5-8-2000

A Mobile Terrain Mapping Robot

Brian Deaton
Trinity University

Jeff Dickerson
Trinity University

Jared Newton
Trinity University

A Mobile Terrain Mapping Robot

Senior Design May 8, 2000

Group Members:

Brian Deaton

Jeff Dickerson

Jared Newton

Group Advisor: Dr. Kevin Nickels

Abstract: *The purpose of this project is to develop a user controlled mobile robot that maps an area of terrain such as a floor of a building. The motivation for this project is to build a robot that can perform tasks that are too menial, difficult, or dangerous to be performed by humans. The robot is a radio-controlled truck with a computer system and sensory equipment attached to it. The user, who stands in the room with the robot, drives the robot through a room with a radio control system while the robot uses its rotating SONAR to detect the surrounding terrain. The robot records its position with every point of the SONAR data so that it can accurately modify a map stored in the memory of its computer system. It calculates this position with a compass and an optical wheel encoder. The time required to map an entire room is anticipated to be less than an hour. Objects such as walls, doorways, trashcans, desks, etc. will be mapped. SONAR data is in the form of the time required for sound to travel to an object and be bounced back. An object's distance from the truck is calculated by knowing the time measurement and the speed that sound travels in air under standard conditions. The SONAR transducer is mounted to a servomotor mast that is controlled by an algorithm so that its direction is always known. The position of the robot is calculated by the distance traveled and the direction it has traveled in. After this data is transmitted to the computer a map of the area can be generated. The success of this project will be determined by the accuracy of the map generated.*

Table of Contents

1 Introduction

Mobile robots are a growing area of research and development in the engineering world. They can often perform tasks too menial, risky, or difficult to be carried out by humans. The goal of this project is to create a user-controlled mobile robot that will successfully use SONAR to map an area of flat terrain of approximately 100 m^2. Some future applications of this design project include the mapping of areas unreachable by humans (such as a distant planet), the mapping of areas dangerous to humans (such as an area contaminated by radiation), or for tracking and environment adaptation for autonomous robotic systems (such as robotic wheelchairs).

A mobile robot needs certain basic components to work properly. It needs a brain to perform functions; it needs input and output capabilities; and it needs a method of moving from one place to another. Since the mobile robot will be mapping an area, it is also crucial for it to have a way of keeping track of its position within the given area. Another necessity for a terrain mapping mobile robot is the ability to detect where objects are within the given area so that a precise map can be generated. All of these necessary segments of the robot also need a way of transmitting data to and from the brain of the robot, so communication is also important. Finally, all of the gathered data needs to be organized and processed into a recognizable map for use by the robot's human users.

2 Base Components

The mapping project will use a computer, a Handy Board controller board and a user controlled vehicle to carry the map generating components around the space to be mapped. When deciding what products to use for these applications it is necessary to first consider what is already owned. Next a cost-benefit analysis of buying a new component is performed. For some applications, the benefits of the new component greatly outweigh the price.

2.1 Computer Considerations

Considerations for the computer include size, weight, speed, memory, cost, and desired functionality. The size and weight of the computer must allow it to be carried by a standard remote control vehicle. This specification requires that a laptop computer instead of a personal computer be used. The laptop will perform mathematic functions that are too time consuming to be calculated on the Handy Board. The laptop will generate the map from the data provided by the peripherals. A laptop donated by Brian Deaton is currently being used for the project, however in the future, a laptop provided the Engineering Science Department at Trinity University will be used.

2.2 Controller Considerations

The controller must be able to handle the data from the position tracking and object detection components. The Motorola Handy Board (HB) plus the Expansion Board was chosen as the data acquisition board. The Handy Board is a 68HC11-based controller board developed by Massachusetts Institute of Technology (MIT) professor Fred Martin. In addition to the sixteen I/O ports provided by the Handy Board, the expansion board provides eight more digital outputs, twelve analog inputs, six servomotor ports, a socket for the Polaroid 6500 SONAR module, and an open prototyping area.

Table X: I/O Ports

	Digital Inputs	Digital Outputs	Analog Inputs	Servo Outputs
Handy Board	9	0	7	0
Expansion Board	0	8	12	6
Total	9	8	19	6

The amount of inputs will be sufficient to handle the data generated by the object detection and position tracking segments of the project. The board's 32 Kbytes of memory allows it to store the data momentarily until it is sent to the laptop. The flexibility that the Handy Board provides makes it the clear choice for the data acquisition board.

The Handy Board plus Expansion Board controls the Polaroid 6500 ultrasonic ranging system (SONAR), the servo motor mast angle, the digital compass, and the optical encoder. The HB has its own

two-megahertz processor with 32 Kbytes of memory that allows for programs to be stored and executed. Programming on the HB to control the peripherals is written in Interactive C, a simpler version of C++. The expansion board provides a nine pin easy connect for the SONAR system. The wiring for the other peripherals is connected through the HB's input and output pins.

2.3 Vehicle Considerations and Alterations

A vehicle is needed to move the object detection components around the room to ensure that the area has been completely mapped. The payload of the vehicle will be approximately 5 kilograms and require approximately .1 m^2 of area to be mounted. The payload and area requirements are determined by measuring and weighing the laptop and the Handy Board (the two largest components). A maximum car speed is set to ensure that all encoder ticks are counted. The full details of this consideration will be discussed in Section 3.3 for the optical encoder.

The initial idea for the vehicle was an RC10 remote control car. It has an aluminum frame that allows for mounting a platform to store the components; however, its speed control and suspension proved to be inadequate. The car accelerated too quickly and would not provide a stable base for the rest of the mapping components.

To accommodate the payload and the space required to carry all of the components required, $1/10^{th}$ scale electric remote control monster truck was purchased. The Traxxas Stampede monster truck has fiber-composite frame and a 26.9cm wheelbase providing both the strength and stability required for the project.

To provide for the peripherals, some vehicle alterations are required. The factory suspension is too soft, so the springs were replaced with solid aluminum tubes. A 31.8 cm x 34.3 cm Plexiglas platform (Figure 1) is mounted to the frame of the truck. This provides a base for the laptop, the Handy Board as well as the mapping peripherals. Because the project only requires one optical encoder for position tracking, a trailing wheel design was implemented (Figure 1).

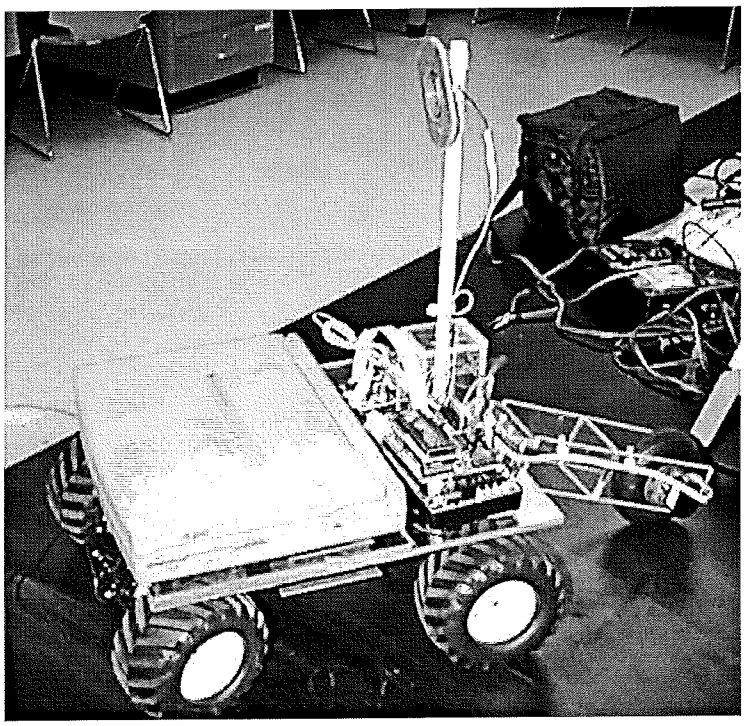


Figure 1: Modifications made to RC vehicle

This design counts the distance that the center of the truck has traveled. The base hardware components are organized as seen below in Figure 2.

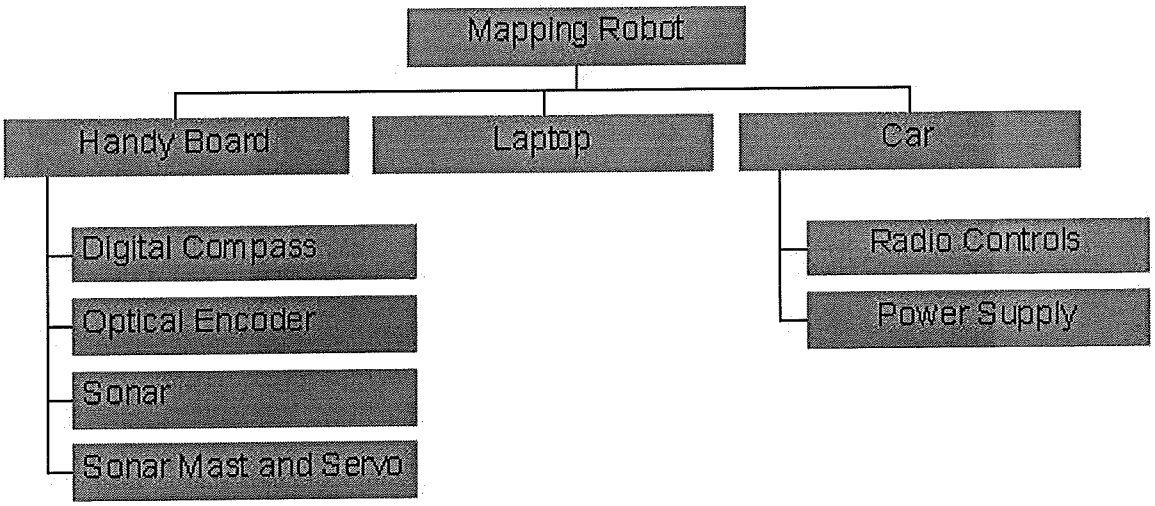


Figure 2: Overall hardware component organization

3 Position Tracking System

In order to generate a map of a terrain, it is necessary to know the position of the robot within terrain itself. Keeping track of the position of the robot is a crucial aspect of the terrain mapper design and requires both the necessary tracking hardware and the software to manipulate the acquired data into a usable form.

3.1 Position Tracking Options

There are many options available for position tracking. The options most focused on are the Global Positioning System (GPS), infrared sensors mounted on the robot, SONAR (SOund Navigation And Ranging) triangulation with the SONAR modules placed inside the terrain area, and odometry.

