RIT Computer Engineering Senior Design Projects II
EECC 657
Final Report

EX-Bot
Map it once and use it forever.

Submitted By: Timothy C. LaForest
TCLaforest@aol.com

____

Eric Shields
ers3268@rit.edu

____

Merve Evran
merveevran@gmail.com

____

Submitted To: Dr. Roy Czernikowski
Date Submitted: February 16, 2006

Table of Contents

Index of Figures

Index of Tables

Introduction

In the proposed system, a robot roams an enclosed area with random objects and operates according to orders from the user interface. The robot gathers sensor information which will be used to construct a map in the user interface. If allowed to, the robot will do this autonomously. If the robot gets stuck in a loop or the user simply wishes to speed the robot's progress to a certain area, the user can click on a virtual map to designate a new place to start mapping from. The map constructed from this information is displayed to the user on the computer running the User Interface. The virtual map is updated as the robot's mapping progresses and is saved until the system is reset or exited. This interface also allows the user to convert the robot to a controlled mode, in which the user can navigate the robot through the mapped area.

Proposed Overall System

The objective of the proposed project is to create a robot that will be capable, with the assistance of a PC, of mapping an area, approximately 7.5ft by 7.5ft (2.3m by 2.3m), then navigating the mapped area. The system will have a java-based user interface with a display of the mapped area and some controls to operate the robot. The computer and the robot communicate via 900MHz RF modules. The robot makes use of ultrasonic range finders, a magnetic compass, and an infrared phototransistor for mapping and navigation. The robot will use the I^{2}C standard for communication with the sensors (except for the phototransistor) and will communicate with the RF module via standard serial communication.

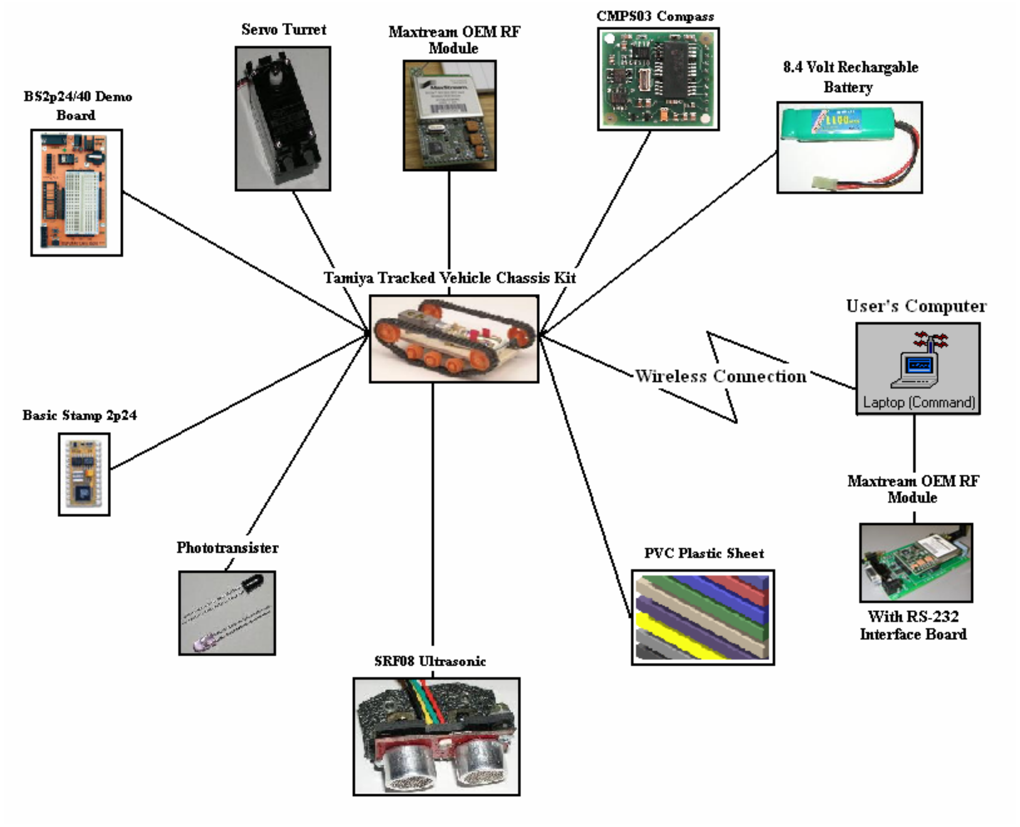


Figure 1: System Overview Component Diagram

The complete system consists of three components: The robot, a laptop serving as the user interface (also known as Control), and the communications between them. A component diagram of the system is shown above in figure 1.

The overall system design is as follows: Both components have an RF wireless module. Using standard RS-232 and UART protocols, a series of signals will be transmitted, in the form of specifically assigned segments of the payload of each packet, to and from the robot.

Some of the operating environment will be created specifically for the demo. This will include 4 outer walls, to eliminate the issues related to outer boundaries, and objects of various shapes and sizes to act as obstacles. The only restriction on the placement of objects is that they cannot be moved without restarting the user interface.

The next three sections detail each of the major pieces of the system, which are stated above.

Robot

The robot is constructed of a base and three tiers. The base is a Robot Tank Base Kit that holds four components: The Tamiya Rubber Tank Tread Kit, High-power Tamiya Twin Motor gearbox kit, Infrared LED and the first of 2 8.4V 1100mAh Batteries (Used to power the H-Bridge used to drive the motors). There are three tiers, constructed of colored PVC plastic sheets, one for the H-Bridge, Compass, and Infrared Phototransistor, one for the BS2p24 processor, RF and development board, and one for the Ultrasonic Ranger turrets and the second 8.4V 1100mAh battery (Used to power the rest of the system). On the second tier sits the BS2p24/40 Demo Board with BS2p24 processor from Parallax, Inc. This board, and processor, was chosen for the fact that it has an embedded I^2C controller and Parallax, Inc provides PBasic, an easy to use programming language.

All interconnections between the devices are done on the development section of the board. The MaxStream 900MHz, 9600 baud, RF transceiver is the only module that is physically located on the development board. The compass and ultrasonic range finders are connected to the I^2C bus. The compass determines the direction that the robot is facing. The phototransistor allows the user's computer to determine the distance travelled by sending a signal to the BS2 everytime a tread hole passes. The RF module will allow communication with the user's computer.

Finally, the robot has 2 SFR08 High Performance Ultrasonic Range Finders on the top tier. Each one sits atop a "turret," a servo motor, which rotates so as to take measurements at different angles. These "turrets" are placed such that they are exactly 5 inches apart at opposite ends of the robot and are in the center of the robot's width. The two sensors will always be facing in opposite directions when taking measurements.

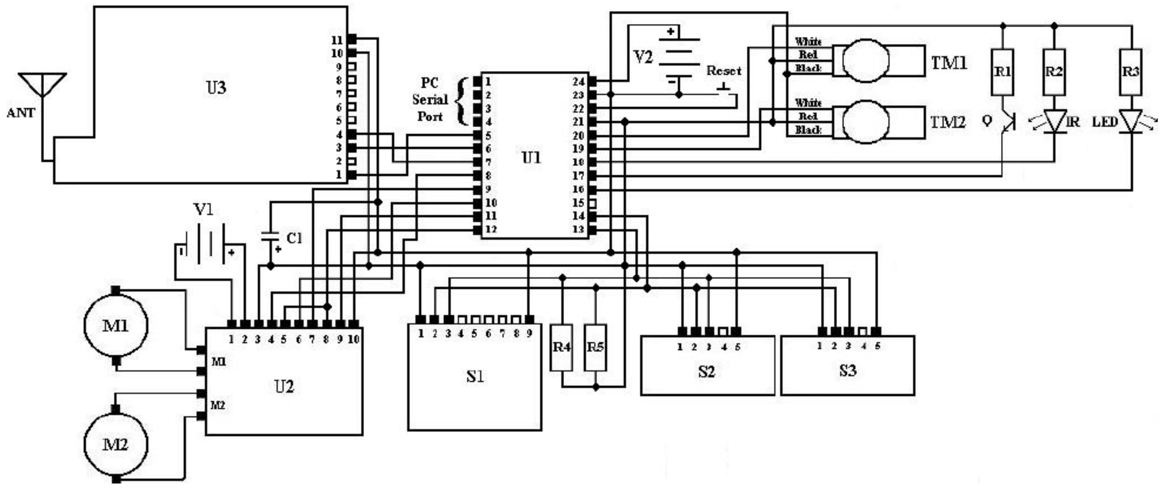


Figure 2: Robot Schematic

The details of robot system are contained in the schematic for the robot, Figure 2, and the component list, Figure 3.

