A GPS Guided Remote Controlled Truck

Scott L. Nykl
Computer Science and Software Engineering Department
University of Wisconsin – Platteville
Platteville, WI 53818
scottnykl@gmail.com

Kitrek T. Riese
Design Engineer – Software
HK Systems
Milwaukee, WI 53151
kitrek@gmail.com

Jeffrey E. Runde
Product Engineer
John Deere Product Engineering Center
Waterloo, IA 50704
jerunde@gmail.com

Michael C. Rowe, Ph.D.
Computer Science and Software Engineering Department
University of Wisconsin – Platteville
Platteville, WI 53818
rowemi@uwplatt.edu

Abstract

This paper describes a final semester project done at the University of Wisconsin Platteville by three software engineering students. The project consisted of a remote controlled truck controlled via a microcontroller. In turn, the microcontroller was interfaced with a portable GPS receiver. This system allowed a user to enter up to ten waypoints. The vehicle then navigated to each of those waypoints in the order in which they were entered. These waypoints were stored as longitude and latitude coordinates. When the vehicle arrived at the final waypoint, it simply shut down its motor and waited to receive an additional set of waypoints.

The vehicle determined its movement based on its present position, as determined by its onboard GPS receiver. Through this communication link, the vehicle was able to obtain its location in terms of longitude and latitude, and its current heading in the form of degrees.

Introduction

Our project, affectionately known as DukE, is an autonomous remote controlled vehicle which navigates its way across the University of Wisconsin Platteville's campus. It is capable of following a sequential list of waypoints entered by the user to find its way to the desired destination. It relies solely on position information received from its onboard GPS receiver to determine its current position and drive itself to the next waypoint. The authors of this paper are all current or former students of the software engineering program at the University of Wisconsin Platteville. We chose to develop this project because it spans many areas of engineering: software, electrical, and mechanical thus providing us with a very interesting systems engineering project not found in any other course we have taken.

Project Goals

In developing this project our primary goal was to develop a system that leveraged the combined functionality of many different subsystems into a functionally useful system that could be completed in the timeframe of a semester final project. This system is an autonomous vehicle that relies on the following subsystems: a Garmin® (18PC Serial) GPS receiver for location determination, a PJRC 8051 microcontroller for algorithm computations and vehicle control, a 20x2 character LCD display to indicate to the user the current state of the system, and a Radio Shack® remote controlled model Hummer® H2® to provide the mechanical propulsion to transport the entire system to the next waypoint.

To complete the project within the allotted time frame we completed a risk assessment study on each subsystem and its interface with the entire system to determine the subsystem's estimated implementation time. This was done by prototyping and proving each of the riskiest functional areas of each subsystem. For example we prototyped the interface between the GPS receiver and the microcontroller by first verifying the receiver's operating characteristics while it was connected to a laptop running Microsoft's® HyperTerminal traveling in an automobile around a campus parking lot. We then wrote an experimental application for the microcontroller which would display the current position data from the GPS receiver in real-time to prove that this interface was functionally possible. It also provided us with a very elaborate yet rudimentary GPS receiver/display device similar to a handheld GPS device used for outdoor navigation.

By prototyping much of the main functionality of the system prior to actually starting implementation we were able to avoid any major problems late in the project that could have prevented its completion and in turn decreased our individual grade point averages.

Prolog

GPS Overview

Our project requires that the system be capable of determining its location on Earth for navigational calculations. A GPS receiver communicating with the microcontroller was chosen to handle this task. The GPS receiver is able to determine its three dimensional location on Earth (latitude, longitude, and altitude) by simultaneously tracking at least four GPS satellites in real time. The receiver works by internally generating the same broadcast signal as the satellite, and comparing it with the signal received from the satellite. It then determines the time difference between both signals. This difference is the amount of time it took for the signal to travel from the satellite to the receiver. The receiver then multiplies this time by the speed of light to determine the distance between it and the satellite. By tracking three satellites the GPS receiver can determine its location on Earth, the forth satellite is necessary to account for the timing inaccuracies of the receiver [1].

NMEA-0183 Standard

The National Marine Electronics Association is responsible for creating and maintaining the interface and communication standards used mainly between marine electronic hardware. Since marine navigation was one of the first civilian uses of GPS the NMEA needed to define it in a standard. This was first covered in the NMEA-0180 and NMEA-0182 standards which were very simplistic and originally created for the Loran-C navigation system. The NMEA then created the NMEA-0183 standard, which provided more complex functionality for GPS; it also still includes Loran-C and other navigation systems.

