Design of an Autonomous Robot System Performing Coordinate Behavior

Junior Design 2007
Group 4

Roque Burleson
Jamey Christy
Andrew Doxon
Ted Schuler-Sandy
Kyle Siemers

Abstract

This paper describes our design used to create two autonomous robots that would be able to push a cardboard box down a lane and end within the lane's boundaries. The robots communicate with each other and a base station. The system works autonomously, except for a start and stop command. We implement a tread system for our drive method to allow for more precise turning. There are painted encodings on the treads used for speed and distance measurement that could be calibrated to the desired precision. Another unique trait of our design is the GUI, which is implemented in a mIRC program.

Our project is related to the following fields of study: robot control, autonomous robotics, radio communication, and drive control.

Table of Contents

I. List of Figures

II. Introduction

The presented design will accomplish the task of having two autonomous robots push a cardboard box down a lane and end within the boundaries. The box is allowed to leave the lane while in transit so long as the robots bring the box back within the lane before the end of the run.

There are several restrictions associated with this task. The robots should be able to communicate with each other and send data to a base station. The Graphical User Interface (GUI) connected to the base station will display all the packets in a readable form.

The robots are required to fit within a six-inch cube and can have no more than three-inches of contact with the box. The box is between one and one-and-one-half meters. The lane is two to two-and-a-half meters in width and roughly four meters long. We are provided three Micazs, one for each robot and one to be used as a base station. In addition to the Micazs we are provided: two H-bridges, two gear boxes, and four wheels. All other items are to be purchased.

The design is divided into 3 main levels. The top level consists of the algorithms used to determine what actions the robots should take to complete the task. The algorithms consist of a Startup program, a Coordinator program, a Found Line program, and an End program. The next level consists of the support modules responsible for communicating the algorithms' commands to the hardware. These support modules consist of: Pulse Width Modulation (PWM) control, data passing, communications, sensor monitoring, and lower level motor control. The bottom level is the hardware itself.

III. Algorithms

To accomplish our task we will be implementing a control algorithm as a state machine. The states represent the set of actions that will be performed. A state machine model was decided upon because of the well-separated modes of operation and the ease of switching between them. These states consist of: Startup, Coordinator, Found Line, and End. These states will then call upon a set of support modules to perform the individual actions. The switching between states will be handled by the states themselves as input demands or the states finish. The transitions between states will be determined by the sensor inputs, and the completion of the initial state.