ANT: 22-guage-wire antenna (3" length for 900 MHz modules)

C1: 5V 1.0F Polarized Capacitor

IR: QED223 Plastic Infrared LED 880nm (on Right Tread)

LED: Test Indicator LED (Red)

M1: FA-130 DC Motor (Connected to Left Tread)
M2: FA-130 DC Motor (Connected to Right Tread)

Q: QSD124 Plastic Silicon Infrared Phototransistor 880nm (on Right Tread)

R1: 10KΩ Resistor
R2: 470Ω Resistor
R3: 470Ω Resistor
R4: 4.7KΩ Resistor
R5: 4.7KΩ Resistor

S1: Devantech CMPS03 Magnetic Compass Module (Centered on Robot)
S2: Devantech SRF08 Ultrasonic Range Finder (Mounted Facing Front)
S3: Devantech SRF08 Ultrasonic Range Finder (Mounted Facing Rear)

TM1: Turret Motor – GWServo S03N STD (Mounted in Front)
TM2: Turret Motor – GWServo S03N STD (Mounted in Rear)

U1: Basic Stamp Microprocessor (BS2p24)
U2: Low Cost Dual 1.1A H-Bridge Module
U3: XCite OEM RF 900MHz Module

V1: 8.4V 1100mAh Rechargeable Battery
V2: 8.4V 1100mAh Rechargeable Battery

Figure 3: Robot Component List

Wireless Communications

Using the wireless RF modules, the robot and the Control use a set communication standard to pass information about the status and next state to perform. The standard packet is shown below in Figure 4.

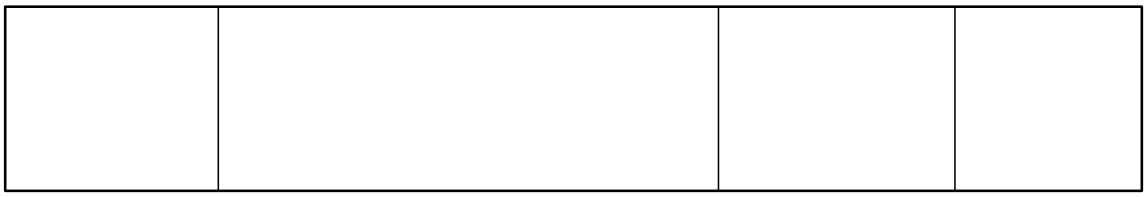


Figure 4: Communication Packet Setup

In most cases, this packet is sent from only once, with the Ultrasonic Measurements zeroed out. However, in the case of mapping mode, this packet will be sent 5 times in order to send all of the sensor readings. This is due to memory constraints of the BS2. A 6th packet is then be sent as part of the normal communications.

The state bits represent what the robot is currently doing or where the user interface wants the robot to be at next, for Robot-to-Control and Control-to-Robot communications, respectively. The state diagram is shown on the next page in figure 5.

State
(8 bits)

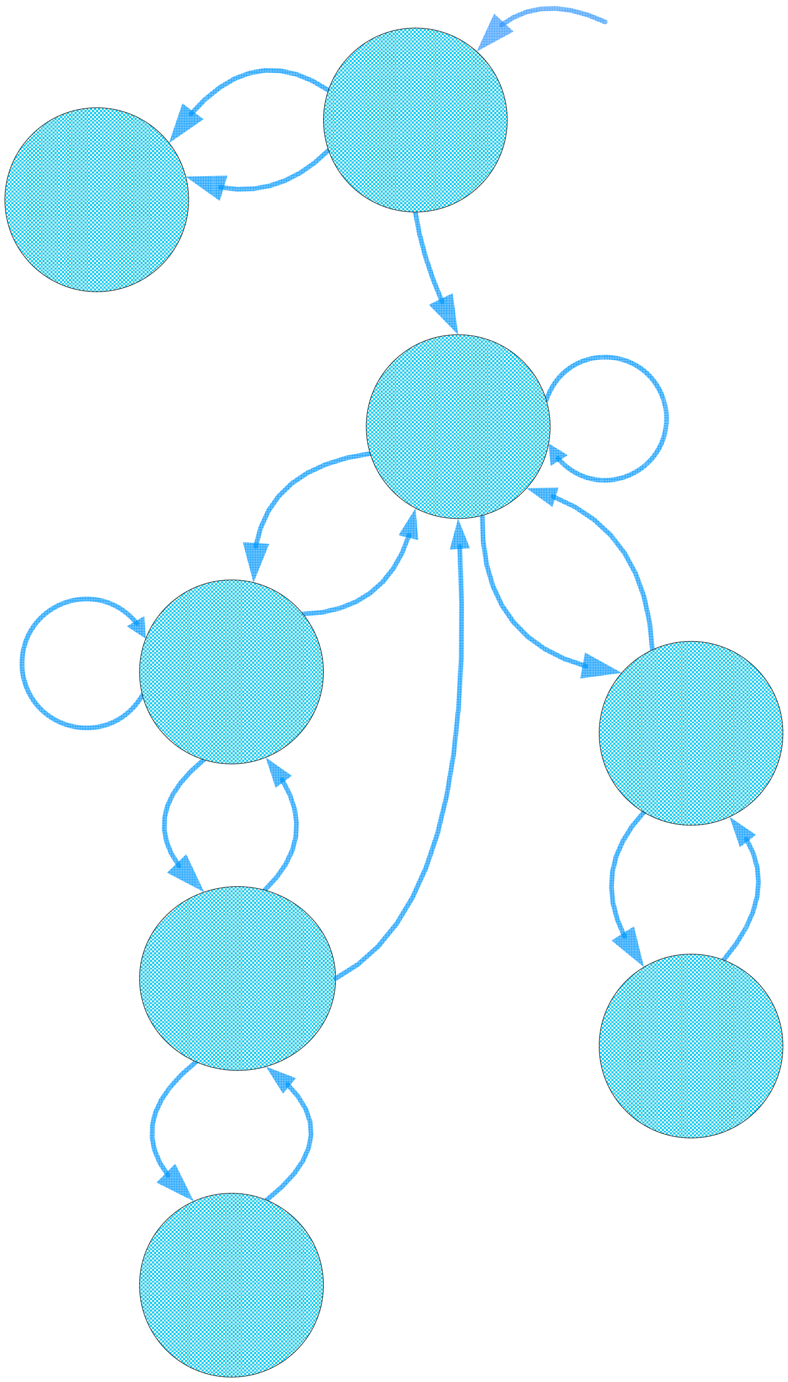


Figure 5: Robot State Diagram

The compass value is an integer from 0 to 3599, representing 0 to 359.9 degrees. This value contains a non-zero value when giving the robot directions in controlled or user-mapping mode or when the robot is communicating its heading to the control.

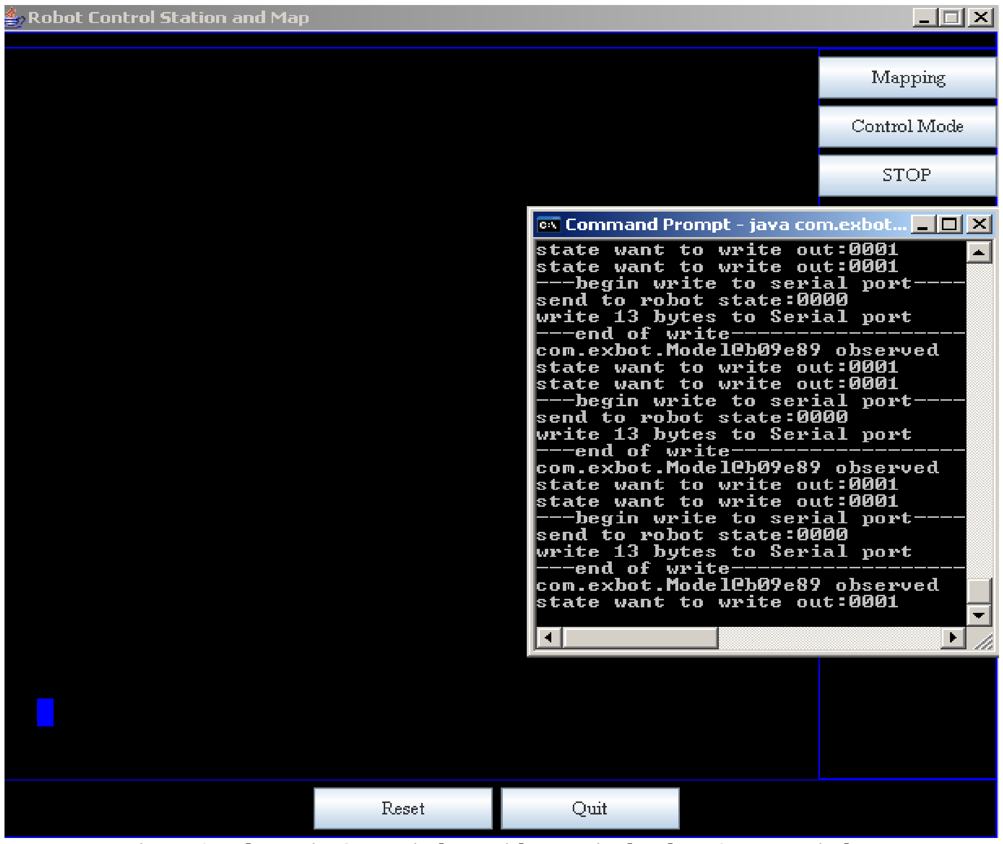
This distance value contains one of three things: Most commonly it contains a number of tread holes that the robot needs to travel. It may also contain all 0's if the data isn't relevant. Finally it may contain an error code. If the value is all ones, then the robot received an invalid state and if the value contains alternating 1's and 0's, then the robot is stuck between two objects and cannot move without the help of the user.

