GPS Gaucho: A GPS Navigating Autonomous Vehicle

Bradley University Department of Electrical and Computer Engineering

Senior Capstone Project
May 16, 2007

Peter Fattore
Andrew Neebel

Advisors:
Dr. In Soo Ahn
Dr. Donald S. Schertz

Abstract

The GPS Gaucho vehicle has been expanded for the last few years and now stabilized for integration of various functions necessary for autonomous navigation. The system consists of a low-level HCS12 system for power train and steering, an EBox-II Windows CE system for remote control and web interface, and a set of GPS receivers for differential GPS (DGPS). All these sub-systems interact through the RS-232 serial connection for efficient and universal communication. Wireless DGPS link between the base station and remote station is maintained via RF RS-232 transceivers controlled by the E-Box. The IEEE 802.11 which the E-Box is running on allows web pages to be served to allow universal client communication and the vehicle control.

Table Of Contents

Abstract 1

List of Figures 3

1. Introduction 4

2. System Description 5

2.1 Block Diagram 5

2.2 Functional Description 6

3. Design/Theory 7

3.1 Differential GPS 7

3.2 EBox-II/Windows CE 9

3.2.1 Building Process 9

3.2.2 Serial Port 10

3.2.3 *Ethernet/Networking* 10

4. Software 11

4.1 Overall Structure 11

4.2 Gaucho 11

4.3 Navigation Control System 13

5. Results 17

6. Conclusion 17

7. Future Work 18

References/Data Sheets 19

Appendix A: Gaucho Control System Beans Settings A1

Appendix B: Gaucho Control System User Code B1

Appendix C: Navigation Control System Code C1

List of Figures

Figure 1 – System Block Diagram of GPS Gaucho 5

Figure 2 – Block Diagram of Differential GPS 7

Figure 3 – Screenshot of OS Deployment Options 10

Figure 4 – Flowchart of Main Gaucho Control Loop 11

Figure 5 – Flowcharts of Movement and Steering Controls 12

Figure 6 – Flowchart of Gaucho A/D Converter Interrupt 13

Figure 7 – Flowchart of Incoming Request Process 14

Figure 8 – Screenshot of User Web Interface 15

Figure 9 – Flowchart of Monitor Process 15

1. Introduction

Our world is becoming automated. Autonomous navigation is a necessity for any device that needs to operate without human interaction. GPS is a fantastic way to drive a vehicle, but alone is not accurate enough to be useful. An application of Differential GPS will be needed to create a working feasible design. Differential GPS is a system where a GPS receiver uses data from a fixed base station to provide much more accurate positioning information. The goal of this project is to build a system that uses Differential GPS as part of an autonomous navigation system. The vehicle the system is running on will be capable of identifying its current location and, based on that information, plot the correct path to another location that is specified by the user through the web interface.

This was the initial intent of the project, but due to a number of setbacks and issues, this objective was not accomplished. The end result of this project is a system that is capable of being controlled remotely through a web interface and being told to execute preprogrammed sequences. There is a Differential GPS system available on the vehicle, but due to limitations encountered in other parts of the system, it has not been integrated into the system.

2. System Description

2.1 Block Diagram

The GPS Gaucho consists of four main parts as shown in Figure 1 below. These parts are the Differential GPS, Web Interface for user control, the Gaucho and its internal control systems and the Navigation System that runs on the EBox A RS-232 serial interface is used to send location data from the Differential GPS to the EBox and from the EBox to the Gaucho. The User Interface runs from a web server on the EBox and is accessed using a web browser.