The GPS system is used extensively in real world applications such as navigation systems. However, civilian use of the GPS system is restricted to an accuracy of about 100 meters. Since this project requires a very high degree of accuracy, GPS clearly will not have sufficient accuracy for this project.

Infrared sensors mounted on the robot provide a cheap method of position tracking. However, due to their limited range (10 – 80 cm), IR would require a large number of readings to accomplish a full mapping of the area.[1]

SONAR triangulation would simply involve mounting SONAR modules inside the terrain area (in this case, the room MEB 322). The information gathered from these fixed SONAR modules is then analyzed to determine the position of the robot. Although the accuracy is high for this method, SONAR triangulation requires a preparation of the terrain area ahead of time and also is not contained on the robot itself. The robot would also not be able to move outside the prepared area, which does not allow much flexibility. Therefore, SONAR triangulation was ruled out for this project.

Finally, odometry provides a relatively inexpensive and accurate method of tracking the robot's position within a terrain. Odometry works by using some device to keep track of the distance that a vehicle has traveled. Since odometry is well understood and widely used, it will be used in this project. Optical

encoders provide the easiest way to implement the odometry configuration for the project because they are used extensively and are easy to work with.

3.2 Odometry

Two optical encoders can be used to calculate the direction that the robot is traveling in. An encoder is attached to each wheel on the rear axle. The difference in the distance traveled by each encoder, a difference that is a result of the turning motion, is used to calculate the angle of rotation. The angle of rotation can then be used to determine the direction of travel.

In using two optical encoders for odometry, there are many sources of error, which accumulates as the encoders are being used. For instance, if the wheel circumferences are incorrectly measured, the translation of ticks into fractions of the circumference will not be accurate. Incorrect circumference measurement is a systematic error, which means it occurs at the same rate as long as the vehicle is moving. Therefore, if the wheel circumference is off by 1%, the final map scale will be off by 1%. If the error is discovered, the scale of the final map can be changed to account for the error. If the wheel circumferences are unequal, a natural turning motion will result in the robot "thinking" it is traveling in a straight line when it is actually turning slightly. This error causes the map to be distorted to varying degrees, depending on the difference in the wheel circumferences and the wheelbase. Another source of error occurs when there is uncertainty about the length of the wheelbase. If the wheelbase is incorrectly measured, the angle of rotation will be calculated incorrectly.[7] Finally, another source of error is wheel slippage because the slipping of one wheel makes the robot think it has rotated through an angle when it hasn't. The reason this error is particularly bad is because the error is not systematic, meaning it can occur randomly at unknown instances to unknown degrees. If the wheel slips 5%, it causes the distance the encoder sees to be off by 5%, but it also may cause the vehicle to think it has turned. Steps to minimize wheel slippage can be taken, such as having wheels with higher coefficients of friction, running the vehicle on a rougher surface, and minimizing quick accelerations of the vehicle. Because of some of these sources of error, another method of implementing odometry may be needed.

A digital compass is a good solution for some of the sources of error. A digital compass provides the angle in which the robot is pointing without the need to calculate the angle of rotation. This eliminates the need for the second optical encoder. Therefore, the wheelbase error and half of the slippage and wheel circumference errors are excluded from the total error, leaving only one circumference error and the slippage of one wheel to contribute to the error.

The single optical encoder is mounted on a trailing wheel placed in the center of the robot, behind the back axle. The wheel is placed in the center because the center distance traveled is the average of the left and right wheel distances.

3.3 Optical Encoder

An optical encoder is a thin, clear disk with small, opaque divisions around its circumference (Figure 3).

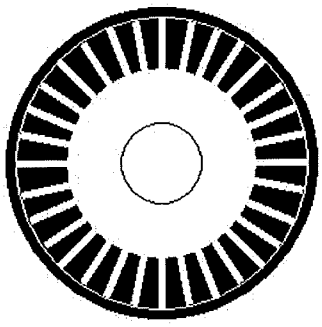


Figure 3: A sample optical disk.

The encoder also has a phototransistor through which the optical disk passes. The phototransistor consists of a light source and a light sensor. The light from the source passes through the optical disk and the sensor receives the light. Each time an opaque region interrupts the light, the output of the phototransistor produces a digital output of "1". The clear region will not intercept the light and will therefore produce an output of "0". A clear region and an opaque region together represent one cycle. If the optical encoder disk is divided into 500 cycles, each cycle represents $1/500^{th}$ of a full rotation of the disk. The angle that the disk has rotated through can then be translated into a distance that the robot has traveled.

In the case of this project, the wheel circumference is 31.9 cm and the encoder has 500 cycles, so each cycle represents (1/500)*31.9 cm, or about 0.64 mm of distance traveled by the robot.

The encoder attaches to the axle of a wheel and as the wheel turns, the encoder outputs a "tick." Ideally, the output of the encoder would generate a hardware interrupt. This would make more efficient use of the processor time because the optical encoder would only have to be attended to when it is producing an output (when the vehicle is moving). Several attempts to implement this procedure proved to be unsuccessful. The alternative procedure of software or timed interrupts is currently implemented. This method generates a one kHz-timed interrupt. Every time this interrupt is generated, the Handy Board looks at the optical encoder to see if a "tick" needs to be counted. This is not processor efficient because the Handy Board looks for an encoder "tick" 1000 times each second even if the vehicle is stationary.

The Handy Board can only handle one function at a time, so the velocity of the robot must be low enough so that the Handy Board can perform other functions between the encoder ticks. At a wheel circumference of 31.9 cm with an interrupt (or poll) every millisecond, the maximum velocity needed to catch every interrupt is 0.64 m/s. However, this would not allow time for other programming to be carried out. Therefore, the maximum velocity must be lower. Currently, the maximum velocity is 0.3 m/s, but anything slower than that is desired.

3.4 Digital Compass

The digital compass consists of a 2-axis magnetometer that measures the magnetic field in a plane. As the compass turns, the magnetic field changes slightly. The compass takes the measurement of the magnetic field and translates it into an angle.[6] Through experimentation, it was found that East = 0 degrees and the angle is measured counter-clockwise. The output of the compass is an input to the Handy Board. The Handy Board updates the current compass angle periodically so that the position of the vehicle can be accurate.

3.5 Position Determination Software

In order to calculate the position of the robot, it is necessary for a program on the Handy Board to keep track of the total number of interrupts from the optical encoder as well as the most recently available compass angle.

Since the Handy Board will be running the peripherals of the robot, and the processor of the HB can only perform one task at a time, it is necessary to determine the priority that each peripheral has. The position of the vehicle in the room is vital. Without the correct position of the vehicle, the map will not be accurate. Therefore, the optical encoder receives the highest priority. The digital compass receives the next highest priority since it is also crucial to know the direction that the vehicle is moving in for the odometry calculations.

The HB can only store a limited amount of data, so it is necessary to upload the data to the laptop frequently. Therefore, the laptop/HB communication receives the third highest priority. Finally, the SONAR readings receive the lowest priority. This is divided into two parts: turning the SONAR mast and getting the SONAR data.

3.5.1 Optical Encoder Programming

The programming overview of the encoder is fairly simple. The program must receive and count the interrupt from the encoder, determine if the robot is moving in forward or reverse, and increment or decrement the total distance (number of ticks) traveled based on that determination. All of this must be done quickly so that the HB can get back to the main program in order to complete other tasks.

Code was written in Assembly language to receive and count interrupts from the encoder. Attempts to get this code working on the Handy Board, however, were unsuccessful. An error message kept occurring which read ".s19 file not contiguous." Despite efforts to fix this problem, the code was never able to be loaded onto the Handy Board. Therefore, assembly code written by Fred Martin of MIT will be used. This code uses timer generated interrupts at a set interval rather than getting hardware (encoder) generated interrupts. This code also counts in the forward direction only. This means that the vehicle cannot travel in

reverse. Otherwise, the encoder count will be incremented despite the reverse motion, resulting in erroneous data.

3.5.2 Compass Programming

The Vector-2X Digital Compass from Precision Navigation can operate in one of two modes. In Master mode, the Handy Board can poll the compass once or choose to receive data continuously. The compass data is then clocked out by the compass at 4 kHz. In Slave mode, the compass is controlled by the Handy Board operating at a maximum clock of 1 MHz. The compass can be polled for data at any time convenient to the Handy Board and clocked out at any speed less than 1 MHz.

Slave mode will be best for this particular application because of the ability the Handy Board has to control the compass at any desired speed. The Handy Board will also not be continuously interrupted by the compass with data. The compass angle will be stored on the HB in two bytes, since the number will range from 0 – 359.

The compass code was first written in assembly and worked on the 68HC11 trainer board. However, attempts to put the code onto the Handy Board were unsuccessful because of the same problems that occurred with the encoder code. Code therefore needed to be written in C language. This code was successful, and the compass can give an angle reading between 0 and 359.

3.6 Position Calculation

The position of the robot is calculated using the data gathered from the optical encoder and the digital compass. First, the digital compass measures the angle at which the robot starts. The compass continues to measure subsequent angles periodically and the most current angle is kept for uploading to the laptop.

Each tick for the encoder wheel represents 0.64 mm of distance traveled. Since the encoder data will be uploaded to the laptop frequently, the vehicle will most likely not have traveled a large distance between uploads. Also, the vehicle will probably not have traveled through a large angle between uploads. Therefore, the distance traveled can be approximated by a straight line. The angle of the straight line can

just be represented by the most recent angle, since the angle should not have changed much. Figure 4A shows the original data gathered by the encoder and compass while Figure 4B shows the approximated straight-line distance and angle.

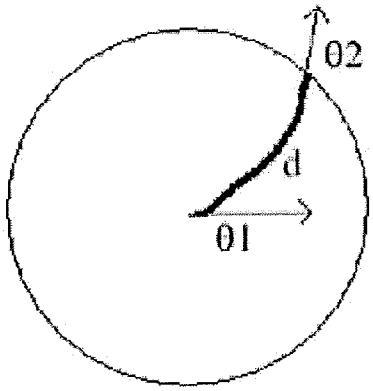


Figure 4A: Original data

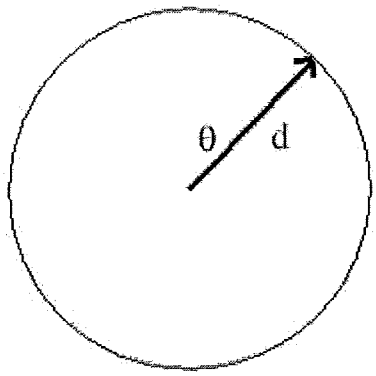


Figure 4B: Straight-line approx.