The user's computer runs a Java program and uses an RS-232 interface, so no restrictions are anticipated on what computer it could be set up on, once the Javax.comm package is installed. Each robot has a single RF module operating in the 900MHz frequency range. This range is satisfactory for use within the US, but is subject to restrictions in other countries.

All of the communication algorithms and code have been implemented on the robot. However, the system testing has not yet been preformed to determine whether the robot communicates successfully with the Control at the right times.

User Interface/Control

Once the files are compiled, the User Interface program is run using by typing "java com.exbot.Project" at the command line. The first window displayed is the main window, which displays the robot in the lower left corner of the room. The large black area is the area in which the map will appear, the buttons on the right communicate with the robot, as well as displaying more options to choose from, and the buttons at the bottom reset and exit the program. The initial window is shown in the following figure. Each pixel represents one 0.5 inch box in the demo environment.


Robot Control Station and Map
Mapping
Control Mode
STOP
Command Prompt - java com.exbot...
Reset
Quit

```
state want to write out:0001
state want to write out:0001
---begin write to serial port----
send to robot state:0000
write 13 bytes to Serial port
---end of write-------------------
com.exbot.Model@b09e89 observed
state want to write out:0001
state want to write out:0001
---begin write to serial port----
send to robot state:0000
write 13 bytes to Serial port
---end of write-------------------
com.exbot.Model@b09e89 observed
state want to write out:0001
state want to write out:0001
---begin write to serial port----
send to robot state:0000
write 13 bytes to Serial port
---end of write-------------------
com.exbot.Model@b09e89 observed
state want to write out:0001
```

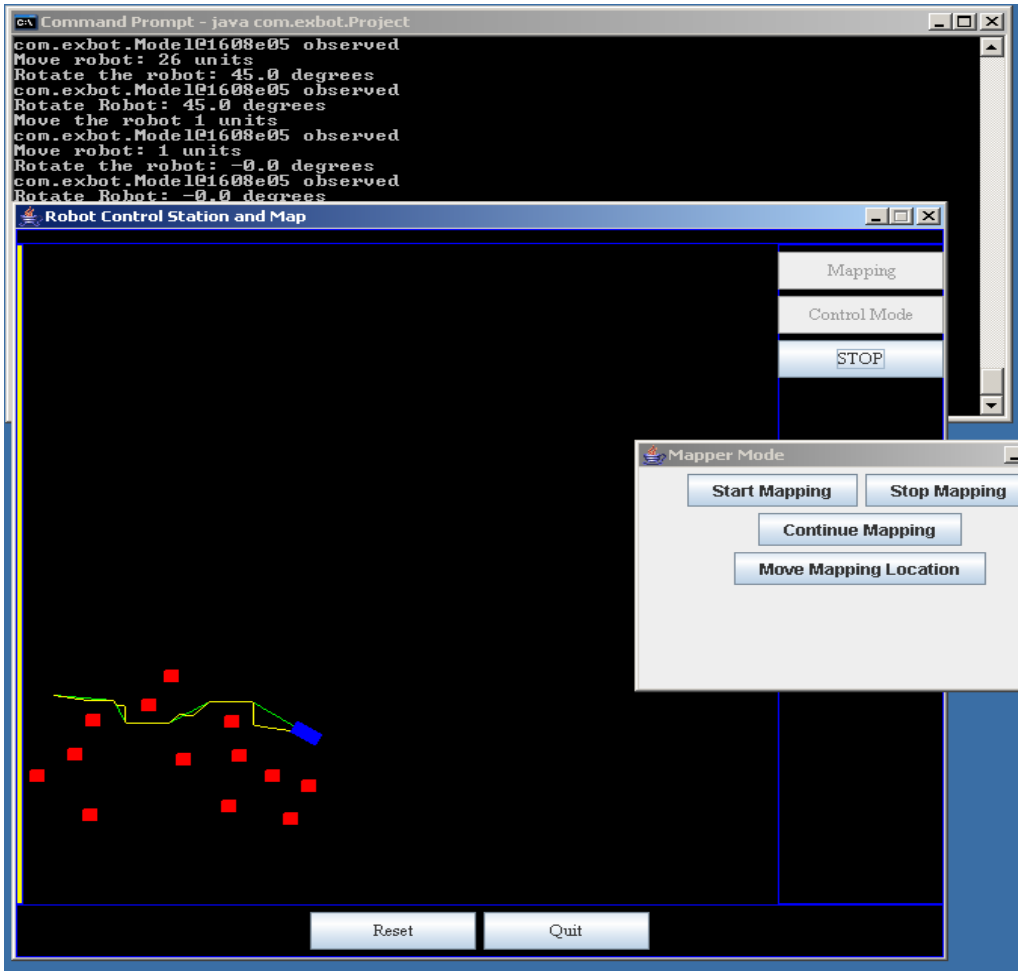
Figure 6: The Main GUI Window, with Terminal Debug Output Window

Since mapping must to be done before the robot is allowed to move, the “Control Mode” button is initially disabled. Clicking on the “Mapping” button opens a new window called “Mapper Mode.” This window contains 4 buttons: “Start Mapping,” “Stop Mapping,” “Continue Mapping,” and “Move Mapping Location.” Clicking on the “Start Mapping” button triggers a packet to be sent to the robot telling it to start Semi-Autonomous Mapping. This process is described in the “User Interface Algorithms” subsection of the “System Functionality and Algorithms” section. Clicking the “Continue Mapping” button lets the robot continue mapping from its current location (usually used after the “Move mapping Location” command). Clicking the “Move Mapping Location” button and then clicking on the map tells the robot to move to that location and wait for a “Continue Mapping” command.

The User Interface now calculates the shortest path to the destination selected by the user. Step by step, it sends the robot packets telling the robot each straight leg of the journey to the final destination. The data in the packets tells the robot to turn to a specified angular direction and the distance to travel till the next point when the robot is given a new direction and distance until it reaches the final destination. Also, the GUI displays the shortest path that was calculated.

Currently, two colors, green and yellow, are shown on the screen that represent the path to be taken by the robot. The yellow is the simplest method of creating the path, which uses straight lines and following of contours. The yellow line is used for testing purposes. A buffer was added so that the line could take into account the width of the robot when calculating paths. The green line represents the most direct route from the current location to the destination with. This algorithm is calculated using the yellow line. The directional commands sent to the robot are derived from the green line. Note that the Red squares in the GUI represent obstacles. Clicking in the map space when the GUI is not in a state to send move commands creates these test obstacles. Further detail about this algorithm can be found in the System Functionality and Algorithms section. As this movement occurs, the user is also given the ability to set the robots final orientation, in degrees, with 0 referring to north.

If the user clicks “STOP” then a packet telling the robot to transition back to idle state is sent at the next point when the robot is ready to receive a packet. Refer to the “User Interface Algorithms” subsection of the “System Functionality and Algorithms” section for more details about what packets contain and when they are sent. Please refer to Figure 7 for a screen shot of the above mentioned information.



```
com.exbot.Model@1608e05 observed
Move robot: 26 units
Rotate the robot: 45.0 degrees
com.exbot.Model@1608e05 observed
Rotate Robot: 45.0 degrees
Move the robot 1 units
com.exbot.Model@1608e05 observed
Move robot: 1 units
Rotate the robot: -0.0 degrees
com.exbot.Model@1608e05 observed
Rotate Robot: -0.0 degrees
```

Figure 7: Obstacles, Mapper Mode Window, and Shortest Path Example

From the Main Window, the "Control Mode" button brings up the same window as the "Mapper Mode -> Move Mapper Location" selection, except that the "Start Mapping" button is not available. This mode otherwise acts the same as "Mapper Mode -> Move Mapper Location."

Pressing the "Reset" button clears all obstacles and the path, if present, from the map, sends a "STOP" packet to the robot, tells the robot to wait more commands.

Pressing the "Quit" button exits the program. No information is saved upon quitting. This also sends a "STOP" packet to the robot.

Testing Strategy

Wireless Module Testing

First, one of the modules was connected to a PC and the X-CTU software installed on that PC. The second module was set up using a loop-back module that would immediately send the same data that it received back over the air. A test program in the X-CTU software was run and the modules functioned. The modules were switched and the test preformed again with similar success.

Next the PC module was sent data from the Control program, looped back, and read back again from the Control program. This ensured that the Control code was capable of sending/receiving information to/from the module.

Next the same was done for the robot. The code would send data to a loop-back module and read it back in.

Finally one module was connected to the PC and one to the robot. The robot would then attempt to send data to the PC and the PC would attempt to read it in. The PC would then send data to the robot and the robot would attempt to read it in.

All of these tests passed successfully.

Robot Testing

The initial testing for the robot involved testing each component's functionality. The first test run involved the motor drive system. The DC motors, controlled by the H-Bridge, were run through a series of commands testing Forward, Reverse, Left Turn, Right Turn and Stop. The motor system passed these tests.