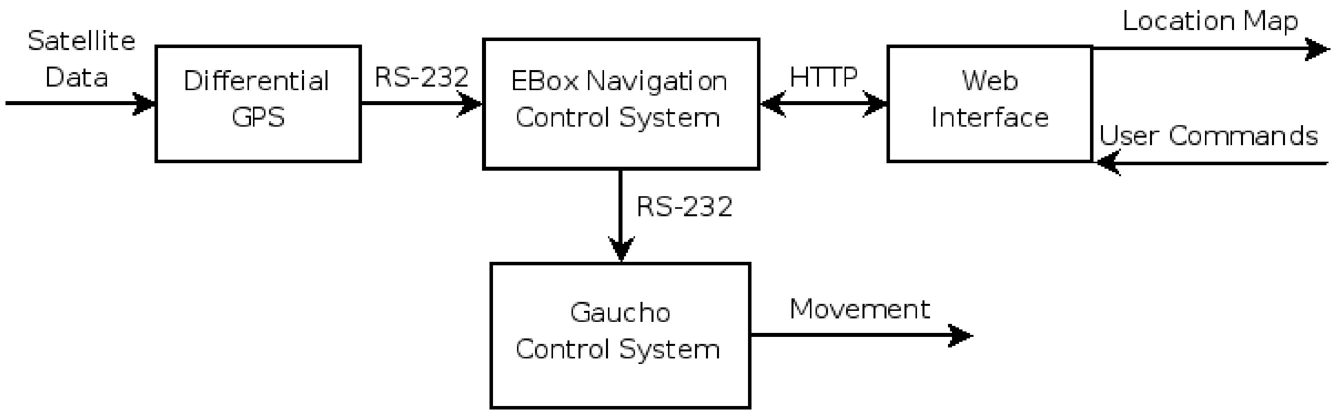


Figure 1 - System Block Diagram of GPS Gaucho

The Differential GPS (DGPS) part of the system consists of two Novatel RT-20 GPS Receivers. These units are capable of being programmed as either a DGPS receiver or base station. One of the units is configured as a base station and provides correction data to the other unit which acts as the receiver. The receiver unit then provides accurate positioning data to the Navigation System which enables it to perform its task in a more precise and reliable fashion.

The Web Interface is provided from a web server that runs as part of the Navigation System. A web-based interface was chosen since this allows for user control from any environment that is capable of running a web browser, be that a computer with Mac OSX, Windows, Linux, or other environments. This allows for a much more flexible system that requires very little in terms of requirements at the user level.

The Gaucho is the vehicle platform that the system resides on. There is a microprocessor mounted on it that takes commands from the Navigation System via a RS-232 serial interface and translates them to appropriate actions in the drive motor and steering actuator. It was initially believed that the control portion of the Gaucho was completed and could be used from the beginning. However, this assumption proved to be false and the entire Gaucho Control System ended up having to be written from scratch. This took a significant portion of time, and as a result, many higher-level tasks that were originally intended to be accomplished were not.

The Navigation System itself is the central part of the project, and, as such, is also the part with the most complexity. This part communicates with all of the other parts of the system through various protocols and, depending on the inputs it receives from the DGPS and Web Interface, tells the Gaucho where to go. This portion of the system was supposed to be able to plot a straight line course to one point and then execute a turn to go to another point. However, because of setbacks with both the Gaucho and the EBox itself, the resulting functionality is much less. Currently, the Navigation System takes remote control commands from the Web Interface and passes them on to the Gaucho and is capable of being programmed with a sequence of commands to execute upon being told to by the Web Interface.

2.2 Functional Description

The Gaucho will be capable of identifying its current location, which becomes the starting point, through the use of the Differential GPS component. When the user identifies a destination to go to, the Navigation Control System can use the starting point and the identified location to calculate the path to follow. For the purposes of our system, this path will be a straight line from the starting point to the end point. Eventually, the system should be capable of calculating and driving from one point to an intermediate point, and then execute a turn to go to a third point. If at any time the Differential GPS quits functioning correctly, then the system will cease performing self-navigation calculations and stop. At this point, the only way to control the system will be through a remote control interface. Having remote control provides a way for the user to guide the Gaucho away from any obstacles that may be blocking the GPS signals. The User Interface will consist of a set of web pages that are served from a server running on the EBox This interface will allows users to either identify a location to navigate to or send remote control commands to the Gaucho.