The NMEA-0183 standard defines the interface and data protocol between a GPS receiver and the other hardware with which it is communicating. In our project the GPS receiver communicates with a microcontroller. The standard also defines how various other marine electronic components communicate. The electrical connection on which communication takes place must follow the EIA-422 standard. This is basically equivalent to an RS-232 connection except that instead of only having one data connection there are two. The channels transmit and receive identical signals but the second channel is the inverse of the first channel [2].

NMEA Protocol

The NMEA protocol defined in the NMEA-0183 standard defines how information is passed between devices such as a GPS receiver and the system using the navigational data. All data elements being transmitted and received are part of a sentence. Each sentence is composed of up to 82 printable ASCII characters. The standard transmission rate defined in the NMEA-0183 is 4800 baud with 8 bits of non parity data plus a stop bit. This means that a device can send a maximum of 480 characters in one second. At 82

characters per sentence this is only six sentences per second. For a real time system this transmission limit can create a large lag in the data as it can be a few seconds old before it is done transmitting. In addition, since the amount of data that can be sent is small, a GPS receiver following the NMEA-0183 standard generally only sends one position update every second [2].

NMEA Sentence Structure

The following table contains some of the most important and commonly used sentence identifiers from the NMEA standard. These sentences are understood by all common devices using GPS data [3].

Sentence	**Description**
$GPGGA	Global positioning system fixed data
$GPGLL	Geographic position - latitude / longitude
$GPGSA	GNSS DOP and active satellites
$GPGSV	GNSS satellites in view
$GPRMC	Recommended minimum specific GNSS data
$GPVTG	Course over ground and ground speed

Figure 2.1.1: NMEA Common Sentence Identifiers

For example the following shows the breakdown of a GPS fixed data sentence.

```
$GPGGA,123621,4805.037,N,01222.000,E,1,07,0.8,507.4,M,46.7,M,,,*32

     GPGGA        Global Positioning System Fix Data
     123621       Fix taken at 12:36:21 UTC
     4805.037,N   Latitude 48 deg 00.037' N
     01222.000,E  Longitude 12 deg 22.000' E
     1            Fix quality: 0 = invalid
                               1 = GPS fix (SPS)
                               2 = DGPS fix
                               3 = PPS fix
                                       4 = Real Time Kinematic
                                       5 = Float RTK
                               6 = estimated (dead reckoning) (2.3 feature)
                                       7 = Manual input mode
                                       8 = Simulation mode
     07           Number of satellites being tracked
     0.8          Horizontal dilution of position
     507.4,M      Altitude, Meters, above mean sea level
     46.7,M       Height of geoid (mean sea level) above WGS84 ellipsoid
     (empty field) time in seconds since last DGPS update
     (empty field) DGPS station ID number
     *32          the checksum data, always begins with *
```

GPS Location Determination

We first programmed our GPS receiver to output only the messages that contained the relevant data for our system. The $GPRMC message contains the following data: location, current course over ground (heading), and the global time (used to ensure that fresh messages are being received.) The Garmin® GPS receiver was easy to program via a Garmin® supplied software application. The GPS unit will continually output this message, once per second, at 4800 baud. It will hold this programming until reprogrammed by the software application, even after a power down cycle. The GPS receiver receives power through a 12V cigarette lighter style jack, which we modified to take 18V via two 9V batteries.

The $GPRMC message is received by the 8051 microcontroller through its auxiliary serial port. Using the auxiliary port allows us to have the GPS unit permanently connected to the microcontroller while programming the unit on the main serial port. This feature becomes very important when the vehicle is fully assembled, and only an access cable connected to the main serial port is available for reprogramming. At this point, the GPS connection can be made permanent within the housing of the vehicle.

When the $GPRMC message is read by the microcontroller, it is stored in an array of characters for parsing and the data is then extracted from the message. An example of the $GPRMC message is shown below in Figure 3.1.1.

```
$GPRMC,002329,A,4244.1137,N,09029.2599,W,000.9,181.5,041205,000.8,W*75
```

Figure 3.1.1: Example $GPRMC GPS Message

The message is parsed using the commas as field separators. The first field of the message is the message name, '$GPRMC', this field is ignored. The global time is taken from the next field. In Figure 3.1.1, the global time is '002329'.

An 'A' is shown in the third field. This simply means the message is valid. If the message were invalid a 'V' would be shown.

The next useful field is field four, which contains the latitude coordinate. In the example above it is '4244.1137'. This coordinate is parsed out of the string and stored as a float value, moving the decimal to the right, giving 42441137. The scaling of the value to a whole number is done to compensate for the systems atan() function which for some reason would not accept decimal values; this has no effect on actual operation because all values are scaled equally. The field following this, containing the 'N' denotes the preceding value as a northern coordinate, rather than a southern. In the northern hemisphere this value will always be 'N', if the GPS message were to be used in the southern hemisphere the value would be 'S'.