Next the I^2C line and all its connected components were tested not only for communication but also for functionality. This involved the Compass and both Ultrasonic devices. First the direction was grabbed from the compass then the Ultrasonic units were commanded to take a reading. The next step was to reverse the direction of the compass and both ultrasonic devices and repeat the first test. These readings were compared and contrasted to verify the functionality of them and the I^2C. All components passed these tests individually. A combined test was not preformed.

The turrets were tested to measure their max angle, min angle and center angle by verifying the length of the pulse used to drive them. This also accomplished the testing of the turret servos functionality. The servos passed this test, but were shown to only have an arc of approximately 90°.

All LEDs were verified for operation and the phototransistor was verified to read from the IR LED.

User Interface Testing

Test classes were created and run to test various aspects of the GUI as its construction progressed. Once each component class was tested successfully, they were merged and the overall system was tested. Due to the methods Java uses, the project was made using several separate files and including them all in a single Package, which was called "com.exbot.Project." Each button and type of robot movement were tested systematically.

To help testing of the path-finding algorithm, a mouse-click based object placement system was created. This allowed the tester to place objects of decent size by simply clicking on the map. Also, a yellow line was used as a basis of the path-finding algorithm.

First attempting to send and receive from a virtual serial port on a laptop that did not have a physical one tested the serial communication with the wireless module. The packet bytes sent and received were stored in files and were examined after the project was run. Once the wireless modules were acquired, the testing proceeded to sending packets to the modules and having the modules loop them back. Tests were completed successfully with the modules and the GUI.

Equipment/Software

The required components for this project to be replicated are listed in the following table.

Manufacturing Costs			
Component	*Cost Per*	*Quantity*	*Total*
Battery	$ 8.99	2	$ 17.98
Battery Recharger	$ 23.99	1	$ 23.99
Compass	$ 49.00	1	$ 49.00
H-Bridge	$ 19.99	1	$ 19.99
IR Detector	$ 0.99	1	$ 0.99
IR Emitter	$ 0.99	1	$ 0.99
Microcontroller and Development Board	$ 199.95	1	$ 199.95
Miscellaneous Electical	$ 80.37	-	$ 80.37
Miscellaneous Mechanical	$ 20.63	-	$ 20.63
Red LED	$ 1.00	1	$ 1.00
RF Devolopment Kit	$ 149.00	1	$ 149.00
Robot Base	$ 39.95	1	$ 39.95
Servo Turrets	$ 18.95	2	$ 37.90
Ultrsonic Sensors	$ 56.99	2	$ 113.98
		Total Manufacturing Cost	$ 755.72

Table 1: Component Listing to Replicate Project

The next table contains the items that were actually purchased in the process of completing this project.

Actual Project Cost				
Component	*Cost Per*	*Quantity*	*Shipping*	*Total*
5V to 3.3V Regulator	$ -	1	$ 9.00	$ 9.00
AC Adapter for Development Board	$ 6.00	1	$ -	$ 6.00
Antenna	$ -	1	$ -	$ -
Battery	$ 8.99	2	$ 7.95	$ 25.93
Battery Recharger	$ 23.99	1	$ 11.90	$ 35.89
Bluetooth Dongle	$ 12.49	1	$ -	$ 12.49
Bluetooth Module	$ 1.98	8	$ -	$ 15.84
Compass	$ 49.00	1	$ -	$ 49.00
Dual, Bidirectional, Voltage Level Translato	$ -	1	$ -	$ -
H-Bridge	$ 19.99	1	$ 30.14	$ 50.13
IR Detector	$ 0.99	1	$ -	$ 0.99
IR Emitter	$ 0.99	1	$ -	$ 0.99
Microcontroller and Development Board	$ 60.00	3	$ 24.00	$ 204.00
Miscellaneous Electical	$ 80.37	-	$ 68.28	$ 148.65
Miscellaneous Mechanical	$ 25.63	-	$ 14.05	$ 39.68
Module Adapter	$ 68.40	1	$ -	$ 68.40
Red LED	$ 1.00	1	$ -	$ 1.00
RF Devolopment Kit	$ 149.00	1	$ 49.00	$ 198.00
Robot Base	$ 39.95	1	$ -	$ 39.95
Servo Turrets	$ 18.95	2	$ -	$ 37.90
SOT-23-8 to DIP Adapter	$ 3.19	1	$ 26.00	$ 29.19
Ultrsonic Sensors	$ 56.99	2	$ -	$ 113.98
Demo Materials	$ 45.00	-	-	$ 45.00
			Total Project Cost	$ 1,132.01

Table 2: Actual Components Bought Listing

All cost values are subject to change at the will of the companies. All components are subject to upgrading, being made obsolete, etc. at the will of the companies. The listed project cost is not necessarily accurate due to fluctuating currency conversion rates, shipping costs, sales, and bulk purchasing.

The robot will be powered by two 8.4V, 1100mAh rechargeable batteries.

The software needed for this project includes the Java Runtime Environment (JRE) as well as a javax.comm communication package from Sun Microsystems, Parallax free Pbasic compiler, and X-CTU software provided by MaxStream .

System Functionality and Algorithms

Robot Algorithms

Any time the robot is turned on or reset, it performs a BIST (Built-In Self-Test). The implementation for this has been coded, but not tested due to the construction of the robot not being complete. The following figure shows the flowchart for the BIST.

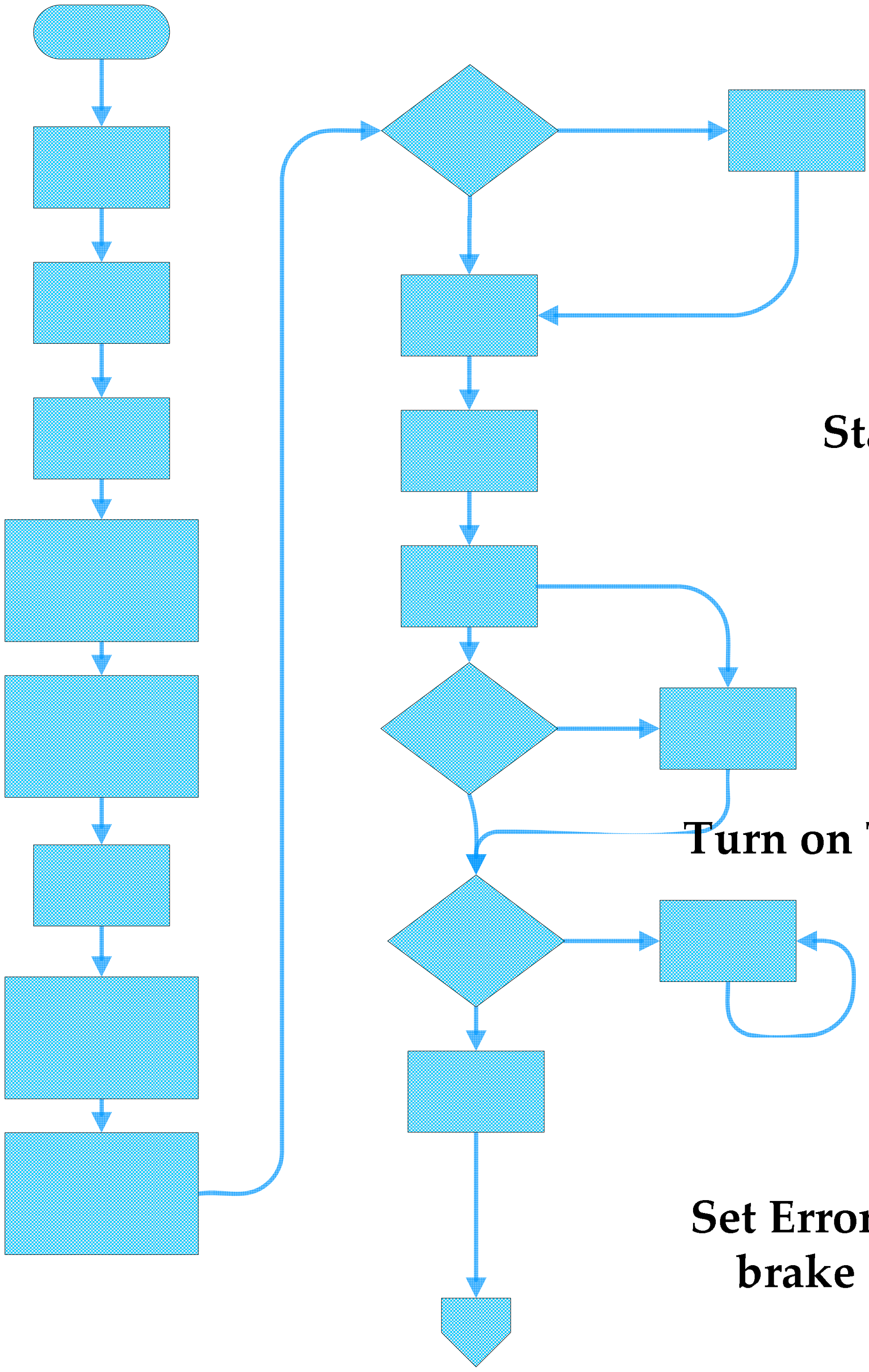


Figure 8: Robot BIST

At the completion of the test, the robot sends a packet to the Control acknowledging that it is in STOP_WAIT and ready for commands. This is the heart of the Main algorithm for the robot. This algorithm is shown below.

