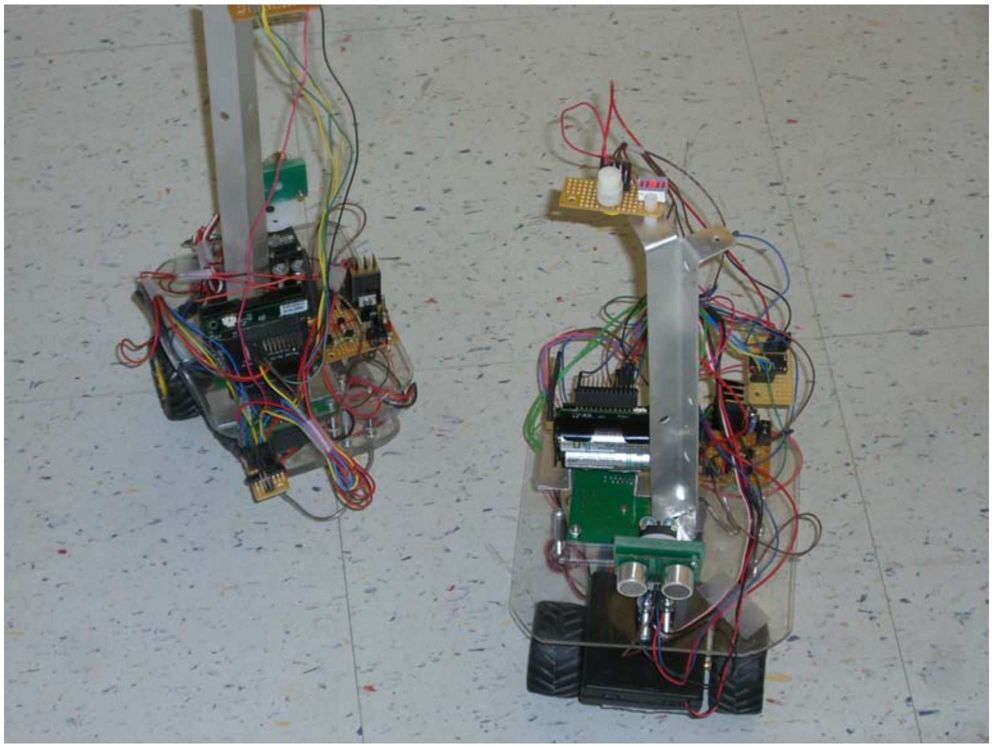
Hunt for the Black Box

May 4, 2006

Prepared For: New Mexico Institute of Mining and Technology
Junior Design - Electrical Engineering 382
Dr. El-Osery
Dr. Wedeward

Prepared By: Group 1
Michael Baltes
Anthony Duran
Steven Myers
Jaisy Perreault
Michael Pursley



Abstract

This paper is a descriptive compilation of the work done by junior design group 1 in building a multi-robot search assembly. With our given project specifications, in which our search field and black box (target) dimensions are specified, the group decided to use an ultrasonic sensor and a digital compass in both our localization and searching algorithms. In knowing the boundaries of our search, we're able to localize each robot from the distances gathered from the ultrasonic sensor and the general direction of the compass. For the searching algorithm we used a pseudo-coordinated pattern and to find the target we use our localization and the known distances to each wall to find any anomalies in distance with the ultrasonic sensor.

Keywords: multi-robot searching, ultrasonic, digital compass, localization, pseudo-coordinated search pattern

Table of Contents

List of Figures and Tables

Introduction

Starting in January, the electrical engineering junior design class was given the task to build and employ a team of robots to find an object in a search field devoid of all other obstacles. Similar to, but much simpler than, robot deployment for mine sweeping, our project is defined by the sensors chosen, the localization schema, and the searching algorithm decided upon.

Due to the importance and implications of such technology, much research has been done on the efficiency, both cost and power, of searching a field with single or multiple robots. Since our search field was designated to be a 3 meter by 3 meter box, and with simplicity in mind, our group decided to use an ultrasonic sensor for object detection since its max range is 3 meters. Since already using an ultrasonic sensor for searching, we decided to also use it to help localize. With our dimensions known and a digital compass telling us the general direction in which our robots are facing, we can figure out our position in the box with the ultrasonic sensor telling us the distance to each wall.

First we will discuss the specific requirements for the project and then briefly go into the choices we made and why, giving a general overview of the final product. Then we will go in depth about the hardware and the software of the project, the final implementation code being Appendix A. We will then conclude with the communication and integration of all the parts including the custom built graphical user interface.

Project Specifications

Designed to test the knowledge and teamwork capabilities learned over the past 3 years as electrical engineering students, our task revolves around networking, communication, and sensors. Required to create a mobile sensor network with at least two mobile sensor nodes in the field that must localize themselves and work independently (i.e. if one node dies, the other nodes must be able to continue with their search), we were given 4 Micazs (basically a microcontroller with the ability to communicate through RF), 3 motors, 3 h-bridges, an interface board for serial communication between the Micazs and a 250 dollar budget.

Our search field is a designated 3 meter by 3 meter box, with our target being a 30 centimeter by 30 centimeter by 30 centimeter box. Our nodes will be randomly placed in the box, must localize themselves and must wirelessly communicate to the basestation their position. We are also required to be able to display each nodes position on a GUI with a refresh time of at most 2 seconds and in the end also display the position of the box. The nodes must then search the box for our target and once found tell the basestation. The basestation will then tell the other nodes in the field the coordinates of the box who must then drive to the given spot to confirm the find.

Sensors

The three types of sensors we used to implement our localization and searching algorithm were an ultrasonic transmitter/receiver, a digital compass, and a servo. All of these sensors each had their own specifications to make them function, and it was our job to meet each of them simultaneously.

The first sensor used for our implementation was the ultrasonic, more specifically the Devantech SRF04 Ultrasonic Range Finder. According to the manufacturer, Acroname, this ultrasonic required 5 volts to function and pulled anywhere from 30 milliamps to 50 milliamps. Its range was from 3 centimeters to 300 centimeters (3 meters), which for our purpose was perfect.

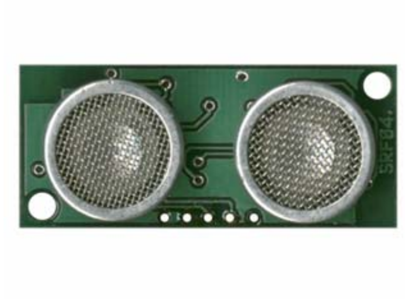


Figure 1
SR04 Ultrasonic Sensor
www.acroname.com

At its maximum range of 3 meters, we found its accuracy to be within 30 cm, or a 10 percent error. It had great sensitivity for its price of usually $25.00 being able to detect a 3 centimeter object from greater than 2 meters. In order to function, the SRF04 required a 10 microsecond input trigger. We used input capture to capture the reflection off the wall or object, with the pulse width proportional to its range. If the reflection was greater than 3 meters away, it would return a 36 microsecond pulse. This sensor was used for our distance readings and object detection.

The second sensor used for our implementation was the digital compass. Our digital compass of choice, due to our budget restraints, was the Dinsmore 1490 Digital Compass. According to Dinsmore's website, this sensor provides eight directions of heading information by measuring the earth's magnetic field using hall-effect technology. The 1490 sensor is internally designed to respond to directional change similar to a liquid filled compass. It will return to the indicated direction from a 90 degree displacement in approximately 2.5 seconds with no overswing. The 1490 can operate tilted up to 12 degrees with acceptable error. It is easily interfaced to digital circuitry and microprocessors using only pull-up resistors. The compass was able to be powered from a minimum of 5 volts and a maximum of 18 volts and pulled approximately 30 milliamps of current. The compass had only 8 distinct directions: North, Northeast, East, Southeast, South, Southwest, West, and Northwest, giving our compass approximately 15 degree accuracy.

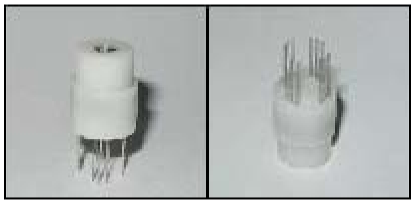


Figure 2
Dinsmore 1490 Digital Compass
www.dinsmoresensors.com

We found this to be somewhat of a hassle, but will talk about how we handled the problem in the actual implementation section. It was really lightweight, only 2.25 grams, and was only 12.7 millimeters in diameter.

The third sensor used for the implementation of our mobile node was the servo, more exactly the Acroname Standard Servo. According to Acroname, this servo was extremely important to our design. On top of the servo was mounted the ultrasonic sensor discussed earlier. The servo had the ability to rotate 360 degrees, but for our purpose we only needed it to rotate 180 degrees. Due to its ability to do so, we were able to prevent having our robot turn with its motors 180

Figure 3
Standard Servo
www.acroname.com

degrees to take 3 ultrasonic readings and instead leave our robot stationary as we rotated the actual ultrasonic sensor using the servo. This servo was hard to integrate into the code due to its need to receive a signal every 20 ms in order to function properly. If the servo did not receive a signal in this amount of time, the internal circuitry would shut down. We used pulse-width in order to rotate the servo to its desired position. This particular sensor required a minimum of 4.8 volts to run. It was very lightweight for its size, only 1.44 ounces. Its ability to rotate quickly, which according to spec was 0.23 seconds through a 60 degree revolution, also was an advantage to our final implementation.

The ability to make each of these sensors function individually was a challenge, but was in general easily done. The main obstacle was trying to integrate all 3 into the same code without having timing issues. It took lots of time and effort in order to do so, but it is achievable and was able to be done.

Implementation

Integrating each of the above sensors into our code was a long, drawn-out process. As mentioned earlier, implementing each of the following sensors individually was the first step, but trying to do so with all 3 in the same code was a challenge.

Our initial plan of attack was to use the basestation timer for the main timer in controlling the nodes. First, we began with motor control. We used the H-bridge manual and a few logic statements in order to get the robot to mobilize. Once we got the robot to move, we realized that the speed was too fast without pulse-width modulation. We foresaw a problem with power consumption, so we used the built-in PWM pins on the Micaz to set the motor to our desired speed. After we were satisfied with the movement of the robots, we proceeded to integrate the digital compass into our code.

The incorporation of the digital compass into our existing code was relatively simple. We simply set up general input/output pins on the Micaz to read from the outputs of the compass. We found immediately that the output on each pin of the compass was not what we expected. Instead of outputting a high signal when pointed in its respective direction, the compass would actually give a low output. For example, when facing north, instead of outputting a high signal on the north pin, it would instead output a zero and set the other three pins high. With some basic logic comparisons in our code, we were able to combine what would be four separate readings from the compass into a single hexadecimal number. As mentioned previously, in order to correct for compass accuracy and gear slippage in the motor, we began to implement a correction scheme. We used the hexadecimal number received from the compass in order to auto-correct our robot to head either north or south, i.e. if it started to veer away from its desired bearing, it would use our motor control code to get back on track. This concluded our initial compass code.

Next step was to add the ultrasonic code to our existing code. It worked flawlessly as a single unit, but once we tried to integrate it, we ran into some disastrous problems. For some reason, our ultrasonic readings began to be completely bogus and our motors were not functioning as before. After some debugging, we found the timing of node was thrown off. While we were taking ultrasonic readings, the basestation continued to send messages to the node, causing an interruption in the ultrasonic reading and giving us false results. This was a major obstacle that we had to troubleshoot. In order to solve this problem, we decided to make the nodes as independent as possible from the basestation. All the major functions we were performing using the basestation's timer, we transferred over to the node's timer. As soon as we did this, we immediately saw better results, and the readings from the ultrasonic were valid.

When we decided the ultrasonic was giving us legitimate data, we started to create the first stages of object detection. Our first plan of attack was to setup our timer in such a way that we would get ultrasonic readings every two seconds. We ran into a few problems during this step in our creation process. One major problem with the ultrasonic sensor was that the readings it would pick up while we were moving proved to be inaccurate. To remedy this conflict of movement and sensor functionality, we decided it would be best to stop our chassis completely every time we took an ultrasonic reading. To execute this, we stopped the motor using a stop-motor function. We also turned off the timer of the micaZ to ensure that no other processes were executing that would interrupt our ultrasonic. Once our ultrasonic fired and received its echo pulse, we restarted the timer and sent a packet back to the basestation for GUI implementation. The node use the reading to decide what future tasks it would carry out. If the reading was greater than 40 centimeters, the robot would continue moving forward for two seconds, and then repeat its ultrasonic process. However, once the ultrasonic reading became less than 40 centimeters, the node would enter another process that used our motor control functions to turn right and move forward for a certain distance. In the first stages of this object identification, the node would then continue trying to move north, even though there was a wall in its way. Our main objective here was to build a foundation for both our localization and searching algorithms.

Our next major and probably most important step in the node-building process was the implementation of the servo into our functioning code. We had found in the past that adding another sensor to an already-functioning code usually caused some problems. This instance was no different from the other times. Once we added the servo code, we tested and began to realize that the servo was functioning properly, but our ultrasonic readings became completely inaccurate. Making our servo function was a big step in our process due to its importance for our searching and localization. This meant that instead of rotating the entire chassis 180 degrees to get 3 readings from our ultrasonic, all we had to do was rotate our ultrasonic sensor 180 degrees with the servo. Without valid ultrasonic readings, though, this would be useless to us. After various hours of debugging, we finally came to the conclusion that we were clearing our pulse-width registers to accommodate the servo, which in turn caused problems for the ultrasonic. To accommodate this, we moved the servo code, which was previously nested in the same task as the ultrasonic, to another part of the code which used the timer to function instead of an interrupt.

