Final Report

EE 382 – Junior Design

Golfing Robot

Team B Navigation

May 6, 2002

Dr. William Rison
Dr. Kevin Wedeward

Brian England
Andreas Garcia
Kristin Herlugson
Chris Montaño

Abstract

The following report documents the Navigation subsystem of a golfing robot. The golfing robot is to use the Global Positioning System (GPS) coordinates of a golf ball and hole, move to the ball, transport the ball to the hole, and place the ball in the hole. The Navigation subsystem is responsible for determining the robot's current position and calculating a course to the position of the ball and the hole. This report will allow for other engineers to reconstruct and troubleshoot the Navigation subsystem design.

Table of Contents

ABSTRACT **2**

TABLE OF CONTENTS **3**

LIST OF FIGURES AND TABLES **4**

INTRODUCTION **5**

NAVIGATION ALGORITHM **6**

HARDWARE **8**

- HC12 HARDWARE 8
- GPS HARDWARE 9
- COMPASS 11
- ETCHED CIRCUIT BOARDS 12
- ENCLOSURES 13

SOFTWARE **15**

- SPI COMMUNICATION 15
- SCI COMMUNICATIONS 16
- GPS PARSING 17
- COMPASS 19
- TILT CORRECTION 20
- MATH FUNCTIONS 21

CONCLUSION **23**

APPENDIX A **24**

APPENDIX B: PIN OUTS & WIRING DIAGRAM **31**

APPENDIX C: BUDGET **34**

APPENDIX D: NAVIGATION SOURCE CODE **35**

List of Figures and Tables

Figure 1: Navigation algorithm flow chart 27
Figure 2: Compass etched board 27
Figure 3: Signal/power distribution etched board 27
Figure 4: Use of Pythagorean Theorem in distance calculation 28
Figure 5a: Distance equations 29
Figure 5b: Distance equations 29
Figure 6: Course calculations 29
Figure 7: Inverse tangent approximation 30
Figure 8: Navigation system wiring diagram 33

Table 1: Vector2X pin assignments 25
Table 2: Vector2X output formats 26
Table 3: Vector2X connections for Navigation application 26
Table 4: GGA sentence structure 28
Table 5: Quadrant assignments 30
Table 6: System pin outs 31
Table 7: Subsystem pin outs 32
Table 8: Navigation final budget 34

Introduction

The purpose of this report is to document our design of a Navigation system for a golfing robot as part of our junior design project so that other engineers may reconstruct and troubleshoot our subsystem. The robot, as a whole, was to find a ball and hole given their locations as determined by a hand-held GPS receiver. The robot must pick up and transport the golf ball to the hole and place the ball in the hole. Each robot has been divided into four subsections; Ball and Hole Location, Navigation, Wireless Communication, and Smart Chassis.

The Navigation subsystem is responsible for determining the location of the robot, relaying the location to a Wireless Communication link, calculating a course to the given location of the golf ball and hole, and sending that course to the chassis. We will provide block diagrams of how to integrate our hardware, as well as the type of data sent between each component. We will provide full circuit diagrams to allow for simple reconstruction of our design. We will also discuss, and breakdown, our final budget.

Navigation Algorithm

A large part of the Navigation subsystem was creating the navigation algorithm (Figure 1); the steps the code would go through to achieve the requirements placed upon the subsystem. It was determined that the Navigation subsystem would be the primary communications center for the robot. The other subsystems; Ball/Hole, Chassis and Wireless Communication, as well as the digital compass are all connected to the navigation subsystem's "hub." The navigation algorithm must work around the limitation that only one slave can communicate at a time. The details of each action will be discussed in detail later, this is an outline of the entire Navigation subsystem.

The first task the Navigation subsystem is responsible for is receiving the longitude and latitude of the ball and the hole from the Wireless Communication subsystem. Once that information is received, the compass is calibrated. Then the HC12 receives the current location of the robot from the GPS receiver, as well as the digital compass. The position coordinates are used to calculate a course, angle and distance, to the ball. The HC12 then sends four bytes to the chassis; distance, a one or zero, an angle byte of less than 255 degrees, and a C4 in hexadecimal. The chassis group asked for the distance to be sent in groups of three centimeters, so the maximum distance it can move is 7.68 meters at a time. If the calculated course produces a distance greater than that, the distance sent to move is 7.68 meters. One byte of data is not enough to interpret a angle of over 256 degrees, so a second byte must be sent to convey to add 256 degrees to the turn angle or not. This byte is the second byte of either a one or zero. The C4 is a check byte that was requested by the chassis group to determine if the set of four bytes received were an actual stream of data.

While the chassis is moving the distance and angle sent, the navigation HC12 sends the robot's current location to the Wireless Communication subsystem at a rate of one transmission every second. The Wireless Communication subsystem is then able to update its graphical interface of the robot's status. When the chassis moves the distance and angle sent to it by Navigation, it sets a completion pin high. When that pin goes high, the robot's current location is compared to the location of the ball again.

The Ball/Hole group requested that the robot be within five meters of the ball or hole before it was initialized. If the robot is not within five meters of the ball, another course is calculated and relayed to the chassis. If the robot is within five meters of the ball, the Ball/Hole group is initiated and their movement commands are passed through the navigation HC12 to the chassis. Once the ball/hole group has located the ball and the chassis has retrieved it, a status bit is sent to the Wireless Communication subsystem. The Navigation HC12 then calculates a course to the hole in the same manner the ball was located.

It was required that there be an emergency stop command for the robot as a precautionary measure. This command is sent by Wireless Communication to Navigation, and then relayed to the Chassis.

Hardware

The hardware of the navigation subsystem consists of a GPS receiver and a digital compass to determine position and heading, an HC12 microcontroller to determine course and control communications with the other robot subsystems, a signal and power distribution, and the enclosures for all of the above. Hardware selection was determined by the requirements placed on the navigation subsystem by the other subsystems and by the component's ability to interface with the provided HC12 microcontroller. We will discuss these requirements, as well as the attributes of each hardware component.

HC12 Hardware

A Motorola MC68HC12 microcontroller along with a memory expansion board is used to perform all the necessary functions. The software and wiring of the HC12 was originally designed for the Byte-Erasable EEPROM. After testing the software in this memory area, many problems arose. The alternative chosen was to use the memory expansion to load the program and use the Expanded Ports A and B on the board. The code was easily modified by altering the HC12.h file by changing the address of the PORTA and PORTB to the expanded PORTA and PORTB. The expansion ports do not output enough voltage for the compass, so the compass had to be rewired to its current state and the code had to be altered slightly.

On the HC12, both the SCI and SPI subsystems are utilized. The SCI is used for communication with the GPS as well as the computer for debugging purposes. The SPI subsystem is used for all other communications throughout the robot. Extensive testing found the SPI subsystem to be unreliable. A separate “communications” HC12 would be a good idea in future systems.

PORTP is used as an auxiliary general purpose I/0 for the compass interface. PORTA is used for the slave selects among the other modules, and Port B is used for status lines from the other modules.

The system also utilizes the input capture port for the stop command. The command is set up as an interrupt to break all communications with other subsystems so the Chassis can me notified immediately.

GPS Hardware

For this project, the coordinates of ball and hole are given as their longitude and latitude positions as determined by a handheld GPS receiver. To create a course to the ball or hole the current location of the robot is also needed. It was decided that a GPS mounted receiver directly on the robot would be used to determine the location of the robot. There are several options in choosing a GPS receiver and what type of GPS would be most useful for the project. There are three types of GPS; standard, differential, and Wide Area Augmentation System (WAAS). Standard GPS receivers provide locations with errors of up to ten to fifteen meters, while differential and WAAS enable receivers reduce this error to around three meters. The Ball and Hole group for Robot B asked to be put within five meters of the ball or hole before they were initialized, therefore a differential or WAAS enabled receiver was needed. Using a differential GPS receiver would require more hardware for the navigation group and more software for Wireless Communication so a WAAS enabled receiver was the desirable choice.

The receiver that was finally decided on was the Garmin GPS 16 WAAS enabled receiver. A major selling point of the Garmin 16 was that it was serially interfaced, therefore it would be easy to integrate with our chosen controller, the HC12 microcontroller. The receiver had a reported accuracy of less than three meters, and after testing found to be actually less than three meters. When the Garmin 16 receiver transmits sixteen strings of data in the format known as NMEA 0183 format, as determined by the National Marine Electronics Association. Only one string of this NMEA data was needed to determine the location of the robot. The parsing of the data is discussed later. Another useful feature of the Garmin 16 was that it has a built in antenna, eliminating the need to purchase and mount an additional antenna. Because the GPS receiver is to be mounted on a chassis with motors and batteries it is essential that the receiver be EMI shielded, another feature of the Garmin 16 receiver. It is also encased in a rugged, waterproof enclosure, so an enclosure did not need to be built for it.