Logically, following this is the longitude coordinate. In the example above this coordinate is shown as '09029.2599'. Again, to maintain consistency, this value is

scaled, and stored as 090292599. The ‘W’ in the field following this data refers to the western hemisphere, if we were in the eastern hemisphere, the value would show an ‘E’.

Directly following this is the speed of the vehicle in knots. Here shown as ‘000.9’. This data is ignored by the system.

In the ninth space the course over ground (heading) is shown. This is used to find the direction the vehicle is moving. It is important to note that this will not be valid unless the unit is moving. In the example above, it is shown to be ‘181.5’. This is a degree measure, using true north as zero, and moving clockwise around the cardinal directions.

The rest of the message is not useful for our purposes and ignored. A detailed description of all the data found in a $GPRMC message is shown below in Figure 3.1.2.

Field	**Example**	**Comments**
Sentence ID	$GPRMC	
UTC Time	124322	hhmmss
Status	A	A = Valid, V = Invalid
Latitude	4156.8822	ddmm.mmmm
N/S Indicator	N	N = North, S = South
Longitude	113649.8273	dddmm.mmmm
E/W	E	E = East, W = West
Speed over ground	005.1	Knots
Course over ground	056.9	Degrees
UTC Date	181205	DDMMYY
Magnetic variation	012.3	Degrees
Magnetic variation	E	E = East, W = West
Checksum	*44	
Terminator	CR/LF	

Figure 3.1.2: $GPRMC GPS Message Field Definitions

Navigation Algorithm

The idea behind the vehicle's navigation algorithm is that it is currently at point (X,Y) and must travel to waypoint (S,T). To do this, the vehicle must perform the following operations:

1. Retrieve its current location from the GPS receiver.
2. Retrieve the direction it is currently facing from the GPS receiver (computed as a difference between the most recently read points); this is its *heading*.
3. Generate an initial bearing vector using the waypoint position and its current position.
4. Classify the initial bearing vector into a direction of type: N, NE, E, SE, S, SW, W, or NW.
5. Execute the "Turning Algorithm" which is described later.
6. Repeat steps 1 through 5 until it is within a specified radius of the waypoint.

The implementation of this algorithm is given in the following example.

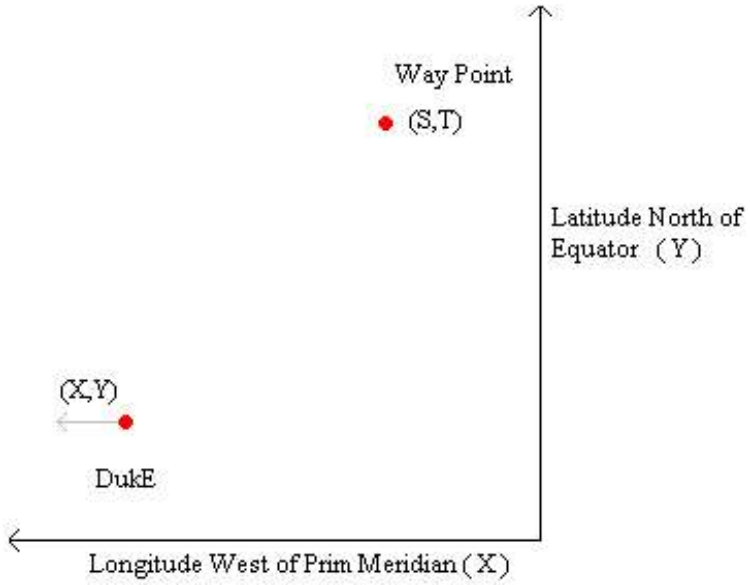

Figure 3.2.A: DukE's Position w.r.t. Waypoint

Figure 3.2.A shows a scenario where the vehicle begins its trek at a longitude and latitude of (X,Y), respectively. We have simplified the navigational calculations by using planar rather than spherical geometries. This simplification constrains our navigation to locations other than the poles or areas that cross 180 degrees E/W. From the diagram, one can see that the vehicle is located southwest from its current waypoint. Moreover, the gray arrow extending from the vehicle shows that its current heading is due west.

At this point, steps 1 and 2 of the navigation algorithm have already been completed; this is because the vehicle receives an updated heading, longitude, and latitude from the GPS receiver once every second.

To complete step 3, the vehicle must calculate the bearing between the current waypoint and itself. This is a simple subtraction of the two points resulting in the vector shown in Figure 3.2.B.