The distance traveled in the x direction is calculated by multiplying the encoder distance, d, by the cosine of the angle θ. The distance traveled in the y direction is calculated by multiplying d by the sine of θ. Figure 5 shows a sample calculation of the x and y values for an angle. All of these calculations are done on the laptop PC using the data gathered by the encoder and the compass.

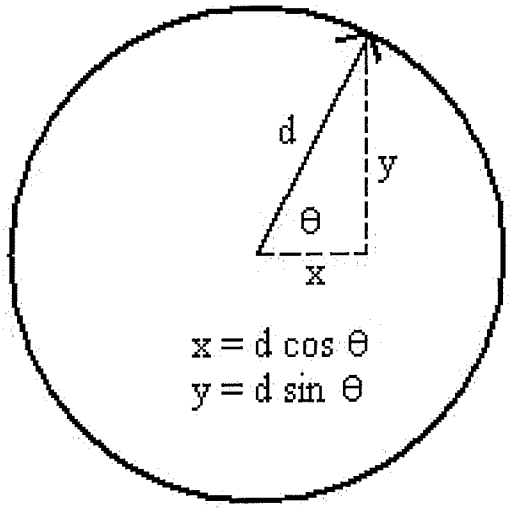


Figure 5: Sample Calculation

4 Object Detection

An essential for map generation is to detect objects in the area to be mapped. For this project, there are two primary options: the Sharp GPD02 infrared sensor and the Polaroid 6500 SONAR ranging module. Either option will be mounted on a mast that rotates 180° to scan the area for objects (Figure 6). These half-rotations will greatly decrease the time required to map an area by scanning the room instead of having to position the car to face each obstruction in an area. The Sharp GP2D02 infrared sensor is capable of detecting objects that are between 10 and 80 cm away with exceptional accuracy and repeatability.[1] At a cost of $21.00 per sensor, this initially appears to be a good option. However, with the possible mapping area reaching 100 m^2, being able to cover an area of only 1 m^2 per half revolution requires a minimum of 200 half revolutions of readings. A half revolution of readings is the readings taken by the object detection device in one half revolution. The number of half revolutions required is based on each point being covered at least twice to allow for a certainty evaluation. A certainty evaluation is required because faulty readings are possible. If an object is detected only once it is possible that a reflection may have distorted the reading, but if an object has been detected multiple times in the exact same place, a degree of certainty is developed. The Polaroid 6500 SONAR ranging module can detect objects up to 10 meters away with an

accuracy of ±3 cm.[5] Each module has a price of $50.00, but the extra area covered per half revolution is worth the increased cost for this application.

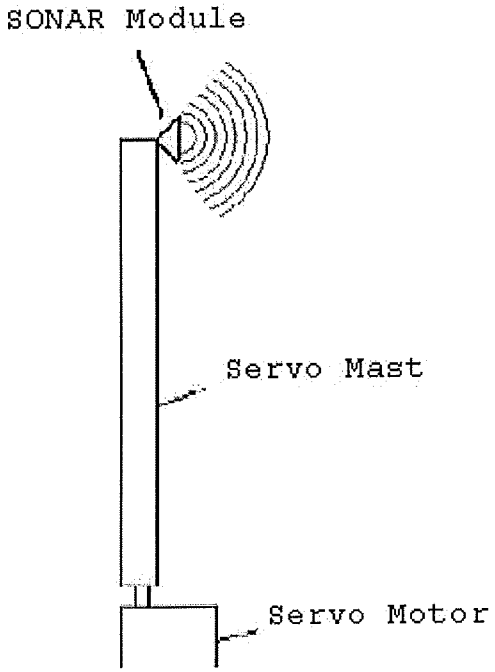


Figure 6: SONAR module mounted on a SONAR mast

The mast is attached to a servomotor that rotates the mast 180°. Each half revolution covers approximately 150 m^2. The increased area covered makes the SONAR module the obvious choice for object detection.

4.1 SONAR Software

The routines used to control the Polaroid 6500 series SONAR module were provided by Dr. Fred Martin of the Massachusetts Institute of Technology Media Lab. These routines are written in Interactive C to be used on the Handy Board. The software routine calculates the time required for a SONAR pulse wave to be transmitted and echoed back to the module. The start time of the pulse is defined by the value of the HB's internal clock. A SONAR pulse is sent immediately after recording the start time. When the SONAR transducer receives the return pulse the HB collects the current time of the internal clock. The

difference between the two time values is stored as a global variable allowing Interactive C to call for its value at any time. The conversion from time to distance will be calculated on the laptop after the data has been transmitted.

See Appendix E for Coding

4.2 SONAR Hardware

The SONAR hardware includes the SONAR transducer and the SONAR module board. The transducer (Figure 7) is a combination speaker/receiver. It performs both functions of transmitting and receiving SONAR pulses. The transducer is connected directly to the SONAR module board that determines if the transducer is transmitting or receiving pulses.

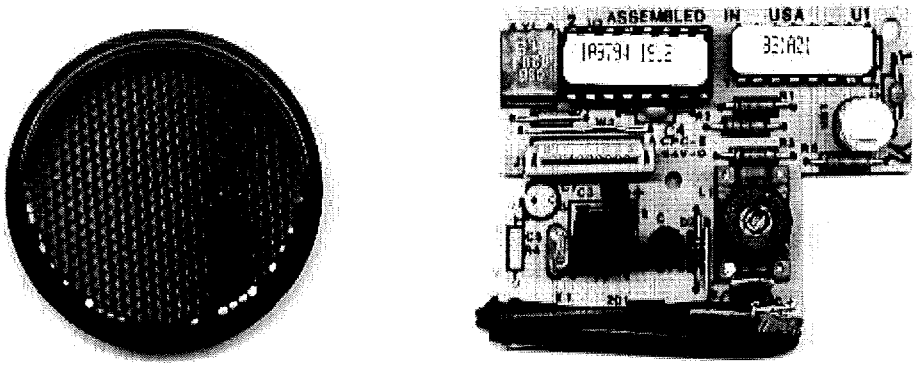


Figure 7: SONAR Transducer and SONAR Module Board

The module board is connected directly to and controlled by the Handy Board.

5 Laptop Software

The Laptop software is organized into three different parts: graphical user interface, map generation, and communications. Each of these different parts of the software is run separately as a thread. A thread is the basic entity to which the operating system allocates processor time. A thread can execute any part of the application's code, including a part currently being executed by another thread. All threads

of a process share the virtual address space, global variables, and operating system resources of the process [9].

When the main thread starts it does some initialization and starts the map generation thread and the communication thread. The main thread is also responsible for handling most of the graphical user interface.

The map generation thread shares a thread-safe list data structure with the communications thread. It waits for the communications thread to receive data from the Handy Board and put that data into the list. The map generation thread continuously check to see if there is any data in the list. When it finds data in the list it removes an element of data from the list. The current position variable is then updated and the data manipulated so that the map modification function can use it. The modified data is then sent to the map modification function.

The map modification function uses the direction and distance reading of the sonar to draw a sonar arc on the map at the current vehicle position. The details of this sonar arc are outlined in Section 6. The map modification function modifies the image of the map that the user sees on the graphical user interface in addition to the actual map represented as a two-dimensional array that is stored in the laptop's memory.

The communications thread opens the serial port and then continuously reads data from it in anticipation of the Handy Board sending a packet of data. This data packet consists of the compass angle, the encoder data, the sonar angle, and the sonar distance reading. The data packet is then stored in the thread-safe list shared by the map generation thread.

Locking mechanisms must be used for global variables so that the threads do not try to modify the same variable at the same time. The map generation thread and the communications thread use a shared list so that the communications thread can transfer the data it receives to the map generation thread. This list has a locking mechanism so that the map generation thread does not try to read the data from the list while the communications thread is writing data to the list. The map itself also has a lock so that a command from the user interface does not try to modify the map at the same time the map generation thread is tries to modify it. When a thread wants to use a shared data structure it merely locks the structure, uses the structure, and then unlocks the structure.

Screenshots of the laptop software are shown below in Figure 8.

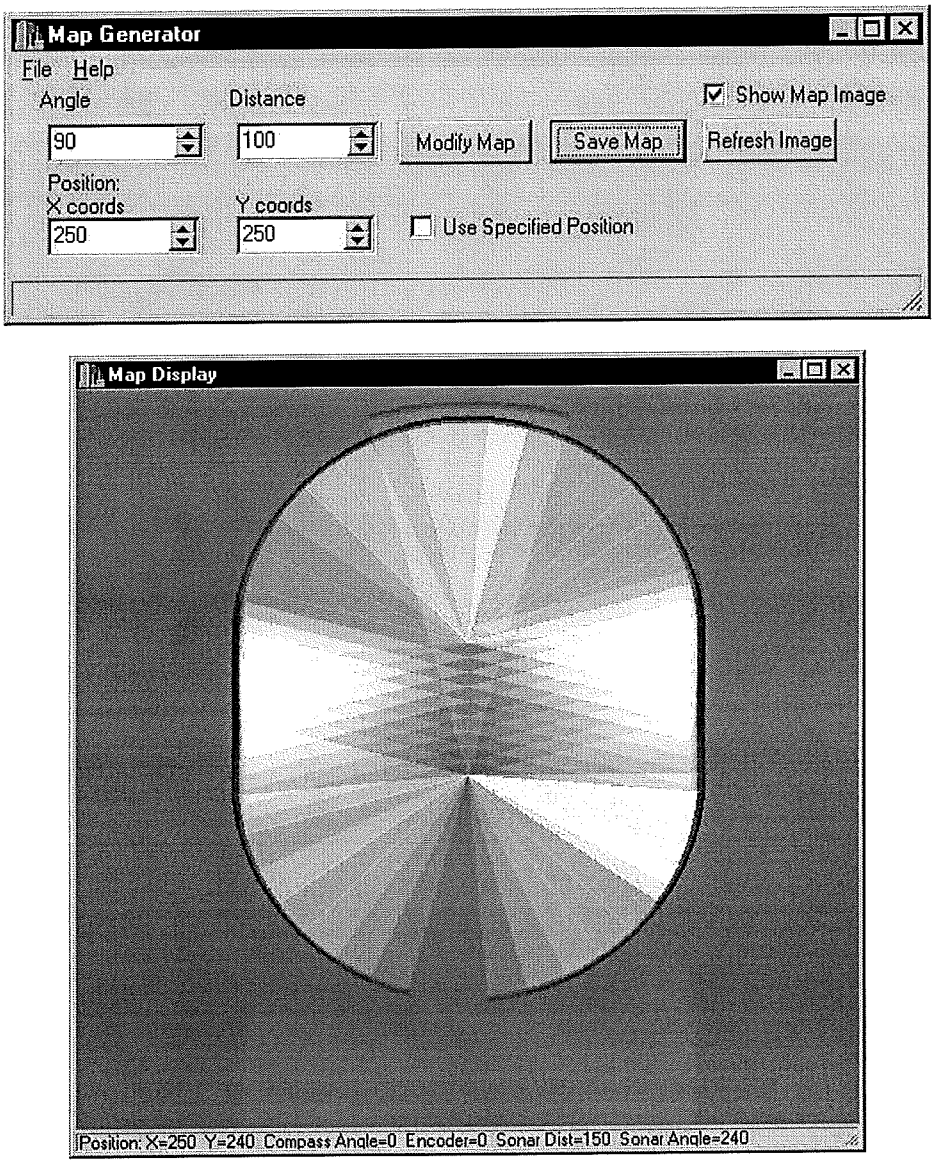


Figure 8: Screenshots from map-generation software

6 Map Generation

As the robot travels through rooms in a building a map is generated on the laptop computer to store the surrounding terrain. This map is continuously updated from the data gathered by the Handy Board as the user drives the robot through the room.