As soon as we did this, our code began to function a lot better. Our ultrasonic readings were valid once again, and we were rotating the servo into 3 distinct positions without any problems. We were now ready to implement our algorithms for localization and searching, but first we had to get our communication scheme functioning.

Communication

Our initial communication scheme we had worked on was using the basestation timer to control the nodes and the nodes performing tasks based on what messages they received and when they received them. Both the node and the basestation were sending and receiving simultaneously. This was able to be done by sending different message types. For our case, the basestation was setup to send only type 4 messages and receive only type 5 messages. Our node was setup to receive only type 4 messages and send out type 5 messages only. As mentioned previously, this did not work due to timing issues mainly with the ultrasonic. We switched over to use the node timer instead for each of the individual nodes to make them as independent as possible. At this point, we had the basestation no longer sending out messages, but only receiving messages of type 5. Doing this solved the timing issues we were having with our ultrasonic sensor. Leaving the basestation in receive-mode was necessary for updating the GUI. The basestation would receive a message every time it received an ultrasonic reading, which occurred approximately every 1.2 seconds or whenever it detected a wall. In this message consisted distance measurements for each direction, the current compass reading, search status (whether the box had been detected), and node ID.

After we began to implement our localization and searching algorithms, we began to realize that our communication scheme was going to need some modifications. In order for our searching algorithm to be coordinated, we needed to setup a communication method where our first node could send out messages only the second node could receive and vice-versa. Creating our timing for this was tricky due to our robot movement being inconsistent. Initially we had it set up where whenever a node came within 40 centimeters to the north or south wall, it would send a message to the opposite node to start its timer. This, after a few tests, did not prove to be adequate due to the fact that the first node did not have enough time to finish its tasks and shut down before the second node reached the opposite wall. In other words, the second node would send out its startup signal for node 1 before node 1 had completed its tasks. To overcome this, we setup a scheme where node 1 would tell node 2 to start its timer when it hit the opposite wall. Node 2, however, did not instruct node 1 to start its timer when it hit the opposite wall, but rather waited until its ultrasonic readings were completed before it sent out a signal telling node 1 to start its timer. This alleviated all the timing problems and prevented both nodes from entering a point where both nodes would deactivate simultaneously.

We also had to setup a communication sequence where when one node found the black box in the arena, it would send out a message of a specific type to the opposite node. When this particular type of message was detected by the other node telling it specific coordinates, the receiving node overrode its search algorithm and entered a sequence of logic that drove it to match the other nodes coordinates. When the coordinates were matched and the desired location achieved, the node shut down its timer and deactivated. Once this communication scheme was made consistent, we were now able to implement our localization and searching algorithms.

Localization

Once this foundation was built, we decided to begin our creation of the localization algorithm. The use of the compass made our localization scheme reasonably easy to implement. Originally, the basic concept of our localization was to orient one node north and the other south, take three readings (north/south, east, and west), rotate our node 180 degrees, and take three more readings. This, however, worked better in theory than it did in actuality, so we opted to take a different approach. Instead, we started node 2 with its timer turned off, and oriented node 1 north. We then had node 1 drive until it hit the north wall.

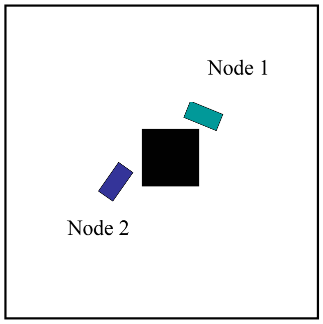


Figure 4
Localization Stage 0

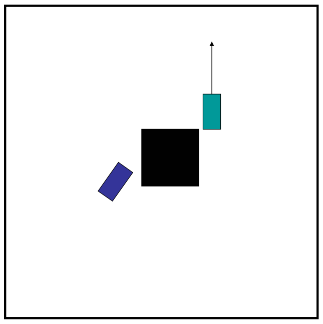


Figure 5
Localization Stage 1

Upon hitting the north wall, the node then sent out a signal that told node 2 to turn its timer on. When node 2 received this signal, it proceeded to orient itself south and start driving toward the south wall.

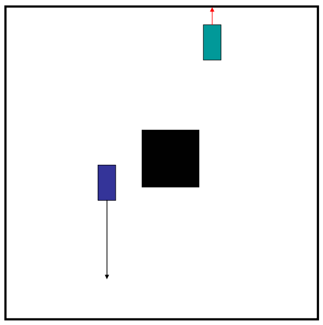


Figure 6
Localization Stage 2

Node 1 would proceed to turn east and drive toward the east wall until it was within 40 centimeters from that wall. About the time this was occurring, node 2 would begin approaching the south wall.

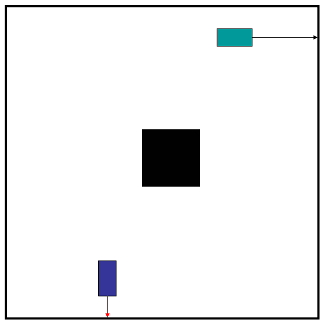


Figure 7
Localization Stage 3

Once node 1 reached the east wall, it would continue to turn south. When this was happening, node 2 would proceed to turn west once it came within 40 centimeters from the south wall.

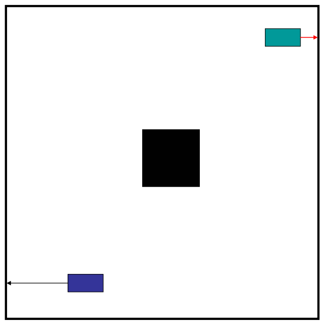


Figure 8
Localization Stage 4

Once node 1 was oriented south, using its servo, it would take three ultrasonic readings (east, south, and west). Node 1 would then shut its timer off and wait for a signal from node 2 to start its searching process. This would end the localization process for this node. Node 2 at this time would be approaching the west wall until it was within 40 centimeters.

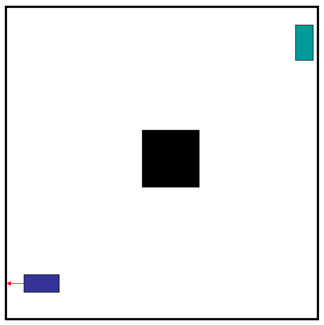


Figure 9
Localization Stage 5

Once node 2 was within 40 centimeters from the west wall, it would proceed to turn north. Once it was oriented north, it would take 3 ultrasonic readings using its servo (west, north, and east), and then send a signal to node 1 to start its timer and start the searching process. Node 2 would then shut off its timer and await a startup signal from node 1. This would end the localization process for node 2 and the overall localization of the two nodes.

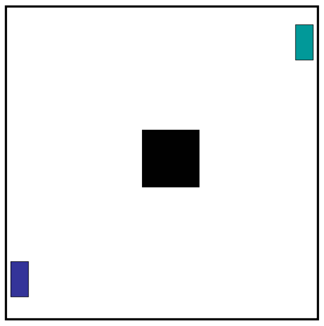
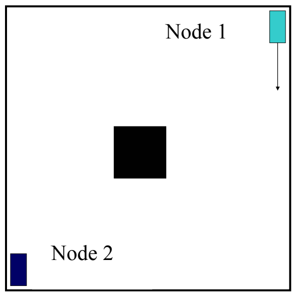


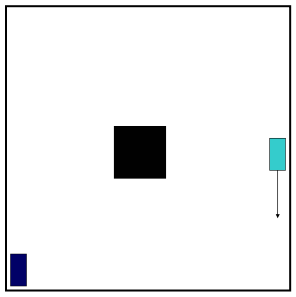
Figure 10
Localization Stage 6

Searching Algorithm

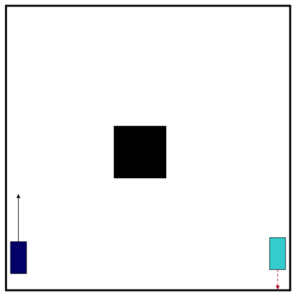
We had always planned on using a pseudo-coordinated searching algorithm, but halfway through the project, it was brought to our attention that a random searching algorithm would be easier to implement. However, we soon found that if we put more than one node in the arena at a time and let them roam, they're ultrasonic readings would interfere with one another. So, we decided to switch back to our coordinated searching algorithm which would cause the robot to sweep the arena until it hit a wall, then turn around and sweep in the opposite direction – the classic lawnmower effect. We used our communication to allow one robot to sweep individually, shooting off an ultrasonic signal every 1.2 seconds (a time we ended up implementing after many trials and lots of error), while the other unused robot remained stationary. The unused robot (initially node 2) would remain in this state, until it received a signal from the roaming node (initially node 1). This signal occurs at different times, depending on the node that is sending the signal. Node 1 sends the signal to start node 2 to as soon as node 1 hits a north or south wall. This is done so that node 2 will not get in the way of node 1's readings at anytime.



a)



b)



c)
Figure 11
Searching Algorithm Stages 1-3

Node 2 then starts heading in its respective direction, also sending ultrasonic signals every 1.2 seconds. During this time, node 1 turns west, and drives forward approximately 10 centimeters. This forward motion is done to ensure that our robot doesn't sweep the same area more than once. Once it offsets itself, node 1 then takes three readings again, relocalizing itself on the new southern wall.

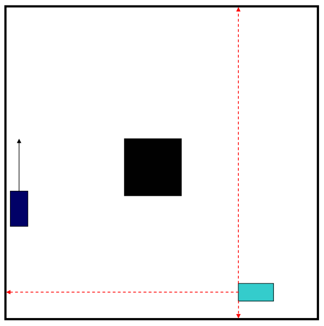


Figure 12
Searching Algorithm Stage 4

As node 2 continues its searching, node 1 realigns itself north to south, but this time facing the opposite direction it faced before hitting the wall. It then takes another three readings, verifying that the node is in the correct placement on the grid, and thus creating less error from bad initial reads on the first set of readings.

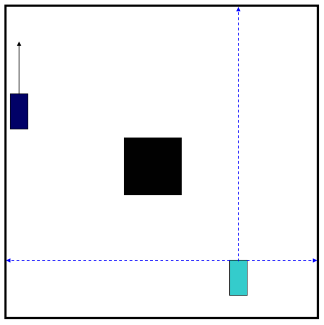


Figure 13
Searching Algorithm Stage 5

After node 1 has taken all of its readings, its timer is turned off, and it becomes the dormant node. Node 2 then hits the north wall, and prepares to turn itself around to sweep in the other direction.

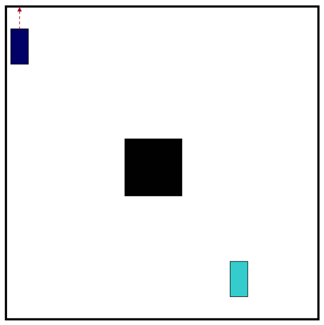


Figure 14
Searching Algorithm Stage 6

It turns itself east, offsets itself 10 centimeters, and takes three readings to localize itself again, just as node 1 did earlier.

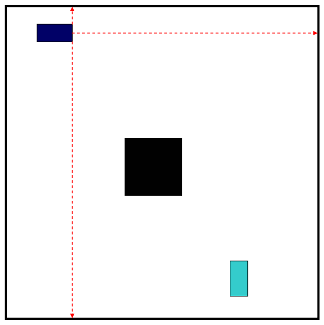


Figure 15
Searching Algorithm Stage 7

Node 2 then realigns itself south and takes another three readings, once again to reduce error that might occur in any sensor readings we take in the turnaround process.

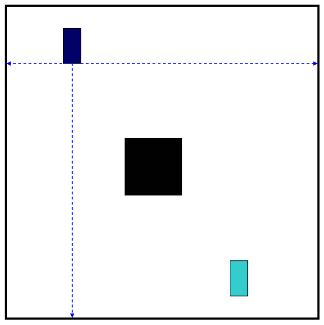


Figure 16
Searching Algorithm Stage 8

As soon as this second set of three readings is completed, node 2 sends out a signal to node 1, which starts node 1's timer again and starts its searching process over again.

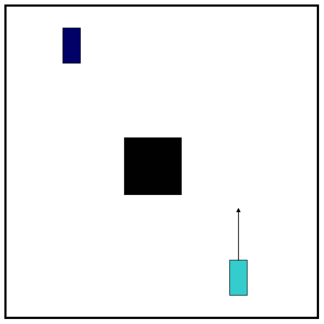


Figure 17
Searching Algorithm Stage 9

The nodes then repeat this process of sweeping the box until a possible detection is present in one of the nodes.

Box Detection

The problem with using ultrasonic sensors over infrared sensors lies within the box detection process. When dealing with lasers, and photo detection, it's easy to put a light sensor on the front of the node and paint the box black. This way, the node knows that it encounters the box when it picks up an absence of light. With our ultrasonic readings, we had to make comparisons of distances to determine whether or not we had a box in our sweeping area. To do this, we had our code save the distance each node recorded every time we hit a north or south wall. This distance was recorded into a variable named x1.