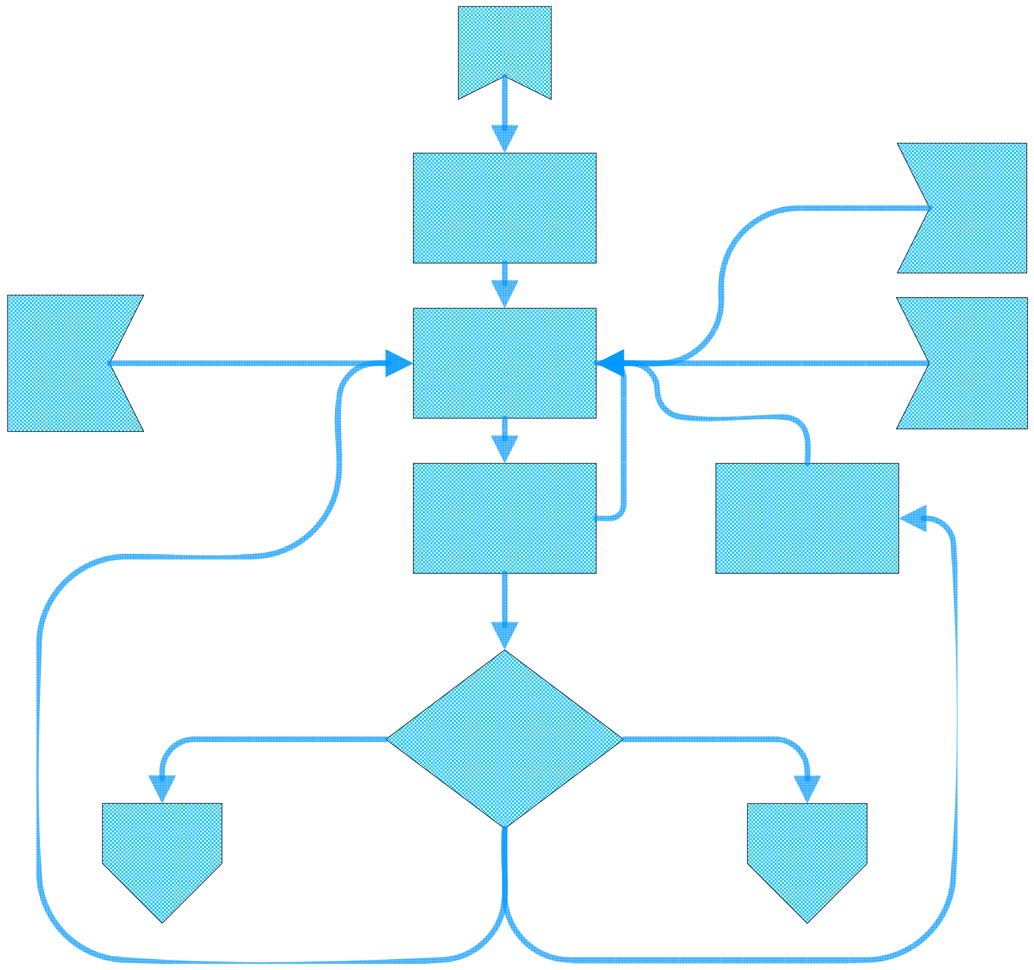


Figure 9: Robot Main Algorithm

This is the heart of the robot because it is where the robot transitions to if it encounters a stop command. From here the robot can be sent to either auto mapping mode or user controlled mode.

If the command is sent to enter Controlled mode, then the robot waits for the user to select a destination. The user will simply click on the map where they wish the robot to go, the Control then calculates the shortest path to that point and sends the step by step instructions to the robot. If the stop command is received, the robot transitions back to Main algorithm. The controlled algorithm is shown below.

From Controlled

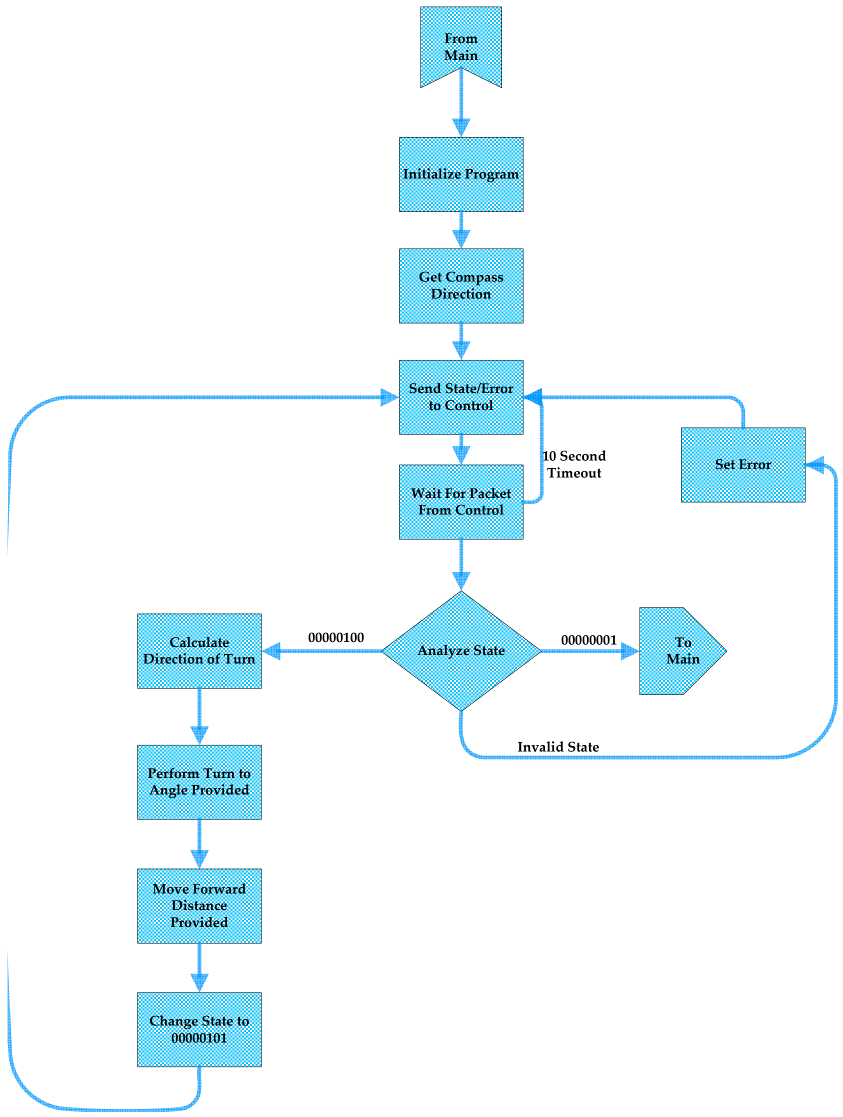


Figure 10: Robot Controlled Algorithm

If the user instead selects auto mapping mode, then the robot takes a series of ultrasonic readings then moves forward or backwards by 5 inches, depending on whether it detects an object too close. If it detects an object too close in both directions, it enters an error state and informs the Control. The Auto Mapping algorithm is on the next page. Each time a turret is moved, it is moved by 10 degrees.

Each of these algorithms is contained within a separate program space in the BS2. This way the relatively small amount of EEPROM and RAM available are still enough to contain what needs to be done.

Also, the communications from the Control will always contain the state that is the desired next step. The robot's communication to the control will always contain the current state the robot is in. If entering controlled mode or user mapping mode, the instructions also contain a distance and compass value (see figure 4 above).