The map is a two dimensional grid based representation of the floor of the chosen room. Each grid cell of this two dimensional grid represents a small square section of the floor. Currently each cell of the grid represents nine square centimeters of floor space. Depending on the computer's memory capacity and the desired resolution the cells can be assigned a different area of floor space.

The cells of the map are assigned a numerical value based on the certainty of an object's existence in the nine square centimeters of floor space represented by the grid cell. The range of these numerical values or occupation values is from 0 to 255. This range was chosen because it is large enough to accommodate this application and, in addition, a number in this range can be stored in an 8-bit register or variable and make effective use of memory. An occupation value of zero assigned to a grid cell means that the robot is sure that that section of the floor is vacant and the occupation value of 255 means that the robot is certain that that section of floor is occupied by a wall or an object.

Initially, the grid cells are assigned values of 127 because this value is in the middle of the range of possible values (0-255). The value 127 in a grid cell signifies that there is a fifty-percent chance that an object is occupying the floor space represented by that grid cell. As the robot collects more sonar data for a particular grid cell the value changes of the grid cell. If a section of the floor is not occupied by an object or a wall then the corresponding grid cell value is reduced every time sonar data is collected for that grid cell. Eventually when enough sonar data is collected, the grid cell that corresponds to the empty floor space will contain an occupation value of zero. A sample map generated the laptop software is shown in Figure 9.

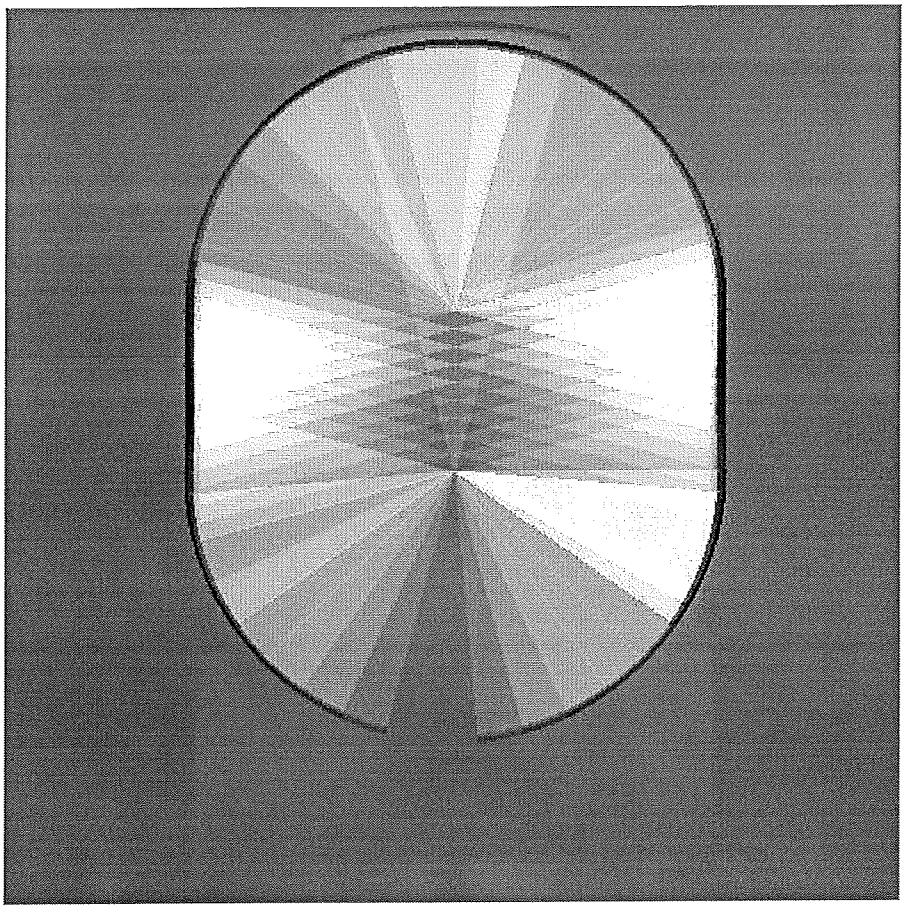


Figure 9: Sample map – color coded to show certainty

The map is stored in a large two-dimensional array. Sonar data is stored to the array in arcs to represent the actual arc shape of the sonar pulses. This method does not draw accurate corners; it merely gives a rough outline of an area of terrain. For instance, if the sonar detects a wall five meters away the radius of the arc is five meters centered at the vehicles current position. The arc's half angle is drawn 10 degrees wide because the sonar is accurate within about 10 degrees. The grid cells on the arc are modified with higher values to represent the presence of an object and the grid cells between the arc and the car within 10 degrees are modified with lower values to represent the absence of objects. For example, if a wall is detected five meters away a 10 degree arc is drawn on the map five meters from the robot's current position in the map. All of the grid cells that lie on the arc are modified with higher values to represent that the robot is more certain that there is an object occupying the space represented by those grid cells. In

addition, all of the grid cells that lie between the arc and the robot's current position are modified with lower values to represent that the robot is less certain that there is an object occupying the space represented by those grid cells.[9]

7 Conclusion

Currently all portions of the project work correctly except for the sonar servo motor control software. If we were to figure out how to control the sonar servo motor with the handy board software we would have a functional terrain mapping system. The experience gained from this project is invaluable.

8 References

1. Detroit, Barry. "Interfacing the Sharp GP2D02 infrared ranging sensor to the HandyBoard." 9 Dec. 99. http://reality.sgi.com/barry_detroit/GP2D02_1.html (9 Dec.99).

2. Arcom Control Systems Web Page. <http://www.arcom.co.uk/products/icp/pc104/processors/Target188.htm> (2 Dec. 1999).

3. Tri-M Systems Web Page. Diamond Systems Garnet MM. <http://www.tri-m.com/products/diamond/garnet.html> (2 Dec.1999).

4. The Handy Board Web Page. <http://lcs.www.media.mit.edu/groups/el/Projects/handy-board/index.html> (2 Dec. 1999).

5. Polaroid 6500 SONAR Data Sheet

6. Precision Navigation Web Page. Vector 2X Compass Module. <http://www.precisionnav.com/vector2xmain2.html> (2 Dec. 1999).

7. Borenstein, Johann and Liqiang Feng. "Measurement and Correction of Systematic Odometry Errors in Mobile robots," IEEE Ttransactions on Robotics and Automation, Vol. 12, No. 5, October 1996.

8. Elfes, Alberto. "SONAR-Based Real-World Mapping and Navigation," IEEE Journal of Robotics and Automation, Vol. RA-3, No. 3, June 1987.

9. Song, Kai-Tai and Charles C. Chang. "Navigation Integration of Mobile Robot in Dynamic Environments," Journal of Robotic Systems, Vol. 16, No. 7, 1999. John Wiley and Sons, New York.

10. (*Microsoft Programmers Guide to Windows 95).

Appendix A Engineering Design Process

A.1 Establishment of design specifications and criteria

From the start, we needed to establish what this project was going to do and decide the best method to accomplish the goal. We knew we wanted to map a terrain, but what kind of terrain? We chose a flat terrain, specifically a classroom of about 100 m^2 in area. We needed a method of gathering various data, so we chose those components as well.

A.2 Analysis

Analysis of the problem involved determining the best way to accomplish the design criteria. We analyzed various computer possibilities, data acquisition methods, and position tracking methods, and eventually made some final decisions.

A.3 Synthesis

The purpose of the project is to synthesize a real-world terrain in a computer-generated map.

A.4 Health and Safety

The safety of the user operating the robot is a concern. If the user begins to take the robot apart or tries to alter the robot in any other way, electrical shock could result. Therefore, it will be necessary to protect the dangerous parts of the robot from interference.

A.5 Social, environmental and political issues

There are no relevant social, environmental or political issues for this project.

A.6 Construction

An optimal construction is desired for this project so that parts can be easily accessed for repair. The materials used in the construction of the robot are also a consideration. For instance, the frame of the vehicle must be strong enough to support the weight of the laptop and other hardware.

A.7 Testing

Testing of the project will be done next semester.

A.8 Evaluation

Evaluation of the project will be done next semester.

A.9 Communication

Communication between group members was vital in the design process up to this point. We divided the project into roughly three parts, and each part has relevance to the other two. If communication were poor, the three parts would not fit together in the end. Weekly meetings with both the advisor and the group alone helped communication greatly.

A.10 Mathematical Modeling

This project involves taking a physical area and creating a map based on data taken from that area. A mathematical model must be made to convert the data into a map. Using a grid system and odometry are both examples of the mathematical modeling that will be done.

A.11 Chemical, electrical and mechanical engineering analogs

There are no analogs relevant to this project.

A.12 Optimization

We will need to optimize the space that the robot takes up in order for it to be useful in the future. Optimizing the programming is another issue, since we don't have unlimited space to store the program or unlimited memory to store the gathered data.

A.13 Ethics

This project could be used for malicious purposes, such as spying, but ethical issues are generally not a concern for this project.

A.14 Aesthetics

Aesthetics in this project is primarily concerned with the programming. Making the programming pleasing and able to be followed by a qualified person is important to this project. If the programming is ugly, the project has little chance of being taken seriously by anyone.

A.15 Robust design

Robust design is another important aspect of our project. We want the robot to handle any kind of user, no matter what unexpected things they try to do with it. Eventually, a robot like this would be autonomous, but that aspect is not crucial to our design at this point.