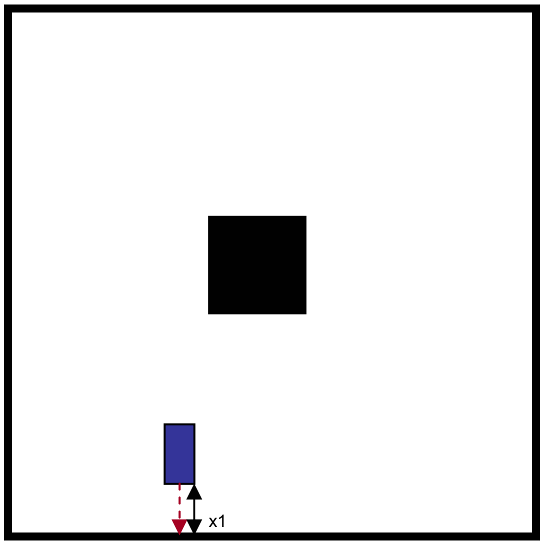


Figure 18
Box Detection Stage 1

We then also saved the distances taken when the node turned east or west. We placed these distances in variables as shown in the diagram:

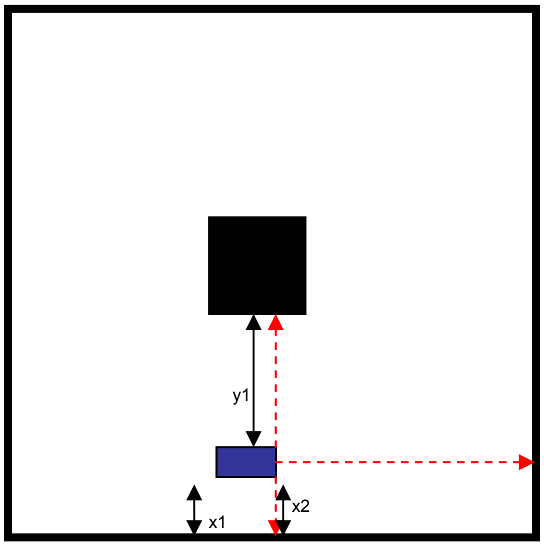


Figure 19
Box Detection Stage 2

We then saved one more reading, labeled y2, after we realigned north/south. The four distances saved were then added together and averaged to reduce error in readings. This averaged number was then compared to the total distance of the arena. If the averaged numbers summed to be less than 3 meters, the robot detected a box and proceeded to run code to verify the presence of the box.

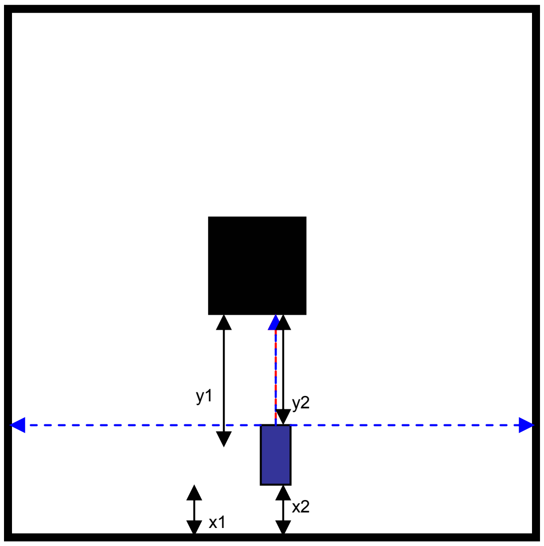


Figure 20
Box Detection Stage 3

To verify the presence of the box, the robot would drive in the box's direction, until the node was within 40 centimeters of the box. At this time, the node would take three readings to determine the coordinates of this possible box detection.

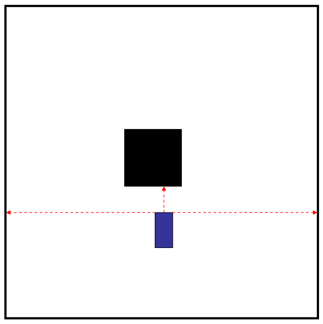


Figure 21
Box Detection Stage 4

The node would then save the distance it was from the box. This distance saved is marked as the red arrow in the following diagrams. The node then compares its east and west readings, and turns towards the greater distance of the two. This is done to ensure that our node doesn't turn towards a small distance and run straight into a wall. After turning east or west, the node then drives forward approximately 30 centimeters, which guarantees that we will clear the box from the line of our next three readings.

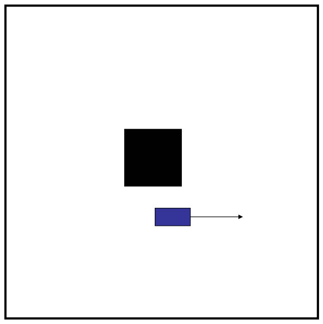


Figure 22
Box Detection Stage 5

After clearing the box, the node takes three more readings. It then compares the red arrow distance with the corresponding blue arrow distance. For instance, in the following diagram, the red arrow distance is compared to the north blue arrow distance. If the blue arrow distance is greater than the red arrow distance + 30 centimeters, we know that we were originally at the box. If not, the node knows that its original assumption of the location of the box was false, and it continues its searching algorithm.

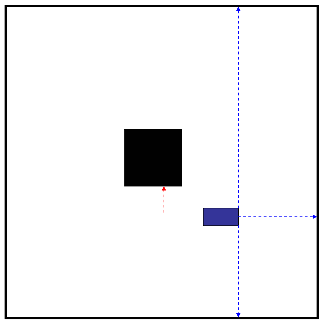
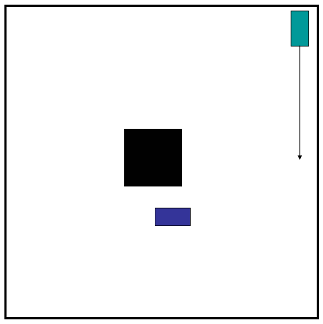
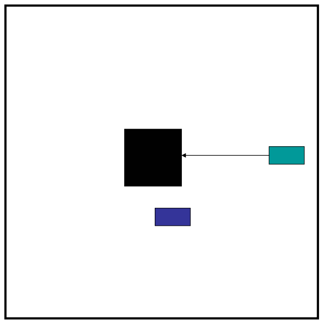


Figure 23
Box Detection Stage 6

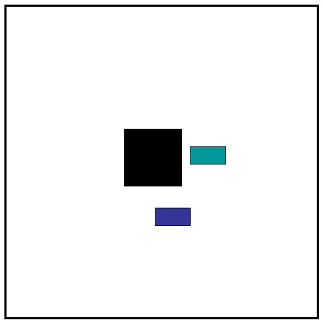
Once the node determines the placement of the box, it reverses itself next to the box and sends out the coordinates of the box to the other node. The second node then drives until its coordinates match those sent by the node that found the box.



a)



b)



c)
Figure 24
Box Detection Stage 7-9

Graphical User Interface

The Graphical User Interface (GUI) was written in Labview 7.1. This symbolic programming language is very useful for instrument control, and was relativity easy to adopt for the project requirements. It was determined that a simple x-y grid system with (0,0) being located in the lower left corner would be the best design to build the program around. This design minimized the logic manipulations needed and simplified the project.

First, data was taken from the serial port. It was then ran through a series of filters that stripped it of the Micaz's packet formatting (first byte, address byte, type byte, length byte, CRC byte, and last byte) leaving just the data. At this point, using the format chosen by the system programmers the data block was broken apart into the individual pieces of data (North distance, East Distance, South distance, West distance, current direction, search status, and node id).

Now the 4 cardinal directions, their distances, and the current direction data were fed into a logic sequence that varied based on the current direction. With the zero point (0,0) located in the lower left corner, it was found that to accurately plot with respect to the north and east distances that they had to be subtracted from the overall size of the arena (300 cm was the value used). An example of the logic sequence for data taken while the direction was north is shown below.

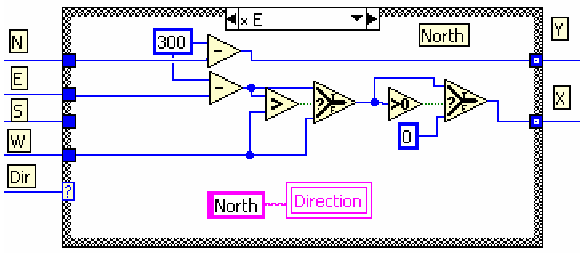


Figure 25
Logic sequence used to determine the (x,y) coordinates of the node when traveling north.

At this point the data was checked to see if it was from node 1 or node 2. This was accomplished with two comparators with one set to 1 and the other comparator set to 2. If the node id matched for the particular comparator than the new (x,y) data was passed and plotted. If it did not match, then a zero was passed. After the comparator, a zero check was implemented to minimize bad serial reads from affecting the plotting accuracy. If a zero was detected, then using a feedback node, the previous data plots were preserved. Below is an example of this implementation.

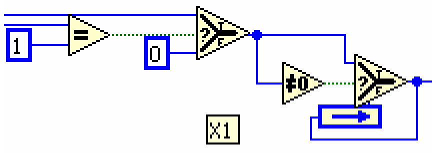


Figure 26
Logic sequence used for the y coordinates of node 1 to determine node source, and check for bad data.

With the data checked and associated with a node it is passed to a multi-xy plot function. This function worked well for this project. It was easily configurable to match the 300 x 300 cm arena, and allowed for on the fly changes to the way the nodes were displayed.

The required start button was implemented in Labview, but due to time restrictions was never implemented on the Micaz side, and therefore could not be tested. The idea behind the implementation was that an array would be created. The bytes that were prone to change would be set as variables, and the ones that were constants were preset. The variables would be taken from logic that set them based on what type of message it was, what the destination address was, the length of the data, the data to be sent, and the CRC byte. The example below is the array that the system programmers agreed would be used to start the nodes on their search sequence.

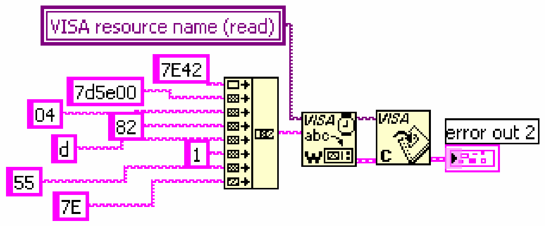


Figure 27
Start button control array with serial write and serial port close functions.

Interfacing and Power Circuitry

The circuitry required to power and interface the different sensors for this project was very basic. The first circuit covered will be the 5 volt regulator. This circuit consists of an LM7805 T0-220 package voltage regulator and heat sink, a 1 µF electrolytic capacitor, and a 0.1 µF electrolytic capacitor.

The second circuit to be discussed is the level converting circuit. Through testing, it was determined that the 2.6-3.3 volt signal from the

Micaz was not sufficient to drive the trigger line of several of the sensors. To remedy this, it was decided to implement a 3-to-5 volt level converter. This circuit consists of an NPN and a PNP transistor in the configuration shown below. The circuit is powered off of the 5 volt regulator, has an input that is connected to the Micaz, and an output that is connected to the input of the desired sensor.

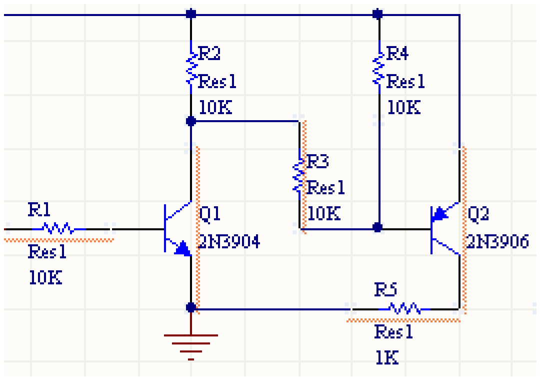


Figure 28
Voltage level converting circuit.

The third circuit to be discussed is the Pulse Width Modulation (PWM) signal multiplier. During the integration process it was determined that the project needed more PWM signals than were available on the Micaz. To remedy this problem, it was decided to control the motors with a single PWM channel. The requirements were that each motor would receive the same PWM signal, and either motor or both could be disconnected from the PWM signal. This was accomplished with a simple dual AND gate. The PWM signal was connected into an input on both AND gates, and a General Purpose Input/Output (GPIO) pin from the Micaz was connected to each of the remaining inputs to the AND gates. The GPIO pins enabled or disabled the PWM signal to the individual motors. Below is an example of how the circuit was wired.

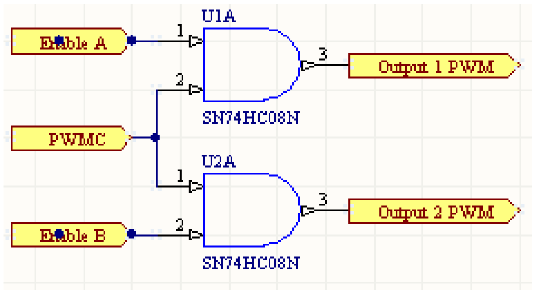


Figure 29
PWM signal multiplier circuit.

The final circuit to discuss is the compass interface board. This simple board allows for easy powering, grounding, and connecting the output lines to jumper wires. In addition, to give a visual aid, leds were connected to the output lines. The example below show how this board was wired.