Vector = (S-X, T-Y)

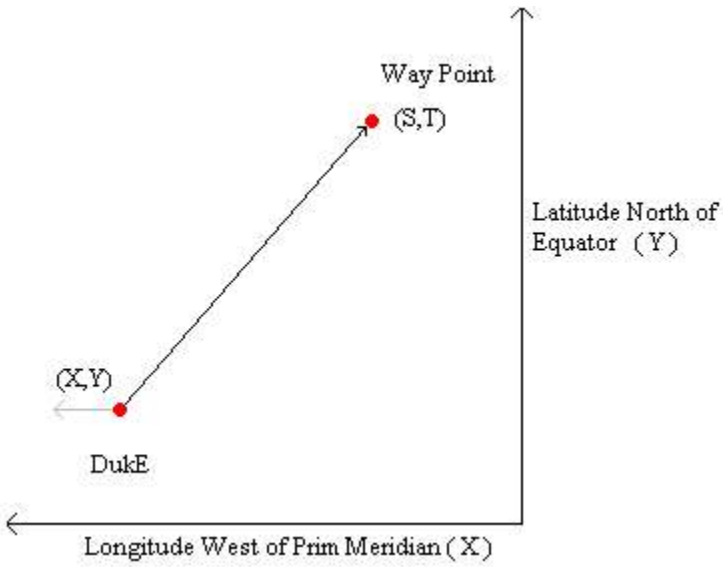

Figure 3.2.B: Initial Bearing Vector

Once the desired bearing is in vector form, step 4 can be executed. The first sub-step is to join the bearing vector with the vehicle's current heading vector tail-to-tail. The second sub-step is to translate the insertion point of these two vectors to the origin of a two-dimensional Cartesian graph. This is shown in Figure 3.2.C below.

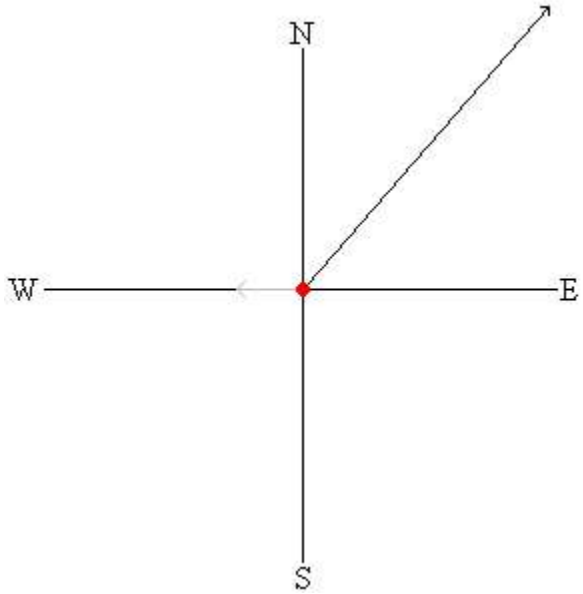

Fig 3.2.C: Translated Bearing Vector

The third sub-step is to calculate the ratio of the bearing vector, that is, latitude/longitude, or Y/X. This ratio represents the tangent of the angle between the bearing vector and the corresponding horizontal axis, which in this instance, is east.

Because the vehicle cannot precisely navigate a bearing that is represented as a single angle, it must classify the bearing's precise angle into a bearing represented by: N, NE, E, SE, S, SW, or NW.

Thus, to determine which quadrant of the Cartesian plane contains the bearing vector, one must look at the sign of the X and Y values of the bearing vector. Figure 3.2.D describes the quadrant mappings:

X sign	Y sign	Quadrant
+	+	1
+	–	4
–	+	2
–	–	3

Figure 3.2.D: Quadrant Specifications

Once the quadrant is determined, the vehicle must specify whether it needs to set the bearing vertically (N or S), horizontally (E or W), or diagonally (NE, SE, SW, NW). This is done in the following manner.

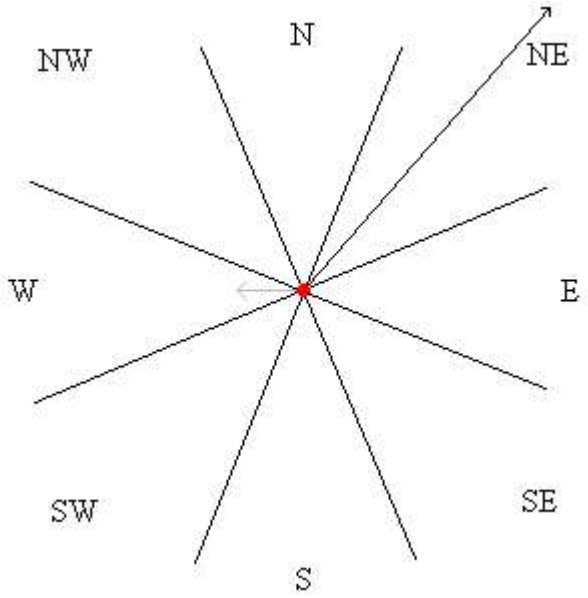

Fig 3.2.E: 8 Octant Bearing Specification

Figure 3.2.E shows a 360 degree compass divided into octants, each allotted 45 degrees; the vertical axis has been rotated 22.5 degrees clockwise to ensure the north octant covers 45 degrees facing true north.