A.16 Economics (life cycle analysis)

Although this project will not be sold or mass produced, it may be used by future design groups. Therefore, it is important to spend money wisely so that the robot will last for years to come.

A.17 Manufacturability, sustainability and reliability

The robot must be reliable as well because if it fails to work in the future, there is something wrong with the design.

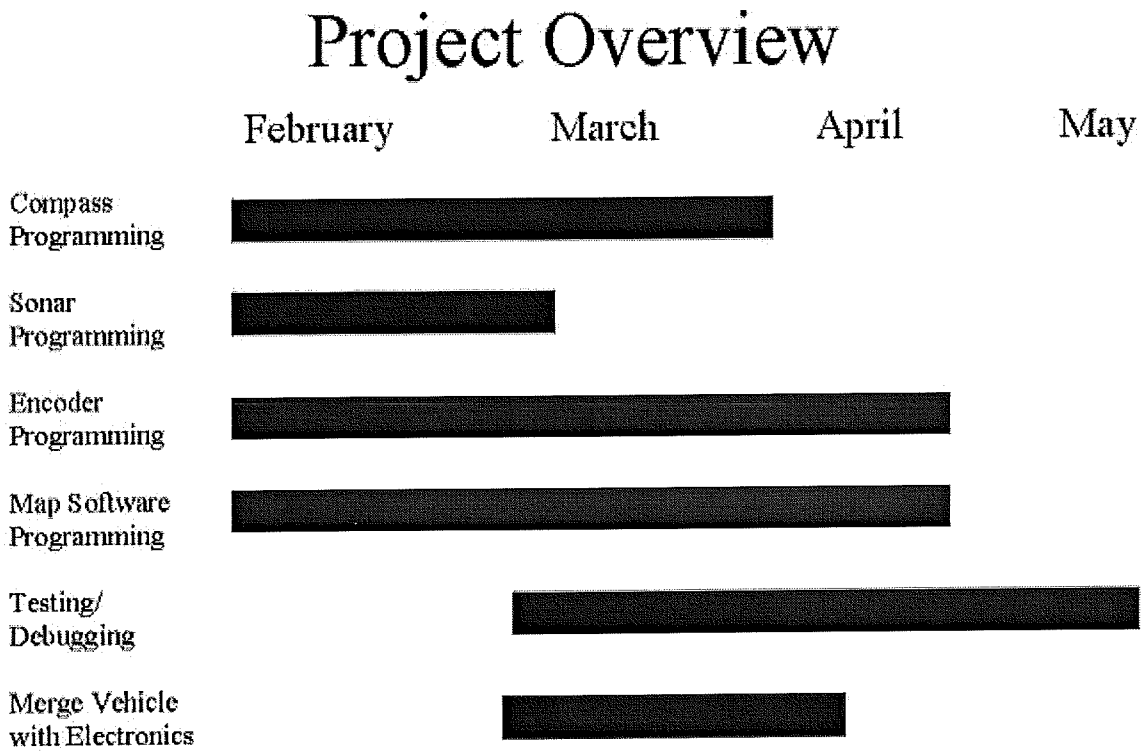
Appendix B - Budget Status

Maximum Budget allotted: $ 1000.00

Budget Spent

SONAR Ranging Module (2)	$50.00
Digital Compass:	$56.50
Optical Encoder:	$39.00
New RC vehicle	$156.00
Handy Board + Expansion Board	$380.00
Servomotor	$17.00
Extras (Batteries, Wheel, etc.)	$50.00
Total:	**$748.50**

Appendix C – Project Overview



Appendix D - Group Member Contributions

Jared Newton: For the project, I have been primarily responsible for organizing group meetings and keeping everyone in the group on the same page. I researched the hardware components. This includes the computer to act as the brain of the project, the SONAR ranging modules, the optical encoder, and the digital compass. Once I found the components Brian and Jeff helped decide what would be best for our application. A large portion of my time working on this project was spent searching through the Internet finding other projects similar to ours and trying to gain an understanding how terrain mapping is traditionally performed. For the paper, my primary responsibilities are the Base Components, Object Detection, and portions of the Abstract.

Brian Deaton: I have been involved with the overall vehicle hardware and software design. I wrote all of the C++ software that runs on the laptop. I have also helped some with the programming for the Handy Board.

Jeff Dickerson: My contribution to the project was researching the position tracking options available, researching the odometry, and flowcharting the odometry programming. This included the decision to purchase the optical encoder and the digital compass, as well as doing initial calculations for the position tracking algorithm. I also did some of the initial calculations for the final mapping algorithm, which takes into account both the odometry data and the SONAR data. For the paper, I wrote the sections involving the Position Tracking System (Section 3) and the Introduction.

Appendix E – Coding

Compass Code

```
void main(){
  int i;
  int c;
  int x;
  int in_byte=0;
  int bit_5=32;   /* 000100000 */
  int SDO=0;
  int temp;
 while(1){
  temp=0;
  SDO=0;
  set_digital_out(6);
  set_digital_out(5);
  set_digital_out(4);
  set_digital_out(3);
  msleep(10L);
  in_byte=0;
  clear_digital_out(6);
  msleep (10L);

  clear_digital_out(5);
  msleep(10L);

  set_digital_out(6);
  msleep(190L);

  set_digital_out(5);
  msleep(20L);

  clear_digital_out(4);
  msleep(20L);



  for (i=0; i<7; i++){
   clear_digital_out(3);
    /* msleep(2L); */
   set_digital_out(3);
    /* msleep(2L); */
    }

   for (i=8; i>=0; i--){
   clear_digital_out(3);
    /* msleep(2L); */
   set_digital_out(3);
    /* msleep(2L); */

   in_byte=peek(0x7FFF);
```

```
   in_byte=in_byte & bit_5;

    if(in_byte > 0){
      temp=1;
     /* 2^i function */

        for (c=i-1; c>=0; c--){
        temp=temp*2;
         }

    SDO=SDO+temp;
}
}
    printf("SDO = %d\n",SDO);
    msleep(500L);

}
}
```