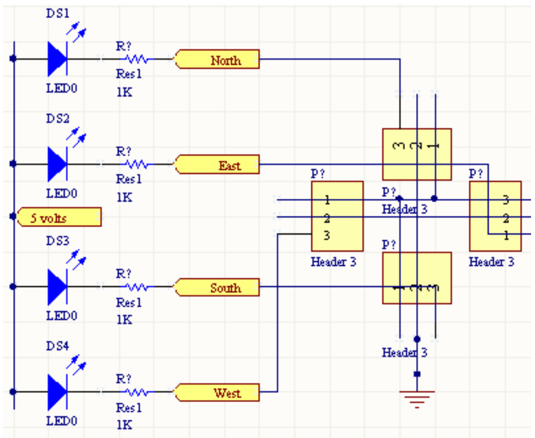


Figure 30
Compass interface circuit with led indicators.

Power Sources for the Mobile Nodes

The main power source for the mobile nodes was chosen to be a 9.6 volt NiMH 1250 mAh battery, which is regulated down with the 5 volt regulator circuit described above. This battery required pressure contacts on its side for proper hookup. In addition, the Micaz was independently powered by two AA batteries. As a last minute addition, a 4xAA battery pack was added. Its purpose was to provide independent power for the servo.

Problems and Recommendations

Throughout this project the group ran into numerous problems. Detailed here are the problems that were faced and how they were overcome, or recommendations on how to possibly overcome these problems.

Power:

Behavior: This problem was detected during the integration phase when the servo was added to the chassis. The behaved randomly if at all when it was connected to the 5 volt regulator circuit, and work to specification when it was connected to its own power source.

Cause: Power calculations were done incorrectly. One possibility is that the lab power supply was inadequate to power the motors, or it was incorrectly displaying the current draw.

Solution: Possible solution is to replace the 1 amp 7805 regulator with a 1.5 amp or even a 3 amp version of the 7805. If weight is not an issue, use of additional battery pack will resolve the issue.

Wiring:

Behavior: Random behavior of individual or all systems on the node. Node was functioning, then stops between test runs.

Cause: Usually a wire or group of wires was knocked loose during handling. This is common when removing the Micaz for re-programming, transporting the node between the bench and the test arena, bumping the node, weak/worn wires/contacts, and weak connections to common ground points.

Solution: Use caution when handling the node. Make a printed circuit board with all necessary connections to minimize jumper wires between boards. Make sturdy wire harnesses. These make re-wiring the node much easier, reduce time spent testing each individual wire, clean up the node, and reduce the chance of a single loose wire.

GUI(plotting):

Behavior: While program is plotting the location of the nodes it occasionally plots the node inaccurately.

Cause: There are several possible causes for this behavior. One is that there was a bad serial port read that resulted in junk data. Another possible cause is that the simple logic used to determine the (x,y) coordinates is insufficient in certain cases. Another possible cause is that the program lacks sufficient data from the Micaz to accurately plot the nodes location.

Solution: There are not really any simple solutions to this problem. Data taken from the serial port is bound to occasionally be out of sync. More detailed checks could be added to the data before processing. As for the faulty logic for certain cases, those cases need to be determined and special cases added to specifically handle them. A known problem is the lack of data from the Micaz. The only way to handle this is to add several more data reading into the behavior code of the node.

Flat tires:

Behavior: Node seems to veer, and tires appear close to flat.

Cause: Weight of the node is improperly distributed, and reaching the nodes limit.

Solution: To stiffen up the tires, fill with 6mm plastic AirSoft rounds. Be careful not to overfill. Note that after fill tires weight may need to be readjusted.

Configuration for the Hardware of the Robot

For the design of our nodes, we tried to keep it as simple as possible. We started with a 12x15 cm base of Plexiglas with squares cut out of the back for the wheels attached to the motor. The center of the base was where we put the battery. The batteries power connectors also worked as a way to keep the battery attached to the node. In the back of the base was a single wire bent so that the node was sitting flat with the middle of the wire touching the ground and the two ends of the wire coming up and screw into the base. This made it so that our node had a tripod base with the two motor wheels in front and the wire in the back.

We used a second layer of Plexiglas that was 15x15 cm that was attached with three spacers for stability, two near the wheels and one near where the wire touches the ground. We used a standard servo, which we attached in the front of the node. On top of the servo was the ultrasonic sensor, which was centered so that it would be able to take readings from in front of the node as well as to the left and right. Just behind the servo was a foot tall length of metal on top of which was the digital compass. The digital compass had to be put at least a foot above the node because the compass would read some interference from the rest of the node.

The back of the node is where the MicaZ was attached to the connector board which allowed us to separate all the information going from the MicaZ to all of the other parts via wires. Through the wires, the MicaZ is connected to the servo, compass, ultrasonic sensor, and H-bridge. The H-bridge is located on the bottom side of the second layer just under where the MicaZ is located. The H-bridge is also connected to the motor, which allows the signal from the MicaZ move the motor. The H-bridge is also connected to a five-volt regulator, which is located on the topside of the second layer right next to the MicaZ.

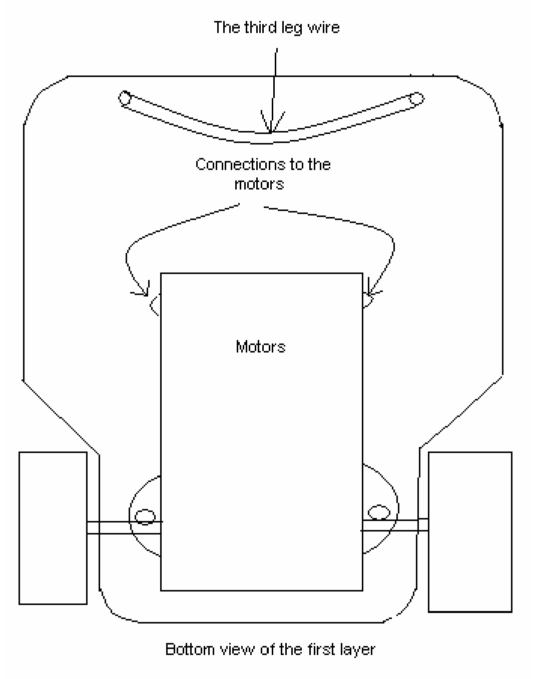


Figure 31

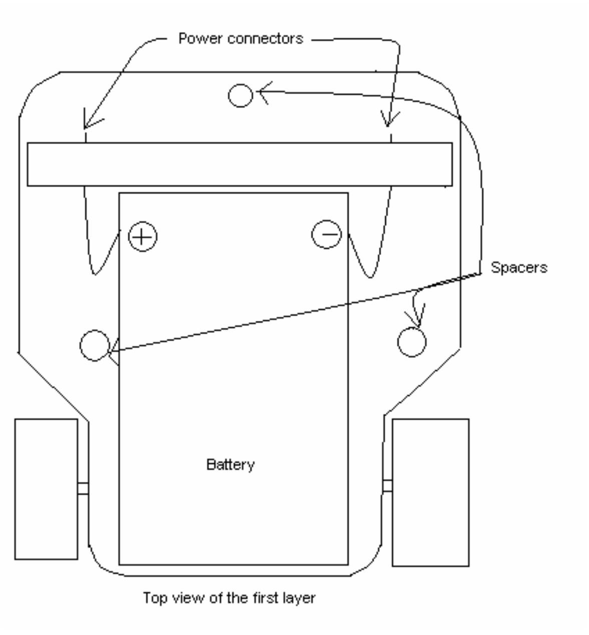


Figure 32

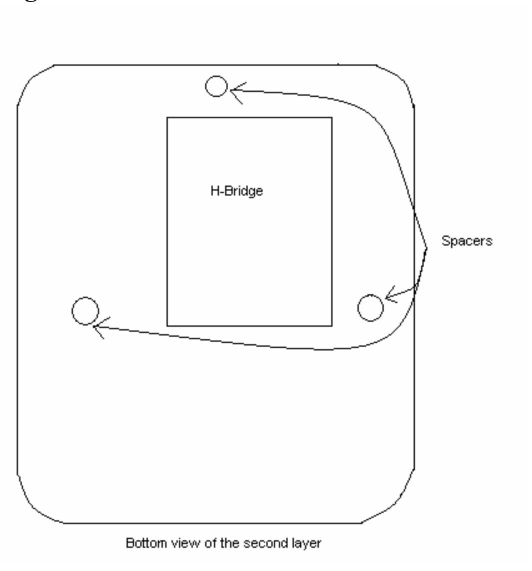


Figure 33

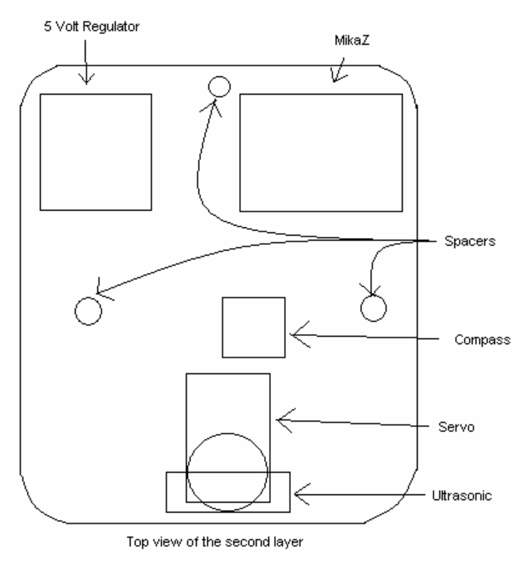


Figure 34

Hardware mistakes

There are a few things that would have made the chassis better. One of these things is the back wire. The wire did not allow for shock absorbing since it was a very rigid wire, so any vibrations the node felt shook the entire node. Vibrations were bad because the part that felt the vibrations the most because of distance from the bottom was the compass and it was the part that needed to remain level. Having either shock absorbing material on the back or having two back wires could have lessened the vibrations of the node. The two back wires would have made it so that the robot would not shake easily from side to side due to its tripod design.

The wires were also an issue since they were everywhere and had a tendency to come unconnected from their connectors, which caused several problems with troubleshooting. This could have been remedied with better wires and also taking out some of the bad wiring that came with some of the circuitry originally.

Final Budget

			TOTAL
ITEM	QTY	PRICE	
Ultrasonic Sensor	2	$25	$50
Servo	3	$10.90	$32.70
Compass	3	$13	$44
Heat Sinks	3	$1.67	$5.41
Battery	2	$11.75	$23.51
Battery boxes	2	$3.08	$6.08
Interface Boards	2	$6.50	$13
Machine shop			$4
Misc. Circuit Board Parts			$37.54
Shipping costs			$1.74
Total			$217.98
Left			$32.02

Figure 35
Final Budget

Conclusion

Our final status on the project is almost complete. When compared to the beginning of the semesters project specifications we felt that we were only truly lacking on two of the points. We had yet to implement a start button on our GUI that would start the entire event and our nodes had yet to become independent. The GUI was thrown together within the last week, which we really have no excuse for, and the nodes had to communicate to each other to start moving. This was so they were not firing the ultrasonic at the same time and interfering with each other's readings, thusly making them reliant on the signal to start moving. Both problems we feel, given a couple more days, we could have implemented.

During the semester we ran into quite a few problems that subsequently put us behind schedule. The integration of the entire project took us longer than we expected, and there were quite a few days that we felt we made no progress at all. We had the ultrasonic code working by itself, as well as the servo, but when trying to combine them all together had quite a few timing/coding problems. Things we believed should work in theory ended up interfering with our other subsystems. During the last week we had problems with one of our chassis, making it necessary to change our three-node system to a two-node system. The wiring of our systems ended up being a large problem for us since a lot of the time we spent trying to debug the code was actually a wiring problem. Given more time we would have built wiring harnesses, alleviating most of our wiring problems, making our chassis look neater, and make debugging simpler.

Overall the project was stressful, challenging and demanding. As a group we feel we have accomplished a lot, both in terms of the actual project and in terms of engineering skills. We learned that when scheduling tasks to overestimate the time of completion, redistribute resources according to priority and timeline, and that planning is the most important part of a project

References

[1] (2006, Feb) Space and Naval Warfare Systems Center: Factors Determining Cost-Optimal Design. [Online] Available: http://www.spawar.navy.mil/robots/research/manyrobo/costfactors.html

[2] (2006, Feb) Space and Naval Warfare Systems Center: Models for Coordinated and Random Searches. [Online] Available: http://www.spawar.navy.mil/robots/research/manyrobo/searchmodels.html

[3] (2006, Feb) Exact String Matching Algorithms. [Online] Available: http://www-igm.univ-mlv.fr/~lecroq/string/index.html

[4] (2006, Feb) Big O Notation. [Online] Available: http://en.wikipedia.org/wiki/Big_O_notation.

[5] C. Z. Janikow, "Adaptable Constrained Genetic Programming: Extensions and Applications," *NASA Summer Faculty Fellowship Program 2004*, vol. 1 and 2, pp 11-1 - 11-7, August 2005.