To initialize the receiver the online manual provided by Garmin was very useful. The receiver outputs data in true RS-232 format. When testing, it was noticed that the GPS's idle state was low. The HC12's SCI system requires an idle high serial bit stream. By implementing an inverter the data from the GPS was converted to a format readable by the HC12's SCI system.

The Garmin 16 has a five-meter cord with an eight-conductor foil shielded RJ-45 termination, the connector most commonly used in networking. By going through the online manual, it was determined that only four of these connections were needed for this project. The connectors were labeled and connected as follows:

1. Pin 1 – Power
2. Pin 2 – Ground
3. Pin 3 – Remote Power to Ground
4. Pin 5 – Port 1 NMEA 0183 Output Data

The other pins were not needed for this project and they were left unconnected. The receiver is a low power receiver. This was desirable because all of the required power of the subsystem, as well as all of the other subsystems, that are on a mobile chassis which relies on batteries for power.

Visual GPS was a useful software tool in testing the accuracy of the GPS receiver and gathering experimental data for verifying the Navigation math functions.

Compass

In order to have the robot know which way it is pointing, some kind of compass must be implemented in the design. A Vector 2X digital compass from Precision Navigation was used. It is a two-axis compass that has magnetic sensor modules and is designed for original equipment manufacturers (OEM) applications. This particular model is very accurate, takes very little power to run, and is very cost efficient. Other models were considered but because of their high cost, the Vector 2X was the final compass decided on. For this particular application, it provided more than enough accuracy. A limitation on the Vector 2X is its inaccuracy when subjected to tilt. This will be discussed in greater detail later.

Etched Circuit Boards

Compass Board

The compass that was chosen, the Vector 2X, did not come in a complete package, therefore, an etched board was made to interface with the compass (Figure 2). It was easy to interface with the compass because of the use of an RJ45 plug, utilizing CAT 5 cable. Each compass pin is connected to a bus line on the right side of the board. On this bus, is the connection to the RJ45 socket and from there, a plug with CAT 5 cable, could easily be inserted.

Signal/Power Distribution Board

The function of the second etched board was to distribute power, data, and the clock to the other three subsystems (Figure 3). Also, an inverter was implemented to take the data from the GPS and invert it to useful information. . The power bus is protected by a 1-amp slow blow fuse and distributes power to the HC12, the compass, the GPS receiver and the inverter to covert the GPS data. Busses for the Master In Slave Out, Master Out Slave In, and clock are connected in the manner that they are because communications between subsystems is via the HC12's Serial Peripheral Interface (SPI). Each individual subsystem that is connected to the navigation subsystem, receives the same data lines, clock lines, and slave select lines used to designate which subsystem is selected. The Ball/Hole subsystem required that it have access to the objective complete flag from Chassis, so another bus on this board is used to connect the flag to both the Ball/Hole and Navigation subsystems.

Enclosures

The central HC12 is enclosed in a plexiglass frame. The frame is constructed of three plexiglass sheets and four brackets held together with nuts and bolts. The frame features integrated mounting hardware for attaching the frame to the chassis via four bolts that extend past the bottom of the frame and bolt to a flat surface provided by the chassis group.

Along with the HC12 in the frame is the etched board that provides busses for power, ground and signal distribution for the navigation. All boards are mounted with standoffs and all wires are either labeled or entered into a matrix organized by color-coding (Tables 6 & 7).

A communications hub is mounted to the front of the frame. The hub consists of six color-coded CAT5 RJ-45 jacks, which snap into a frame and are covered by a wall plate. Connections between the hub, the HC12, and busses are made with eight-conductor, twisted-pair, CAT5 cable with headers soldered onto each wire.

Compass

Due to the fact that the Vector 2X is susceptible to the elements and is not equipped with integrated protection, it is mounted in a sealed enclosure. The compass board is mounted with standoffs inside of a Rubbermaid 1.4 qt Servin' Saver® Round container. Two CAT5 cables run through the side of the enclosure and are kept in place by strain reliefs. In one cable, two conductors are used for power and ground while the other six are left unconnected. All eight conductors in the second cable are used for signal lines to the compass board. The enclosure is mounted using a nut and bolt, positioned in the center of the round container, which allows for aligning the compass with the front of the robot.

GPS

The GPS requires no special enclosure, and is mounted flush by utilizing the three threaded inserts integrated into the GPS receiver.

Software

Due to the Navigation subsystem being the “brain” of the golfing robot, software was the largest concern of the design. First, a navigation algorithm was mapped out. Then, each component of the Navigation subsystem was approached separately and integrated into the algorithm, as each proved functional. Outside of the entire navigation algorithm, software includes SPI communication, SCI communication, digital compass and GPS receiver integration, and math algorithms to create a course.

SPI communication

The HC12’s integrated Serial Peripheral Interface (SPI) is used to retrieve data from the digital compass and to communicate with the other subsystems. The navigation HC12 is setup in master mode and all other systems, including the compass, are configured as slaves. Each slave is given a separate slave select line (SS), which is active low. Pins from PORTA are used to implement these slave selects. Due to the unique communication requirements of this robot, there are times when a slave must initiate a data transfer. For example, when the robot is powered up, the Navigation subsystem waits for the Wireless Communications subsystem to send initial instructions. To allow the slave subsystems to indicate when they are ready to transmit data, ready lines are designated for the Ball/Hole and Communications subsystems on PORTB. The speed of the SPI system is determined by the maximum recommended speed of the digital compass, which is 1Mhz for the Vector2X.

SCI communications

Primarily, the HC12's on-board Serial Communications Interface (SCI) is used to connect to a PC running a terminal program. In this application, the SCI port is also used to communicate with the GPS receiver. Communicating with more than one system at a time could pose a problem because both could try and transmit at the same time. A method was found for using a single SCI port to communicate with both a PC and the GPS receiver. When a computer is connected to the SCI port and a key is pressed, a byte is transferred to the HC12 through the SCI receive data pin (PORTSC0). This is the same pin that must be connected to the GPS data out line for communications with the GPS receiver. As soon as the GPS is powered up, it starts to stream data on its output line. This means that before the GPS is connected to the HC12, the program must be loaded and started, otherwise the streaming data from the GPS receiver will appear as keystrokes at the DBug12 prompt. Once the program has started, it is possible to connect the GPS receiver's output line to PORTS0 and read the streaming GPS data as a series of normal SCI transfers, one byte at a time. This procedure requires that no further keystrokes be made after the GPS receiver is connected to PORTS0, so as to avoid interfering with the SCI data transfers. The data string that is utilized is enabled by default; therefore, the GPS receiver can be left in its default mode. As a result, the GPS receiver's input line can be left disconnected since the configuration settings are not being changed. Not connecting the GPS receivers input line to the HC12's SCI transmitted data line (PORTS1) allows a computer's serial input line to remain connected to PORTS1. In short, it was discovered that connecting a computer to the HC12's serial connector, loading the program, starting the program, and then connecting the GPS receiver to PORTS0, streaming GPS data can be read through the SCI system while still using "printf" statements to display information on a computer, all on one SCI port.

The GPS receiver defaults to a 4800bps transmission rate. As no changes are being made to the default configuration, the GPS parsing algorithm must set the baud rate to 4800bps for reading the GPS sentences. The HC12 will then read and transmit at 4800bps over the SCI port. DBug12 initializes the SCI system to 9600 baud, so code must be loaded and then started with the terminal program set to communicate at 9600bps. In order for "printf" statements to function properly when using the GPS receiver and the computer on the same SCI port, the terminal settings must be changed to 4800bps after the program is started.

This method was chosen to avoid adding a UART chip to the HC12. A UART chip adds another SCI port to the HC12. Adding another SCI port could possibly aid in debugging, however, no configuration changes are made to the GPS receiver, and keystrokes will not need to be entered after the code is started. Thus, not only is a UART unnecessary, its complexity (as seen by the groups that encountered problems implementing one) overshadows any potential usefulness.

GPS Parsing

When powered on, the GPS receiver streams data at a constant interval dependent on how the receiver is configured. Using the HC12's SCI system, the transmitted sentences are received and then parsed using an algorithm designed to search for the correct sentence and extract the data of interest.