3. Design/Theory

3.1 Differential GPS

Any type of autonomous vehicle needs precise driving. Typical GPS is inadequate for stringent positioning requirements to the centimeter accuracy. Normal public GPS is accurate to about 20 feet. Worst case, a vehicle wobbling within this gap is unacceptable. A better system needs to be used, this is where Differential GPS (DGPS) comes into play.

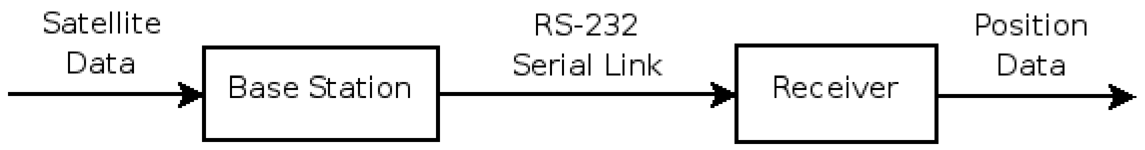


Figure 2 - Block Diagram of Differential GPS

As shown in Figure 2 above, Differential GPS is a two part system consisting of a base station and remote station. The base station is at a fixed location. It compares real time data to its known location and generates correction coordinates (RT20). The remote station receives this information, applies the correction data, and outputs corrected coordinates in standard GPS notation (Proprietary, NMEA, RTCM, RTCA). These systems individually will be described.

Using the two GPS unit system, accuracy is increased. The technical specifications say that after 3 minutes of use, 20 cm accuracy is obtained and after 10-20 minutes of use, 3-4 cm accuracy is in play. Both the base station and the remote station must maintain 4 satellites in their view to maintain the RT20 DGPS system. Also, keeping the remote station still for any length of time will improve the results from the accuracy algorithm. The log commands listed later on below build the DGPS system called RT20.

Communication to and from these units involves serial communication. The setting for this communication is 9600 Baud, 8 bits, 1 stop bit, no parity and no flow control. Though this is not fast by today's standards, it is adequate for our GPS communication. Each GPS unit has two serial ports, either can be used for programming, sending data, or receiving data. The commutations that will be described between units use RF serial transceivers. The basic antenna allows an approximate 1 block radius for communication. This can be increased with higher gain antennas. For our testing, it was acceptable for the time being. Since design on all subsystems was split into components, a more powerful RF serial transceiver can be applied. The base station was deployed from an antenna on the top of Jobst Hall. The fixed position of this antenna has been set to:

- 40.699629905355 North Latitude
- 89.61684333092 West Longitude
- 198.75 meters Altitude

This was determined from averaging data over a few week period of time. Though this is not 100% accurate, it was appropriate for the time limitations and applications for our project. The station is able to output precise coordinates, but not necessarily accurate. The position is fixed on the base station by using the following command:

```
Fix position 40.699629905355 -89.61684333092 199 base0
```

Once the position is fixed, logs needed to be sent to the remote station. There are two main logs and a third optional log: RTCM3, RTCM59, and RTCM. RTCM3 reports the fixed location that was entered into the base station every 10 seconds. RTCM59 reports the base station's satellite observation data every two seconds. This log is limited to 12 satellites. The RTCM log is used for a quick 1 meter position for a worst case scenario. This is transmitted every 5 seconds. These transmitted logs are setup on the base station with the following commands:

```
Log com2 rtcm3 ontime 10
Log com2 rtcm59 ontime 2
Log com2 rtcm ontime 5
```

In order to prevent a user from having to enter all of these commands every time the unit powers up, the GPS receivers utilize a built-in flash memory that the configuration can be saved to. If there is a saved configuration, it is then automatically loaded upon power-up. After entering the previous 4 commands, the next command is used to save the data.

```
Saveconfig
```

If an error was made, or if a completely new configuration is needed. It is possible to wipe the flash memory with:

```
Creset
```