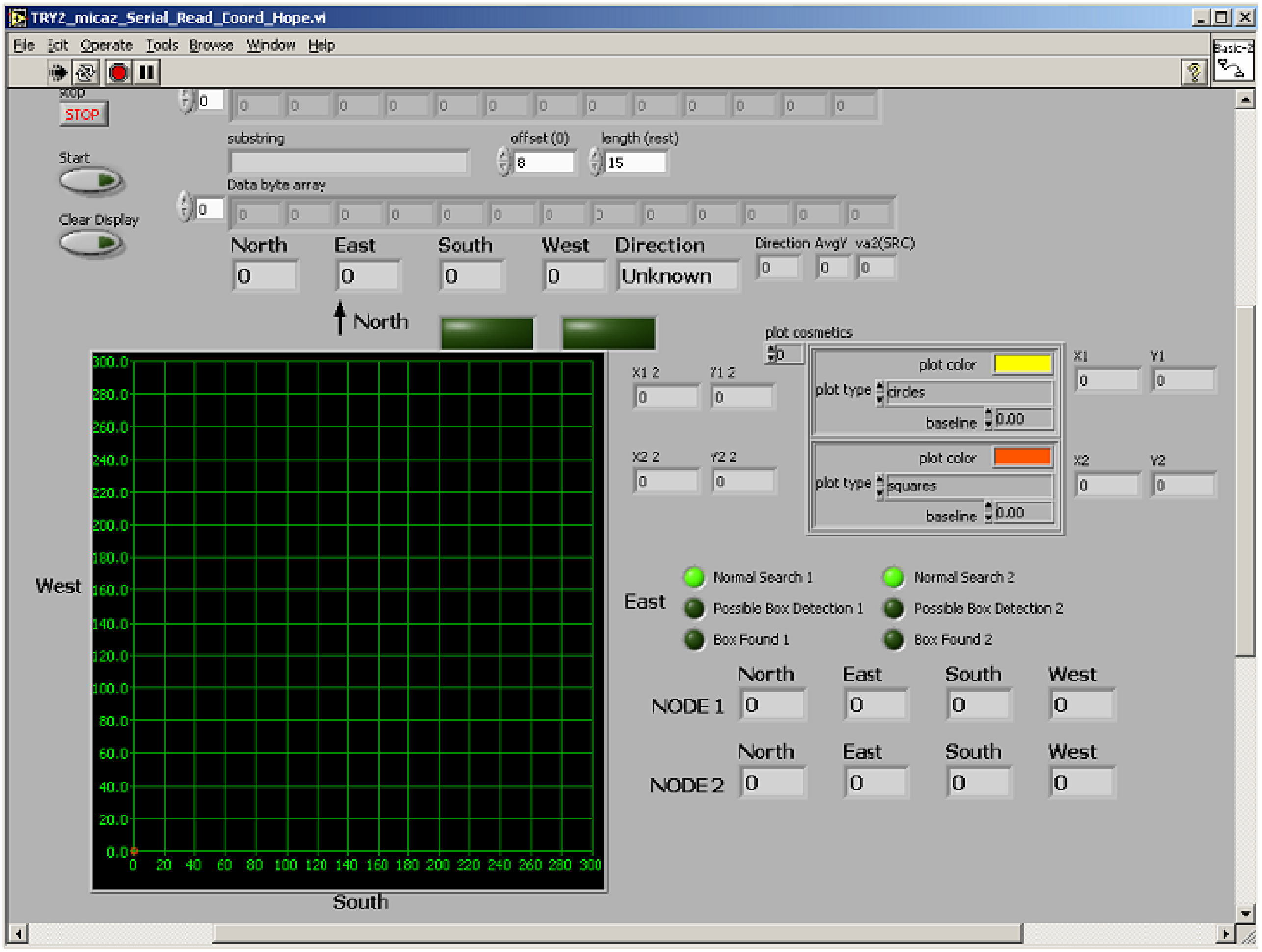
[6] B. Joshi, D. Morris, N. White, and R. Unal, "Optimization of Operations Resources via Discrete Event Simulation Modeling," 6th AIAA/USAR/NASA/ISSMO Symposium on Multidisciplinart Analysis and Optimization, September 1996

[7] (2006, Feb) Genetic Algorithms and Evolutionary Computation. [Online] Available: http://www.talkorgins.org/faqs/genalg/genalg.html#examples:robotics

[8] S. Koenig, B. Szymanski and Y. Liu, "Efficient and Inefficient Ant Coverage Methods," *Annals of Mathematics and Artificial Intelligence - Special Issue on Ant Robotics*, vol. 31, pp. 41-76, 2001

[9] W. Burgard, M. Moors, C. Stachniss and F. Schneider, "Coordinated Multi-Robot Exploration," *IEEE Transactions on Robotics*, vol. 21, pp.376-378, 2005.

Appendix A
Screenshot of the GUI



offset (0): 8
length (rest): 15
Direction: Unknown
plot type: circles, baseline 0.00
plot type: squares, baseline 0.00

Appendix B
Final Code

Main Basestation Code

```
// $Id: ReceiveDataM.nc,v 1.0 4/10/2005$
/*
 * Authors: Aly El-Osery
            Steven Myers
            Anthony Duran
 * Date last modified:  4/10/2005
 *
 */

/**
 * @author  Aly El-Osery
            Steven Myers
            Anthony Duran
 */


includes IntMsgg;

module ReceiveDataM {
  provides interface StdControl;
  uses {
    interface ReceiveMsg;
    interface MsgOutput;
    interface StdControl as CommControl;
    interface Leds;
  }
}
implementation {

  command result_t StdControl.init() {
    call Leds.init();
    call Leds.redOff();
    call Leds.yellowOff();
    call Leds.greenOff();
    return call CommControl.init();
  }

  command result_t StdControl.start() {
    return call CommControl.start();
  }

  command result_t StdControl.stop() {
    return call CommControl.stop();
  }

  event TOS_MsgPtr ReceiveMsg.receive(TOS_MsgPtr m) {
    IntMsgg *message = (IntMsgg *)m->data;
    call Leds.redToggle();
    call MsgOutput.output(*message);

    return m;
  }

  event result_t MsgOutput.outputComplete(result_t success) {
    return SUCCESS;
  }

}
```