Sonar Code

```
/*
        sonar.c
        Polaroid 6500 routines for Handy Board / Interactive C
        by Fred Martin, fredm@media.mit.edu
        Sat Nov 22 13:57:35 1997

        echo signal is connected to tic3/pa0;
        init signal is pd5 (SS);
        binh signal is pd4 (SCK)
*/
        int x=10;
void sonar_init() {
        bit_set(0x1009, 0x30);          /* ddrd */
        bit_set(0x1021, 1);                     /* at tctl2, */
        bit_clear(0x1021, 2);           /* set tic3 for rising edge */
}


int sonar_sample() {
        int start_time;

        poke(0x1023, 1);                        /* clear tic3 flag */

        start_time= peekword(0x100e);   /* capture start time */
        bit_set(0x1008, 0x20);                  /* trigger pulse */

        while (!(peek(0x1000) & 0x1)) {   /* wait until receive echo */
                if ((peekword(0x100e) - start_time) < 0) {
                        /* if too much time has elapsed, abort */
                        bit_clear(0x1008, 0x20);
                        return -1;
```

```
                }
                defer();                                /* let others run while waiting */
        }

        bit_clear(0x1008, 0x20);            /* clear pulse trigger */

        return peekword(0x1014) - start_time;       /* tic3 has time of echo */
}

int sonar_closeup() {
        int start_time;

        poke(0x1023, 1);                            /* clear tic3 flag */
        start_time= peekword(0x100e);
        poke(0x1008, 0x20);

        while ((peekword(0x100e) - start_time) < 1000);

        bit_set(0x1008, 0x30);                      /* turn on BINH */

        while (!(peek(0x1000) & 0x01)) {
                if ((peekword(0x100e) - start_time) < 0) {
                        /* if too much time has elapsed, abort */
                        bit_clear(0x1008, 0x30);
                        return -1;
                }
                defer();
        }

        bit_clear(0x1008, 0x30);

        return peekword(0x1014) - start_time;       /* 0x1014 is tic3 */
}

void main()
{
        sonar_init();

        while (1) {
                int result;
                result= sonar_closeup();
                if (result != -1) printf("%d\n", result);
                else printf("*******\n");
                msleep(50L);
            send_int(x);
       }
}
void send_int(int x){
  msleep(10L);
  poke(0x102f, x);
}
```

```
/*****************************************************************************
*       Main.cpp
*       by Brian Deaton
*
*****************************************************************************/


//---------------------------------------------------------------------------
#include <vcl\vcl.h>
#pragma hdrstop

#include "Main.h"
#include "SerialThread.cpp"
//#include "MapDisplay.cpp"

//---------------------------------------------------------------------------
#pragma link "CSPIN"
#pragma resource "*.dfm"
TMainForm *MainForm;

//---------------------------------------------------------------------------
__fastcall TMainForm::TMainForm(TComponent* Owner)
        : TForm(Owner)
{
        TSerialThread *SerialThread = new TSerialThread(false);
        TMapThread *MapThread = new TMapThread(false);
        if(SerialThread == NULL  ||  MapThread == NULL)
                cerr << "Thread Creation Failed";

}
//---------------------------------------------------------------------------
void __fastcall TMainForm::FormCreate(TObject *Sender)
{
        Application->OnHint = ShowHint;
}
//---------------------------------------------------------------------------
void __fastcall TMainForm::ShowHint(TObject *Sender)
{
        StatusLine->SimpleText = Application->Hint;
}
//---------------------------------------------------------------------------
void __fastcall TMainForm::FileNew(TObject *Sender)
{
        //--- Add code to create a new file ---
}
//---------------------------------------------------------------------------
void __fastcall TMainForm::FileOpen(TObject *Sender)
{
        if (OpenDialog->Execute())
        {
                //---- Add code to open OpenDialog->FileName ----
        }
}
//---------------------------------------------------------------------------
void __fastcall TMainForm::FileSave(TObject *Sender)
{
        //---- Add code to save current file under current name ----
}
//---------------------------------------------------------------------------
void __fastcall TMainForm::FileSaveAs(TObject *Sender)
{
        if (SaveDialog->Execute())
        {
                map->dump_map_to_file(SaveDialog->FileName);
        }
}
//---------------------------------------------------------------------------
void __fastcall TMainForm::FilePrint(TObject *Sender)
{
        if (PrintDialog->Execute())
```

```
        {
                //---- Add code to print current file ----
        }
}
//---------------------------------------------------------------------------
void __fastcall TMainForm::FilePrintSetup(TObject *Sender)
{
        PrintSetupDialog->Execute();
}
//---------------------------------------------------------------------------
void __fastcall TMainForm::FileExit(TObject *Sender)
{
        Close();
}
//---------------------------------------------------------------------------
void __fastcall TMainForm::HelpContents(TObject *Sender)
{
        Application->HelpCommand(HELP_CONTENTS, 0);
}
//---------------------------------------------------------------------------
void __fastcall TMainForm::HelpSearch(TObject *Sender)
{
        Application->HelpCommand(HELP_PARTIALKEY, Longint(""));
}
//---------------------------------------------------------------------------
void __fastcall TMainForm::HelpHowToUse(TObject *Sender)
{
        Application->HelpCommand(HELP_HELPONHELP, 0);
}
//---------------------------------------------------------------------------
void __fastcall TMainForm::HelpAbout(TObject *Sender)
{
        //---- Add code to show program's About Box ----
}
//---------------------------------------------------------------------------
void __fastcall TMainForm::ModifyMapClick(TObject *Sender)
{
        COORDS  temp;
        if(PositionCheckBox->Checked){
                temp.x=map->position.x;
                temp.y=map->position.y;
                PositionLock->Acquire();
                map->position.x=XcoordSpinEdit->Value;
                map->position.y=YcoordSpinEdit->Value;
                PositionLock->Release();
        }
        map->modify_map((PI*AngleSpinEdit->Value)/180,DistSpinEdit->Value);
        if(PositionCheckBox->Checked){
                PositionLock->Acquire();
                map->position.x=temp.x;
                map->position.y=temp.y;
                PositionLock->Release();
        }

}
//---------------------------------------------------------------------------

void __fastcall TMainForm::SaveMapClick(TObject *Sender)
{
        if (SaveDialog->Execute())
        {
                map->dump_map_to_file(SaveDialog->FileName);
        }
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::RefreshImageClick(TObject *Sender)
{

//      int color_base= pow(2,16)+pow(2,8)+1;
```

```
//      int color_base_BR = 127*pow(2,16)+1;
//      int color_base_G=pow(2,8);
     for(int m=0;m<rows;m++) {
        for(int n=0;n<cols;n++) {
            MapDisplayForm->Image->Canvas->Pixels[n][rows-m] = 65793*map->get_map_value(n,m);
            }
        }

}
//---------------------------------------------------------------------------

void __fastcall TMainForm::ShowMap(TObject *Sender)
{
        if(MapImageCheckBox->Checked)
                MapDisplayForm->Show();
        else
                MapDisplayForm->Hide();
}
//---------------------------------------------------------------------------
```

```
/*****************************************************************************
*       Main.h
*       by Brian Deaton
*
*****************************************************************************/

//---------------------------------------------------------------------------
#ifndef MainH
#define MainH
//---------------------------------------------------------------------------
#include <vcl\sysutils.hpp>
#include <vcl\windows.hpp>
#include <vcl\messages.hpp>
#include <vcl\sysutils.hpp>
#include <vcl\classes.hpp>
#include <vcl\graphics.hpp>
#include <vcl\controls.hpp>
#include <vcl\forms.hpp>
#include <vcl\dialogs.hpp>
#include <vcl\stdctrls.hpp>
#include <vcl\buttons.hpp>
#include <vcl\extctrls.hpp>
#include <vcl\menus.hpp>
#include <Classes.hpp>
#include <ComCtrls.hpp>
#include <Controls.hpp>
#include <Dialogs.hpp>
#include <Menus.hpp>
#include "CSPIN.h"
#include <StdCtrls.hpp>
#include <ExtCtrls.hpp>
#include <queue>
//---------------------------------------------------------------------------
class TMainForm : public TForm
{
__published:
        TMainMenu *MainMenu;
        TMenuItem *FileNewItem;
        TMenuItem *FileOpenItem;
        TMenuItem *FileSaveItem;
        TMenuItem *FileSaveAsItem;
        TMenuItem *FilePrintItem;
        TMenuItem *FilePrintSetupItem;
        TMenuItem *FileExitItem;
        TMenuItem *HelpContentsItem;
        TMenuItem *HelpSearchItem;
        TMenuItem *HelpHowToUseItem;
        TMenuItem *HelpAboutItem;
        TStatusBar *StatusLine;
        TOpenDialog *OpenDialog;
        TSaveDialog *SaveDialog;
        TPrintDialog *PrintDialog;
        TPrinterSetupDialog *PrintSetupDialog;
        TButton *ModifyMapButton;
        TButton *SaveMapButton;
        TCSpinEdit *AngleSpinEdit;
        TCSpinEdit *DistSpinEdit;
        TLabel *Label1;
        TLabel *Label2;
        TButton *Button1;
        TCSpinEdit *XcoordSpinEdit;
        TCSpinEdit *YcoordSpinEdit;
        TLabel *Position;
        TLabel *Xcoords;
        TLabel *Ycoords;
        TCheckBox *PositionCheckBox;
        TCheckBox *MapImageCheckBox;
        void __fastcall FormCreate(TObject *Sender);
        void __fastcall ShowHint(TObject *Sender);
        void __fastcall FileNew(TObject *Sender);
```

```
        void __fastcall FileOpen(TObject *Sender);
        void __fastcall FileSave(TObject *Sender);
        void __fastcall FileSaveAs(TObject *Sender);
        void __fastcall FilePrint(TObject *Sender);
        void __fastcall FilePrintSetup(TObject *Sender);
        void __fastcall FileExit(TObject *Sender);
        void __fastcall HelpContents(TObject *Sender);
        void __fastcall HelpSearch(TObject *Sender);
        void __fastcall HelpHowToUse(TObject *Sender);
        void __fastcall HelpAbout(TObject *Sender);
        void __fastcall ModifyMapClick(TObject *Sender);
        void __fastcall SaveMapClick(TObject *Sender);
        void __fastcall RefreshImageClick(TObject *Sender);
        void __fastcall ShowMap(TObject *Sender);
private:        // private user declarations
public:         // public user declarations
        virtual __fastcall TMainForm(TComponent* Owner);

};
//---------------------------------------------------------------------------
extern TMainForm *MainForm;



//---------------------------------------------------------------------------
#endif
```

```
/*****************************************************************************
*       SerialThread.cpp
*       by Brian Deaton
*
*****************************************************************************/

//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "SerialThread.h"
#include "MapThread.cpp"

#pragma package(smart_init)
//---------------------------------------------------------------------------
//   Important: Methods and properties of objects in VCL can only be
//   used in a method called using Synchronize, for example:
//
//      Synchronize(UpdateCaption);
//
//   where UpdateCaption could look like:
//
//      void __fastcall TSerialThread::UpdateCaption()
//      {
//        Form1->Caption = "Updated in a thread";
//      }
//---------------------------------------------------------------------------
__fastcall TSerialThread::TSerialThread(bool CreateSuspended)
        : TThread(CreateSuspended)
{
}
//---------------------------------------------------------------------------
void __fastcall TSerialThread::Execute()
{
       DCB dcb;
        HANDLE hCom;
        DWORD dwError;
        BOOL fSuccess;
        DWORD BytesToRead=1;
        unsigned long BytesRead;
        char text[]=" :)       ";
        PacketPtr  data;
        TList* TempList;

    //These need to be changed
        short* compass_angle;
        short* sonar_dist;
        short* encoder_data;
        short* sonar_angle;

        hCom = CreateFile("COM1",
            GENERIC_READ | GENERIC_WRITE,
            0,    /* comm devices must be opened w/exclusive-access */
            NULL, /* no security attrs */
            OPEN_EXISTING, /* comm devices must use OPEN_EXISTING */
            0,    /* not overlapped I/O */
            NULL  /* hTemplate must be NULL for comm devices */
            );

        if (hCom == INVALID_HANDLE_VALUE) {
            dwError = GetLastError();
        }

        /*
         * Omit the call to SetupComm to use the default queue sizes.
         * Get the current configuration.
         */

        fSuccess = GetCommState(hCom, &dcb);
```

```
        if (!fSuccess) {
            ShowMessage("COM port cannot be opened");
        }

        /* Fill in the DCB: baud=9600, 8 data bits, no parity, 1 stop bit. */

        dcb.BaudRate = 9600;
        dcb.ByteSize = 8;
        dcb.Parity = NOPARITY;
        dcb.StopBits = ONESTOPBIT;

        fSuccess = SetCommState(hCom, &dcb);

        if (!fSuccess) {
            ShowMessage("COM port cannot be opened");
        }


//--------------------
        data = new TDataPacket;

         while(1) {

           compass_angle= new short;
           sonar_dist= new short;
           encoder_data= new short;
           sonar_angle= new short;

           if(!ReadFile(hCom, compass_angle, 2, &BytesRead, NULL)){
               ShowMessage("Cannot read COM1");
               }
           if(!ReadFile(hCom, encoder_data, 2, &BytesRead, NULL)){
               ShowMessage("Cannot read COM1");
               }
           if(!ReadFile(hCom, sonar_dist, 2, &BytesRead, NULL)){
               ShowMessage("Cannot read COM1");
               }
           if(!ReadFile(hCom, sonar_angle, 2, &BytesRead, NULL)){
               ShowMessage("Cannot read COM1");
               }


           data->compass_angle = *compass_angle;
           data->encoder_data = *encoder_data;
           data->sonar_dist = *sonar_dist;
           data->sonar_angle = *sonar_angle;


            TempList = DataList->LockList();
            try
                {
                  TempList->Add(data);
                }
            __finally
                {
                  DataList->UnlockList();
                }


           delete compass_angle;
           delete sonar_dist;
           delete encoder_data;
           delete sonar_angle;

           compass_angle=NULL;
           sonar_dist=NULL;
           encoder_data=NULL;
           sonar_angle=NULL;

            // delete TempList;   ???? really delete this????
```

```
        }  //end while

        CloseHandle(hCom);  //yes this is unreachable but windows closes
                            //all files on exit anyway
}
//---------------------------------------------------------------------------
```

```
/*****************************************************************************
*       SerialThread.h
*       by Brian Deaton
*
*****************************************************************************/

//---------------------------------------------------------------------------
#ifndef SerialThreadH
#define SerialThreadH
//---------------------------------------------------------------------------
#include <Classes.hpp>
//---------------------------------------------------------------------------
class TSerialThread : public TThread
{
private:
protected:
        void __fastcall Execute();
public:
        __fastcall TSerialThread(bool CreateSuspended);
};


//---------------------------------------------------------------------------
#endif
```

```
/*****************************************************************************
*       MapThread.cpp
*       by Brian Deaton
*
*****************************************************************************/

//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "MapThread.h"
#include "map.cpp"
#include "DataList.cpp"
#pragma package(smart_init)
//---------------------------------------------------------------------------
//   Important: Methods and properties of objects in VCL can only be
//   used in a method called using Synchronize, for example:
//
//      Synchronize(UpdateCaption);
//
//   where UpdateCaption could look like:
//
//      void __fastcall TMapThread::UpdateCaption()
//      {
//        Form1->Caption = "Updated in a thread";
//      }
//---------------------------------------------------------------------------

__fastcall TMapThread::TMapThread(bool CreateSuspended)
        : TThread(CreateSuspended)
{

}
//---------------------------------------------------------------------------
void __fastcall TMapThread::Execute()
{
        TList* TempList;
        PacketPtr  data;
        TDataPacket tempdata;
        bool NewData_Flag=0;

        while(1) {

                TempList = DataList->LockList();
                try
                    {

                      if(TempList->Count > 0){
                        data = (PacketPtr) TempList->First();
                        if(data != NULL) {
                                tempdata.sonar_dist = data->sonar_dist;
                                tempdata.compass_angle = data->compass_angle;
                                tempdata.encoder_data = data->encoder_data;