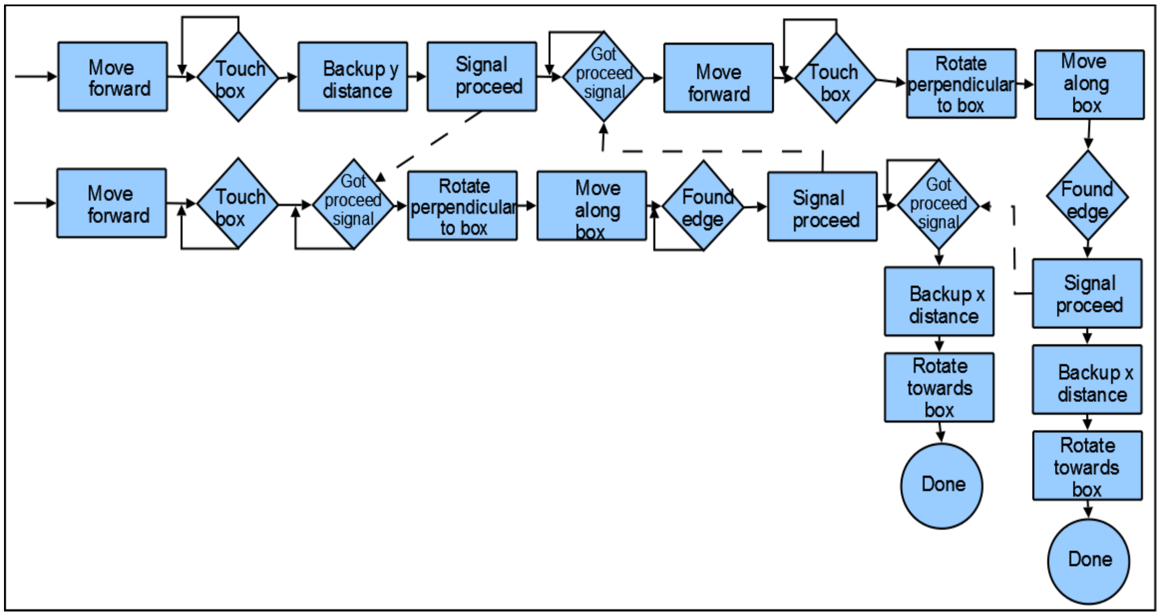

Figure 1: Startup flow chart

The Startup state, shown in Figure 1, initializes the robots when they receive the start signal from the base station. Due to this, Startup will only need to be run once. Other parts of the program require knowing which side of the box the two robots are on. The Startup program automatically does this by setting the primary robot to go to the right side and the secondary robot to the left side. It is responsible for making sure that the robots will be placed in the optimum position, at the ends of the box, to complete the task. The first action of the Startup state has the robots drive forward until they both touch the box, then align themselves so that both pressure switches are depressed. At this point, the robot that is defined to be the secondary will then back up to allow the primary robot to pass by. The primary robot will then rotate to the right until it is parallel to the box. The robot will proceed along the side of the box until it detects the edge with the box sensor. The primary robot will then signal the secondary robot to align in a similar manner as above but to the other end of the box. When both of the robots are located at the edge of the box they will back up and rotate so they are perpendicularly aligned at the edge of the box. Once the robots are aligned, the program will transition into the Coordinator state.

During the last stage of the Startup program the timer subsystem for the Coordinator is initialized. The timer for the initialization is turned off. This way we can minimize the processing done by the Micaz. The Coordinator program's job is to drive the robots straight down the lane. To do this, the robot runs through a series of comparisons between its bumper states and those of the other robot. There are 7 basic states that the Coordinator program runs checks for. The following table shows these states. The highlighted numbers are the bumpers on the robot performing the instructions. A "1" represents a depressed bumper switch and a "0" represents a free bumper.

Command	**States involved**
Drive Forward	0000, 1111, 1100, 0100, 1000
Stop	0011
Turn Right	0001, 1101, 1001
Turn Left	0010, 1110, 0110
Forward Slow	0111, 1011
Special (right)	0101
Special (left)	1010

Figure 2: State diagram of Coordinator

The first case, "Drive Forward", occurs when neither bumper on the robot is depressed or in the ideal case where both bumpers of the robots are depressed. The "Stop" case will occur when one robot is aligned with the box and the other is not touching it. The "Turn Right" and "Turn Left" states occur when one bumper on the robot is not depressed and the other is, and the other robot is not in the same bumper configuration. In this case the robot will turn until both of the robot's bumpers are depressed. The "Forward Slow" case occurs when one robot is fully connected with the box and the other robot is in the "Turn Right" or "Turn Left" state. This state is used because a turning robot is significantly slower than a robot driving straight. This state is simply the drive forward state with a speed modulator built in to slow the robot down. Finally the special cases are to help cope with angle correction.

In each of the special cases, the robots try and balance the box so that it stays perpendicular to the lane. By the time the robots reach the special case they will know which side of the box they are on. This is due to the Startup algorithm. The special cases should only activate if the box has gotten askew but the robots are still facing straight down the lane. In this case, the robot that is further forward will slow down to allow the lagging robot to catch up. As soon as the robots have both bumper sensors depressed they resume the move state and continue down the lane.

Periodically during this time the robots are polling the line sensors to see if they cross a line. When this occurs the Coordinator program places itself in a waiting status and tells the Found Line program to run.

The Found Line program follows a simple algorithm. A short synopsis of the algorithm is displayed in Figure 3.