Northeast Node Main Code

```
/* Authors: Aly El-Osery
            Steven Myers
            Anthony Duran
*/

includes IntMsgg;

module RfmToIntM {
      provides interface StdControl;
      uses{
            interface StdControl as CommControl;
            interface Leds;
                  interface ReceiveMsg as ReceiveIntMsg;
            interface SendMsg as Send;
                  interface Timer;
      }
}

implementation{
      TOSH_ASSIGN_PIN(PW8, G, 0);
      TOSH_ASSIGN_PIN(PW9, G, 1);
      TOSH_ASSIGN_PIN(PW10, G, 2);
      uint16_t x1, x2, y1, y2, avgY;
      int check;
      int box;
      int hookerbot;
      int state;
      int sense;
      int thing;
      int north;
      int south;
      int stopping;
      int goingnorth;
      int goingsouth;
      int otherway;
      int limit;
      int servo;
      int stoprobot;
      int increase;
      int bling;
      int reversing;
      int temp;
      int hope;
      int servohope;
      int begin;
      int drunk;
      int wall;
      int detect;
      int firstime;
      int verify;
      int timing;
      bool pending;
      struct TOS_Msg data;
      IntMsgg *distmsg = (IntMsgg *)data.data;
      int first;  //# of times trigger function called
      uint16_t widthcnt;      //65534 counter for trigger
      int start;
      uint16_t way;
      uint16_t comp;
      uint16_t nine;

      command result_t StdControl.init() {
                  x1 = x2 = y1 = y2 = 0;
                  avgY = 0;
                  box = 0;
                  check = 0;
            state = 0;
                  stoprobot = 0;
            sense = 0;
                  first = 0;
                  servo = 0;
                  widthcnt = 0;
                  start = 0;
                  thing = 0;
                  drunk = 30;
                  increase = 0;
                  wall = 0;
                  bling = 0;
                  temp =  0;
                  goingsouth = 0;
                  goingnorth = 0;
                  limit = 0;
                  stopping = 0;
                  begin = 0;
                  detect = 0;
                  timing = 0;
                  hope = 0;
                  servohope = 0;
                  hookerbot = 0;
                  firstime = 0;
                  verify = 0;
                  north = 0;
                  south = 0;
                  reversing = 0;
                  way = 0x000e;
                  call CommControl.init();
            call Leds.init();
            return SUCCESS;
      }

      command result_t StdControl.start() {
            call CommControl.start();
                  return call Timer.start(TIMER_REPEAT, 300);
      }
```

```
      command result_t StdControl.stop() {
                  call Timer.stop();
            return call CommControl.stop();
      }

event TOS_MsgPtr ReceiveIntMsg.receive(TOS_MsgPtr m) {

    IntMsgg *message = (IntMsgg *)m->data;

      if(message->avgY == 0){
            call Timer.start(TIMER_REPEAT, 300);
      }
      else if(message->avgY == 1){
            call Timer.start(TIMER_REPEAT, 300);
      }
      else if(message->avgY == 2){
            box = 3;
            otherway = message->dir;
            goingnorth = message->north;
            goingsouth = message->south;
            call Timer.start(TIMER_REPEAT, 300);
      }

    return m;
}

task void Forward(){
    TOSH_CLR_PW0_PIN();
    TOSH_SET_PW2_PIN();
    TOSH_CLR_PW6_PIN();
    TOSH_SET_PW4_PIN();
}

task void TurnRight(){
      TOSH_CLR_PW0_PIN();
      TOSH_CLR_PW2_PIN();
      TOSH_SET_PW6_PIN();
      TOSH_CLR_PW4_PIN();
}

task void TurnLeft(){
      TOSH_SET_PW0_PIN();
      TOSH_CLR_PW2_PIN();
      TOSH_CLR_PW6_PIN();
      TOSH_CLR_PW4_PIN();
}

task void MotorStop(){
    TOSH_CLR_PW0_PIN();
    TOSH_CLR_PW2_PIN();
    TOSH_CLR_PW6_PIN();
    TOSH_CLR_PW4_PIN();
}

task void ReverseThatBiatch(){
      TOSH_SET_PW0_PIN();
    TOSH_CLR_PW2_PIN();
    TOSH_SET_PW6_PIN();
    TOSH_CLR_PW4_PIN();
}

task void Print_Robot(){
            call Send.send(2, sizeof(IntMsgg), &data);
}

task void Print_BaseStation(){
            call Send.send(0, sizeof(IntMsgg), &data);
}

task void BoxFound(){
      box = 2;
      distmsg->avgY = box;
      if(reversing <= 9){
            post ReverseThatBiatch();
            reversing++;
      }
      else{
            post MotorStop();
            distmsg->north = north;
            distmsg->south = south;
            post Print_Robot();
            post Print_BaseStation();
            call Timer.stop();
      }
}



task void SetupSonar(){

      int temp2 = 0;
      //Left Wheel
      OCR3CH = 0x00;
      OCR3CL = 0x00;
      TCCR3B = 0x42;                  //set clock with 8 prescaler, about
912.5kHz,
      TCCR3A = 0x00;

      DDRE = 0x80;                    //Set PE7 as input
      DDRC = 0x00;                    //set PC7 as output

      atomic{
            TOSH_SET_PW1_PIN();        //sets pw6 high
            while(temp2 < 250){
                  temp2++;
            }
            temp2 = 0;
            TOSH_CLR_PW1_PIN();
      }
      SREG = 0x80;      //Global Interrupt enable (pg9)
      ETIMSK = 1<<TICIE3;                   //enable Timer3, capture, and
interrupt
      return;
}
```

```
task void BoxMove(){
      if(increase == 4){
            increase = 0;
            post MotorStop();
            call Timer.stop();
            post SetupSonar();
      }

      else{
            OCR3AH = 0x46;                //sets period to ~20ms
            OCR3AL = 0x50;
            OCR3CH = 0x2A;
            OCR3CL = 0x30;
            TCCR3A = 0x2B;
            TCCR3B = 0x5A;
            increase++;
            if(way == 0x000e  || way == 0x000c || way == 0x0006){
                  if(distmsg->north >= goingnorth){
                        post Forward();
                  }
                  else if(otherway == 0x000d || otherway == 0x000c ||
otherway == 0x0009){
                        if(limit == 0){
                              if(start <= 3){
                                    OCR3CH = 0x20;
                                    OCR3CL = 0x00;
                                    post TurnLeft();
                                    start++;
                              }
                              else{
                                    limit = 1;
                              }

                        }
                        else if(distmsg->north > 10){
                              post Forward();
                        }
                        else if(distmsg->north <= 10){
                              post MotorStop();
                        }
                  }

                  else if(otherway == 0x0007 || otherway == 0x0006 ||
otherway == 0x0003){
                        if(limit == 0){
                              if(start <= 3){
                                    OCR3CH = 0x20;
                                    OCR3CL = 0x00;
                                    post TurnRight();
                                    start++;
                              }
                              else{
                                    limit = 1;
                              }

                        }
                        else if(distmsg->north > 10){
                              post Forward();
                        }
                        else if(distmsg->north <= 10){
                              post MotorStop();
                        }
                  }
            }

            else if(way == 0x000b || way == 0x0009 || way == 0x0003){
                  if(distmsg->south >= goingsouth){
                        post Forward();
                  }
                  else if(otherway == 0x000d || otherway == 0x000c ||
otherway == 0x0009){
                        if(limit == 0){
                              if(start <= 3){
                                    OCR3CH = 0x20;
                                    OCR3CL = 0x00;
                                    post TurnRight();
                                    start++;
                              }
                              else{
                                    limit = 1;
                              }

                        }
                        else if(distmsg->south > 10){
                              post Forward();
                        }
                        else if(distmsg->south <= 10){
                              post MotorStop();
                        }
                  }

                  else if(otherway == 0x0007 || otherway == 0x0006 ||
otherway == 0x0003){
                        if(limit == 0){
                              if(start <= 3){
                                    OCR3CH = 0x20;
                                    OCR3CL = 0x00;
                                    post TurnLeft();
                                    start++;
                              }
                              else{
                                    limit = 1;
                              }

                        }
                        else if(distmsg->south > 10){
                              post Forward();
                        }
                        else if(distmsg->south <= 10){
                              post MotorStop();
                        }
                  }
            }
```

```
      }
}

task void BoxTest(){

      if(firstime == 0 || firstime == 2){
            post MotorStop();
            servohope = 1;
            if(timing < 7){
                  if(timing == 0){
                        OCR3AH = 0x46;                //sets period to
~20ms
                        OCR3AL = 0x50;
                        TCCR3B = 0x5A;                //set clock with
8 prescaler, about 912.5kHz,
                        TCCR3A = 0x2B;
                        hope++;
                  }
                  if(hope == 1 && timing == 0){
                        // Left
                        OCR3B = 0x07be;          //551us
                  }
                  else if(hope == 2 && timing == 0){
                  // Neutral
                        OCR3B = 0x04da;     //1.25ms
                  }
                  else if(hope == 3 && timing == 0){
                        // Right
                        OCR3B = 0x01fb;      //2.15ms
                  }
                  else if(hope == 4 && timing == 0){
                        OCR3B = 0x04da;
                  }
                  timing++;
            }
            else{
                  if(hope == 4){
                        sense = 0;
                        hope = 0;
                        temp = 0;
                        if(firstime == 0){
                              firstime = 1;
                        }
                        else{
                              firstime = 3;
                        }
                  }
                  timing = 0;
                  call Timer.stop();
                  post SetupSonar();
            }
      }

      else if(firstime == 1){
            if(verify <= 3){
                  if(way == 0x000e){
                        hookerbot = way;
                        detect = distmsg->north;
                        if(distmsg->west > distmsg->east){
                              way = 0x0007;
                              post TurnLeft();
                        }
                        else{
                              way = 0x000d;
                              post TurnRight();
                        }
                  }
                  else if(way == 0x000b){
                        hookerbot = way;
                        detect = distmsg->south;
                        if(distmsg->west > distmsg->east){
                              way = 0x0007;
                              post TurnRight();
                        }
                        else{
                              way = 0x000d;
                              post TurnLeft();
                        }
                  }
                  verify++;
            }
            else{
                  post Forward();
                  if(verify >= 12){
                        verify = 0;
                        firstime = 2;
                  }
                  else{
                        verify++;
                  }
            }
      }

      else if(firstime == 3){
            if(hookerbot == 0x000e){
                  if(north >= (detect + 30)){
                        post BoxFound();
                  }
                  else{
                        box = 0;
                        sense = 0;
                        servohope = 0;
                        avgY = 0;
                        call Timer.start(TIMER_REPEAT, 300);
                  }
            }
            else if(hookerbot == 0x000b){
                  if(south >= (detect + 30)){
                        post BoxFound();
                  }
                  else{
                        box = 0;
                        sense = 0;
                        servohope = 0;
                        avgY = 0;
                        call Timer.start(TIMER_REPEAT, 300);
                  }
            }
      }
}

event result_t Timer.fired(){
            TOSH_MAKE_PW8_INPUT();
            TOSH_MAKE_PW9_INPUT();
            TOSH_MAKE_PW5_INPUT();
            TOSH_MAKE_PW7_INPUT();
            distmsg->val2 = 0x01;
      distmsg->dir = (0x0001 & (TOSH_READ_PW8_PIN() * 0x00FF)) |
(0x0002 & (TOSH_READ_PW9_PIN() * 0x00FF)) | (0x0004 &
(TOSH_READ_PW5_PIN() * 0x00FF)) | (0x0008 & (TOSH_READ_PW7_PIN() *
0x00FF));
            comp = distmsg->dir & way;
            nine = comp | way;
            TOSH_SET_PW3_PIN();
            TOSH_SET_PW10_PIN();
            TOSH_MAKE_INT0_OUTPUT();
            TOSH_MAKE_INT1_OUTPUT();

            if(begin == 0){
                  OCR3AH = 0x46;                //sets period to ~20ms
                  OCR3AL = 0x50;
                  OCR3B = 0x04da;     //1.25ms
                  TCCR3A = 0x2B;
                  TCCR3B = 0x5A;
            }

            if(drunk == 30){
                  distmsg->north = 100;
                  drunk = 0;
            }

            if(distmsg->dir == way){
                  if(wall == 1){
                        temp = 10;
                        sense = 4;
                        wall = 0;
                  }
                  else{
                        state = 4;
                  }
            }
      else if(comp == 0x000c){
                  state = 3;
            }
      else if(comp == 0x0003){
                  state = 3;
          }
          else if(comp == 0x0006){
                  state = 2;
      }
          else if(comp == 0x0009){
                  state = 2;
      }
          else if(nine == 0x000e){
                  state = 3;
            }
          else if(nine == 0x000b){
                  state = 2;
      }

            if(way == 0x000d){
                  if(distmsg->dir == way){
                        state = 4;
                  }
                  else if(comp == 0x000c){
                        state = 2;
                  }
                  else if(comp == 0x0009){
                        state = 3;
                  }
            }

            if(box == 3){
                  post BoxMove();
            }

            else if(box == 2){
                  post BoxFound();
            }

            else if(sense == 4){

                  post MotorStop();

                  if(temp == 10){
                        if(bling < 7){
                              if(bling == 0){
                                    OCR3AH = 0x46;
      //sets period to ~20ms
                                    OCR3AL = 0x50;
                                    TCCR3B = 0x5A;                //set
clock with 8 prescaler, about 912.5kHz,
                                    TCCR3A = 0x2B;
                                    servo++;
                              }

                              if (servo == 1 && bling == 0){
                                    // Left
                                    OCR3B = 0x07be;          //551us
                              }
                              else if (servo == 2 && bling == 0){
                                    // Neutral
                                    OCR3B = 0x04da;     //1.25ms
                              }
                              else if (servo == 3 && bling == 0){
                                    // Right
                                    OCR3B = 0x01fb;      //2.15ms
                              }
                    else if (servo == 4 && bling == 0){
                        OCR3B = 0x04da;
                    }
                    bling++;
                }

                else{
                    if(servo == 4){
                        sense = 0;
                        servo = 0;
                        temp = 0;
                        if(begin != 0 && distmsg->dir ==
way){
                            stoprobot = 1;
                        }
                    }

                    bling = 0;
                    call Timer.stop();
                    post SetupSonar();
                }
            }

            else{
                sense = 0;
                call Timer.stop();
                post SetupSonar();
            }
        }

        else if(sense == 15){

            OCR3AH = 0x46;                //sets period to ~20ms
            OCR3AL = 0x50;
            OCR3CH = 0x20;
            OCR3CL = 0x00;
            TCCR3A = 0x2B;
            TCCR3B = 0x5A;

            if(thing == 0){
                if(start <= 3){
                    if(way == 0x000e){
                        if(begin == 0){
                            post TurnRight();
                        }
                        else{
                            post Print_Robot();
                            post TurnLeft();
                        }
                    }
                    else{
                        post Print_Robot();
                        post TurnRight();
                    }
                    start++;
                }
                else{
                  if(begin == 0){
                        post MotorStop();
                        start = 20;
                  }
                  else{
                        post Forward();
                        //sense = 4;
                        check = 1;
                  }

                  if(start >= 12){
                        start = 0;
                        thing = 1;
                  }
                  else{
                        start++;
                  }
            }
      }

      else{
            if(way == 0x000e){
                  if(begin == 0){
                        way = 0x000d;
                  }
                  else{
                        way = 0x000b;
                  }
            }
            else if(way == 0x000b){
                  way = 0x000e;
            }
            else{
                  begin = 1;
                  way = 0x000b;
            }
            sense = 4;
            wall = 1;
            thing = 0;
            start = 0;
      }


}



else{
      sense++;
      OCR3AH = 0x46;                //sets period to ~20ms
      OCR3AL = 0x50;
      OCR3CH = 0x2A;
      OCR3CL = 0x30;
      TCCR3A = 0x2B;
      TCCR3B = 0x5A;

      if(way == 0x000e && distmsg->north < 40){
```

```
                        x1 = distmsg->north;
                        temp = 10;
                        sense = 15;
                        start = 0;
                        if(box == 1){
                              sense = 0;
                              temp = 0;
                              post BoxTest();
                        }
                  }

                  else if(way == 0x000b && distmsg->south < 40){
                        x1 = distmsg->south;
                        temp = 10;
                        sense = 15;
                        start = 0;
                        if(box == 1){
                              sense = 0;
                              temp = 0;
                              post BoxTest();
                        }
                  }

                  else if(((way == 0x000d) || (way == 0x0007)) &&
((distmsg->south < 40) || (distmsg->north < 40))){
                        if(box == 1){
                              sense = 0;
                              temp = 0;
                              post BoxTest();
                        }
                        else{
                              distmsg->north = 100;
                              distmsg->south = 100;
                        }
                  }

                  else if(way == 0x000d && distmsg->east < 40){
                        temp = 10;
                        sense = 15;
                        start = 0;
                  }


                  else{

                        if(stopping == 1){
                              stopping = 0;
                              call Timer.stop();
                        }

                        //Robot turns right YELLOW ON
                        else if(state == 2){
                                    OCR3CH = 0x16;
                                    OCR3CL = 0x00;
                              post TurnRight();
                        }
                        //Robot turns left RED ON
                        else if(state == 3){
                                    OCR3CH = 0x16;
                                    OCR3CL = 0x00;
                              post TurnLeft();
                        }

                        //Robot Reverses ALL ON
                        else if(state == 4){
                              post Forward();
                        }
                  }

            }
}

task void Print_Data(){
            call Send.send(0, sizeof(IntMsgg), &data);
}

task void Addition(){
      avgY = (x1 + x2 + y1 + y2 + 0x19)/2;
      if (avgY < 260){
            box = 1;
      }
}

task void AtInterrupt(){
      int cmdistance;
      uint16_t useconds;
      atomic{
            if( first == 0 ){
                  TCNT3H = 0x00;
                  TCNT3L = 0x00;
                  first = 1;
                  TCCR3B = 0x02;                    // set clock with
8 prescaler, about 912.5kHz,
                                                    // set trigger on
falling edge
            }
            else{

                  ETIMSK = 0<<TICIE3;               //disable Timer3,
capture, and interrupt
                  useconds = widthcnt;
                  cmdistance = useconds/58;
                  first = 0;            // reset for next cycle

                  if(servohope == 1){
                        servo = hope;
                  }

                  if(way == 0x000e){
                        if(servo == 1){
                              distmsg->west = cmdistance;
                        }
                        else if(servo == 2 || servo == 0){
```

```
                    distmsg->north = cmdistance;
              }
              else if(servo == 3){
                    distmsg->east = cmdistance;
              }
        }

        else if(way == 0x000b){
              if(servo == 1){
                    distmsg->east = cmdistance;
              }
              else if(servo == 2 || servo == 0){
                    distmsg->south = cmdistance;
              }
              else if(servo == 3){
                    distmsg->west = cmdistance;
              }
        }

        else if(way == 0x000d){
              if(servo == 1){
                    //distmsg->north = cmdistance;
                    north = cmdistance;
              }
              else if(servo == 2 || servo == 0){
                    distmsg->east = cmdistance;
              }
              else if(servo == 3){
                    //distmsg->south = cmdistance;
                    south = cmdistance;
              }
        }

        else if(way == 0x0007){
              if(servo == 1){
                    south = cmdistance;
                    //distmsg->south = cmdistance;
              }
              else if(servo == 2 || servo == 0){
                    distmsg->west = cmdistance;
              }
              else if(servo == 3){
                    north = cmdistance;
                    //distmsg->north = cmdistance;
              }
        }

        if(stoprobot == 1){
              stopping = 1;
              stoprobot = 0;
        }

        if(check == 1){
              y1 = distmsg->east;
              x2 = distmsg->west;
              if(servo == 3){
                    check = 2;
```

```
                        }
                  }
                  else if(check == 2 && servo == 2){
                        if(way == 0x000b){
                              y2 = distmsg->south;
                        }
                        else if(way == 0x000e){
                              y2 = distmsg->north;
                        }
                        post Addition();
                        check = 0;
                  }
                        distmsg->avgY = box;
                        call Timer.start(TIMER_REPEAT, 300);
            }
            post Print_Data();
      }
}

TOSH_INTERRUPT(SIG_INPUT_CAPTURE3){
      widthcnt = ICR3;
      post AtInterrupt();
}

event result_t Send.sendDone(TOS_MsgPtr msg, result_t success){
      if (pending && msg == &data){
            pending = FALSE;
      }
      return SUCCESS;
}
}
```