The NMEA 1803 standard dictates that data be transmitted in the form of sentences comprised of printable ASCII characters. Each sentence begins with a ‘$’, followed by a two letter talker ID, a three letter sentence ID, and the comma delimited data strings. The delimiting commas are sent regardless of the omission of a data field. An optional checksum may follow the data fields, which includes an ‘*’ character followed by two hex digits. The sentence is terminated by a carriage return and a line feed.

The Global Positioning System Fix Data (GGA) sentence is enabled by default and contains the appropriate latitude and longitude information for input into the math functions. The format of the ‘GGA’ sentence is shown in Table 4. The latitude and longitude data fields are sent including leading-zeros, which maintains a constant string length for these fields. The ‘GGA’ sentence is unique among the default sentences in that its name is the only one that contains a ‘GA’ string of characters. The parsing algorithm searches for a ‘G’ followed by an ‘A’. When a ‘GA’ string is found, it is known that the sentence currently being transmitted is the desired sentence. Taking advantage of the comma delimiters and the constant structure of the data fields, the algorithm starts counting commas after detecting a ‘GA’.

Once two commas are detected, the next nine bytes are stored into an array. These nine bytes represent the latitude as a string of characters. After the ninth byte of the latitude data field is stored, two more commas are counted. The next ten bytes are stored in another array representing the longitude.

After the current latitude and longitude information is acquired in character string format, it must be converted to floating point values of seconds. The reasoning behind ignoring the minutes is addressed in the discussion of the math function.

Compass

Compass Calibration

Due to the fact that magnetic fields can disrupt the calculation of a heading by the compass, it needs to be calibrated. By calibrating, the static magnetic fields in the host system can be compensated for. Calibration cannot compensate for any magnetic field that is not part of the host system.

There are certain steps, as well as certain pin settings that need to be followed in order to calibrate the compass:

1. *Make sure P`/C is set high*
2. *Set the CAL low for 10ms*
3. *After 10ms set CAL back high*
4. *Rotate the compass by 180 degrees*
5. *Set the CAL low for 10ms*
6. *After 10ms set CAL back high*

To check that the steps are being followed by the compass, the CI pin will be set high after the first low/high pulse of the CAL pin and the will be set back low after the second low/high pulse of the CAL pin. If there is a need to abort the calibration or to clear a previous calibration, perform the low/high pulse of the CAL pin twice without moving the compass. Upon completing the steps for calibration, the compass will be able to take a reading without interference from the host system.

Retrieve reading

As in calibration of the compass, there are certain steps and certain pin settings that need to be followed in order to take a heading reading. Timing issues are a concern when dealing with taking a reading. For example, if a pin needs to be toggled, the time that it is kept low or high will determine if the reading is taken correctly. The following steps must be taken:

1. *Set P`/C low*
2. *After 10ms, EOC will drop low, indicating that one reading has been taken*
3. *After EOC is dropped low, set P`/C back high*
4. *After 80-100ms of going low, EOC will be set back indicating that the data can be retrieved*
5. *Wait 10ms after EOC goes high, then drop SS low. Once this is done, data will begin clocking to the HC12*
6. *It takes 16 rising edges to get all the data so after those 16 rising edges, set SS back high.*
7. *The data should now have been sent to the HC12*

Tilt Correction

The Vector 2X compass has two degrees of error in every degree the compass is tilted. Because the robot was to be designed to navigate on the flat chipping green of the New Mexico Tech Golf course, tilt was determined not to be a factor. After observing the actual green, it was decided that a tilt of maybe up to ten degrees could occur. Using trigonometry it can be shown that if the robot moves the maximum distance of 7.68 meters and it is tilted ten degrees, the error in the angle calculations would cause a shift of 2.68 meters to the left or the right of the desired location. This distance is still within the parameters of error of the Garmin 16 receiver so it could not be distinguished from the greater error in the receiver. Also, if tilt did cause an error in the calculation of the distance, when the chassis moved the erroneous distance the location would be checked again and another course calculated. The chances of the robot stopping on the same degree of tilt, therefore possibly compounding the error, is so slim it was determined the extra time and money to implement tilt-correction would be wasted

Math Functions

The math functions used for this application were broken down into two parts: Distance and Course functions. The methods used in the functions were not the first chosen. The initial choice was to use either the Haversine Formulas or the Inverse Vincenity/Sadano equations. There were two problems with these methods; the compiler and/or HC12 had problems calculating trigonometric and square root functions, and because the calculated numbers were so small, rounding and decimal drop off caused calculation errors. To cope with the compiler/HC12 not calculating the trigonometric and square root function properly, Taylor series (Figure 7) and Newton-Raphson (Figure 5b) methods were introduced to calculate the appropriate functions. Power and factorial functions were created to make the approximations as accurate as possible. Testing on these new functions caused even further problems in that the compiler would not allow floating point variables to be passed to functions.

This leads to the current functions used. Assuming the robot travels less than 100 meters, one can assume the earth is flat and only the seconds of arc change. This allows the use of approximations without function calls and much simpler mathematics.

The limitations that were found were not to be of the actual calculations, but of the formulas used. Another limitation on the current functions is the assumption that only seconds of latitude and longitude are going to change. The GPS outputs latitude/longitude in DDDMM.mmmm (D represents degrees, M represents minutes, m represents fractions of minutes). The functions only use the fractions of minutes and assume the robot will never cross over a minute line. This limitation could be corrected by adding the rest of the output string in, but was not needed for the specific application.

Distance Function

The distance function uses the Pythagorean theorem to calculate distance (Figures 4 & 5a). The difference in longitude and latitude between the current position and desired destination are used as the sides of a triangle with the course being the hypotenuse between the two. This assumes the earth is flat, an assumption that can be made when calculating a course of short distances. Due to problems with the math libraries, the Newton-Raphson Method was used to find the square root of the argument.

Course Function

The Course function has a dual purpose. It first calculates the true course from north given two latitude and longitude points. Secondly, it calculates the turn angle given the current heading from north.

To simplify calculations, the true course (β) was calculated in the first quadrant. Then, depending on the values of Δlat and Δlon, two different Taylor series approximations were used and a certain modification was done to get final true course (Table 5). Turn angle (θ) is defined as the difference between true course and the heading from north (α), which is acquired by the compass (Figure 6).

Conclusion

At this time, SPI communications between subsystems is not fully functional and requires debugging. Communications have, however, been achieved in test situations consisting of the Navigation module communicating with a single subsystem. Due to the issues with communications, the Navigation algorithm has not been thoroughly tested.

The Navigation module performs all other required tasks, including determining the current robot location, orientation, and a course to the ball or hole. Using a test program that allows the Navigation subsystem to operate independent of the other subsystems, the module can get data from the GPS receiver and digital compass, and calculate a course to a predetermined location.

With a little more debugging time, this module would become fully integrated with the other robot subsystems, and utilize its current navigation abilities.