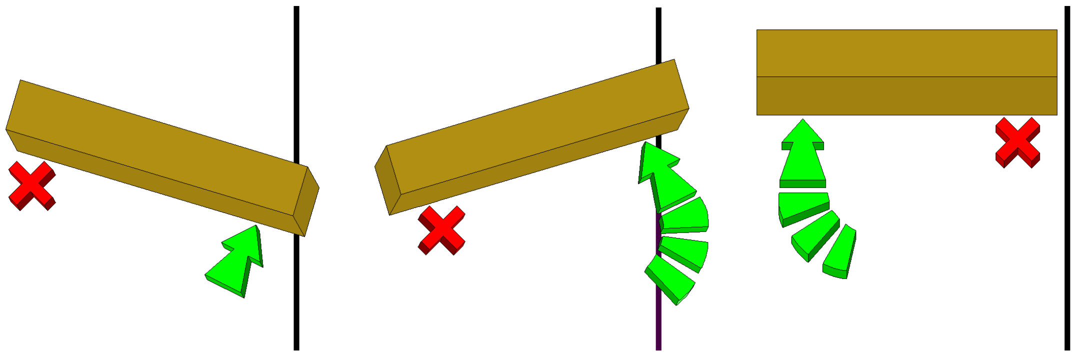

Figure 3: Found Line Algorithm

The primary robot will drive forward around a semicircle, stopping once the line is passed for the second time. The secondary robot will then move to position the box such that it is perpendicular to the lane and both bots can continue with the Coordinator program.

Initially, the primary robot, which is the robot that detected the line, sends a message to the other robot, which causes that robot to enter the Found Line program as well. The primary robot then proceeds to drive in a circle. To do this, the inner tread is set to PWM at a slightly slower speed than the outer tread. During this process, the robot can end the Found Line state in three situations. The first is where, immediately after starting, the robot's opposite line sensor detects the end of the lane. The second exit to the Found Line state is the false line alarm. This occurs after the robot has driven a significant distance around its circle and the opposite line sensor has not detected a line. This state protects against short small crossovers. The third and final exit is caused when the robot senses the line with the opposite sensor and proceeds around the circle until it encounters the line again. During each of the three states, the robot records the distance traveled so that it may inform the other robot. After any one of the three exit conditions are met the primary robot sends a corresponding signal and waits until the other robot says that it has finished with the Found Line program.

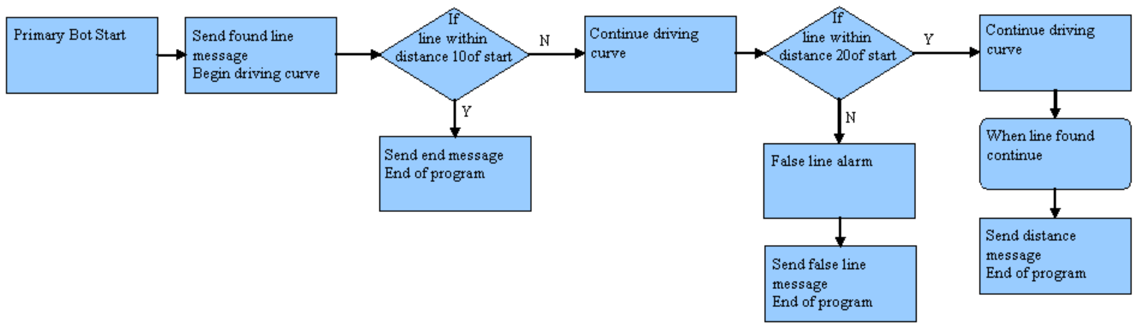
Primary Bot Start
Send found line message Begin driving curve
If line within distance 10of start
N
Y
Send end message End of program
Continue driving curve
If line within distance 20of start
Y
N
False line alarm
Send false line message End of program
Continue driving curve
When line found continue
Send distance message End of program

Figure 4: Found Line of the Primary Robot

The Found Line program for the secondary robot is significantly less complex. After receiving the found line signal and being kicked into the Found Line program, the secondary robot stops and waits for further input as shown in Figure 5. Once the primary robot sends its signal, the secondary robot proceeds based on that signal. In the case of the end-of-the-lane signal, the secondary robot catches up so that the box is at the edge of the lane then sends the proceed signal. In the second case, where the false line alarm signal is sent, the secondary robot drives a preset distance forward around the same circle as the other robot. In the third case, the secondary robot uses the same program as the second case, but travels half of the distance that the primary robot traveled. In all three cases the secondary robot sends a continue signal which lets both robots either enter the end state or continue forward with the Coordinator.