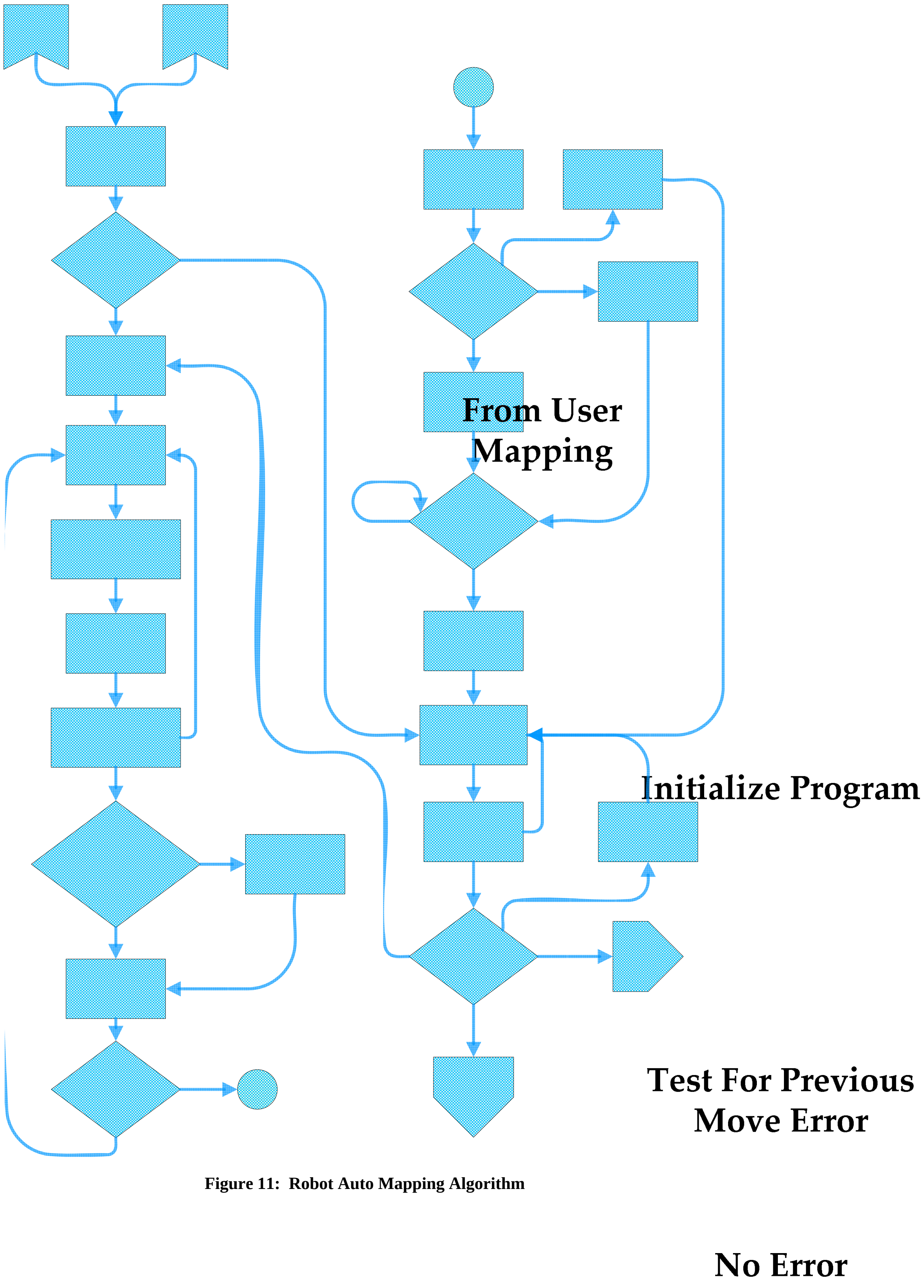


Figure 11: Robot Auto Mapping Algorithm

Now the user has the option to send the robot into User Mapping mode. This mode is very similar to Controlled mode, except that the robot immediately enters Auto Mapping mode upon reaching its destination and receiving confirmation from the Control. The User Mapping algorithm is shown below.

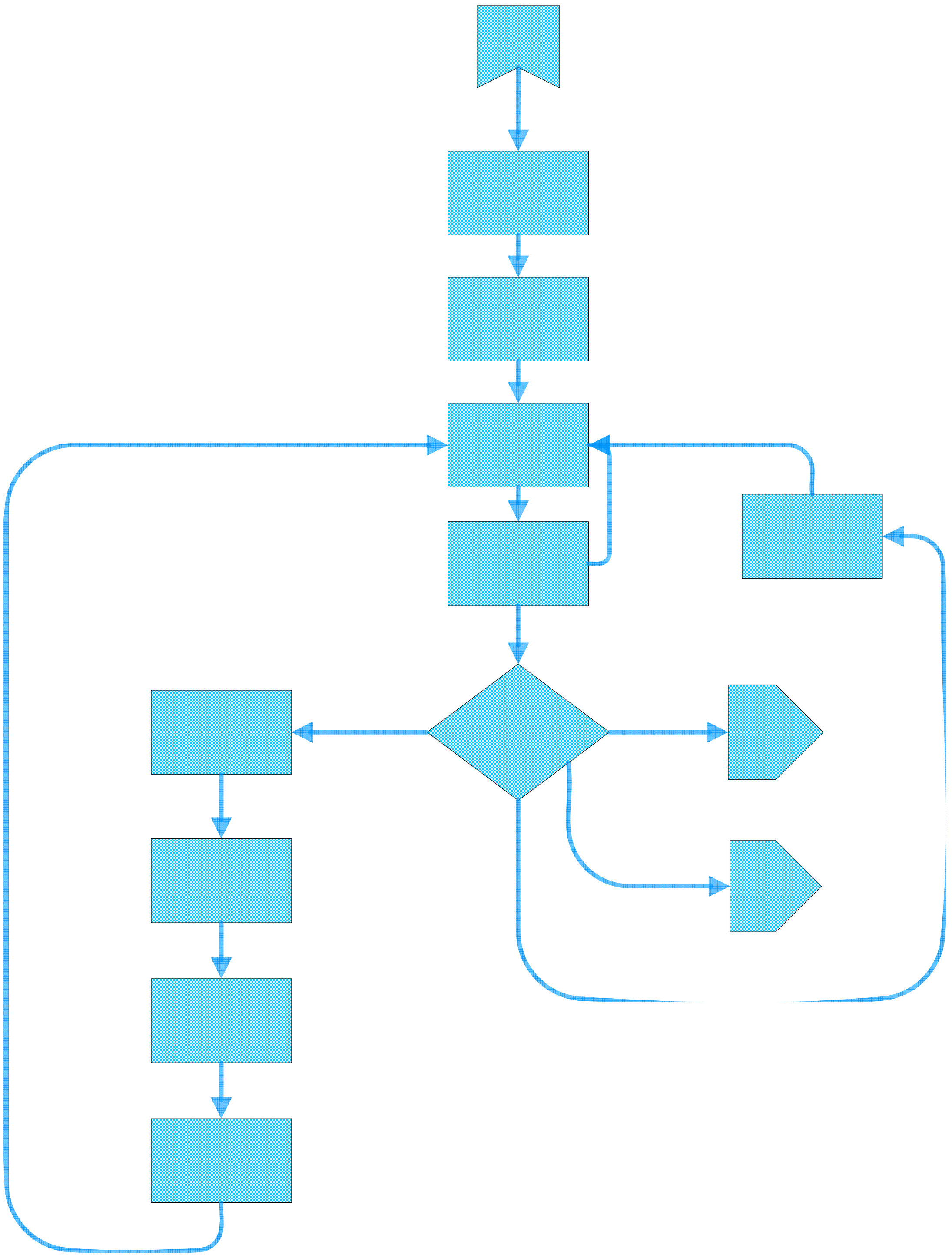


Figure 12: Robot User Mapping Algorithm

Unfortunately, as with the BIST, these algorithms were only derived and coded. No testing could be done due to the fact that the robot was not fully built and the Control's ability to communicate was limited.

Once the data is obtained from the ultrasonic sensors in the Auto Mapping mode, it is sent to the Control to be processed. In order to process it, another algorithm needed to be created. For this, a chart of theta values (representing the angle range that the measurement was taken in) for a given distance value. If the distance is not directly represented in the chart, then the angle theta would be linearly interpolated from the next highest and next lowest distance on the chart. This algorithm, called the bubble algorithm, is shown below. There are 9 bubbles for each turret, one every 10°. There are a varying number of lines per bubble, separated by 1°; the number of lines is equal to the theta for that bubble.