Appendix A

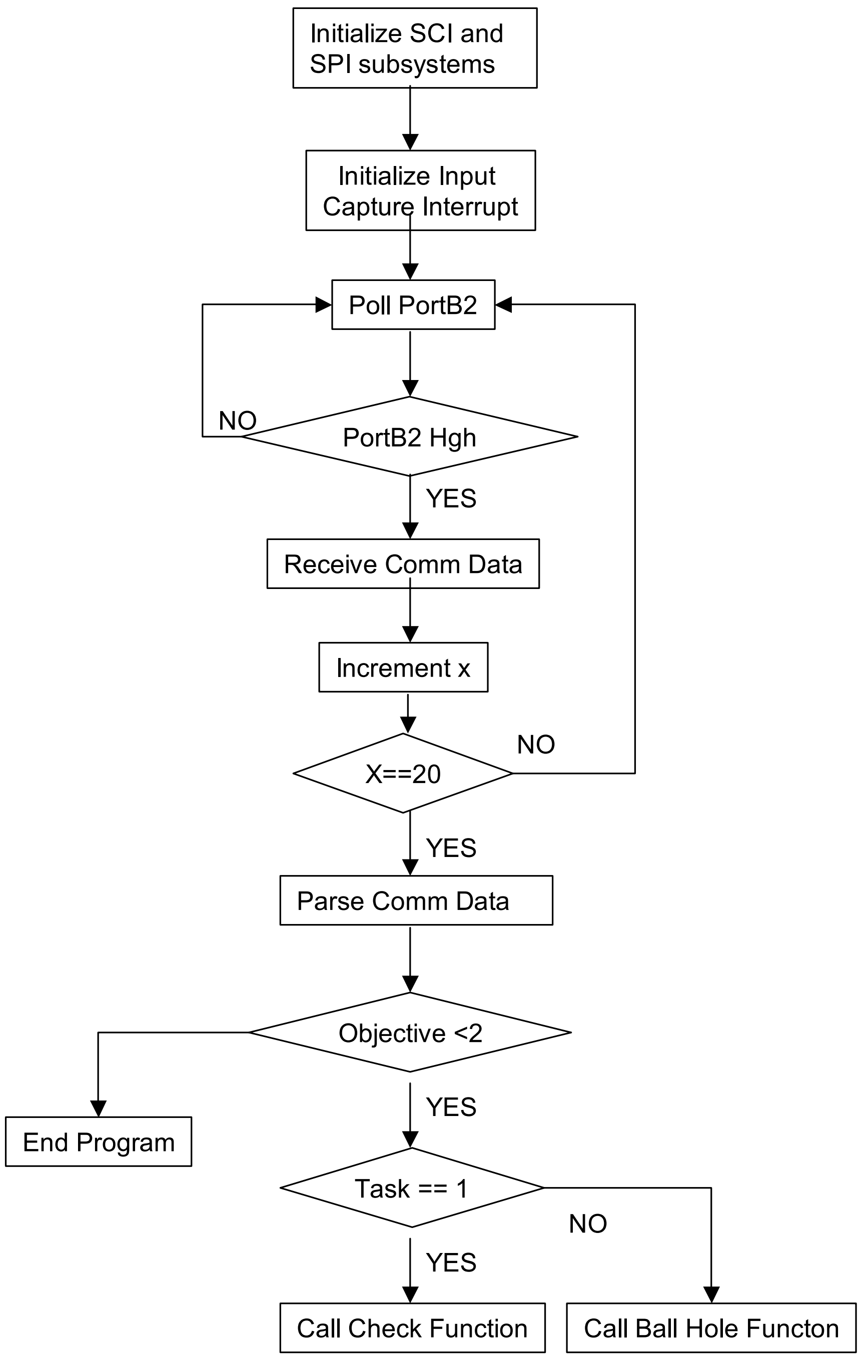


Figure 1: Navigation algorithm flow chart

PIN	NAME	FUNCTIONALITY
P1	*SCLK*	*Serial Clock. Max rate is 1Mhz. Data is clocked out on the rising edge.*
P2	*SDO*	*Serial Data Output*
P3	*SDI*	*Serial Data Input*
P4	*SS*	*Slave Select(low enable)*
P5	*P/C*	*Poll(low enable)/Continuous(high enable)*
P6	*CAL*	*Calibration(low enable)*
P7	*RES*	*Resolution. Determines resolution for calculations*
P8	*M/S*	*Master(low enable)/Slave(high enable)*
P9	*BCD/BIN*	*Binary Coded Decimal(low enable)/Binary(high enable)*
P10	YFLIP	*Flips the Y Axis. Changed depending on whether the pins of the compass are pointed down or up*
P11	*XFLIP*	*Flips the X Axis. Changed depending on whether the pins of the compass are pointed down or up*
P12	*CI*	*Calibration Indicator. Indicates when compass is being calibrated*
P13	*EOC*	*Drops low when P/C is dropped low. Goes high when heading is calculated*
P14	*RAW*	*Selects raw outputs of the X and Y readings if set to low*
P15	*POWER*	*Power. Must be tied to +5 volts*
P16	*GND*	*Ground*
P17	*RESET*	*Reset(low enabled)*

Table 1: Vector2X pin assignments

	bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
BCD Bits		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

BCD bits represent following values:

200	100	80	40	20	10	8	4	2	1
MSB									LSB

ex: bit 1,2,3,6,7=1 implies that heading value = 80 + 40 + 8 + 4 + 2 = 144 degrees

	bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Binary Bits		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Binary bits represent following values:

256	128	64	32	16	8	4	2	1
MSB								LSB

ex: bit 4,7=1 implies that heading value = 128 + 16 = 144 degrees

Table 2: Vector2X output formats

PIN	**NAME**	**SETTING**	**FUNCTION OF SETTING**
P1	*SCLK*	*Input*	
P2	*SDO*	*Output*	
P3	*SDI*	*Not Connected*	
P4	*SS*	*Input**	
P5	*P`/C*	*Input**	
P6	*CAL*	*Input**	
P7	*RES*	*High*	*High resolution*
P8	*M`/S*	*High*	*Slave mode*
P9	*BCD`/BIN*	*High*	*Binary output*
P10	*YFLIP*	*High*	*Set for compass pins facing down*
P11	*XFLIP*	*Low*	*Set for compass pins facing down*
P12	*CI*	*Output*	
P13	*EOC*	*Output*	
P14	*RAW`*	*Not Connected*	*Not in raw mode*
P15	*POWER*	*Five volts*	
P16	*GND*	*Zero volts*	
P17	*RESET`*	*Input**	

* Pin controlled and changed by the HC12

Table 3: Vector2X connections for Navigation application

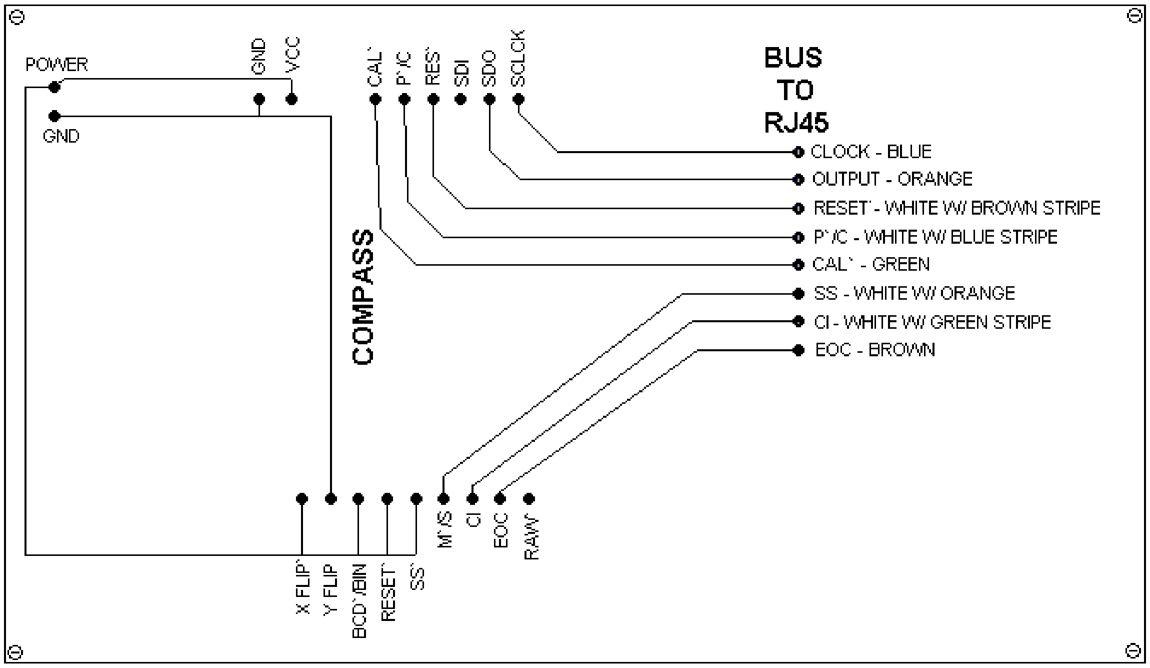


Figure 2: Compass etched board

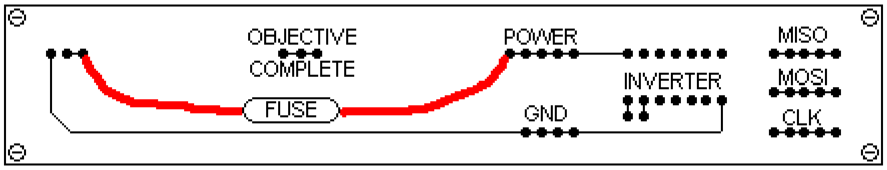


Figure 3: Signal/power distribution etched board

3.2.4 Global Positioning System Fix Data (GGA)

$GPGGA,<1>,<2>,<3>,<4>,<5>,<6>,<7>,<8>,<9>,M,<10>,M,<11>,<12>*hh<CR><LF>

<1>	UTC time of position fix, hhmmss format
<2>	Latitude, ddmm.mmmm format (leading zeros will be transmitted)
<3>	Latitude hemisphere, N or S
<4>	Longitude, dddmm.mmmm format (leading zeros will be transmitted)
<5>	Longitude hemisphere, E or W
<6>	GPS quality indication, 0 = fix not available, 1 = Non-differential GPS fix available, 2 = Differential GPS (DGPS) fix available, 6 = Estimated
<7>	Number of satellites in use, 00 to 12 (leading zeros will be transmitted)
<8>	Horizontal dilution of precision, 0.5 to 99.9
<9>	Antenna height above/below mean sea level, -9999.9 to 99999.9 meters
<10>	Geoidal height, -999.9 to 9999.9 meters
<11>	Differential GPS (RTCM SC-104) data age, number of seconds since last valid RTCM transmission (null if not an RTCM DGPS fix)
<12>	Differential Reference Station ID, 0000 to 1023 (leading zeros will be transmitted, null if not an RTCM DGPS fix)