This command should be performed prior to the commands before *saveconfig* is called. The remote station is used to deliver the finalized differential GPS coordinates. Similar to the method to output logs, there is a method to receive logs. Since the remote station does the calculations, it accepts the data with a single command:

```
Accept com2 rt20
```

This is a universal command that will listen for all data and compile coordinates out of it. The more information it receives, the more accurate it will be. The sending of the additional RTCM log has no change in programming on the remote station due to the single command line to receive. Since all the processing is taken care of internally, all that has to be done is to set the output port. This will output standard GPS coordinates which are indistinguishable from normal coordinates aside from the accuracy of the data:

```
Log com1 P20A 1
```

Saveconfig should also be called again to make this system operate independently. Considering the application on a remote device, a collision or bump could disrupt power and reset the unit. The system should be able to operate again without the user inputting the data again to program.

3.2 EBox-II/Windows CE 6

The EBox-II encompasses all of the subsystems on the GPS Gaucho. It is a compact small form factor x86 computer. Due to its low power requirements, it is a great choice to run the high-end control system. The EBox-II supports a variety of operating environments from XP Embedded, CE, Linux, etc. For our use, we chose Windows CE 6. This operating system natively supports the .Net 2.0 compact framework. Previous versions required an additional install or build option for the operating system.

3.2.1 Building process

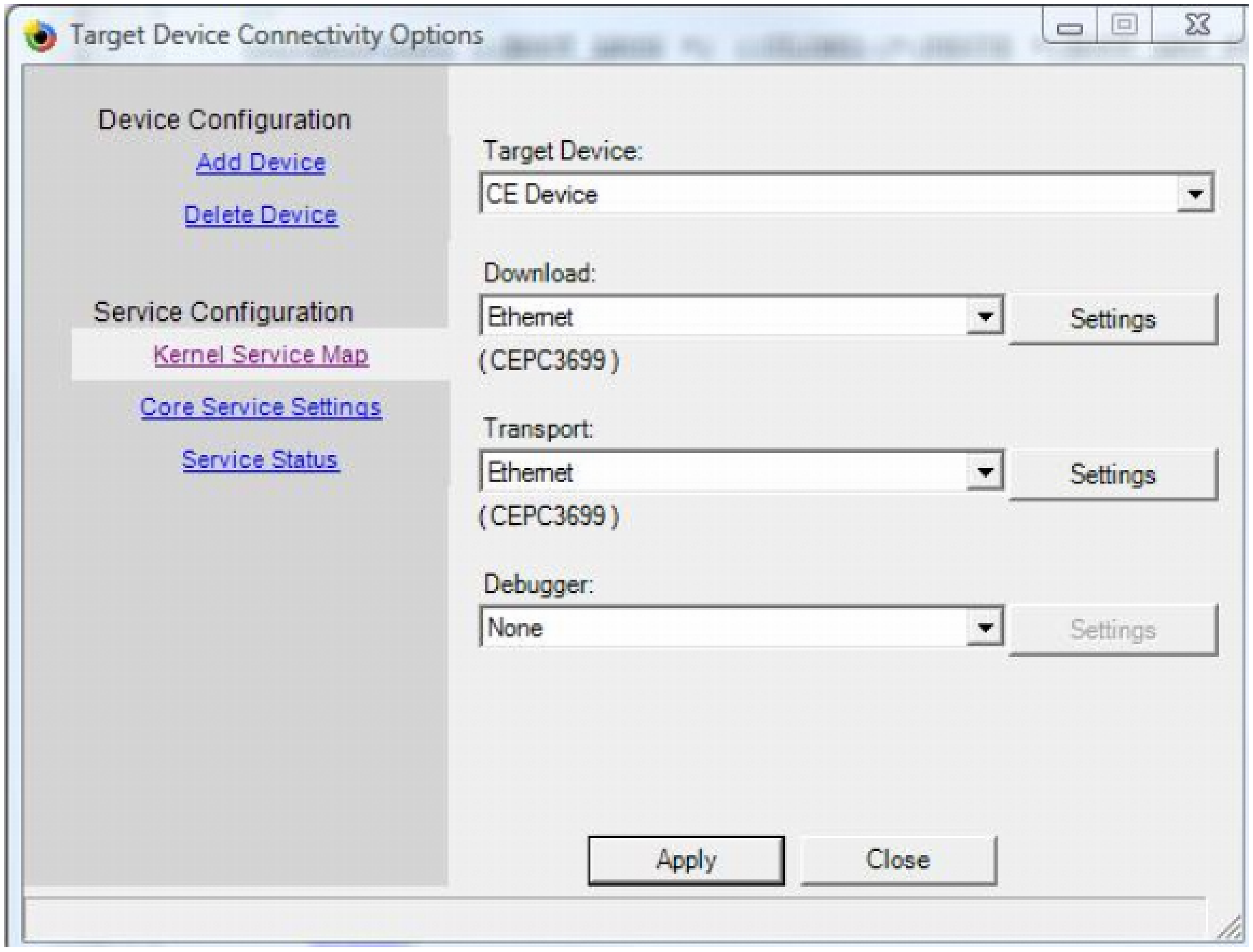
As with any .Net deployment, building and deploy is quick and easy. Complex network calls or serial data communications can be programmed with a few lines of code compared to raw C++ or P/Invokes. Using Visual Studio 2005 Service Pack 1, go to File> New Project. Select Visual C#>Smart Device>Windows CE 5.0. After selecting the appropriate name and locations and selecting OK, you can see the creation of the project. Once this is completed, it is possible to compile applications that will work on the Windows CE platform.