                                tempdata.sonar_angle = data->sonar_angle;
                                TempList->Remove(data);
                                NewData_Flag=1;
                          } // end if 2
                        }//end if 1
                      }   //end try
                __finally
                    {
                      DataList->UnlockList();
                    }
               //  delete TempList;   ??? do i really need to delete this???


               if(data!=NULL && NewData_Flag){
                  map->modify_map(extract_data_from_packet(tempdata), data->sonar_dist);
                  NewData_Flag=0;
```

```
                }

        }   //end while
}
//---------------------------------------------------------------------------
```

```
/*****************************************************************************
*       MapThread.h
*       by Brian Deaton
*
*****************************************************************************/

//---------------------------------------------------------------------------
#ifndef MapThreadH
#define MapThreadH
//---------------------------------------------------------------------------
#include <Classes.hpp>
//---------------------------------------------------------------------------
class TMapThread : public TThread
{
private:
protected:
        void __fastcall Execute();
public:
        __fastcall TMapThread(bool CreateSuspended);
};
//---------------------------------------------------------------------------
#endif
```

```
/*****************************************************************************
*       DataList.cpp
*       by Brian Deaton
*
*****************************************************************************/

//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "DataList.h"

double extract_data_from_packet(TDataPacket data){
        double angle;
        int x,y;

        angle = (data.compass_angle) + 90;
        while(angle > 360)
                angle-=360;
        while(angle < 0)
                angle+=360;

        angle = (PI*angle)/180;

        x=(data.encoder_data)*cos(angle);
        y=(data.encoder_data)*sin(angle);
        PositionLock->Acquire();
        map->position.x+=x;
        map->position.y+=y;
        PositionLock->Release();

    //????????
       // data.sonar_angle-=90;
        angle = angle + ((PI*data.sonar_angle)/180);
    //?????????

        MapDisplayForm->MapStatusBar->SimpleText = "Position: X=" + IntToStr(map->position.x)
                + "  Y=" + IntToStr(map->position.y) +"  Compass Angle="
                + IntToStr(data.compass_angle) + "  Encoder="
                + IntToStr(data.encoder_data) + "  Sonar Dist="
                + IntToStr(data.sonar_dist) + "  Sonar Angle="
                + IntToStr(data.sonar_angle);

        return angle;
}


//---------------------------------------------------------------------------
#pragma package(smart_init)
```

```
/*****************************************************************************
*       DataList.h
*       by Brian Deaton
*
*****************************************************************************/

//---------------------------------------------------------------------------
#ifndef DataListH
#define DataListH
//---------------------------------------------------------------------------

typedef struct ADataPacket {
short sonar_dist,compass_angle;
int encoder_data;
short sonar_angle;
}TDataPacket;
typedef TDataPacket* PacketPtr;

double extract_data_from_packet(PacketPtr data);

TThreadList *DataList = new TThreadList();


#endif
```

```
/*****************************************************************************
*       MapDisplay.cpp
*       by Brian Deaton
*
*****************************************************************************/

//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "MapDisplay.h"
#include "globals.h"

//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TMapDisplayForm *MapDisplayForm;
//---------------------------------------------------------------------------
__fastcall TMapDisplayForm::TMapDisplayForm(TComponent* Owner)
        : TForm(Owner)
{
        Image->Canvas->Lock();
        Image->Canvas->Brush->Color = 8355711;      //grey 127
        Image->Height=rows;
        Image->Width=cols;
        Image->Canvas->FillRect(Rect(0,0,rows,cols));
        Image->Canvas->Unlock();

}
//---------------------------------------------------------------------------
```

```
/*****************************************************************************
*       MapDisplay.h
*       by Brian Deaton
*
*****************************************************************************/

//---------------------------------------------------------------------------
#ifndef MapDisplayH
#define MapDisplayH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ExtCtrls.hpp>
#include <ComCtrls.hpp>
//---------------------------------------------------------------------------
class TMapDisplayForm : public TForm
{
__published:    // IDE-managed Components
        TImage *Image;
        TStatusBar *MapStatusBar;
private:        // User declarations
public:         // User declarations
        __fastcall TMapDisplayForm(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TMapDisplayForm *MapDisplayForm;
//---------------------------------------------------------------------------
#endif
```

```
/*****************************************************************************
*       map.cpp
*       by Brian Deaton
*
*****************************************************************************/

#pragma hdrstop
#include <condefs.h>
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <math.h>

#include "map.h"
#include "MapDisplay.cpp"

//---------------------------------------------------------------------------
#pragma argsused

extern TMap *map= new TMap();

TMap::TMap() {

        position.x=cols/2;
        position.y=rows/2;

        for(i=0;i<rows;i++) {
           for(j=0;j<cols;j++) {
               map[i][j]=127;
           }
        }
}

TMap::~TMap() {
}

bool TMap::modify_map(/*changed from short*/double angle, double dist) {
        //angle is measured up from positive x-axis.
        //everything in this function is done in relative coords
        //until map_plot is called

        if(angle<0 || angle>(2*PI) || dist<=0)
                return 0;

        COORDS rm,r1,r2;
        /*changed from short*/double r1_angle,r2_angle;

        r1_angle=angle + (PI/18);    //(PI/18)/*10 degrees*/;
        r2_angle=angle - (PI/18);

        get_relative_coords(rm,angle,dist);
        get_relative_coords(r1,r1_angle,dist);
        get_relative_coords(r2,r2_angle,dist);

        // Lock Globals to avoid simultaneous thread access
        MapDisplayForm->Image->Canvas->Lock();
        MapLock->Acquire();

//------The arc is on the left side of the circle:
        if(r1.x <= 0 && r2.x <= 0) {
           sweep_left(r1_angle,r2_angle,dist);
        }
//----------------------End left side

//------The arc is on the right side of the circle:
        if(r1.x >= 0 && r2.x >= 0) {
           sweep_right(r1_angle,r2_angle,dist);
        }
//----------------------End right side
```

```
//------The arc is on the middle of the circle:
        if((r1.x <= 0 && r2.x >= 0) || (r1.x >= 0 && r2.x <= 0))  {
           sweep_left(r1_angle,r2_angle,dist);
           sweep_right(r1_angle,r2_angle,dist);
        }
//----------------------End middle

        draw_arc(r1,r2,angle,dist);

        MapLock->Release();
        MapDisplayForm->Image->Canvas->Unlock();

        return 1;
}

void TMap::sweep_left(double r1_angle, double r2_angle, double dist){

        COORDS r,s1,s2,ch,cl;
        int vbar,m;

        if (r2_angle < PI/2) {

           get_relative_coords(r,r1_angle,dist);

           for(vbar=0;vbar >= r.x;vbar--) {

                get_y(s1,r1_angle,vbar);
                circle_edge_x(ch,dist,vbar);

                for(m=ch.y-1;m>=s1.y;m--) {
                       map_minus(vbar,m);
                   }
              } //end for
           } //end if

        else if (r1_angle > (3*PI)/2) {

           get_relative_coords(r,r2_angle,dist);

           for(vbar=0;vbar >= r.x;vbar--) {

                get_y(s2,r2_angle,vbar);
                circle_edge_x(ch,dist,vbar);
                cl.y=-ch.y;

                for(m=s2.y;m>cl.y;m--) {
                        map_minus(vbar,m);
                   }
              } //end for

           } //end else if

        else {
           for(vbar=0;vbar >= -dist;vbar--) {

                get_y(s1,r1_angle,vbar);
                get_y(s2,r2_angle,vbar);
                circle_edge_x(ch,dist,vbar);
                cl.y=-ch.y;
                cl.x=ch.x;

                if((ch.y > s2.y) && (s1.y > cl.y)) {
                   for(m=s2.y;m>=s1.y;m--) {
                      map_minus(vbar,m);
                       }
                   }


                 else if((s2.y >= ch.y) && (s1.y >= cl.y)) {
                   for(m=ch.y-1;m>=s1.y;m--) {
```

```
                        map_minus(vbar,m);
                        }
                    }


                else if((ch.y >= s2.y) && (s1.y <= cl.y)) {
                    for(m=s2.y;m>cl.y;m--) {
                        map_minus(vbar,m);
                        }
                    }

                else if((s2.y > ch.y) && (s1.y < cl.y)) {
                    for(m=ch.y-2;m>cl.y+1;m--) {
                        map_minus(vbar,m);
                        }
                    }
                } //end for
            } //end else
}

void TMap::sweep_right(double r1_angle, double r2_angle, double dist){

        COORDS r,s1,s2,ch,cl;
        int vbar,m;

        if ((r1_angle > PI/2) && (r1_angle < (3*PI)/2)) {

            get_relative_coords(r,r2_angle,dist);

            for(vbar=1;vbar <= r.x;vbar++) {

                get_y(s2,r2_angle,vbar);
                circle_edge_x(ch,dist,vbar);

                for(m=ch.y-1;m>=s2.y;m--) {
                        map_minus(vbar,m);
                    }
                } //end for
            } //end if

        else if ((r2_angle < (3*PI)/2) && (r2_angle > PI/2)){

            get_relative_coords(r,r1_angle,dist);

            for(vbar=1;vbar <= r.x;vbar++) {

                get_y(s1,r1_angle,vbar);
                circle_edge_x(ch,dist,vbar);
                cl.y=-ch.y;

                for(m=s1.y;m>cl.y;m--) {
                        map_minus(vbar,m);
                    }
                } //end for

            } //end else if

        else{
            for(vbar=0;vbar <= dist;vbar++) {

                get_y(s1,r1_angle,vbar);
                get_y(s2,r2_angle,vbar);
                circle_edge_x(ch,dist,vbar);
                cl.y=-ch.y;
                cl.x=ch.x;

                if((ch.y > s1.y) && (s2.y > cl.y)) {
                    for(m=s1.y;m>=s2.y;m--) {
                        map_minus(vbar,m);
                        }
```

```
                    }


                else if((s1.y >= ch.y) && (s2.y >= cl.y)) {
                   for(m=ch.y-1;m>=s2.y;m--) {
                       map_minus(vbar,m);
                       }
                   }


                else if((ch.y >= s1.y) && (s2.y <= cl.y)) {
                    for(m=s1.y;m>cl.y;m--) {
                        map_minus(vbar,m);
                       }
                   }

                else if((s1.y > ch.y) && (s2.y < cl.y)) {
                    for(m=ch.y-2;m>cl.y+1;m--) {
                        map_minus(vbar,m);
                       }
                   }
                } //end for
            } //end else

}
void TMap::get_relative_coords(COORDS &point,/*changed from short*/double angle,double dist) {
        point.x=(dist*cos(angle));
        point.y=(dist*sin(angle));
}

void TMap::get_absolute_coords(COORDS &point,/*changed from short*/double angle,double dist) {
        point.x=position.x*(dist*cos(angle));
        point.y=position.y*(dist*sin(angle));
}

void TMap::get_y(COORDS &point, /*changed from short*/double angle, double x_coord) {
        point.y=x_coord*tan(angle);
}

void TMap::circle_edge_x(COORDS &point, double dist, double x_coord) {
        double angle=acos(x_coord/dist);
        point.y=dist*sin(angle);
        point.x=x_coord;
}

void TMap::circle_edge_y(COORDS &point, double dist, double y_coord) {
        double angle=asin(y_coord/dist);
        point.x=dist*cos(angle);
        point.y=y_coord;
}

void TMap::map_minus(int x_coord, int y_coord) {
        unsigned int x=x_coord + position.x;
        unsigned int y=y_coord + position.y;
        if(y<=rows && x<=cols){
           if(map[y][x] > 231) {
                map[y][x]=255;
                MapDisplayForm->Image->Canvas->Pixels[x][rows-y] = 65793*map[y][x];
           }
           else {
                   map[y][x]= map[y][x]*1.1;
                   MapDisplayForm->Image->Canvas->Pixels[x][rows-y] = 65793*map[y][x];
           }
        }
}

void TMap::map_plus(int x_coord, int y_coord) {
        unsigned int x=x_coord + position.x;
        unsigned int y=y_coord + position.y;
        if(y<=rows && x<=cols){
```

```
                map[y][x]= map[y][x]*0.9;
                MapDisplayForm->Image->Canvas->Pixels[x][rows-y] = 65793*map[y][x];
        }
}

void TMap::draw_arc(COORDS r1, COORDS r2, double angle, double dist) {
        COORDS ch;
        int vbar,hbar;

           if(abs(r2.x-r1.x)>=abs(r2.y-r1.y)) {
                for(vbar=(r2.x>=r1.x ? r2.x : r1.x);vbar>=(r2.x>=r1.x ? r1.x : r2.x);vbar--){
                        circle_edge_x(ch,dist,vbar);
                        ch.y=(sin(angle)>=0?1:-1)*ch.y;
                        if(ch.x<=(r2.x>=r1.x ? r2.x : r1.x) && ch.x>=(r2.x>=r1.x ? r1.x : r2.x)){
                                if(sin(angle)>=0) {
                                        map_plus(vbar,ch.y);
                                        map_plus(vbar,ch.y+1);
                                  }
                                else {
                                        map_plus(vbar,ch.y);
                                        map_plus(vbar,ch.y-1);
                                }

                        } //end if
                } //end for
           }
           else {
                for(hbar=(r2.y>=r1.y ? r2.y : r1.y);hbar>=(r2.y>=r1.y ? r1.y : r2.y);hbar--){
                        circle_edge_y(ch,dist,hbar);
                        ch.x=(cos(angle)>=0?1:-1)*ch.x;
                        if(ch.y<=(r2.y>=r1.y ? r2.y : r1.y) && ch.y>=(r2.y>=r1.y ? r1.y : r2.y)){
                                if(cos(angle)>=0) {
                                        map_plus(ch.x+1,hbar);
                                        map_plus(ch.x+2,hbar);
                                }
                                else {
                                        map_plus(ch.x-1,hbar);
                                        map_plus(ch.x-2,hbar);
                                }

                        }
                }
           }
}

bool TMap::dump_map_to_file(AnsiString filename) {

        long r,c;
        ofstream outfile(filename.c_str(), ios::out);

        if (!outfile) {
                cerr << "Map output file could not be opened" << endl;
                return false;
        }

        outfile << "P5" << endl
                <<  cols << ' ' << rows << endl
                << "255" << endl;

        for(r=rows-1;r>=0;r--) {
           for(c=0;c<cols;c++) {
               outfile << map[r][c];
           }
        }
        outfile.close();

        return true;
}

char TMap::get_map_value(unsigned int x, unsigned int y) {
```

```
        return map[y][x];
}
```

```
/*****************************************************************************
*       map.h
*       by Brian Deaton
*
*****************************************************************************/

#ifndef MAP_H
#define MAP_H

#include "globals.h"

struct xycoords {
        int x,y;      //!! used to be unsigned int
};
typedef struct xycoords COORDS;


class TMap {

public:

        short sonar_dist,compass_angle;
        unsigned int encoder_data;
        double sonar_angle;

        COORDS position;

        TMap();
        ~TMap();
        bool modify_map(/*changed from short*/double angle, double dist);
        bool dump_map_to_file(AnsiString filename);
        char get_map_value(unsigned int x, unsigned int y);

private:
        unsigned char map[rows][cols];
        unsigned int i,j;
        void get_relative_coords(COORDS &point,/*changed from short*/double angle,double dist);
        void get_absolute_coords(COORDS &point,/*changed from short*/double angle,double dist);
        void get_y(COORDS &point, /*changed from short*/double angle, double x_coord);
        void circle_edge_x(COORDS &point, double dist, double x_coord);
        void circle_edge_y(COORDS &point, double dist, double y_coord);
        void map_minus(int x_coord, int y_coord);
        void map_plus(int x_coord, int y_coord);
        void draw_arc(COORDS r1, COORDS r2, double angle, double dist);
        void sweep_left(double r1_angle,double r2_angle,double dist);
        void sweep_right(double r1_angle, double r2_angle, double dist);

};

#endif
```

```
/*****************************************************************************
*       globals.h
*       by Brian Deaton
*
*****************************************************************************/

#ifndef GLOBALS_H
#define GLOBALS_H

#include <SyncObjs.hpp>

extern TCriticalSection* MapLock = new TCriticalSection();
extern TCriticalSection* PositionLock = new TCriticalSection();


#define PI 3.14159265358979

extern const unsigned int rows=500;
extern const unsigned int cols=500;

#endif
```