Since the bearing is stored as a ratio of Y/X and since it already knows the quadrant in which the bearing vector rests, all it must do is compare the ratio of the tangent to each 45

degree line segment that lies within the bounding quadrant. Although this calculation may cause a divide by zero error at the Prime Meridian, it was determined this would not affect our vehicle's performance in the North America. Figure 3.2.F describes this process:

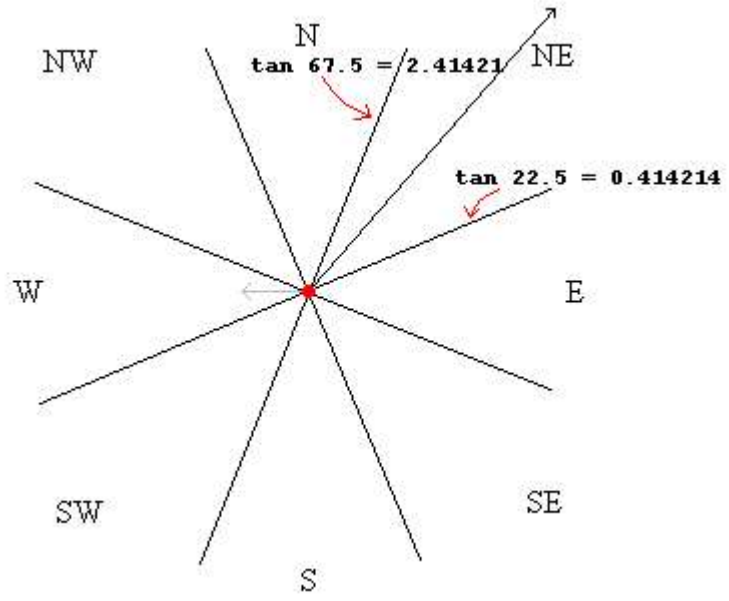

Figure 3.2.F: Final Bearing Classification

Thus, if Y/X is less than 0.414214, it sets the final bearing to E or W. If Y/X is between [0.414214, 2.41421] it sets the final bearing to NE, SE, SW, or NW, and if Y/X is greater than 2.41421 it sets the final bearing to N or S.

In this particular example, since Y is positive and X is positive, the vehicle knows that the bearing resides in quadrant 1. Additionally, since Y/X falls within [0.414213, 2.41421] it knows that its final bearing shall be set to NE.

Now that the bearing has been established for this iteration, and the vehicle's current heading is west, it must execute one iteration of the turning algorithm, as described in section "turning algorithm".

Finally, after one iteration of the turning algorithm completes, the vehicle repeats steps 1 through 5 of the navigation algorithm until the vehicle arrives within a predefined radius of the current waypoint. Then the vehicle sets the current waypoint to the next waypoint and begins the process all over again. After arriving at the final waypoint, the vehicle turns off its motors and waits.

Turning Algorithm

The vehicle is controlled by five basic motions: straight, small left turn, small right turn, large left turn, and large right turn. It is important to note that during the turns, the vehicle is moving in the forward direction. Once the vehicle begins its path to the first waypoint, it moves in the forward direction continuously until it reaches the final waypoint, where it is placed in neutral.

The vehicle does have the ability to move in reverse. However, this is not used by the software. Although a complex 'Y' style turn could be used to position the vehicle, simple right and left turns using forward are much simpler, and tend to hold the vehicle to a steadier course through the waypoints.

In making a decision on which of these control signals to deploy, the received heading is used in conjunction with the bearing. The heading is taken directly from the GPS message, and the bearing is calculated by the navigation algorithm. For example, if the vehicle was heading due west, and has a bearing of northeast, a large right turn is necessary to make the heading equal to the bearing.