Southwest Node Main Code

```
/* Authors:    Aly El-Osery
               Steven Myers
               Anthnoy Duran
*/

includes IntMsgg;

module RfmToIntM {
      provides interface StdControl;
      uses{
            interface StdControl as CommControl;
                  interface ReceiveMsg as ReceiveIntMsg;
            interface Leds;
            interface SendMsg as Send;
                  interface Timer;
      }
}

implementation{
      TOSH_ASSIGN_PIN(PW8, G, 0);
      TOSH_ASSIGN_PIN(PW9, G, 1);
      TOSH_ASSIGN_PIN(PW10, G, 2);
      uint16_t x1, x2, y1, y2, avgY;
```

```
int check;
int box;
int hookerbot;
int state;
int sense;
int thing;
int north;
int south;
int servo;
int bling;
int goingnorth;
int goingsouth;
int otherway;
int limit;
int reversing;
int temp;
int hope;
int servohope;
int begin;
int drunk;
int increase;
int wall;
int stoprobot;
int detect;
int firstime;
int verify;
int timing;
int stopping;
bool pending;
struct TOS_Msg data;
IntMsgg *distmsg = (IntMsgg *)data.data;
int first;  //# of times trigger function called
uint16_t widthcnt;      //65534 counter for trigger
int start;
uint16_t way;
uint16_t comp;
uint16_t nine;

command result_t StdControl.init() {
            x1 = x2 = y1 = y2 = 0;
            avgY = 0;
            box = 0;
            check = 0;
      state = 0;
      sense = 0;
            increase = 0;
            first = 0;
            servo = 0;
            widthcnt = 0;
            start = 0;
            thing = 0;
            drunk = 30;
            wall = 0;
            bling = 0;
            temp =  0;
            begin = 0;
            goingsouth = 0;
```

```
                  goingnorth = 0;
                  limit = 0;
                  detect = 0;
                  timing = 0;
                  hope = 0;
                  stoprobot = 0;
                  servohope = 0;
                  hookerbot = 0;
                  firstime = 0;
                  verify = 0;
                  north = 0;
                  south = 0;
                  reversing = 0;
                  stopping = 0;
                  way = 0x000b;
                  call CommControl.init();
            call Leds.init();
            return SUCCESS;
      }

      command result_t StdControl.start() {
            return call CommControl.start();
                  //return call Timer.start(TIMER_REPEAT, 300);
      }

      command result_t StdControl.stop() {
                  call Timer.stop();
            return call CommControl.stop();
      }

event TOS_MsgPtr ReceiveIntMsg.receive(TOS_MsgPtr m) {

    IntMsgg *message = (IntMsgg *)m->data;

      if(message->avgY == 0){
            call Timer.start(TIMER_REPEAT, 300);
      }
      else if(message->avgY == 2){
            box = 3;
            otherway = message->dir;
            goingnorth = message->north;
            goingsouth = message->south;
            call Timer.start(TIMER_REPEAT, 300);
      }

    return m;
}

task void Forward(){
    TOSH_CLR_PW0_PIN();
    TOSH_SET_PW2_PIN();
    TOSH_CLR_PW6_PIN();
    TOSH_SET_PW4_PIN();
}

task void TurnRight(){
      TOSH_CLR_PW0_PIN();
```

```
        TOSH_CLR_PW2_PIN();
        TOSH_SET_PW6_PIN();
        TOSH_CLR_PW4_PIN();
}

task void TurnLeft(){
        TOSH_SET_PW0_PIN();
        TOSH_CLR_PW2_PIN();
        TOSH_CLR_PW6_PIN();
        TOSH_CLR_PW4_PIN();
}

task void MotorStop(){
    TOSH_CLR_PW0_PIN();
    TOSH_CLR_PW2_PIN();
    TOSH_CLR_PW6_PIN();
    TOSH_CLR_PW4_PIN();
}

task void ReverseThatBiatch(){
        TOSH_SET_PW0_PIN();
    TOSH_CLR_PW2_PIN();
    TOSH_SET_PW6_PIN();
    TOSH_CLR_PW4_PIN();
}

task void Print_Robot(){
            call Send.send(1, sizeof(IntMsgg), &data);
}

task void Print_BaseStation(){
            call Send.send(0, sizeof(IntMsgg), &data);
}

task void BoxFound(){
        box = 2;
        distmsg->avgY = box;
        if(reversing <= 9){
            post ReverseThatBiatch();
            reversing++;
        }
        else{
            post MotorStop();
            distmsg->north = north;
            distmsg->south = south;
            post Print_Robot();
            post Print_BaseStation();
            call Timer.stop();
        }
}

task void SetupSonar(){

        int temp2 = 0;
        //Left Wheel
        OCR3CH = 0x00;
        OCR3CL = 0x00;
```

```
      TCCR3B = 0x42;                //set clock with 8 prescaler, about
912.5kHz,
      TCCR3A = 0x00;

      DDRE = 0x80;                  //Set PE7 as input
      DDRC = 0x00;                  //set PC7 as output

      atomic{
            TOSH_SET_PW1_PIN();      //sets pw6 high
            while(temp2 < 250){
                  temp2++;
            }
            temp2 = 0;
            TOSH_CLR_PW1_PIN();
      }
      SREG = 0x80;      //Global Interrupt enable (pg9)
      ETIMSK = 1<<TICIE3;                 //enable Timer3, capture, and
interrupt
      return;
}

task void BoxMove(){
      if(increase == 4){
            increase = 0;
            post MotorStop();
            call Timer.stop();
            post SetupSonar();
      }

      else{
            OCR3AH = 0x46;                //sets period to ~20ms
            OCR3AL = 0x50;
            OCR3CH = 0x2A;
            OCR3CL = 0x30;
            TCCR3A = 0x2B;
            TCCR3B = 0x5A;
            increase++;
            if(way == 0x000e  || way == 0x000c || way == 0x0006){
                  if(distmsg->north >= goingnorth){
                        post Forward();
                  }
                  else if(otherway == 0x000d || otherway == 0x000c ||
otherway == 0x0009){
                        if(limit == 0){
                              if(start <= 3){
                                    OCR3CH = 0x20;
                                    OCR3CL = 0x00;
                                    post TurnLeft();
                                    start++;
                              }
                              else{
                                    limit = 1;
                              }

                        }
                        else if(distmsg->north > 10){
                              post Forward();
                        }
                        else if(distmsg->north <= 10){
                              post MotorStop();
                        }
                  }

                  else if(otherway == 0x0007 || otherway == 0x0006 ||
otherway == 0x0003){
                        if(limit == 0){
                              if(start <= 3){
                                    OCR3CH = 0x20;
                                    OCR3CL = 0x00;
                                    post TurnRight();
                                    start++;
                              }
                              else{
                                    limit = 1;
                              }

                        }
                        else if(distmsg->north > 10){
                              post Forward();
                        }
                        else if(distmsg->north <= 10){
                              post MotorStop();
                        }
                  }
            }

            else if(way == 0x000b || way == 0x0009 || way == 0x0003){
                  if(distmsg->south >= goingsouth){
                        post Forward();
                  }
                  else if(otherway == 0x000d || otherway == 0x000c ||
otherway == 0x0009){
                        if(limit == 0){
                              if(start <= 3){
                                    OCR3CH = 0x20;
                                    OCR3CL = 0x00;
                                    post TurnRight();
                                    start++;
                              }
                              else{
                                    limit = 1;
                              }

                        }
                        else if(distmsg->south > 10){
                              post Forward();
                        }
                        else if(distmsg->south <= 10){
                              post MotorStop();
                        }
                  }

                  else if(otherway == 0x0007 || otherway == 0x0006 ||
otherway == 0x0003){
```

```
                        if(limit == 0){
                              if(start <= 3){
                                    OCR3CH = 0x20;
                                    OCR3CL = 0x00;
                                    post TurnLeft();
                                    start++;
                              }
                              else{
                                    limit = 1;
                              }

                        }
                        else if(distmsg->south > 10){
                              post Forward();
                        }
                        else if(distmsg->south <= 10){
                              post MotorStop();
                        }
                  }
            }
      }
}

task void BoxTest(){

      if(firstime == 0 || firstime == 2){
            post MotorStop();
            servohope = 1;
            if(timing < 7){
                  if(timing == 0){
                        OCR3AH = 0x46;                //sets period to
~20ms
                        OCR3AL = 0x50;
                        TCCR3B = 0x5A;                //set clock with
8 prescaler, about 912.5kHz,
                        TCCR3A = 0x2B;
                        hope++;
                  }
                  if(hope == 1 && timing == 0){
                        // Left
                        OCR3B = 0x07be;          //551us
                  }
                  else if(hope == 2 && timing == 0){
                  // Neutral
                        OCR3B = 0x04da;     //1.25ms
                  }
                  else if(hope == 3 && timing == 0){
                        // Right
                        OCR3B = 0x01fb;      //2.15ms
                  }
                  else if(hope == 4 && timing == 0){
                        OCR3B = 0x04da;
                  }
                  timing++;
            }
            else{
                  if(hope == 4){
```

```
            sense = 0;
            hope = 0;
            temp = 0;
            if(firstime == 0){
                firstime = 1;
            }
            else{
                firstime = 3;
            }
        }
        timing = 0;
        call Timer.stop();
        post SetupSonar();
    }
}

else if(firstime == 1){
    if(verify <= 3){
        if(way == 0x000e){
            hookerbot = way;
            detect = distmsg->north;
            if(distmsg->west > distmsg->east){
                way = 0x0007;
                post TurnLeft();
            }
            else{
                way = 0x000d;
                post TurnRight();
            }
        }
        else if(way == 0x000b){
            hookerbot = way;
            detect = distmsg->south;
            if(distmsg->west > distmsg->east){
                way = 0x0007;
                post TurnRight();
            }
            else{
                way = 0x000d;
                post TurnLeft();
            }
        }
        verify++;
    }
    else{
        post Forward();
        if(verify >= 12){
            verify = 0;
            firstime = 2;
        }
        else{
            verify++;
        }
    }
}

else if(firstime == 3){
```

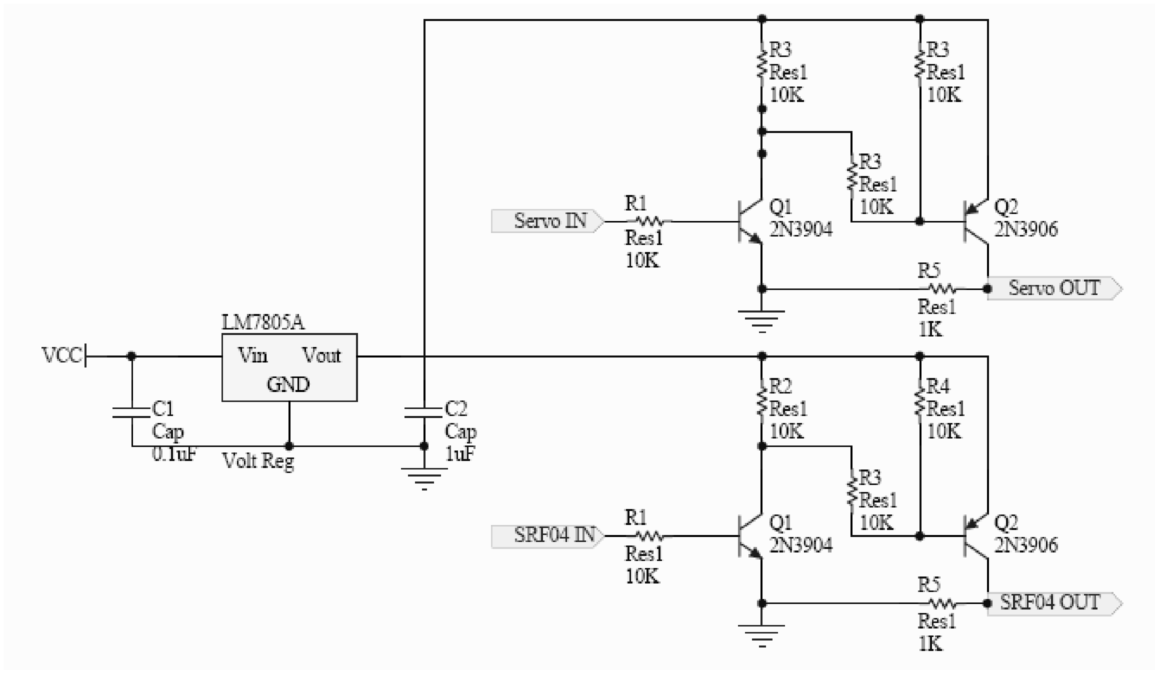
```
            if(hookerbot == 0x000e){
                  if(north >= (detect + 30)){
                        post BoxFound();
                  }
                  else{
                        box = 0;
                        sense = 0;
                        servohope = 0;
                        avgY = 0;
                        call Timer.start(TIMER_REPEAT, 300);
                  }
            }
            else if(hookerbot == 0x000b){
                  if(south >= (detect + 30)){
                        post BoxFound();
                  }
                  else{
                        box = 0;
                        sense = 0;
                        servohope = 0;
                        avgY = 0;
                        call Timer.start(TIMER_REPEAT, 300);
                  }
            }
      }
}

event result_t Timer.fired(){
            TOSH_MAKE_PW8_INPUT();
            TOSH_MAKE_PW9_INPUT();
            TOSH_MAKE_PW5_INPUT();
            TOSH_MAKE_PW7_INPUT();
            distmsg->val2 = 0x02;
      distmsg->dir = (0x0001 & (TOSH_READ_PW8_PIN() * 0x00FF)) |
(0x0002 & (TOSH_READ_PW9_PIN() * 0x00FF)) | (0x0004 &
(TOSH_READ_PW5_PIN() * 0x00FF)) | (0x0008 & (TOSH_READ_PW7_PIN() *
0x00FF));
            comp = distmsg->dir & way;
            nine = comp | way;
            TOSH_SET_PW3_PIN();
            TOSH_SET_PW10_PIN();
            TOSH_MAKE_INT0_OUTPUT();
            TOSH_MAKE_INT1_OUTPUT();

            if(begin == 0){
                  OCR3AH = 0x46;                //sets period to ~20ms
                  OCR3AL = 0x50;
                  OCR3B = 0x04da;     //1.25ms
                  TCCR3A = 0x2B;
                  TCCR3B = 0x5A;
            }

            if(drunk == 30){
                  distmsg->south = 100;
                  drunk = 0;
            }
            if(distmsg->dir == way){
                  if(wall == 1){
                        temp = 10;
                        sense = 4;
                        wall = 0;
                  }
                  else{
                        state = 4;
                  }
            }
      else if(comp == 0x000c){
                  state = 3;
            }
      else if(comp == 0x0003){
                  state = 3;
          }
          else if(comp == 0x0006){
                  state = 2;
      }
          else if(comp == 0x0009){
                  state = 2;
      }
          else if(nine == 0x000e){
                  state = 3;
            }
          else if(nine == 0x000b){
                  state = 2;
      }

            if(way == 0x0007){
                  if(distmsg->dir == way){
                        state = 4;
                  }
                  else if(comp == 0x0003){
                        state = 2;
                  }
                  else if(comp == 0x0006){
                        state = 3;
                  }
            }

            if(box == 3){
                  post BoxMove();
            }

            else if(box == 2){
                  post BoxFound();
            }

            else if(sense == 4){

                  post MotorStop();

                  if(temp == 10){
                        if(bling < 7){
                              if(bling == 0){
```