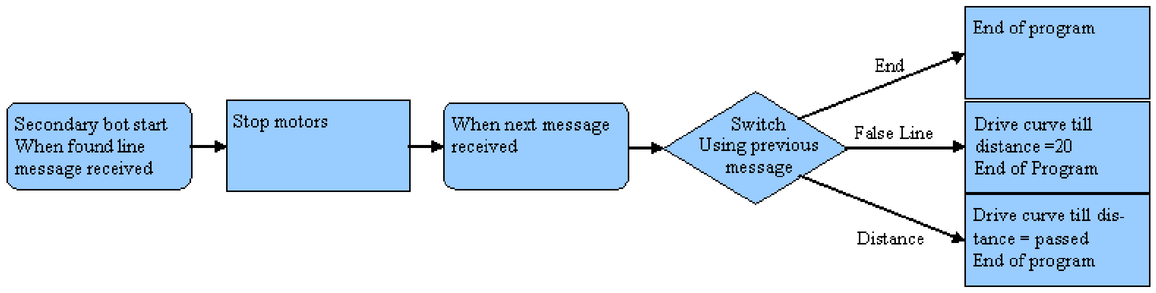
Secondary bot start When found line message received
Stop motors
When next message received
Switch Using previous message
End
False Line
Distance
End of program
Drive curve till distance =20 End of Program
Drive curve till dis- tance = passed End of program

Figure 5: Found Line of the Secondary Robot

The end state can be reached by one of two methods: the end signal being received from the GUI, or the Found Line state determining that the line encountered was the end of-the-lane line. In this state the robot should turn off the motors and timers and wait till the robot is turned off or reset.

IV. Support Modules

The support modules of each robot are the basic foundations upon which all of the high level control is built. There are 5 basic support modules that are used: communications, identification, PWM_motor control, and sensors. The communications support module is sent data from anywhere in the robot. It takes the data and packages it so that it may be sent across the RF port of the robot. The communications module also receives all messages and un-packages them into their data components. It then passes the received data to the algorithm that was intended to use it. The identification module tells the other components whether this robot is the primary or secondary robot. This is used primarily in the communications system and the Startup program as an identifier.

One of the most important support modules is the PWM_motor control. It is responsible for managing the PWM controlling the motors speed and direction. The module was set up so that each side of the robot can be run at separate speed. Instructions to both motors can also be given in the form of the following: go forward, stop, reverse, turn right, turn left, and pivot. These allow for easy programming of movement in the upper level algorithms. The final support system is the sensor system. During startup, the sensor system passes the memory address of the sensor values to all other components. The sensor support system periodically checks and updates the memory containing sensor data. This way, the other programs can look up the current state of the sensors without wasting time checking the actual pin that the sensor is connected to.

The implementation of the algorithms and the supporting modules was in the nesC language for the TinyOS operating system that runs on the Micaz. This language is a C derivative for network embedded systems with TinyOS driving much of its' design. It is a component-based language that provides a unique method of linking components together. The system is an event driven system.

The graphical user interface, seen in Figure 6, used for our project runs on a computer attached to the Micaz interface board through a USB connection. The attached Micaz acts as the base station. The GUI is written in the mIRC scripting language with the help of Dialog Studio, a design kit that generated the code when the various items were placed on a workspace. Once the main GUI layout was completed the code was copied into the mIRC scripting editor where the remaining code was implemented. This code included all the functionality for the GUI as well as how to update the display when certain packets were sent by either of the two robots out in the field. The GUI also included code for sending the start and stop signals to our robots.