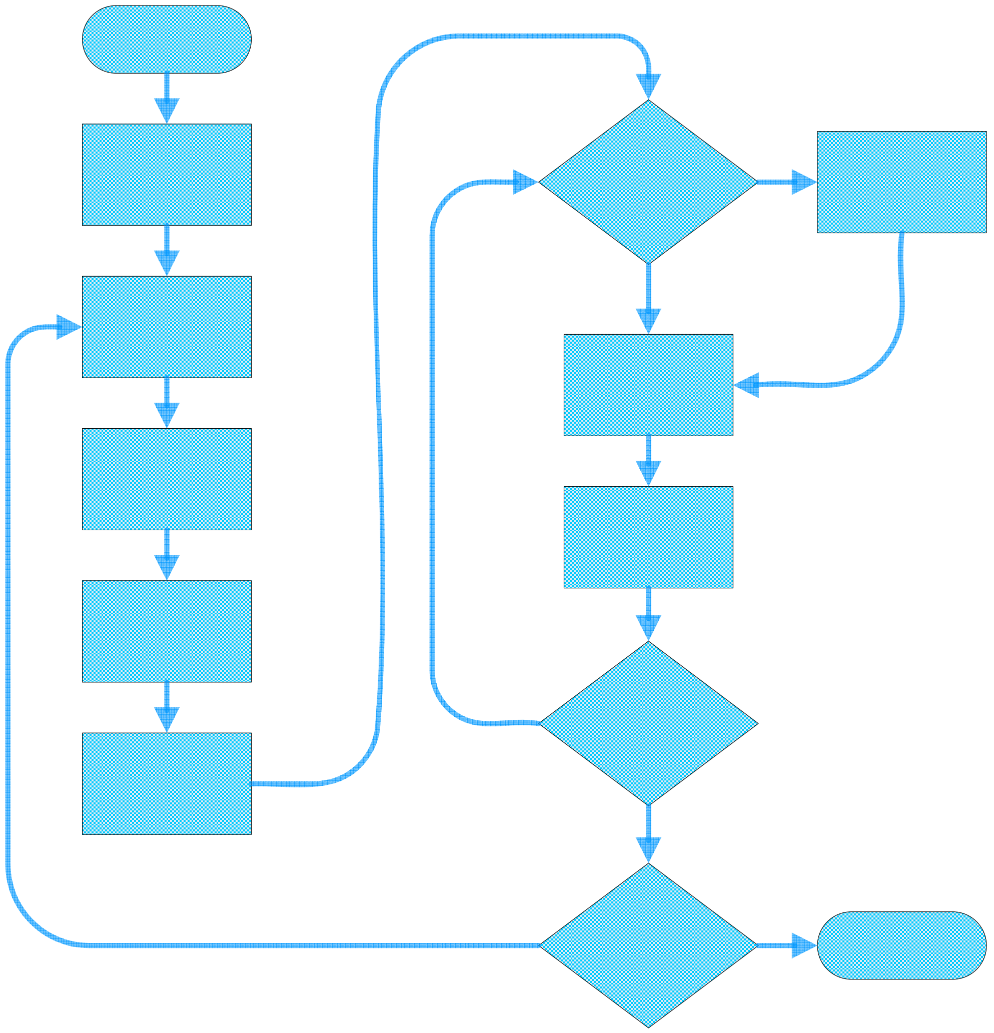


Figure 13: Bubble Algorithm

Path-Finding Algorithms

The "best path" algorithm is used to direct the robot around obstacles in the most direct route. The algorithm used in this program is based on the A* (A Star) best path search.

The A* algorithm is based off Dijkstra's Algorithm. Dijkstra's algorithm is guaranteed to find the shortest path from one point to another while A* is not guaranteed to find the best path, but a very good path. The A* algorithm works by starting from the starting point and working its way outward. For each adjacent vertex (in this case a vertex is a pixel) from the starting point it assigns it a value or a "cost of movement." It then places each vertex in an ordered queue. The queue is ordered by the vertex's value. The vertex with the lowest cost is always at the head of the queue. Each cost is equated by a heuristic. The Heuristic in this program is called the "Manhattan distance." Each vertex's cost is the sum of its distance from the starting point and its distance from the ending point. Each vertex is only visited once. A* is a better choice than Dijkstra for computer programs because Dijkstra will look at many unneeded vertices. A* saves a lot of time because it looks, based on a heuristic, at only the most probable moves.

The vertex at the head of the queue is always to be examined first. When it is examined, all of its adjacent vertices, if they were not already visited, are given its cost and it has a link to the current vertex. When the end point is found, we can traverse the vertexes back to the beginning to find our path. Figure 14 demonstrates this algorithm.

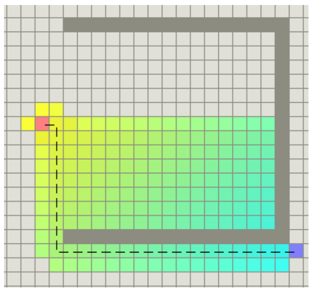


Figure 14: A* Path-Finding Algorithm

End	2	3	4	5	6	7	Start	8
(7+0) = 7	(6+1) = 7	(5+2) = 7	(4+3) = 7	(3+4) = 7	(2+5) = 7	(1+6) = 7	(0+7) = 7	(1+8) = 9
2	3	4	5	6	7	Start	X	Start

Table 3: A* Algorithm Calculation

In Table 3, the top row is the vertex's number, the middle row is square's cost (distance to start + distance to end), and bottom row is the linking of the previous vertex.

From Start, the adjacent moves are 9 and 7 so it moves to vertex 7. Vertex 7 has one adjacent vertex (vertex 6) with a score of 7; seven is less then 9 so we move to that vertex. This continues to the end, at which point the vertices chosen are traversed in reverse to find create the selected path.

The path-finding algorithm with less turns, which is represented by the green line, uses the result of the above algorithm to create the simplest and straightest path. The importance of implementing this algorithm is to save time and complexity in the robot's movement. The more turns the robot is required to take, the more instructions it must receive, the longer it takes, and the higher the chance for errors to occur.

The general algorithm is as follows:

- Start at the robot's position
- Draw a straight line to the destination (where the user clicked)
- Once the line is drawn, check to see if the line has hit any obstacles
- If no obstacles were encountered, then the path is done
- If an object is hit, then more work must be done:
    - ❖ Move the destination by 1 pixel and try again
    - ❖ If an object is still hit, repeat the previous step
    - ❖ Continue until no object is hit
    - ❖ Perform the same between the new destination point and the original destination point

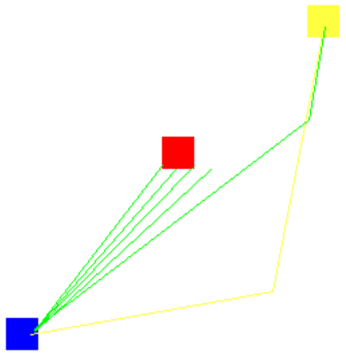


Figure 15: The Less Turns Path-Finding Algorithm

Communications Algorithms

Since RF communication is used for the communication between Robot and the PC system, the RF module was treated as a client that is sending and receiving data from the PC server. The control GUI classes are set to implement Listeners and Observers to respond to button press events and serial data updates such as sending and receiving data from/to the PC.

The packet design and the protocol could be listed as the most important elements. Initially a simple protocol, similar to High-Level Data Link Control (HDLC), was used, but due to the changes of the hardware specification and hardware limitations, a fixed length algorithm was used in the end. This means all the packets need to be of fixed length, 13 bytes for this project, and all packets not equal to the length throw errors. Due to the short range of the communication and the internal error correcting of the modules, the data was sent in groups with no specific flags or additional error checking.

The user interface will maintain the last packet it sent out so as to send it out again if an error message is received. There are two types of packets, Read_in and Send_out. The Read_in packets are accepted in sets of 1, 2, or 6.

The format of all packets sent out is: 1 byte for state, 8 bytes of 0's (since no ultrasonic data is sent), 2 bytes for the compass direction, and 2 bytes for the distance. More than 1 packet will never be sent in a set.

A standalone thread class is used to be able to read the incoming packets effectively. Each packet read in is parsed and the information contained in it is used to determine what type of communication was received. The javax.comm package is used to create the events that this thread watches for, as well as providing the functionality to send data to the serial port. After reading in the packet, it is stored in ReadInPacket.packet for use where necessary in the code.

Packet Parsing Algorithm

1. Information read in must be in 13, 26 or 78 bytes sets. If it is different a debug statement is printed and the information is thrown away for testing and data insurance reasons.
2. If it's 13 bytes, the byte array is decoded to a Packet, which is saved in ReadInPacket, and the Observers are notified to perform an update() in order to let other methods make a call to the decoded information.
3. If it's 26 bytes, that means a Map_Travelling_wait or Control_waiting state is received. If there is no error in the first packet, then the second packet is parsed.
4. If it's 78 bytes, that means these are Take_Reading packets, ArrayUtil is used to parse the packets and put the ultrasonic data in an int array.

Since the packet transfer is in bytes and the program calculations are in integers, encoding/decoding functions were implemented, from native arrays to byte arrays and from state string to state byte.

The following table shows what communication the GUI expects to send or receive based on the current state and the button pressed.

Button or State	TESTING	STOP_READY	Taking_Readings	Map_ Travelling_ Wait	Control_ Waiting
INCOMING COMMUNICATION FROM ROBOT AND RESPONSE, IF NOT RELATED TO A BUTTON					
	Recieve Testing state with all zeros in the data part/ Send echo	After testing is echoed back successfully 1 packet will be received	9 packets from data readings followed by 1 Request for instructions	1 packet received showing state asking for further instructions	1 packet received showing state asking for further instructions
MAIN WINDOW BUTTONS:					
Control		Control_Waiting sent to robot			
STOP		Stop_Ready sent to robot	Stop_Ready sent to robot	Stop_Ready sent to robot	Stop_Ready sent to robot
Reset		Stop_Ready sent to robot	Stop_Ready sent to robot	Stop_Ready sent to robot	Stop_Ready sent to robot
Quit		Stop_Ready sent to robot	Stop_Ready sent to robot	Stop_Ready sent to robot	Stop_Ready sent to robot
BUTTONS IN POP UP FOR MAPPING MODE					
Start Mapping		Taking_Readings sent to robot			
Stop Mapping		Stop_Ready sent to robot			
Continue Mapping			Taking_Readings sent to robot		
Move Mapping Location			Map_Traveling_ wait sent to robot		
BUTTONS IN POPUP WINDOW FOR CONTROLLED MODE					
Rotate Robot X Degrees					Control_Traveling sent to robot
Move					Control_Traveling sent to robot
Close					Stop_Ready sent to robot
BUTTONS FOR POP UP IN MAPPING MODE WHEN MOVE MAPPING LOCATION PRESSED :					
Rotate Robot X Degrees				Map_Traveling sent to robot	
Move				Map_Traveling sent to robot	
Close				Taking_Readings sent to robot	

Table 4: GUI Communications Overview

Analysis of Difficulties

This project was fraught will difficulty from the start.