Memorandum

To:	Terrain Mapper Group (Brian Deaton, Jeff Dickerson, Jared Newton)
From:	Dr. Kevin Nickels, Senior Design Group Administrator
Date:	May 9, 2000
Subject:	End of Semester Senior Design Comments/Evaluation

Evaluation

1. If I believe the report, the project can do everything (together) except move the sonar? If that's true (I don't think that it is), you should show some actual maps generated by moving the robot around a room with a fixed sonar mast. If it's not, you should give an accurate status of the project. If you put your names (and mine) on a report, it should be *accurate* and *complete*!!

2. My knowledge of the current state of the project is that each individual portion (except possibly the intelligent control of the sonar mast...I don't think I've even seen an algorithm for that) works in isolation, but that the integral product hasn't been shown to work. I've seen the following demonstrations:

   (a) map generator using manual data input.

   (b) sonar mast movement

   (c) sonar measurement (output to HB screen)

   (d) encoder counting (forward only, output to HB screen)

   I've not seen the following demonstrations, that might have been expected:

   (a) compass readings (output to HB screen) / I believe that this has been done, I just haven't seen it.

   (b) Intelligent control of sonar mast.

   (c) compass & encoder readings (output to HB screen)

   (d) compass & encoder readings (output to laptop screen)

   (e) map generator using HB inputs – final product!

3. Based on the incomplete project and the inattention to scheduling (at the very least, for the reports), I'm assigning the following grades:

Name	Project Grade	Report Grade	Overall Grade
Brian	C		75 (C)
Jeff	C-	C	70 (C-)
Jared	B		82 (B-)

Report Comments

1. Organization
   (a) There should be a paragraph at the end of §1 describing the organization of the rest of the report, and explaining how to read it.
   (b) §2 purports to describe the base components of the system, then starts to discuss the decisions involved. There's no clear tie-in to the report. If you're trying to address the analysis you chose to do, why not say that? It seems like this section is trying to do **two** things.
   (c) §3 is well organized, and seems to go through the hardware and software needed to understand the position tracker.
   (d) §4 organization looks ok, TOC is poorly formatted.
   (e) §5, §6 seem (from the TOC) like they could be combined... what's the difference between software that happens to be on the laptop, and software that generates the map?
   (f) App B - isn't this the final budget? Not a status report?
   (g) §8 - references normally go before appendices.
2. §1 - Introduction
3. §2 - Base Components
   (a) will use the components? future tense?
   (b) Table X ?
4. §3 - Position Tracking System
   (a) 3.2 Odometery – a 5% wheel slip won't cause **your** design to think that it's turned.
   (b) 3.6 - There are *a lot* of assumptions sprinkled through here. A list or table would help the reader understand where your method is applicable. Also, an explicit description (pseudo-code, maybe) would make the method more understandable.

5. §4 - Object Detection
6. §5 - Laptop Software
7. §6 - Map Generation
   (a) Seems like this should be a sub-section of §5.
8. §7 - Conclusion
   (a) You can't conclude anything about your methods for solving odometry, or interfacing differing applications/hardware, mixing off-the-shelf hardware with custom applications, mixing "public" software applications/modules with custom code, or system-level engineering design?
9. Appendix A - Engineering Design Process
   Isn't this last semesters?
10. Appendix B - Budget Status
11. Appendix C - Project Overview
12. Appendix D - Contributions
13. Appendix E - Coding
    This isn't in the TOC? Will there be another appendix for the other software?
14. §8 - References

cc: Dr. Paul Giolma, Senior Design Adminstrator