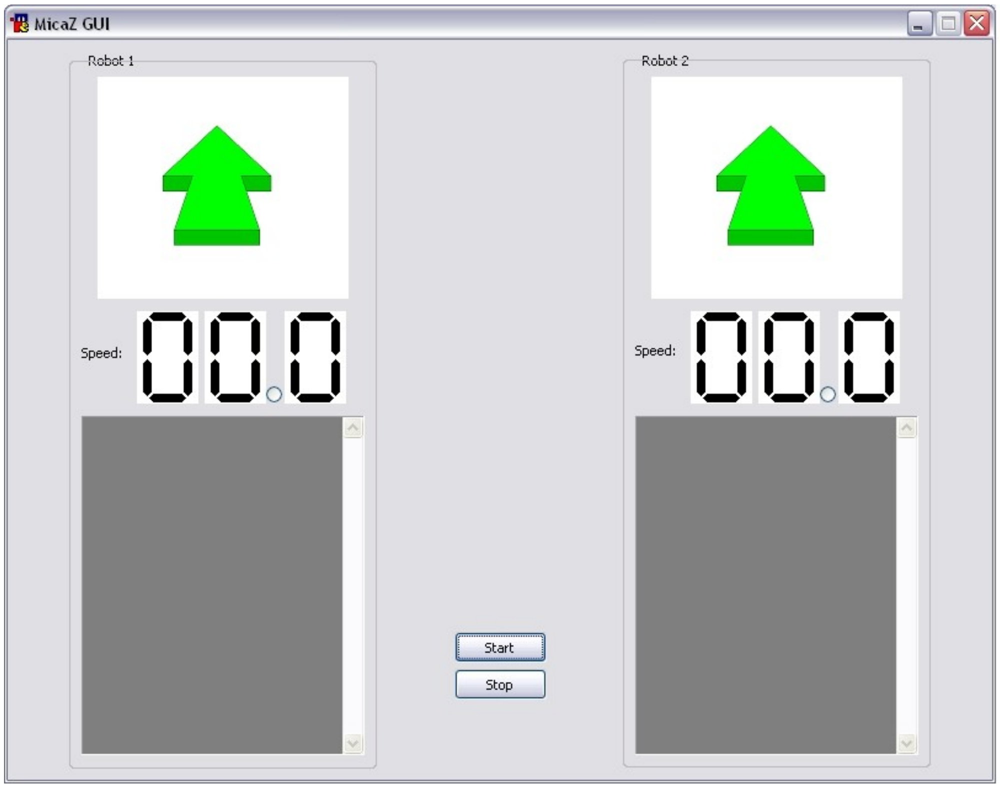

Figure 6: GUI

The mIRC scripting language is a language that was developed by the designer of mIRC itself, and is only usable in that application. Although mIRC scripting is primarily used for interfacing with both mIRC and IRC, it can be used to create GUIs so long as it is running inside the mIRC application. The mIRC scripting language is used for the GUI for three main reasons: the ease of making a GUI using the development kit, allows for rapid code implementation, and built in socket communication.

The GUI displays all required commands from the two robots in the field. The display of these commands includes both an image and a textual representation in a bilateral display system, with one side for each robot's status. Also included in the display is the speed of the robots. There are two buttons, "Start" and "Stop", that a user can click to send out the respective signals.

In order to interface with the Micaz from mIRC, an intermediary program had to be written that could convert serial packets into TCP/IP packets and vice-versa. This program was designed and implemented in the python scripting language. This script continually reads from both the socket used by mIRC and the serial port used by the base station. Packets received over the serial port are filtered by group number and discarded if they did not match ours. The data part of the packet was stripped out and sent across a socket to mIRC where the data was analyzed by the GUI code and displayed in the correct places. When the "Start" or "Stop" buttons are clicked, a short data packet is sent to the socket where the python script would then append all the other data to make it a complete packet for the Micaz. The packet is then sent over the serial port to the Micaz.

The robot's code is implemented on a Crossbow Micaz, which is an 8-bit high performance, low power microcontroller. The Micaz is approximately 1.5inches by 2.5inches in size. It contains the microcontroller interface, a wireless antenna, and a battery case that holds two AA batteries. The Micaz, although unable to communicate directly with the computer, comes with an interface board for programming and basic communication. There are 128KB of in-system re-programmable flash memory for program storage. The Micaz runs at 8MHz allowing for rapid execution of code segments. The six PWM channels are used to manage the drive control subsystem of our code, allowing for different rates of speed for either side of our robot. The Micaz is able to operate in a voltage range between 2.7 to 5.5 volts. The AA batteries used kept it at 3V. The Micaz's wireless radio has a 2.4GHz frequency and is able to transfer and receive 256kbps.