Similar to this process is how the CE operating system is built. Considering any embedded deployment of a system like this needs to have a small operating image size. While creating a new project, select Platform Builder. After the project creation, there is a 'Catalog Items View' that lists the optional components for the operating system. Some of the additions are as follows: USB Storage Class Driver, Ram and Rom File System, and Hive Registry. The USB Storage Class Driver allows USB flash drives to operate. The file system allows permanent and temporary data transactions. The registry option allows the registry to be written to the permanent storage so data can be saved and used after a cold boot.

In addition to configuring the separate modules, 2 environment variables need to be set: BSP_VS2005_CORECON and IMGRAM128. Each should have a value of 1. The first variable allows enhanced communication for deploying applications to the EBox-II. The second variable tells the CE operating system exactly how much RAM the device has. The CE image will fail to load if this is not set.

The deployment of an operating system build is a complex process. Within Visual Studio and the platform project is open, click on Target > Connectivity Options. It is necessary to select correct connectivity for the Download and transport. The option will be Ethernet. This will work as long as the devices are on the same network. When the EBox-II is put into eboot mode, the connectivity options will locate the EBox-II automatically if you click on settings. This is done through UDP discovery. The interface used for setting these options is shown on the next page in Figure 3.

Now that the setup is complete, it is possible to deploy the operating system. The EBox-II will request an image when it is in eboot mode. Then with Visual Studio, you can deploy by hitting F5 or click on Target>Connect Device to deploy the latest built image. The operating system image (nk.bin) is sent via TFTP and loaded on the EBox-II. To save the image for more permanent use, the nk.bin file should be moved to the root of the hard drive of the device. This can be done with the network deployed image via the GUI and transferring the nk.bin over USB flash drive or network share.


Target Device Connectivity Options

Target Device: CE Device

Download: Ethernet (CEPC3699)

Transport: Ethernet (CEPC3699)

Debugger: None

Figure 3 - Screenshot of OS Deployment Options

3.2.2 Serial Port

The serial port on the EBox-II is a standard computer serial port and can be addressed and called in the same fashion. This version of the EBox is limited to a single serial port. A new version (EBox-2300) has a second serial port. This would allow the EBox to pull GPS coordinates on one port and control the Gaucho on the other port.