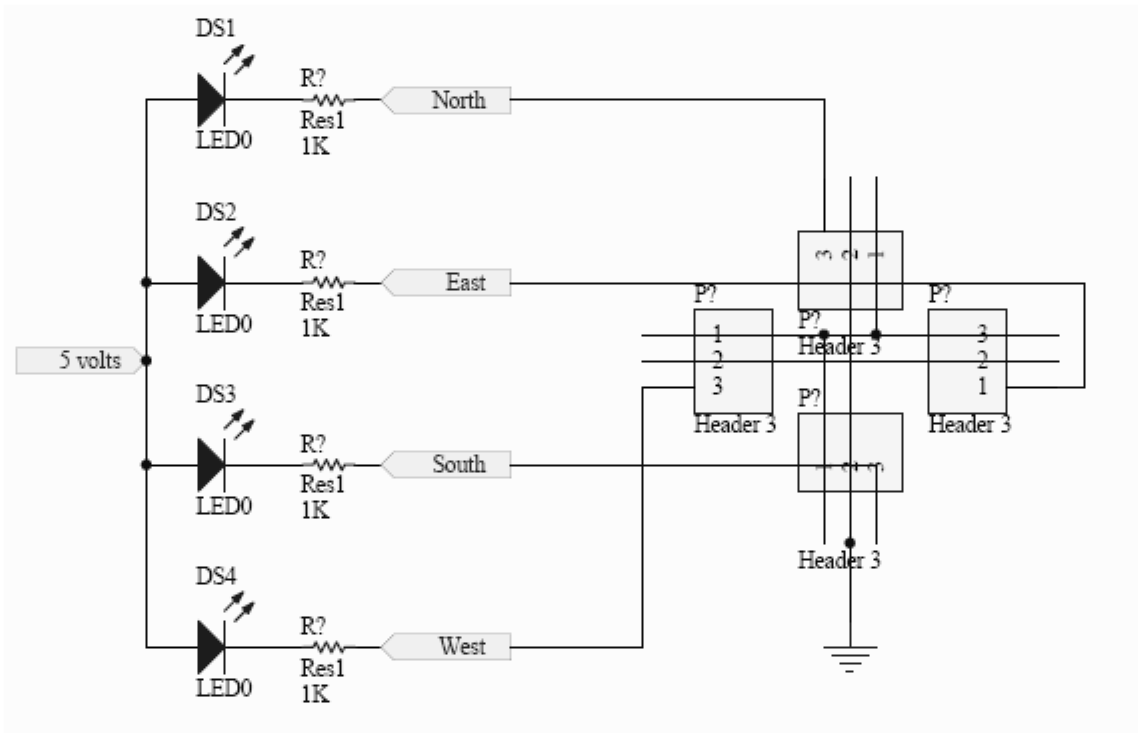
```
                                    OCR3AH = 0x46;
        //sets period to ~20ms
                                    OCR3AL = 0x50;
                                    TCCR3B = 0x5A;                  //set
clock with 8 prescaler, about 912.5kHz,
                                    TCCR3A = 0x2B;
                                    servo++;
                              }

                              if (servo == 1 && bling == 0){
                                    // Left
                                    OCR3B = 0x07be;           //551us
                              }
                              else if (servo == 2 && bling == 0){
                                    // Neutral
                                    OCR3B = 0x04da;      //1.25ms
                              }
                              else if (servo == 3 && bling == 0){
                                    // Right
                                    OCR3B = 0x01fb;       //2.15ms
                              }
                              else if (servo == 4 && bling == 0){
                                    OCR3B = 0x04da;
                              }
                              bling++;
                        }

                        else{
                              if(servo == 4){
                                    sense = 0;
                                    servo = 0;
                                    temp = 0;
                                    if(begin != 0 && distmsg->dir ==
way){
                                          stoprobot = 1;
                                    }
                              }

                              bling = 0;
                              call Timer.stop();
                              post SetupSonar();
                        }
                  }

                  else{
                        sense = 0;
                        call Timer.stop();
                        post SetupSonar();
                  }
            }

            else if(sense == 15){

                  OCR3AH = 0x46;                  //sets period to ~20ms
                  OCR3AL = 0x50;
                  OCR3CH = 0x20;
                  OCR3CL = 0x00;
```

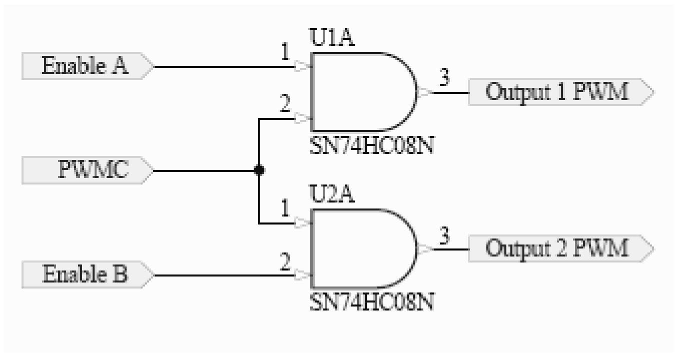
```
TCCR3A = 0x2B;
TCCR3B = 0x5A;

if(thing == 0){
      if(start <= 3){
            if(way == 0x000b){
                  if(begin == 0){
                        post TurnRight();
                  }
                  else{
                        post TurnLeft();
                  }
            }
            else{
                  post TurnRight();
            }
            start++;
      }
      else{
            if(begin == 0){
                  post MotorStop();
                  start = 20;
            }
            else{
                  post Forward();
                  //sense = 4;
                  check = 1;
            }

            if(start >= 12){
                  start = 0;
                  thing = 1;
            }
            else{
                  start++;
            }
      }
}

else{
      if(way == 0x000b){
            if(begin == 0){
                  way = 0x0007;
            }
            else{
                  way = 0x000e;
            }
      }
      else if(way == 0x000e){
            way = 0x000b;
      }
      else{
            begin = 1;
            way = 0x000e;
      }
      sense = 4;
      wall = 1;
```

```
                        thing = 0;
                        start = 0;
                }

        }



        else{
                sense++;
                OCR3AH = 0x46;                  //sets period to ~20ms
                OCR3AL = 0x50;
                OCR3CH = 0x2A;
                OCR3CL = 0x30;
                TCCR3A = 0x2B;
                TCCR3B = 0x5A;

                if(way == 0x000e && distmsg->north < 40){
                        x1 = distmsg->north;
                        temp = 10;
                        sense = 15;
                        start = 0;
                        if(box == 1){
                                sense = 0;
                                temp = 0;
                                post BoxTest();
                        }
                }

                else if(way == 0x000b && distmsg->south < 40){
                        x1 = distmsg->south;
                        temp = 10;
                        sense = 15;
                        start = 0;
                        if(box == 1){
                                sense = 0;
                                temp = 0;
                                post BoxTest();
                        }
                }

                else if(((way == 0x000d) || (way == 0x0007)) &&
((distmsg->south < 40) || (distmsg->north < 40))){
                        if(box == 1){
                                sense = 0;
                                temp = 0;
                                post BoxTest();
                        }
                        else{
                                distmsg->north = 100;
                                distmsg->south = 100;
                        }
                }

                else if(way == 0x0007 && distmsg->west < 40){
                        temp = 10;
                        sense = 15;
                        start = 0;
                  }


                  else{

                        //Robot turns right YELLOW ON
                        if(stopping == 1){
                              stopping = 0;
                              post Print_Robot();
                              call Timer.stop();
                        }

                        else if(state == 2){
                                    OCR3CH = 0x16;
                                    OCR3CL = 0x00;
                              post TurnRight();
                        }

                        //Robot turns left RED ON
                        else if(state == 3){
                                    OCR3CH = 0x16;
                                    OCR3CL = 0x00;
                              post TurnLeft();
                        }

                        //Robot Reverses ALL ON
                        else if(state == 4){
                              post Forward();
                        }
                  }

            }
}

task void Print_Data(){
            call Send.send(0, sizeof(IntMsgg), &data);
}

task void Addition(){
      avgY = (x1 + x2 + y1 + y2 + 0x19)/2;
      if (avgY < 260){
            box = 1;
      }
}

task void AtInterrupt(){
      int cmdistance;
      uint16_t useconds;
      atomic{
            if( first == 0 ){
                  TCNT3H = 0x00;
                  TCNT3L = 0x00;
                  first = 1;
                  TCCR3B = 0x02;                      // set clock with
8 prescaler, about 912.5kHz,
                                                        // set trigger on
falling edge
            }
            else{

                  ETIMSK = 0<<TICIE3;                 //disable Timer3,
capture, and interrupt
                  useconds = widthcnt;
                  cmdistance = useconds/58;
                  first = 0;              // reset for next cycle

                  if(servohope == 1){
                        servo = hope;
                  }

                  if(way == 0x000e){
                        if(servo == 1){
                              distmsg->west = cmdistance;
                        }
                        else if(servo == 2 || servo == 0){
                              distmsg->north = cmdistance;
                        }
                        else if(servo == 3){
                              distmsg->east = cmdistance;
                        }
                  }

                  else if(way == 0x000b){
                        if(servo == 1){
                              distmsg->east = cmdistance;
                        }
                        else if(servo == 2 || servo == 0){
                              distmsg->south = cmdistance;
                        }
                        else if(servo == 3){
                              distmsg->west = cmdistance;
                        }
                  }

                  else if(way == 0x000d){
                        if(servo == 1){
                              //distmsg->north = cmdistance;
                              north = cmdistance;
                        }
                        else if(servo == 2 || servo == 0){
                              distmsg->east = cmdistance;
                        }
                        else if(servo == 3){
                              //distmsg->south = cmdistance;
                              south = cmdistance;
                        }
                  }

                  else if(way == 0x0007){
                        if(servo == 1){
                              south = cmdistance;
                              //distmsg->south = cmdistance;
                        }
                        else if(servo == 2 || servo == 0){
                              distmsg->west = cmdistance;
                        }
                        else if(servo == 3){
                              north = cmdistance;
                              //distmsg->north = cmdistance;
                        }
                  }

                  if(stoprobot == 1){
                        stopping = 1;
                        stoprobot = 0;
                  }

                  if(check == 1){
                        y1 = distmsg->east;
                        x2 = distmsg->west;
                        if(servo == 3){
                              check = 2;
                        }
                  }
                  else if(check == 2 && servo == 2){
                        if(way == 0x000b){
                              y2 = distmsg->south;
                        }
                        else if(way == 0x000e){
                              y2 = distmsg->north;
                        }
                        post Addition();
                        check = 0;
                  }
                  distmsg->avgY = box;
                  call Timer.start(TIMER_REPEAT, 300);
            }
            post Print_Data();
      }
}

TOSH_INTERRUPT(SIG_INPUT_CAPTURE3){
      widthcnt = ICR3;
      post AtInterrupt();
}

event result_t Send.sendDone(TOS_MsgPtr msg, result_t success){
      if (pending && msg == &data){
            pending = FALSE;
      }
      return SUCCESS;
}
}
```

Appendix C
Schematics






U1A
Enable A
1
2
3
Output 1 PWM
SN74HC08N
PWMC
U2A
1
2
3
Output 2 PWM
Enable B
SN74HC08N