Table 4: GGA sentence structure

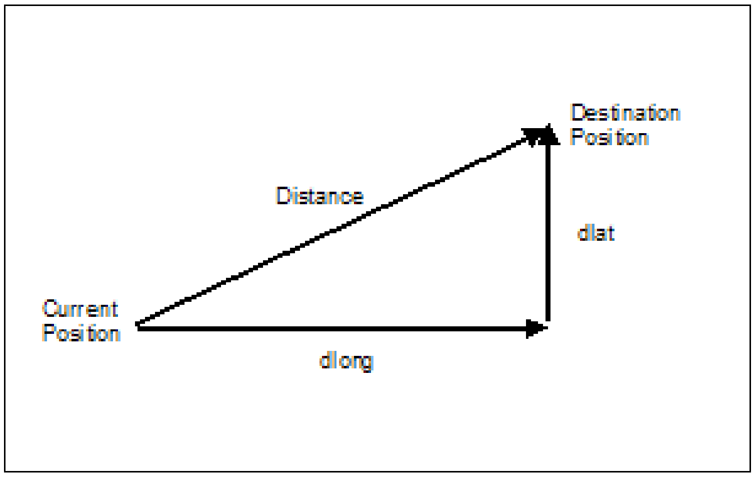


Figure 4: Use of Pythagorean theorem in distance calculation

$$\boxed{\begin{aligned}&Distance = \sqrt{\Delta lat^2 + \Delta lon^2}\\&\Delta lat = dest\_lat - current\_lat\\&\Delta lon = current\_long - dest\_long\end{aligned}}$$

Figure 5a: Distance equations

$$\boxed{distance_n = distance_n - \frac{distance_n \wedge 2 - (\Delta lat^2 + \Delta lon^2)}{2 * distance_n}}$$

Figure 5b: Distance equations

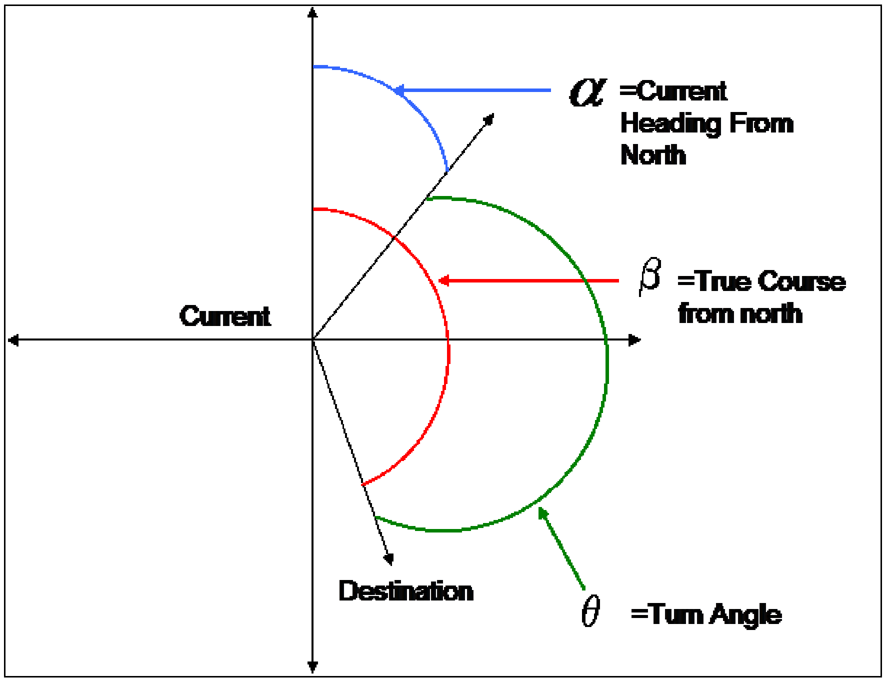

α =Current Heading From North
β =True Course from north
Current
Destination
θ =Turn Angle

Figure 6: Course calculations

$$\tan^{-1} x = \begin{cases} x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \cdots & |x| < 1 \\ \pm\frac{\pi}{2} - \frac{1}{x} + \frac{1}{3x^3} - \frac{1}{5x^5} + \cdots & \begin{cases} + \text{ if } x \geq 1 \\ - \text{ if } x \leq -1 \end{cases} \end{cases}$$

Figure 7: Inverse tangent approximation

Δlat	**Δlon**	**True Course Direction**	**β = final true course**
Positive	Negative	NE	β= β
Negative	Negative	SE	β = PI - β
Negative	Positive	SW	β= β + PI
Positive	Positive	NW	β= 2*PI - β

Table 5: Quadrant assignments

Appendix B: Pin outs & Wiring diagram

Output		Group	Function	Bus Color
PORTA0		Compass	Poll'/Cont	Brown
PORTA1		Compass	SS	Orange
PORTA2		Compass	Cal	Blue
PORTA3		Compass	Reset	Yellow
PORTA4		Chassis	SS	Black
PORTA5		Comm	SS	White
PORTA6		Ball/Hole	SS	Green
PORTA7				Red

Input		Group	Function	Bus Color
PORTB0		Compass	EOC	Yellow
PORTB1		Chassis	Completion Flag	Orange
PORTB2		Comm	Ready	Black
PORTB3		Ball/Hole	Ready	White
PORTB4		Chassis	Objective Completed	Red
PORTB5				Blue
PORTB6				Brown
PORTB7				Green

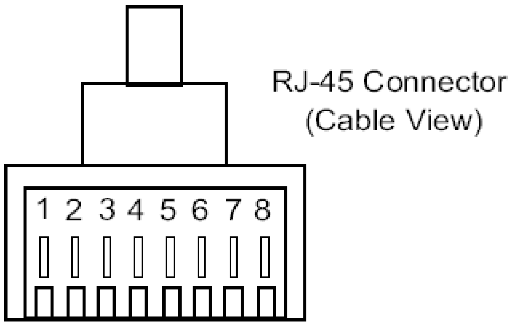
***NOTE: Ports A & B are actually expanded ports A & B*

RJ-45	**Compass** (Purple)		**Chassis** (Green)		**Comm** (Orange)	
Pin 1	SS	PORTA1	SS	PORTA4	SS	PORTA5
Pin 2	MISO		MISO		MISO	
Pin 3			MOSI		MOSI	
Pin 4	Clock		Clock		Clock	
Pin 5	Poll'/Cont	PORTA0	Completion Flag	PORTB1	Ready	PORTB2
Pin 6	Cal	PORTA2	Objective Completed	PORTB4	Stop	PORTB5
Pin 7	Reset	PORTA3				
Pin 8	EOC	PORTB0				

RJ-45	**Ball/Hole** (Yellow)		**GPS** (White)		**Power** (Blue)
Pin 1	SS	PORTA6	Power		Power from Chassis
Pin 2	MISO		Ground		Ground from Chassis
Pin 3	MOSI		Remote Power	Ground	
Pin 4	Clock				
Pin 5	Ready	PORTB3	Read	PORTS0	
Pin 6	Objective Complete	PORTB4			
Pin 7					Power to Compass
Pin 8					Ground to Compass

Table 6: System pin outs

RJ-45	Chassis	Communications	Ball/Hole
Pin 1	SS	SS	SS
Pin 2	MISO	MISO	MISO
Pin 3	MOSI	MOSI	MOSI
Pin 4	Clock	Clock	Clock
Pin 5	Completion Flag	Ready	Ready
Pin 6	Objective Completed	Stop	Objective Completed
Pin 7			
Pin 8			



RJ-45	Color
Pin 1	White w/Orange Stripe
Pin 2	Orange
Pin 3	White w/Green Stripe
Pin 4	Blue
Pin 5	White w/Blue Stripe
Pin 6	Green
Pin 7	White w/Brown Stripe
Pin 8	Brown