3.2.3 Ethernet/Networking

There is an on board 10/100 Ethernet port to allow network communication. Since wired communication is not applicable to a mobile device, a Linksys wireless bridge was implemented. It simply was programmed to connect to the wireless SSID named ‘BUwireless’. Beyond that configuration, it acts as a simple pass-through and the connected device, which in this case is the EBox, is on the network.

4. Software

4.1 Overall Structure

There are two different sets of software on this system. The Gaucho contains an HCS12 microcontroller that processes commands sent through a serial interface and drives the steering actuator and drive motors correctly. This portion is programmed using Embedded C and Metrowerks Codewarrior 3.1. The second part of the system is the Navigation Control System itself, which runs on the EBox The control system is written using C# and the .NET Framework, and runs under Windows CE 6.0. The environment used to code this system is Visual Studio 2005. The two sets of software communicate back and forth through the use of a RS-232 serial interface, which is more than adequate for the small amount of data that needs to be sent to provide coordination.

4.2 Gaucho

As stated before, the Gaucho Control System is programmed using Embedded C. All of the initialization and hardware access methods were automatically generated by the Processor Expert beans that come with Codewarrior. This left only the actual identification and execution of commands to be written. The basic flow of the main loop of the Gaucho software is shown in the flowchart in Figure 4 below. Most of the system works through polling as shown below, but part of the system had to be implemented through an interrupt.

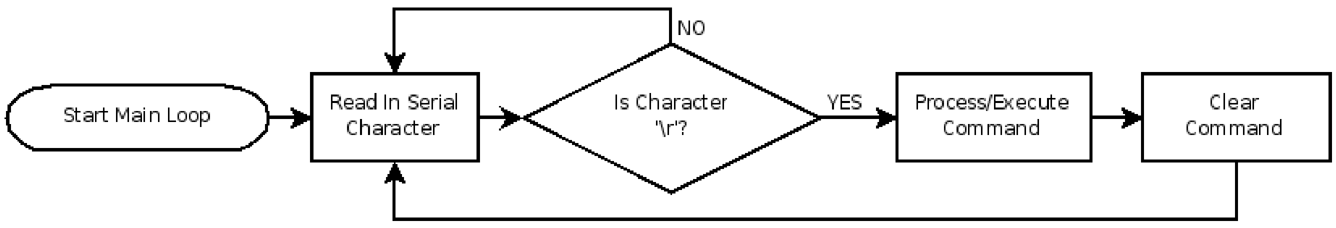


Figure 4 - Flowchart of Main Gaucho Control Loop

As shown above, the Gaucho constantly watches for new data from the higher level system or the user, depending on what is currently controlling it. As the gaucho receives letters of the command, it adds them to a string where it builds the command. Upon receiving the final character, which has been defined as being the escape sequence '\r', or a carriage return, the Gaucho then processes the string and takes the appropriate action. Currently, all of the commands are one character long, and the Gaucho decides what action to take based on what the the first character is. The commands that are currently supported by the Gaucho are:

- F for forward movement
- B for backwards (reverse) movement
- R for turning the wheels to the right
- L for turning the wheels to the left
- C for centering the steering
- H to halt the Gaucho
- X to kill the system (stop everything now)

The main execution loop is shown in Figure 4, but one part is left unclear – the process/execute command step. The different movements the Gaucho can make, forward/backward vs. turning, are carried out in different manners. Both of the drive motors and the linear actuator for steering are driving by pulse-width modulation (PWM) signals coming from the HCS12. However, the direction and on/off controls vary depending on which part of the system is being controlled. Figure 5 below shows the flowcharts for executing the forward and backward movement commands and the left, right, and center steering commands.

