   if (encoder1_velocity > encoder0_velocity)
```

```
                        spin_power = spin_power + 2;
                else
                        spin_power = spin_power - 2;
                motor(0, (spin_sign * spin_power));
        }
/*      printf("0vel: %d        1vel: %d\n", encoder0_velocity, encoder1_velocity);*/
        if (encoder0_counts >= counts_target)
        {       off(0);
                mot0=1;
        }
        if (encoder1_counts >= counts_target)
        {
                off(1);
                mot1=1;
        }
    }
    ao();
}
```

## int move(float move_distance, int move_power, float *x_rovptr, float *y_rovptr, float *theta_rovptr, int check_IR_flag )

```
{
/* Must load fencdr0.icb and fencdr1.icb                                        */
/* The move function forces the wheels to turn at the same rate         */
/* only that both wheels travel = counts_distance                               */
/* distance is in mm, -ve distance goes backwards                               */
/* move_power must be positive and @70                                          */
/*#define COUNTS_PER_MM 3.73                                                    */

    int distance_sign=1, motor_offset=1, mot0=0, mot1=0,obstacle_flag=0;
    int counts_distance, i=0, velocity_delta; /* (does not)allows for distances over 28ft to be
passed to it*/
    encoder0_counts = 0;
    encoder1_counts = 0;
    counts_distance = (int)((float)COUNTS_PER_MM * move_distance);
/* Setting distance sign */
    if (move_distance < 0.0)
    {       distance_sign = -1;
            motor_offset = 2; /* offset of 2 is for backwards direction. */
    }
/* start moving */
    motor(0, distance_sign * (move_power + motor_offset)); /*to initially adjust for motor
imbalance @ move_power=70*/
    motor(1, distance_sign * move_power);
/*  printf("m:x=%f y=%f\n", (*x_rovptr), (*y_rovptr));                          */

    while((mot0 || mot1) == 0)      /*using OR so that distance is determined by the
dependable encoder1     */
    {
            if (encoder0_counts >= counts_distance)
            {       off(0);
                    mot0=1;
            }
            if (encoder1_counts >= counts_distance)
            {
                    off(1);
                    mot1=1;
            }
            if (encoder0_counts > 32700 || encoder1_counts > 32700)
            {
                    tone(1200.0, 0.6);      /*Play different beep sound     */

                    break;                  /*prevents overflow of counters, then this
function will be called again by main     */
            }
            i++;
            if (i%50==0) {printf("0vel: %d     1vel: %d  1pwr:%d\n", encoder0_velocity,
encoder1_velocity, move_power);}
```

```
   /*   This next IF statement keeps both wheels rotating at the same rate */
                velocity_delta = (int)Rovabs((float)encoder0_velocity - (float)encoder1_velocity);
                if (mot0==0 && mot1==0 &&(velocity_delta > VELOCITY_TOLERANCE)&& (velocity_delta <
12))
                {
                        if (encoder0_velocity > encoder1_velocity)
                        {
                                if (move_power < 100)
                                        move_power = move_power + 2;
                        }
                        else
                        {
                                if (move_power > 40)     /* extra precaution to prevent invalid
encoder0_velocity from stopping motor1    */
                                        move_power = move_power - 2;
                        }
                        motor(1, move_power);
                }
   /*           printf("0vel: %d        1vel: %d\n", encoder0_velocity, encoder1_velocity);*/
   /*   This next IF statement polls the IR sensors for obstacles and if detected          */
   /*   then stops the rover, updates coordinates, and returns and notifies calling function
        */
                if (check_IR_flag == 1)
                {
                        obstacle_flag = check_IR();
                        if (obstacle_flag > 0)
                        {
   /*                           msleep(800L);    because IR sensitivity too high  */
                                ao();
                                *x_rovptr   =   (*x_rovptr)   +   ((((float)(/*encoder0_counts   +
*/encoder1_counts)) / 1.0) / (float)COUNTS_PER_MM) * sin(*theta_rovptr);
                                *y_rovptr   =   (*y_rovptr)   +   ((((float)(/*encoder0_counts   +
*/encoder1_counts)) / 1.0) / (float)COUNTS_PER_MM) * cos(*theta_rovptr);
                                return obstacle_flag;
                        }
                }
        }
        ao();
        beep();
   /*   Update the coordinates of the rover...                                          */
        *x_rovptr  =  (*x_rovptr)  +  ((((float)(/*encoder0_counts  +  */encoder1_counts))  /  1.0)  /
(float)COUNTS_PER_MM) * sin(*theta_rovptr);
        *y_rovptr  =  (*y_rovptr)  +  ((((float)(/*encoder0_counts  +  */encoder1_counts))  /  1.0)  /
(float)COUNTS_PER_MM) * cos(*theta_rovptr);

   /*   printf("6) x = %f y = %f\n", *x_rovptr, *y_rovptr);
        sleep(5.0);
        printf("Diff 0 & 1 -     %d & %d\n", encoder0_counts - counts0_target, encoder1_counts -
counts1_target);
        sleep(5.0);
   */
        return 0;
   }
   /* If 0 returned then there are no obstacles detected by the IR*/
   /* If 7 returned then there is an obstacle on the right                */
   /* If 8 returned then there is an obstacle on the left         */
```

## int check_IR()

```
   {
        if (digital(7) == 1)
        {
                msleep(1100L);
                return (7);
        }
        if (digital(8) == 1)
        {
                msleep(1100L);
                return (8);
        }
        if (digital(9) == 1)
```

```
        return (7);
    if (digital(10) == 1)
        return (8);
    else
        return (0);
}
```

```
/* This function will avoid an obstacle by turning 90 degrees, moving forward   */
/* 3ft, turning 90 degrees back to its original heading and moving forward      */
/* another 3ft.  Which way it turns depends on the object_detected number:      */
/* 1 represents an obstacle on the front left or straight ahead, 2 represents   */
/* an obstacle on the front right (so it turns left)                            */

/* #define AVOID_OBSTACLE_DISTANCE 914.4      3 feet in mm */
```

## void AvoidObstacle(float *x_rovptr, float *y_rovptr, float *theta_rovptr, int object_detected)

```
{
/*   assumes obstacle is on the front left, ie object_detected = 8      */
    float avoid_angle = 1.571;
    printf("AvoidObtacle routine\n");
/*   msleep(1500L);*/
    printf("int:x= %f, y= %f\n", (*x_rovptr), (*y_rovptr));
/*   msleep(3000L);*/
    if (object_detected == 7)        /*obstacle on front righthand side*/
        avoid_angle = - 1.571;
    *theta_rovptr = (*theta_rovptr) + avoid_angle;
    if ((*theta_rovptr) > PI)
        *theta_rovptr = (*theta_rovptr)-(2.0 * PI);
    spin(avoid_angle);                              /*Turn either +/- 90 degrees     */
    msleep(200L);
    move(AVOID_OBSTACLE_DISTANCE, 90, x_rovptr, y_rovptr,theta_rovptr, 0);
    *theta_rovptr = (*theta_rovptr) - avoid_angle;
    if ((*theta_rovptr) > PI)        /*should probably be placed in caldistvec as well */
        *theta_rovptr = (*theta_rovptr)-(2.0 * PI);
    spin(-avoid_angle);                                     /*Turn +/-90 degrees     */
    msleep(200L);
    move(AVOID_OBSTACLE_DISTANCE, 90, x_rovptr, y_rovptr,theta_rovptr, 0);
    msleep(200L);
}
```

## float Rovabs(float x)

```
{
    if (x >= 0.0)
        return x;
    else
        return (-x);
}
```