Table 7: Subsystem pin outs

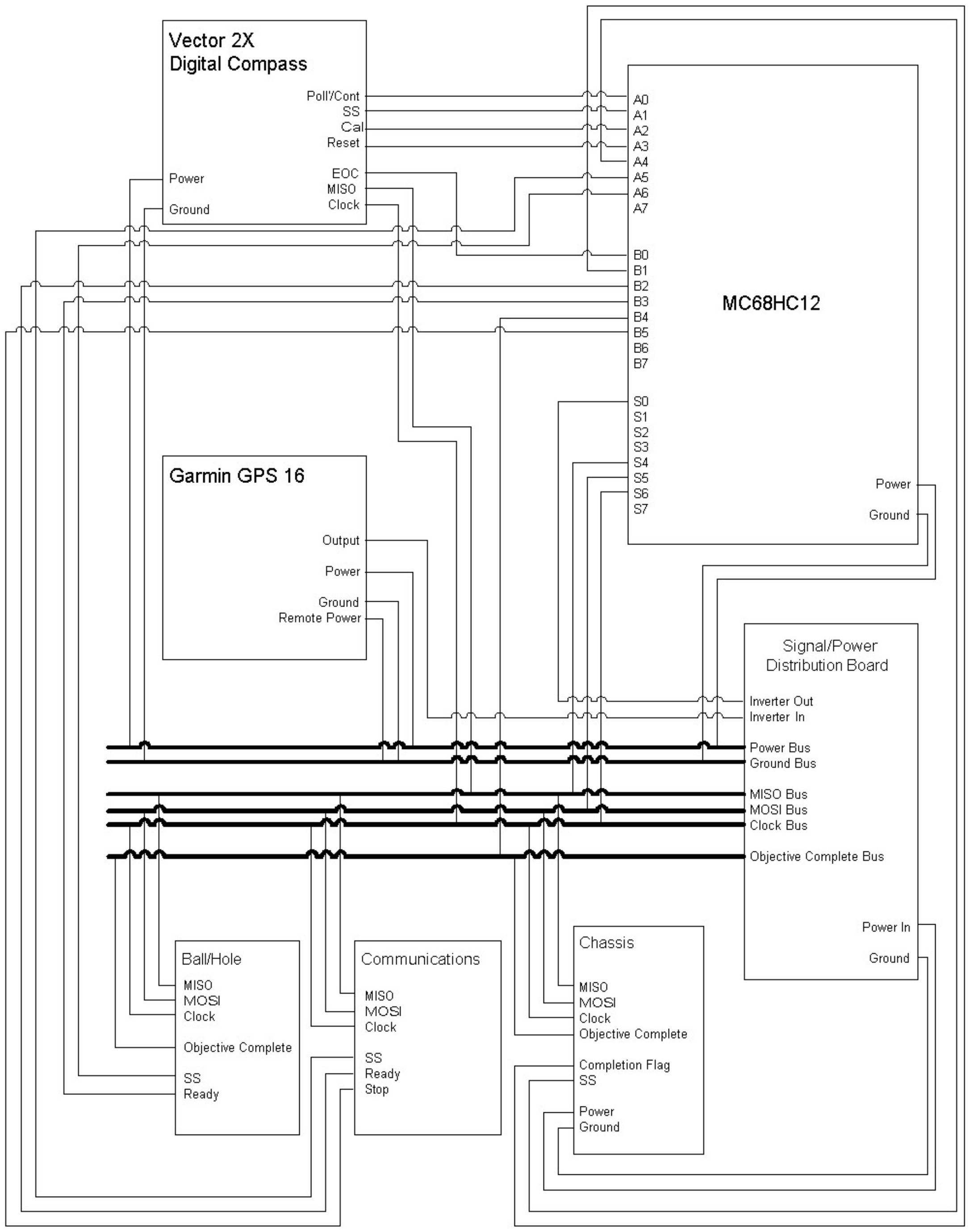


Figure 8: Navigation system wiring diagram

Appendix C: Budget

	Quantity	Unit Price	Total
Hardware			
Vector 2x compass	1		$58.25
Garmin 16 GPS receiver	1		$164.00
HC12 microcontroller			
3054-87 1.4 qt. Servin' Saver Round	1		$2.11
Miscellaneous			
Snappable 30 Pin Sip Socket	1		$1.00
Strain Relief	10	$0.10	$1.00
Standoffs	10	$0.13	$1.25
Copper Etching Board	1		$8.50
Cables	4	$1.94	$7.76
RJ45 colored jacks	6	$5.59	$33.54
RJ45 quicksnap frame	1		$3.95
wall plate	1		$0.69
bolts, nuts, brackets, and washers			$6.50
fuse holder, fuses, DB9 connector			$5.00
shipping on misc. items			$14.62
Total			**$308.17**

Table 8: Navigation final budget

Appendix D: Navigation source code

Header Files, Define, Function Prototypes, Global Variables

```
#include <hc12.h>
#include <stdio.h>

#define D_1MS (8000/4)       //constant used in delay function
#define CM_MIN 18.5208522          //used to convert distance in radians to centimeters
#define chassis_SS (0x10)     //Chassis Slave Select(Port A4)
#define comm_SS (0x20)       //Comm Slave Select (Port A5)
#define ball_hole_SS (0x40)  //Ball/Hole Slave Select (Port A6)
#define compass_SS (0x02)   //Compass Slave Select (Port P3)
#define comm_ready (0x04)  //Ready pin used by Comm for SPI transfer (Port B2)
#define ball_hole_ready (0x08)  //Ready pin used by Ball/Hole for SPI Transfer (Port B3)
#define PI 3.141592653589793238 //Constant PI

void gps();
void delay(unsigned int ms);
void check();
void ball_hole();
void send(int SS, int x);
void receive(int SS, int x, int ready, int j);
int compass();
void distance();
int turn_angle(int x);
void conversion();

char receive_array[30] = {0};  //Global array to store data received on SPI transfer
char send_array[15] = {0};     //Global array to store data transmitted on SPI transfer

double ball_pos[2] = {0};       //Array for ball lat/long in deciminutes
double hole_pos[2] = {0};       //Array for hole lat/long in decimunites

double current_pos[2] = {0}; //Array for present lat/long in decimunites
double current_dest[2] = {0};//Array for destination lat/long in decimunites

long int dist = 0;                    //Distance from present to destination position
int task = 1;                         //Initialize task to 1, enter check function first
int objective = 0;                    //Variable for obejective complete 0-looking for ball
                                      //1-have ball, 2 dropped ball in hole
```