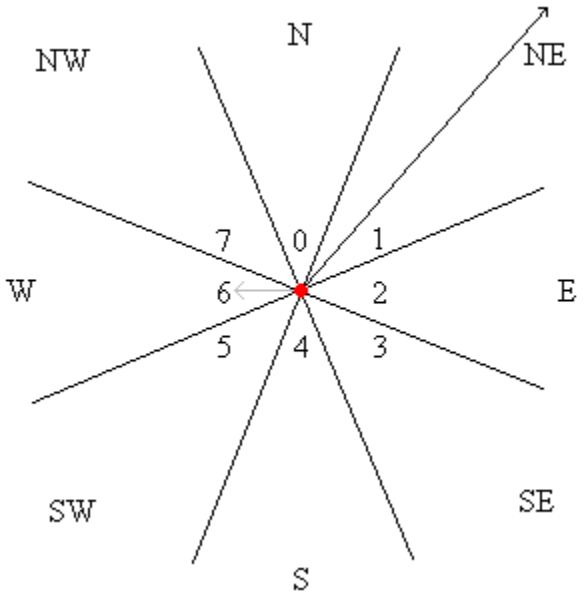

Figure 3.3.1: Numerical Values for Directions

This is calculated by first normalizing the heading. Heading normalization is done by subtracting eight (the total number of directions) and then taking the absolute value of the difference. Each possible direction is given a value as shown in Figure 3.3.1. With a current heading of west, or six, normalizing this results in the value of two. This means to change the vehicle's heading to north it must move two directional units right.

Now this result is added to our bearing. In the example, the bearing is northeast, which has a value of one giving us a new bearing of three. This means we must make a turn

covering three octants to place our desired heading directly on the bearing between the vehicle and the waypoint.

Because the value is less than four, a turn to the right will be most efficient. If the value was greater than four, the desired bearing is to the left of the current heading, and finally if the value is equal to zero, then the current heading is the same as the bearing and there is no need to turn.

This covers which direction the vehicle must turn to correct its heading. The decision between making a big or a small turn is made by looking at the difference between the current heading and desired bearing. When a change of only one directional unit is needed left or right, a small left or right turn is made, otherwise a large right or left turn is made respectively.

Since the control algorithm is executed approximately once every second, which corresponds to the position update rate of the GPS receiver used, the vehicle cannot be allowed to turn right or left during the entire second or the vehicle will over steer its bearing and will be forced to turn again. While executing a large left or right turn, the vehicle is only allowed to only turn for half of a second. The vehicle then travels straight for the remaining time until the next location update is received and the control algorithm is executed again. This also ensures that the next GPS heading update is accurate. A small turn is similar to a large turn except the amount of time the vehicle is allowed to turn is reduced to one tenth of a second. Typically a large turn is approximately 90 degrees and a small turn rotates the vehicle approximately 15 degrees. These values vary greatly based on the coefficient of friction between the vehicle tires and the ground. The once per second position update rate of the GPS receiver was not sufficient to provide high precision feedback for turning.

System Hardware

The vehicle is controlled through its wireless remote, which travels inside the vehicle along with the other control electronics. Velocity control is achieved by removing the two potentiometers controlling the vehicle's propel and steering and replacing them with resistor sets which mimic certain desired speeds and directions. These resistor sets are activated via Port A of the 8051 microcontroller, which is running the control software. The microcontroller also interfaces with a Garmin® 18PC GPS receiver though its auxiliary serial port. The user can enter waypoints and start/restart/reset the navigation of the vehicle via the two push buttons that are connected to the microcontroller's interrupt pins, INT0 and INT1 are used. The microcontroller and GPS unit are powered via two 9V batteries connected in series. The remote and relay/resistor sets are powered via one 9V battery. A 7.2V NI-CD battery powers the driving and steering motors of the vehicle.