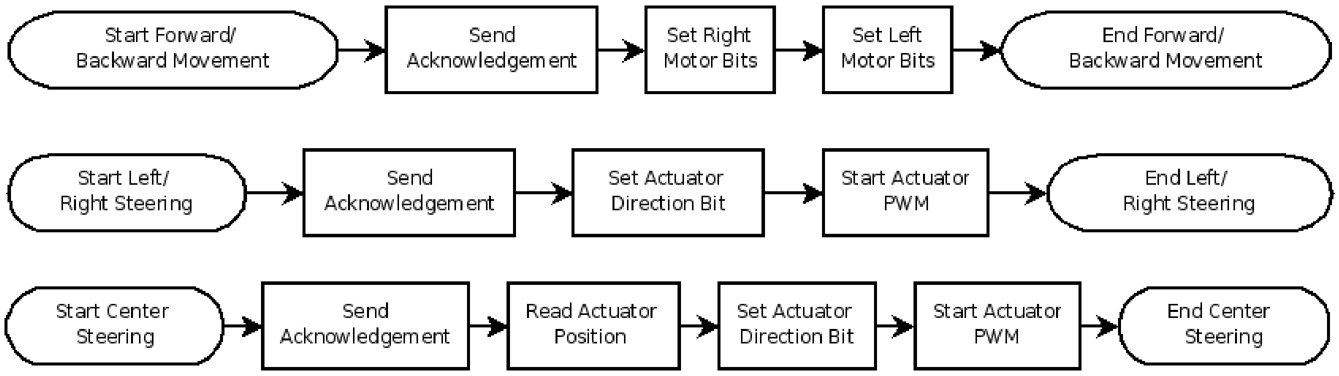


Figure 5 - Flowcharts of Movement and Steering Controls

In the case of the drive wheels, the PWM signals only controls the speed, while on/off and direction are controlled by a pair of bits, one for each direction, that are set and cleared. Each drive motor has it's own pair of bits and PWM signal. When both bits are 0, the motors are off, while they drive in one direction or the other if either bit is set. If both bits are set, it causes a short across the motor terminals, which implements a braking action in the motors. However, this braking behavior has not been used at all yet, and the Gaucho instead just coasts to a stop when the motors are turned off. If desired, each motor can be driven separately, but in order to make things simpler, the drive motors are driven in tandem.

The steering system is slightly different. While there is still a bit output that controls the direction the actuator moves, instead of turning the actuator on and off with the bit, the power is controlled directly from whether or not the PWM signal is running. When the pulse is not running, then the actuator doesn't move and then when the PWM signal is started the actuator is driven. Due to the limited range of the steering, there is little reason to have a variable speed control, and so the actuator is always driven at the same speed. There is also a slight difference between the left/right steering and the center steering. If the wheels are being centered, the system first has to check to see if the the wheels are right or left of center, and the based on that set the actuator bit the correct direction.

As stated before, part of the system was implemented through use of an interrupt. One of the problems that was encountered was in the steering system where the system had to be able to both start and stop the actuator. This had a tendency to prevent the Gaucho from responding to other commands until the steering actuator finished moving. The program flow of the interrupt is shown on the next page in Figure 6. Upon the interrupt firing, the system looks to see if the steering wheels are being moved to a particular direction. If so, the system checks to see if the actuator has reached the corresponding position, upon which it stops moving the actuator. If neither condition is true, then the interrupt exits and waits for the next interrupt to fire.