Main Robot Function

```
//************MAIN Robot FUNCTION***********************************

main()
{

int x;

//*********************SCI SETUP*************************************
SC0BDH = 0x00;      //set SCI to 4800 Baud
SC0BDL = 0x68;
//******************************************************************


//*********************Setup SPI**************************************

DDREXP = 0X01;     //Set Expansion Port A for output and Expansion Port B for input
PORTA = 0XFF;      //Initialize all Port A/Slave Selects High
DDRS = 0xE0;       //SS, SCLK, MOSI OUTPUTS
SP0CR1 = 0x54;     //Master Mode, Clock Phase idel High
SP0CR2 = 0x00;     //Not bidirectional, normal mode
SP0BR = 0x02;      //SPI clock set to 1Mhz

//******************************************************************


//****************Setup Input Capture **********************************

TSCR = 0X80;                          //Turn on Timer
TIOS = TIOS &~0X01;                   //Channel 0 acts as input compare
TCTL4 = (TCTL4 | 0X01) & ~0X02;       //Set Channel 0 to capture rising edge
TFLG1= 0X01;                          //Clear the Interrupt
TMSK1 = TMSK1 | 0X01;                 //Hardware Interrupt Enabled
enable();                             //Enable all interrupts

//******************************************************************


//*******Wait for instructions from communications**************************

for (x=0; x<20; x++)                          //Receive 20 Bytes
{
        while((PORTB & 0x04) != 0x04);        //Wait for comm to be ready
(PORTB2 high)
```

```
	receive(comm_SS, 1, comm_ready,x);		//Call receieve function with Comm
setup

}

//*******************************************************************

//*******************Parse communications data****************************

	ball_pos[0] =
10000*((receive_array[2]+.01*receive_array[3]+.0001*receive_array[4])/60);
	ball_pos[1] =
10000*((receive_array[7]+.01*receive_array[8]+.0001*receive_array[9])/60);
	hole_pos[0] =
10000*((receive_array[12]+.01*receive_array[13]+.0001*receive_array[14])/60);
	hole_pos[1] =
10000*((receive_array[17]+.01*receive_array[18]+.0001*receive_array[19])/60);


	current_dest[0] = ball_pos[0];		//Set current position to ball position
	current_dest[1] = ball_pos[1];



while(objective < 2)	//Do this until ball is dropped
	{

	if(task == 1)		//If not close enough to the ball/hole
		check();	//Jump to check function

	if (task ==2)		//If close enough to the ball/hole
		ball_hole();	//Jump to Ball_Hole function

	}	//End while

}

//****************************End Main**********************************
Check Function



//*************************Check Function*******************************
//Check function is used to update current position, send movement commands to
//the chassis, and update position to the communications group
```

```
void check()
{
int x;
int heading = 0;                //Heading acquired by the compass
int turn = 0;                   //Angle needed to turn by chassis


gps();          //Jump to GPS function to get current position

distance();     //Jump to Distance function to calculate distance to go


if (dist <= 500)          //If distance is less than 500 cenitmeters then set task = 2
        {
                task = 2;
                return;         //Return to main function
        }


heading = compass();            //Set heading to current heading from north


if (dist >= 765)                //765 is maximum distance chassis can move
        send_array[0] = 0xff;   //If greater than this set send_array[0] to 0xff

else
        send_array[0] = dist/3;         //else send chassis units of 3 centimeters


turn = turn_angle(heading);     //set Turn to current turn_angle

if (turn >= 255)                //if greater than 255, set first byte to 1
        {                       //and second byte to turn-255
        send_array[1] = 0x01;
        send_array[2] = turn - 256;
        }
else
        {                       //else set first byte to 0
        send_array[1] = 0x00;//and seconde to value of turn
        send_array[2] = turn;
        }

send_array[3] = 0xc4;           //Error Checking byte


send(chassis_SS, 4);    //Send distance, turn angle and error checking byte to chassis
```

```
while ((PORTB & 0x02) == 0)          // Wait for chassis to send "movement complete"
signal
{

        //Update location to communications group (every 5 seconds)(while we wait)
        gps();
        delay(5000);   //delay 5 seconds
        send_array[0] = 0x01;          //Set progress byte to 1 - Searching with GPS
        conversion();                  //Convert location data to seconds and fill array
        heading = compass();           //Set heading to current compass heading

        if (heading >= 255)            //if heading > 255 set first byte to 1
                {                      //second byte to heading-255
                send_array[7] = 0x01;
                send_array[8] = heading - 256;
                }
        else                           //else set first byte to 0, second to heading
                {
                send_array[7] = 0x00;
                send_array[8] = heading;
                }

        send(comm_SS, 9);              //send comm progress byte, location, heading



}       //End while


return;                   //Return to main

}
//*********************End Check************************************


//****************Ball Hole Function***********************************
//Ball hole function is used as a relay between ball/hole and chassis.
//All commands are routed through us to chassis


void ball_hole()
{

PORTA &= !ball_hole_SS;               //Initiate ball/hole by toggling slave select
```

```
delay(100);
PORTA |= ball_hole_SS;


send_array[0] = 0x02;          // set send_array to comm status bit for ball/hole searching
send(comm_SS, 1);              // tell communications that ball/hole has started its search


//*******Repeat this stuff until we have completed objective******************
while(1)
{

while ((PORTB & 0x08) == 0);                // wait for ball_hole to be ready

receive(ball_hole_SS, 3, ball_hole_ready);  // Receive directions from ball hole

        send_array[0] = receive_array[0];   //Distance
        send_array[1] = receive_array[1];   //Heading byte 0
        send_array[2] = receive_array[2];   //Heading byte 1
        send_array[3] = 0xc4;               //Error byte

        send(chassis_SS, 4);                // Send directions to chassis


while ((PORTB & 0x02) == 0);                // Wait for chassis to send "complete" signal

if((PORTB & 0x10) == 0)                     // did we complete objective? if no start
while over
{
        objective = objective++;       //if objective complete increment to next objective

        send_array[0] = 0x04;          // 0x04 - objective complete
        send(comm_SS, 1);              // tell comm that we have completed objective

                if(objective == 1)
                {
                        current_pos[0]=ball_pos[0];   //set current pos to ball pos
                        current_pos[1]=ball_pos[1];
                        current_dest[0]=hole_pos[0]; //set destination to hole pos
                        current_dest[1]=hole_pos[1];
                        task = 1;
                        return;                       //return to main and enter check
function
                } //end if

                if(objective==2)
                {
                        exit();            //Mission complete end program
                }

} //end if

} //end while

} //end ball_hole
//*************End Ball Hole Function************************************
```