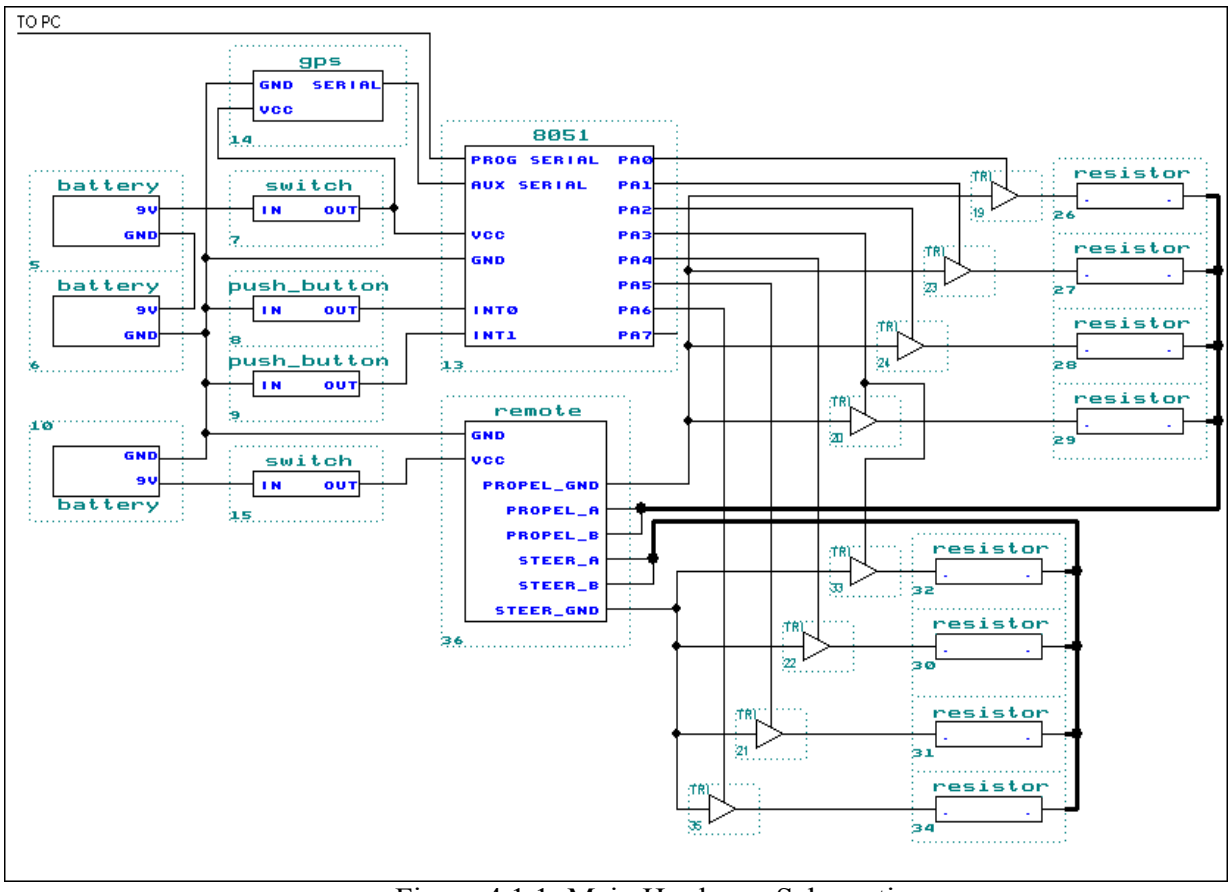

Figure 4.1.1: Main Hardware Schematic

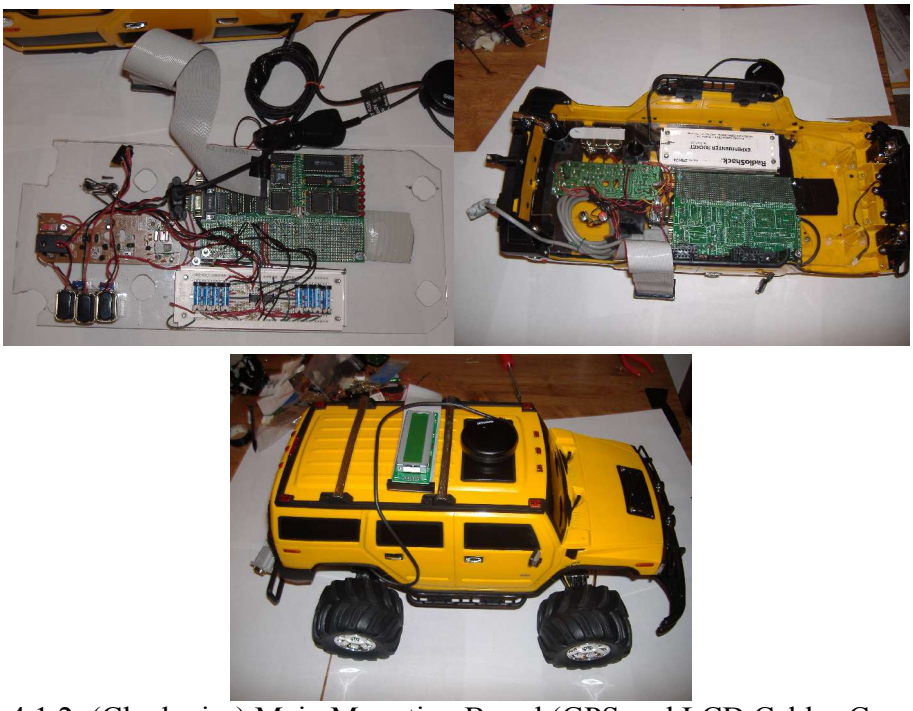

Figure 4.1.2: (Clockwise) Main Mounting Board (GPS and LCD Cables Connected), Main Mounting Board Installed in the Vehicle Body, Fully Assembled Vehicle

System Software

The software controlling the vehicle follows the general state diagram shown below in Figure 4.2.1.