V. Hardware

The hardware is designed to contain the Micaz and all other components used. We constructed two robots, each consisting of two levels. There is also room for a third level, which could be easily added if more space is required. The chassis are made from aluminum plates. The reason we chose aluminum is that it is sturdy and fairly easy to machine into whatever shape we needed. Also, we were able to obtain free aluminum from the scrap pile in the machine shop, which cut down on our costs. We cut three separate plates per robot. Two were square pieces of aluminum, one of which was used for the second levels, and the other for the third levels if needed. The other plate was cut into a rectangular shape, to allow room to mount the wheels. Since the robot has to fit inside a 6”x6”x6” box, we had to make sure that the bottom level with wheels attached would not exceed a width of 6 inches. To hold the levels together, we used two long bolts that we obtained from a box containing parts from old design projects. We drilled through the separate levels, and then attached them to each bolt by placing a nut both above and below the plate. This scheme held the plates firmly in place while allowing us to change the spacing, providing room to mount components onto the separate levels.

The underside of the bottom level holds the wheels, gearbox, and axle mount and sensor bar. Mounting these components on the bottom gave us room on the top for other components, while maintaining sufficient clearance above the ground. The sensor bar was mounted so that the sensors could be adjusted to be the correct distance above the ground. The backset of wheels was mounted in the gearbox. The second set of wheels was mounted on the front of the robot using a custom axle mount. The axle mount consisted of three machined aluminum blocks. The blocks were mounted to the platform using screw taps. The axles were held in place with holes drilled through the center of each block. Originally, we started by using one block and two L-shaped pieces of aluminum to mount the wheels, but we found that these were not at all sturdy enough, thus three blocks were used. The wheel axles would then be threaded through one side block and partially into a middle block. Two lock collars with setscrews were used to hold the axles in place. The sensor bar itself was a long piece of L-shaped aluminum. Three sensors were mounted on each side of the bar, one facing outwards, two facing down. These are the box, line, and speed sensors. The sensors are mounted with screws, which, due to the long hole in the in the middle of the sensors, makes them adjustable.

The top side of the bottom level holds the front bumper and the H-bridge. The front bumper consists of two snap switches, an aluminum bar, and two pieces of eraser. The snap switches have a bar, which is hook shaped at the end. They are mounted to the bottom level. The aluminum bar is mounted to the switches by a small length of copper wire. The wire is threaded through the bar and around the hook part of the switch. This holds the bar onto the switch while also allowing for the change in distances created by the pressing of the switches. We found that the bar itself was not thick enough, since it rested slightly behind the edge of the wheels. Because of this, we decided to attach two pieces of eraser to the bar, which ensured that it would hit the box before the wheels did. The H-bridge is also mounted on the top of the bottom level. It is attached via four screws that are threaded up through the bottom level, and stick out above it. The H-bridge itself rests on four short plastic cylinders, and is held down by four nuts.

The second level has nothing mounted on the underside. The upper side holds the MicaZ itself, a breadboard used for circuitry, and the batteries. The MicaZ is mounted in a thin tin seating that is bent into a form that would securely hold it in place. There is no topside, thus allowing easy removal and replacement of the MicaZs, and the sides are slightly lower thus allowing easy access to the on/off switch. The seating is then attached to the aluminum by double-stick tape, which holds it in place quite well. In order to wire up the circuits required for the sensors and the power regulators, we took a normal breadboard and cut it into fourths so that one of the fourths would be small enough to fit onto the robot. The batteries themselves are simply mounted to the plate using double stick tape.

Our testing has so far shown that our design is sturdy enough to work for our purposes. While several elements could be slightly better, such as using printed circuits boards instead of a breadboard, the changes are not necessary to get our robot to perform correctly.

One of the most unique aspects of our approach is our drive system. Out of the many different steering options we considered, we decided to use a skid-steer driving method. In a skid-steer system, the wheels on each side of the unit are tied together in such a way so that all wheels turn at the same rate and in the same direction. This allows each side of the vehicle to be considered a single drive unit, reducing the overall drive systems complexity. A skid-steer system also has the advantage of a centralized point of rotation, which allows our robot to turn inside its own radius. This system can also easily take advantage of a steering method called counter-rotation. In this method, the robot can make sharp turns by having the wheels on the turning side run in reverse, and the wheels on the opposite side run forwards. This causes the system to turn in place around its center of rotation. The robot can make varying degrees of turns by changing the relative speed of the wheels on each side. Having both sides rotate in the same direction but at different rates allows the vehicle to make shallow adjustment turns while in motion. Straight-line motion can be achieved and easily maintained, by having both sides run at the same speed.