Send Function

```
//********************Send Function************************************
//Send function is a generic function to send data to any of the other modules
//It has for input, the slave select of the module(SS) and the number of bytes
//being sent(x)

void send(SS, x)
{
        int i= 0;
        PORTA &= !SS;                   //Select the slave
        for (i; i < x; i++)             //Repeat x times
        {
                SP0DR = send_array[i];                  //Send data to slave
                while((SP0SR & 0x80) == 0);             //Wait for transfer to
complete
                delay(100);                             //Delay 100 ms and repeat
        }

        PORTA |= SS;            //Deselect the slave

        return;                 //Return
}
//*************End Send function*******************************************



//*************Receive Function*******************************************
//Receive function is used to to receieve data over the SPI system from other modules
//It has for input, the slave select(SS), number of bytes being received(x), and
//the ready line(ready)

void receive(SS, x, ready,j)
```

```
{


        int i= 0;
        for(i = 0; i<x; i++)                //Repeat x times
        {
                PORTA &= !SS;                       //Select the slave
                delay(100);                         //Delay 100 ms
                SP0DR = 0xaa;                       //Send junk data to slave
                while((SP0SR & 0x80) == 0);         //Wait for transfer to complete
                receive_array[j] = SP0DR;           //fill receive arrray
                PORTA |= SS;
        }
        return;
}
//********End Recieve Function*****************************************




//***********Delay Function**********************************************
//Delay function is used to delay ms microseconds.
void delay(unsigned int ms)
{
        int i;

        while (ms>0)
        {
                        i=D_1MS;
                        while (i>0)
                        {
                                i--;
                        }
                        ms--;
        }
}
//**********End Delay Function*********************************************


//*************************Compass Function****************************

int compass()
{
int head;
char heading[2];
```

```
DLCSCR =0X00        //Set DLC to normal operating mode
DDRDLC = 0X04       //Set pin 2 of PortDLC as input
DDRP = 0xff;        //Set PORTP output
PWCTL = 0x00;       //Normal drive capability on port P
PORTP = 0xF0;       //Initialize Port P high, compass pins are active low

//*****************SPI Setup*********************************
DDRS = 0xE0;                //SS, SCLK, MOSI OUTPUTS
SP0CR1 = 0x5C;              //Master mode, Clock Polarity and Phase high
SP0CR2 = 0x00;              //Not bidirectional, normal mode
SP0BR = 0x02;               //SPI clock set to 1Mhz
//*******************************************************

//*********************Power up*****************************
PORTP = 0x70;               //Set reset low
delay(100);                 //Delay 100ms
PORTP = 0xf0;               //Toggle reset back high
delay(750);                 //Delay 750 ms
//*******************************************************


//****************Retrieve data*********************************

        PORTP &= 0xEF;                          //Set PC low
        delay(10);                              //Delay 10 ms
        while ((PORTDLC & 0x04) == 0x01);       //Wait for EOC to go low
        PORTP |= 0x10;                          //Set PC high
        while ((PORTDLC & 0x04) == 0x00);       //Wait for EOC to go high
        delay(10);

        PORTP = (PORTP & 0xdf);                 //Lower Slave select
        delay(5);

        SP0DR=0Xd6;                             //initiate transfer
        while ((SP0SR & 0x80) == 0);            //Wait for register to fill
        heading[0] = SP0DR;                     //Fill heading array

        delay(2);
        SP0DR=0x48;                             //initiate second transfer
        while ((SP0SR & 0x80) == 0);            //Wait for register to fill
        heading[1] = SP0DR;                     //Fill Heading array

        if (heading[0]==0x01)                   //First byte of heading is 1
        {
                head = 256 + heading[1];        //Add 256 to heading[1]
        }                                       //and set equal to head
        else
                head=heading[1];                    //else set head to heading[1]

        PORTP = (PORTP & 0xdf);                     //Raise slave select
        return(head);                               //Return Current heading

}

//**********************End Compass Function***************************



//********************GPS Function*************************************

void gps()
{

char lat[10];           //Array to store latitude data from GPS
char lon[11];           //Array to store longitude data from GPS

char num1 = 0;          //Generic char variable

int y;                  //Generic int variable
int lat_dec[10];        //Array used to convert char to int
int lon_dec[11];        //Array used to convert char to int

int x = 0;

//*********Continue until G and A have been read consecutively******************

while (x != 2)
{
x = 0;

while ((SC0SR1 & 0x20) == 0);       // Wait for Receive data register full flag to be set
num1=SC0DRL;                        // Read SCI data register into variable

        if (num1 == 0x47)     // if data is an G, increment X
                x++;

while ((SC0SR1 & 0x20) == 0);       // Wait for Receive data register full flag to be set
num1=SC0DRL;                        // Read SCI data register into variable

        if (num1 == 0x41)     // if data is an A, increment x
                x++;
}
```

```
//*********** Have received a G then an A***********************************


//****Continue until 2 commas have been found ( , = 0X2C)**********************

x = 0;
while (x != 2)
{

while ((SC0SR1 & 0x20) == 0);        // Wait for Receive data register full flag to be set
num1=SC0DRL;                         // Read SCI data register into variable

        if (num1 == 0x2C)     // if data is a ",", increment x
        x ++;

}

//************ Received 2 commas ****************************************


//***************** Now store latitude Data*********************************

for (x=0; x<9; x++)
        {
        while ((SC0SR1 & 0x20) == 0);        // Wait for Receive data register full flag to
be set

        lat[x]=SC0DRL;                       // Read SCI data register into variable

}

//****************** Latitude stored in lat[]*********************************


//***************Count 2 more commas***********************************
x = 0;

while (x != 2)
{

        while ((SC0SR1 & 0x20) == 0);        // Wait for Receive data register full flag to
be set
        num1=SC0DRL;                         // Read SCI data register into variable

        if (num1 == 0x2C)                    // if data is a ",", increment x
        x++;

} // End while

//*******************Received 2 commas*********************************


// ***************Now store longitude information**************************


for (x=0; x<10; x++)
{
        while ((SC0SR1 & 0x20) == 0);       //Wait for Receive data register full flag to
be set

        lon[x]=SC0DRL;                      // Read SCI data register into variable

}

//***********************Longitude stored in long[]********************

lat[9] = '\0';              //End lat string with null character
lon[10] = '\0';             //End long string with null character

y=0;

//**********************Convert Latitude data to integer value***************

for (x=0; x<9; x++)
{

        if (lat[x] == 0x2e)             //if period, skip this value
                x++;

        lat_dec[y] = (int)lat[x] - 48;  //convert character number to integer number
        y++;

}
//****************Latitude Data Converted********************************


//********************Convert Longitude Data to Integer Value*************

y = 0;
for (x=0; x<10; x++)
{
        if (lon[x] == 0x2e)             //if Period, skip this value
        x++;
        lon_dec[y] = (int)lon[x]-48;    //convert character number to integer number
        y++;
}
//**************Longitude Data converted**********************************

//****Set current position to deci-minute values and store as a whole number*********

current_pos[0] = (1000*lat_dec[4]) + (100*lat_dec[5]) + (10*lat_dec[6]) + (lat_dec[7]);
current_pos[1] = (1000*lon_dec[5]) + (100*lon_dec[6]) + (10*lon_dec[7]) +
(1*lon_dec[8]);

//******************************************************************

return;                 //Return to calling function

}

//**********************End GPS****************************************


//*********************Distance function**********************************
//Distance function calculates the distance between two points

void distance()
{

        int i;
        double x;               //Initial value used for approximation
        double dlat;            //Difference in Latitude
        double dlon;            //Difference in Longitude
        double tempd;           //Temporary value used to calculate distance
        double d;               //Sum of the squares of dlat and dlon
        dlon = current_dest[1] - current_pos[1];        //calculate differnce in lon
        dlat = current_dest[0] - current_pos[0];        //calculate different in lat
        d=dlon*dlon  + dlat*dlat;               //set d to sum of squares of dlon and dlat
        x = 10000;                              //Initial guess
        for(i = 0; i<50; i++)                   //Iterate 50 times
        {
                tempd = x -( (x*x - d)/(2*x)); //calculate next term
                x = tempd;
        }
        dist = tempd * CM_MIN;                  //Convert from Minutes to Radians
        return;                                 //Return to calling function
```

```
}
//***************End Distance Function**********************************

//***************Turn Angle Function************************************
//Turn Angle function calculates true course based on two points, and then
//Calculates a turn angle based on true course and heading from the GPS
//Has course has input which is heading from north

int turn_angle(course)
{

        int turn_angle;         //Value returned by function, angle from north to turn
        double dlon;            //Difference in longitude
        double dlat;            //Difference in Latitude
        double a;               //Arctan argument dlon/dlat
        double true_course;     //True Course from north
        dlon = current_pos[1] - current_dest[1];        //Calculate dlon and map into N-E
plane
        dlat = current_dest[0] - current_pos[0];        //calculate dlat
        a = (dlon/dlat);        //set a to arctan argument
        if (dlon <0)            //if dlat or dlon is negative, make a positive
                a = (-1)* a;
        if  (dlat <0)
                a = (-1) * a;

        if (a > 1)              //if a< 1 use first atan taylor series expansion
                {
                true_course = PI/2 -(1/a) + (1/(3*a*a*a)) - (1/(5*a*a*a*a*a)) +
(1/(7*a*a*a*a*a*a*a));
                }
        else if (a <= 1)        //if a<=1 use second atan taylor series expansion
                {
                true_course = a - (a*a*a)/3 + (a*a*a*a*a)/5 - (a*a*a*a*a*a*a)/7 +
(a*a*a*a*a*a*a*a*a)/9;
                }

        //Edit true course to put into the correct quandrant
        if (dlon >0 & dlat<0)
                true_course = PI - true_course;
        if (dlat < 0 & dlon < 0)
                true_course = true_course +PI;
        if (dlat >0 & dlon<0)
                true_course = 2*PI - true_course;
        true_course = true_course *180/PI;              //True course to degrees

        //Calculate turn angle
```

```
	if (course > true_course)
	turn_angle = 360+true_course-course;
	if(course < true_course)
	turn_angle = true_course - course;

	return(turn_angle);	//Return turn angle.

}


//*************End Turn Angle Function**********************************

//*******************Input Capture Interrupt******************************
@interrupt void tic0_isr(void)
{
	send_array[0] = 0;
	send_array[1] = 0;
	send_array[2] = 0;
	send_array[3] = 0xc4;
	send(chassis_SS, 4);		//Send chassis all zeros to stop them
	TFLG1 = 0X01;			//Clear Interrupt
	exit();
}
//***************End Input Capture Interrupt******************************


//*******************Test Code Main Function*****************************
void main()
{
	int heading;
	int compass_heading;

//Parse simulated Comm Data

	ball_pos[0] =
10000*((receive_array[2]+.01*receive_array[3]+.0001*receive_array[4])/60);
	ball_pos[1] =
10000*((receive_array[7]+.01*receive_array[8]+.0001*receive_array[9])/60);
	hole_pos[0] =
10000*((receive_array[12]+.01*receive_array[13]+.0001*receive_array[14])/60);
	hole_pos[1] =
10000*((receive_array[17]+.01*receive_array[18]+.0001*receive_array[19])/60);
	current_dest[0] = ball_pos[0];
	current_dest[1] = ball_pos[1];

	SC0BDH = 0x00;	//Set Baud Rate to 4800
	SC0BDL = 0x68;

	while(1)		//Do forever
	{
	gps();			//Call GPS, and get current position
	distance();		//Call Distance and calculate current distance
	compass_heading = compass();		//Set current heading to compass heading
	heading = turn_angle(compass_heading);	//Get current turn angle
	//Print All Data
	printf("Heading from north: %d\n\r", compass_heading);
	printf("Turn_angle = %d\n\r", heading);
	printf("Distance in CM  = %d\n\r", (int)(dist));
	delay(2000);
	printf("\n\n\n\n\n\n\n\n\n\n");

	}

}

//******************End Test Main Code**********************************


//****************Redefine PutChar Function*******************************
//This function is used to allow printf statements to work with the HC12

int putchar(char x)
{
	while ((SC0SR1 & 0x80) == 0);
	SC0DRL = x;
	if (x == 0x0a)
	{
		while ((SC0SR1 & 0x80) == 0);
		SC0DRL = 0x0d;
	}
}
```

Appendix E: References

Garmin International. Garmin GPS 16 Technical Specifications. (2002, May). Available:
http://www.garmin.com/products/gps16/techspec.html

PNI Corporation. Vector 2X user manual. (2002, May). Available:
http://www.pnicorp.com/technical-information/pdf/vector-2x.pdf

Apollocom. Visual GPS. (2002, May). Available:
http://www.apollocom.com/VisualGPS/default.htm

Peter Bennett. NMEA FAQ. (2002, May). Available:
http://vancouver-webpages.com/peter