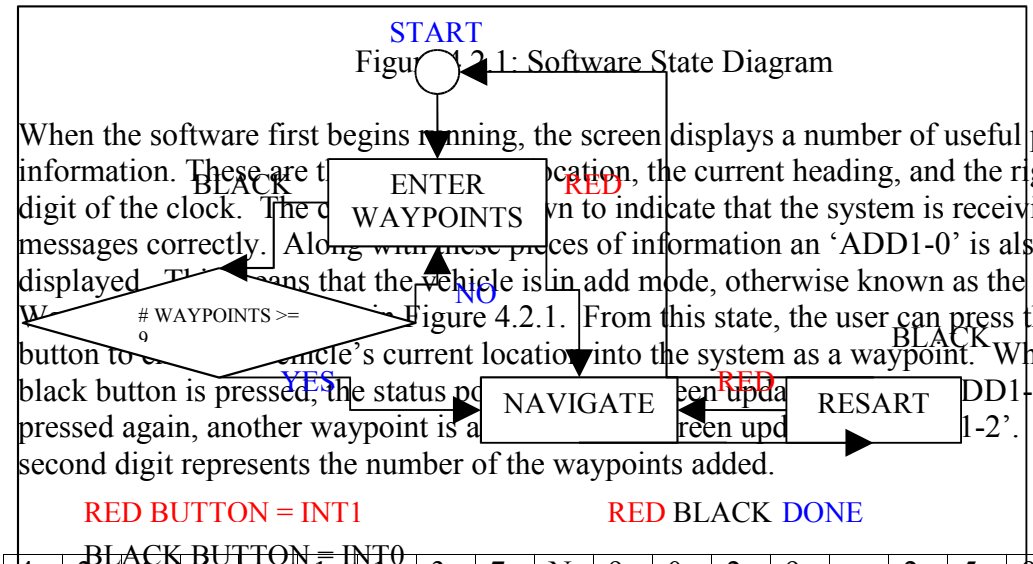

Figure 4.2.1: Software State Diagram

When the software first begins running, the screen displays a number of useful pieces of information. These are t[illegible]cation, the current heading, and the right most digit of the clock. The c[illegible]vn to indicate that the system is receiving messages correctly. Along with these pieces of information an 'ADD1-0' is also displayed. This means that the vehicle is in add mode, otherwise known as the "Enter Waypoints" mode in Figure 4.2.1. From this state, the user can press the black button to enter the vehicle's current location into the system as a waypoint. When the black button is pressed, the status portion of the screen upda[illegible]DD1-1'. If pressed again, another waypoint is a[illegible]reen upd[illegible]1-2'. The second digit represents the number of the waypoints added.

4	2	4	4	.	1	1	3	7	N	9	0	2	9	.	2	5	9	9	W
5		N	E		1	8	0	.	5		A	D	D	2	-	1		F	L

Figure 4.2.2: Example LCD Display

After the desired number of waypoints has been entered, pressing the red button will cause the vehicle to go into navigate mode as depicted above. The vehicle will wait 5 seconds before beginning navigation in order to allow the user to place the vehicle on the ground. At this point, the screen will also be updated with the vehicle's current bearing. This is the direction it must go in order to reach the first of the waypoints. How the heading change will be carried out is also displayed. For example, if the vehicle had a heading of 180 degrees, and a bearing of East, then it would need to move forward and turn left in order to face East, so 'FL' (forward, left) would be displayed.

When placed in 'NAV' mode the display will as change from 'ADD' to 'NAV'. Once the vehicle has navigated to the first waypoint, the display will change from reading '1-2' to '2-2' meaning that it is attempting to reach the second of two waypoints.

Once the vehicle has reached the final waypoint it will automatically shift into neutral and stop. At this point it goes into the 'RST' state. This stands for restart; it gives the user the option of pressing the red button and navigating through the same waypoints again, or pressing the black button and entering new waypoints.

It is important to note, that if any button is pressed during 'NAV' mode, the vehicle will automatically move to the restart state. This allows the user to pick up the vehicle and stop the wheels from spinning without restarting the 8051 microcontroller.

Summary

In summary this project consisted of taking a standard remote controlled vehicle and adding a microcontroller and a GPS receiver to it. This allowed the vehicle to intelligently navigate from one waypoint to another autonomously. Therefore this project met our primary goal of developing a system which leveraged the functionality of many different subsystems, spanning the areas of software, electrical, and mechanical engineering into a coherent and useful system.

References

[1] *GPS Technologies and Alternatives,* from
http://www.rand.org/publications/MR/MR614/MR614.appa.pdf

[2] NMEA Standards and History from
http://www.nmea.org

[3] *NEMA DATA*, from
http://www.gpsinformation.org/dale/nmea.htm#GGA