These advantages make skid-steering an effective choice for our design. Being able to rotate within the wheelbase of the robot allows the robots to rotate in place when finding the edge of the box without having to worry about accidentally hitting it. By only needing to monitor the speed of each side, we have a large degree of control and accuracy in our movements with relatively few sensors. The only difficulty with skid steering is its slightly higher complexity to implement.

There are two primary methods to create a skid-steer system. The main method to implement skid-steering in industrial machinery is to use a common transmission for each side. This is a very effective method of ensuring equal power distribution. However, such transmissions are very complex and are beyond our ability to create using our existing transmission. The second method that can be used is to have only one set of wheels on each side powered and have the remaining wheels on each side coupled to the drive wheel using a track. This method can be inefficient due to track slippage, but is much easier to implement, only requiring a free rotating axle for the other wheels and a track to link them together. This has the added bonus of allowing speed measurements to be taken by encoding the tracks, which simplifies the sensor system and ensures that we are measuring the actual ground speed of the robots.

After several different designs for tracks, we determined that a three-stage track system would work best for our needs. The first stage consists of a rubber o-ring on the outside rim of each of our wheels. The purpose of this o-ring is to add extra traction to compensate for removing the tires to implement the tracks. These are unmodified, and were sized to fit the rims. The second stage consists of a rubber o-ring that runs between the inner rims of the wheels on each side. This o-ring provides most of the coupling forces between the two sides, and also a large portion of the traction. Like the outer o-ring, this o-ring is unmodified and was sized to match the track length of the system. The final stage of the track is the main track. While designated the main track, this portion actually provides only a small portion of the traction and coupling in the system. Its main function is to provide encoding for the speed sensors. This stage is made from a remote controlled car track, which has been cut to correct width and length. Despite careful measurements when re-cutting, the tracks stretched slightly, which allowed it to slip under power, greatly reducing its coupling ability. By adding the inner o-ring mentioned above much of the strain was taken off, allowing the track to rotate smoothly and without slipping. In addition to being cut to size, these tracks have had graduations painted on the inner surface to serve as optical encodings for the speed sensors. While we originally planned on using a mask to paint the treads, establishing such a mask proved to be too time consuming and it still allowed paint to leak through. However, the material we used for the track had a uniform ribbing on the inside, giving us a reference to hand paint the marking using model paint and a fine brush. Overall the hand painting was very effective, with the difference between the largest and smallest markings being less than 2mm.

After developing a plan for our algorithm we determined that we would need the following sensor inputs: speed measurement for each side, line detection for figuring out if we cross the line on either side, object detection for finding the edge of the box, and pressure detection for finding which sides are in contact with the box. We originally planned on using a sensor to determine the robots position with respect to each other, but we were unable to find a sensor with adequate range and sensitivity to do what we intended, so our algorithm was modified to perform the task without the sensor. After deciding what each sensor needed to do, we needed to decide which type of sensor would best perform each task. For the speed measurement, line detection, and edge of box detection, we decided to use optical encoders. For detecting box contact we decided to use pressure sensors.

Optical encoders are a meet our requirements for many reasons. First, since they detect light and dark regions they can easily distinguish the black tape used for marking the lane, the black and white graduations on the tracks, and the edge of the box. Second, the sensors work without having to be in contact with an object, which reduces friction losses in the case of the line or speed sensors, damage to the track or sensor in the case of the speed sensors, and inadvertent moving of the box in the case of the edge detectors. Since the speed measurement was most critical to our design we looked at sensors that would meet those requirements first. The sensor we ended up using, the QRB1134, can be used for all three applications. This simplified our design, since we only have to refer to one data sheet to configure the sensors, and we were able to easily install the remaining sensors once one initial sensor was setup. The QRB1134 consists of an infrared emitting diode paired with a phototransistor sensitive to the same wavelength. When connected as shown in diagram 1, the sensor will output logic high when there is a reflective object and a logic low when there is no object or a non-reflective object. By changing the external resistor values, the optimum range can vary between .2 inches and 6 inches. The sensors are mounted on sliding mounts so that they can be moved to the optimum height and rotated to the optimum angle for best resolution. We use a total of 6 QRB1134's on each robot, one on each side for each sensor system.