The project was behind schedule right away due to a combination of changing professors and a lack of communication from a hopeful sponsor. This resulted in the late discovery that the proposal written for Senior Design I was not accomplishable in one quarter. This resulted in a complete redevelopment of the project for the design review at the start of Senior Design II. This in turn caused numerous additional delays due to the lack of explicit detail in the new proposal, which had to be written in very short order.

Once the project was truly underway, many more problems cropped up.

The Java API (Application Programming Interface) for communication with the serial port of a computer was difficult to integrate, especially when trying to use a virtual serial port created by a Bluetooth USB device. This led to many problems with establishing packet sizes and communication timing.

Also changes to the packet algorithm required more classes to be implemented on short notice. The packet size had to be made divisible by bytes (in other words a multiple of 8 bits) for the information to be successfully sent over the wireless and the individual sections of the packet also had to be divisible by bytes due to the RS232 requirements and javax.comm package requirements.

The ultrasonic data algorithms were built on a turret arc of 180 degrees. Unfortunately it was discovered that they only seemed to allow a 90 degree arc. This caused many changes in the communications and algorithm processing, as well as lowering the overall accuracy obtained from each mapping position.

The sensor offsets on the robot, in conjunction with the GUI using the exact center of the robot as the “origin” of the robot, rather than one of the sensors, caused extensive problems with the algorithm that interprets the ultrasonic data.

The original motors for the robot needed to be replaced with higher torque ones at the last minute, which required disassembling the robot and the motor gear box and reassembling it again.

Several times, a group member would be incapacitated for a period of time due to sickness or disability problems. This caused problems with planned due dates and meetings, as well as class attendance. Also several times the project had to be delayed due to the ridiculous amount of work that had to be done for other classes being taken at the time, which was at least 5 fold the work load of any quarter previous.

It was discovered that the Bluetooth modules purchased for the project didn’t have the capability to connect with the robot in the way that was needed, so the whole Bluetooth concept had to be dropped near the end of the project and replaced with easier to use RF modules.

Conflicts in the way the coordinate systems were set up in the User Interface and the Robot caused problems with integration until the cause was determined.

Lack of willingness to modify one’s own part to help interact with the rest of the project was another difficulty. Many painful compromises needed to be made in communication size and other aspects due to this problem.

Power draw on the robot caused the microcontroller to shutdown when powering the motors; resulting in another, separate battery being added. In addition, several parts of the original robot communication system operated at different input voltages, so voltage level translators had to be bought and installed in several places, and then all of it was removed when the Bluetooth was replaced with the RF modules.

Finding and implementing the path finding algorithm was an extensive task. First the algorithm used would not actually find all paths or by any means a good path if it did find one. Then the algorithm would find a path, but it would use far more turns than the robot could feasibly accomplish without a much better guidance system.

Trying to find a way to place an updatable map in the GUI was another major problem. Java applets do not function inside of swing GUI elements and there was no easy way to create a map without the use of the applet. This resulted in using a rougher map system without a lot of the features the Applet offered.

Getting the servos to stop at exact degree measurements proved to be difficult. When sending pulses to the turrets they would stop at approximately the same position every time but there was a small margin of error. The placement of the turrets also made it hard to get the center location to point directly forward/backward for the center reading and the use of the sensors in object avoidance while traveling. It was also discovered that sending information to the turrets too quickly would cause strange behavior.

The Ultrasonic measurements showed an amount of error. When taking a reading, the actual measurable distance seemed to be longer then the reading received from the sensor.

Figuring out why the communications were displaying very strange results, such as only sending every other byte of information, was tricky. It was finally discovered that using the PBasic Serial in and out commands along with the ability to organize things by words was unacceptable. After changing the code to always use bytes, the communications worked.

Testing different algorithms to determine what would work for the path finding, bubble, etc. was a lengthy task.

Creating an environment where the user interface could test communication with the modules required that the interface be set up to test both send and receive at the same time, since they could not be tested individually due to the module functionality.

Due to timing and other factors, one person ended up footing the majority of the bill for the project up front, this caused a few additional delays do to income availability.

Things That Would Be Done Differently

The project development should have continued even though there was the possibility of gaining funding from Honeywell, Inc. The robot components were half ordered and waiting for testing and assembly at this point. Continuing with this development would have placed the robot completion many weeks sooner, allowing time for testing and system integration. The result was that the final parts had to be ordered during the quarter and assembly and test had to take place in short order.

The ultrasonic rangers would have been placed on a single rotation platform. This would eliminate the problems stemming from trying to ensure that separate servos are turning exactly the same amount. The servo would then be set up in continuous rotation mode if partial rotations were possible, or additional rangers would be added to make up the loss in angular arc.

The thought of Bluetooth, a recently new standard, for use in the project was an interesting idea, but upon reflection the tried and true methods of standard RF communication were much more appropriate for a project with this amount of additional work. In addition, changing something as integral as the communication standard mid-project can easily lead to more delays and problems with the code and assembly conversions.

Regardless of the amount of work that was required for other classes, better time-slicing would have eliminated a few of the problems, such as the very late discovery that the Bluetooth modules would not work. Rather than working exclusively on one assignment for days, moving back and forth is the better approach.

Team Responsibilities

Timothy C. LaForest

- Initial and secondary project concept
- Initial component research
- Initial component purchases
- Robot Algorithm Design
- Initial Robot Flowchart Design (Revision was a collaboration with Eric)
- Robot State Table Design (Revision was a collaboration with Eric)
- Robot design
- Robot construction/wiring
- Robot programming
- Robot and Robot Component Testing
- Robot Communication Testing (collaboration with Eric)
- Initial algorithm research used for proposal last quarter

Eric Shields

- All Versions of All Documentation submitted
- Website Design and Implementation
- Wireless Communication Testing
- Collaboration with Tim for the Integration of the Wireless Communication into the Robot
- Collaboration with Merve for the Integration of the Wireless Communication into the User Interface
- Inter-team Mediation and Translation – Acting as a Buffer to Avoid Conflicts and Helping to Translate Between Merve and Tim

- Collaboration with Merve on the initial Path Finding Algorithm
- Design and Implementation of the Bubble Algorithm
- Testing of the Bubble Algorithm (Collaboration with Merve)
- Design of the initial, incomplete Path-Finding Algorithm
- Oversee Most areas of Testing, Especially Those Related to the Interaction of System Components
- Go-To Guy: If either Merve or Tim had a Problem I'm the One They Would Ask. This Led to My Involvement in a Large Quantity of the Coding in at Least a Small Way.

Merve Evran

- User Interface Design and Implementation
- Research and Implementation of the New and Improved Path Finding Algorithm
- Communication Between the Serial Port on the PC and the User Interface
- Java and Javax Research
- Integration of the RF Communication Modules into the Control Code
- Testing of the Control to RF Modules Communication (Collaboration with Eric)
- Integration of the Bubble Algorithm into the Control Code
- Testing of the Bubble Algorithm (Collaboration with Eric)
- Maintain Communication Packet Functionality While Specifications Changed

Time Line

Item Description	Primarily Responsible Member	Completion Date
Setup User Interface	ME	12/6/2005
Finalize Project Algorithms	ALL	1/12/2006
Conclude Bluetooth Research	ERS	12/16/2006
Create Website Template	ERS	12/13/2005
Aquire All Hardware	TCL	12/16/2005
Implement Buttons/Map Function	ME	12/16/2005
Test Component Functionality	TCL	1/12/2006
Implement Mapping Software	ME	1/31/2006
Construct Robots	TCL	1/31/2006
Write Avoidance Alrorithm	TCL	1/13/2006
Rewrite Project Proposal	ERS	1/24/2006
Implement Packet interface	ERS	1/31/2006
Test Avoidance Algorithm	TCL	1/31/2006
Write Path Finding Algorithm	ME	1/31/2006
Test User Interface Functionality	ME	1/31/2006
Test Path Finding Algorithm	ME	1/31/2006
Find New Java Interface Tool	ME	1/31/2006
Update website	ERS	1/31/2006
Test Terrain Mapping	ALL	1/31/2006
Bluetooth Module Functionality	ERS	N/A
Order RF Modules	ERS	2/7/2006
RF Module setup	ALL	2/10/2006
System Functionality Testing	ERS	2/14/2006
Write Final Report	ERS	2/14/2006
Demo Poster	TCL	2/14/2006
Team Assesments	ALL	2/22/2006
Finish Website	ERS	2/14/2006

Table 5: Project Timeline

Conclusion

This project started as a great idea with a lot of enthusiasm. Unfortunately, it fell on some very hard times and had to be continuously altered to be simpler and/or cheaper. The final product was nothing like what we wanted at the start and is, by itself, not overly useful, simply a small stepping stone to larger things. This project was a great learning experience in feasibility and the usefulness of predicting constraints. This knowledge will go far in the corporate world.