For the box contact sensor we decided to use a physical touch sensor instead of an optical sensor because we want to be sure that the robots are actually touching the box and not just close enough to trigger the sensor. The type of sensor we are using is a double pole, single throw snap switch. It is wired so that, when the robot is not in contact with the box, the switch is connected to the ground terminal and logic low is reported. When the robot is in contact with the box, the switch depresses and the throw is now connected to the VCC terminal and logic high is reported. The arms for the switches are connected to a bumper to distribute the weight of the box when pushing. They are connected on the outside edge of the bumper to give better sensitivity if the box is angled. The original switch we were going to use had a longer contact arm than the one we finally used, however that particular model was out of stock. To compensate for the decreased length of the arm and give better surface contact with the box, we added rubber standoffs to the edges of the bumper.

To connect the various components together a breadboard is cut to size with several ground and power buses. While not as stable as a printed circuit board (PCB), a breadboard allowed changes to the design to be implemented as needed to compensate for unforeseen difficulties. Several times in our initial design we had to radically modify the sensor and board layout. If we had had to re-etch a new circuit board, it would have been time and cost prohibitive. Even in final testing, we had to modify our power supply design because our initial design made some incorrect assumptions and we were not getting enough power to drive the motors. Since we used a breadboard, we easily remedied the problem in a few minutes. Had we used a PCB it would have taken several hours to make the same changes. Once we had a final working prototype design, we could have rebuilt onto a PCB if time had allowed. This would have given us a much cleaner and less cluttered design, and we could have used leads that were cut to length. We compensated for this lack by carefully color-coding and labeling all of our wiring.

VI. Power

Our final power supply design consists of three parts. The first part is the Micaz's self-contained battery pack consisting of 2 1.5V AA batteries. The second part consists of a 5V DC power regulator supplied by a 9V battery. The power regulator allows us to generate a constant 5V even when the battery begins to lose charge and its voltage begins to drop. This supply powers the various sensor systems. Originally this was also supposed to supply the power for the H-bridge and motor, but we discovered in final testing that the current drawn by the sensors took too much power from the motors and left the motors unable to move the fully loaded chassis. To fix this we added the third part of our power system, a second 5V regulator powered by two 9V batteries in parallel. This resulting configuration has enough current to drive the motors and H-bridge effectively.

Our power budget was measured and calculated directly from one of our robots. The power consumption of the infrared sensors and pressure sensors were a total of 1W, while the regulator used for these sensors was far over this amount at 1.4W. Our motors on the other hand only consumed half a watt on their own, with an additional .4W from the voltage regulator on the system. These totals, combined with the power consumed by the h-bridge and Micaz, came to just over a total of three watts. Using the rated time on our 9V volt batteries we calculated the run time of our robots to be about two hours using three such batteries, assuming they maintained a stable charge. Our power budget, had it been calculated like this previously, would have shown us that we needed the three 9V batteries per robot instead of the two we had originally planned for.

VII. Budget

Rechargeable Batteries	$18.95	4	0	0
Batteries	$23.00	8	6	$17.25
Pair of wheels	$11.70	2	4	$23.40
Eraser	$1.17	2	2	$1.17
Battery Connectors	$1.80	6	6	$1.80
Treads	--	--	4	--
O-Rings Small	--	--	8	--
O-Rings Large	--	--	4	--
Aluminum	--			--
Total	$117.51			$80.46

Figure 7: Budget

The budget shown above is for how much our group has spent (Cost column) and how much it would cost to build the robots we used (System Cost column). As seen in the above chart we managed to stay within the $150 budget allocated by the Electrical Engineering Department. Most items were purchased with extra quantity in case of damages caused through out the semester. The following items are donations and do not have listed costs: Treads, O-Rings, and Aluminum. The parts provided by the class that we did not purchase extra are not listed here.

VIII. Conclusion

With the above design, we believe we can successfully accomplish the task given to us. Here are some of the benefits of our design. By using a tracked skid system we have better turning ability and a shorter turning radius. Our Startup procedure allows us to compensate for not knowing the initial positions of the robots. By using a sensor bar with adjustable mount we can fine-tune the sensors for optimum performance. Finally by using a multilevel design we can easily add additional modules and further refine the design.