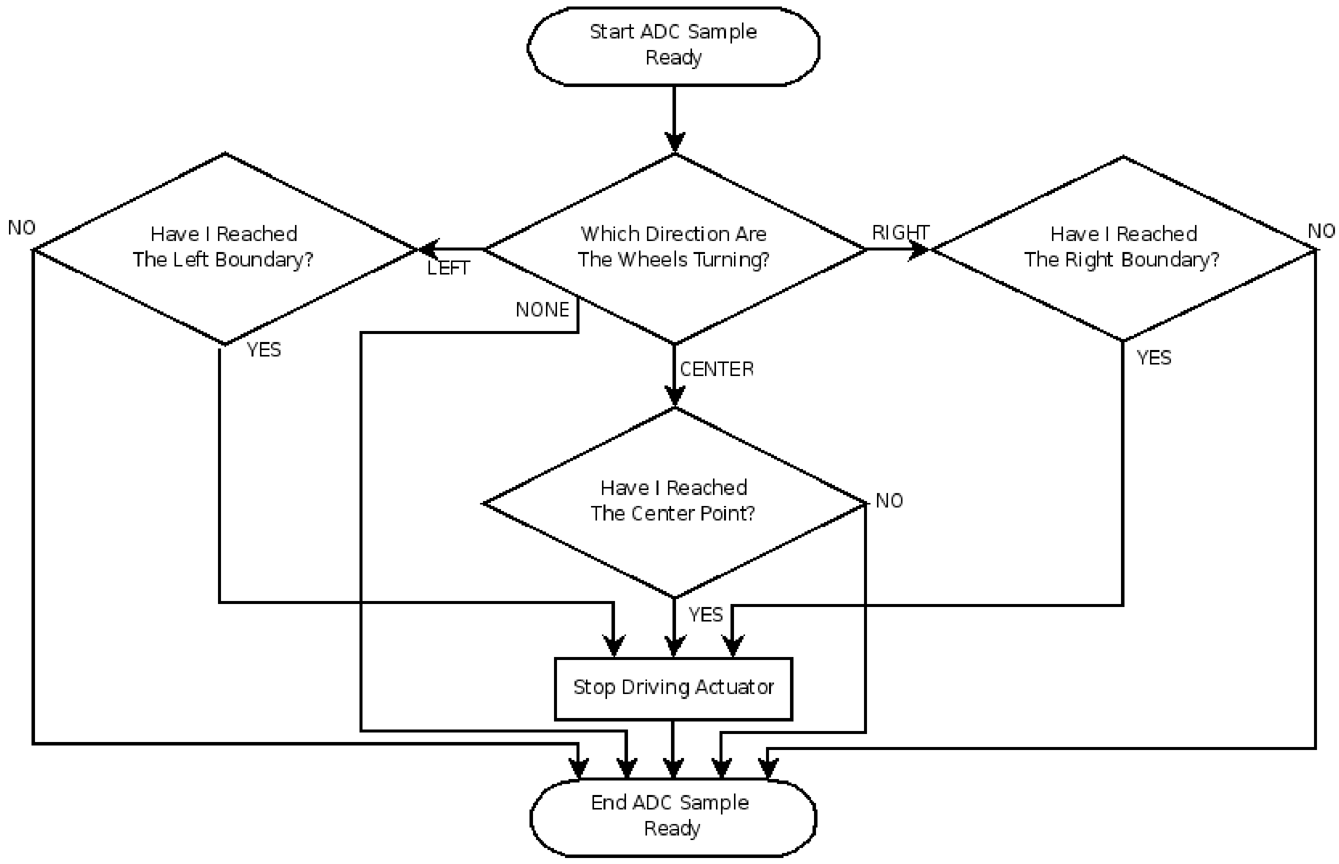

Start ADC Sample Ready
Which Direction Are The Wheels Turning?
LEFT
RIGHT
NONE
CENTER
Have I Reached The Left Boundary?
Have I Reached The Right Boundary?
Have I Reached The Center Point?
NO
YES
NO
YES
NO
YES
Stop Driving Actuator
End ADC Sample Ready

Figure 6 - Flowchart of Gaucho A/D Converter Interrupt

The string building behavior of the system was built to allow for future extensibility. Commands are capable of being modified to allow for more precise control of the Gaucho. An example of this is in the forward and reverse movement commands. Currently forward and reverse use a fixed speed that has been hard coded. This has led to problems where speeds that are almost too fast inside are barely able to start the Gaucho moving outside. This can be improved upon by adding a number to the command that the Gaucho can use to determine the speed. An example of this could be using numbers from 1 to 9, where 1 is the lowest power that can move the Gaucho and 9 is the fastest safe speed.

4.3 Navigation Control System

The Navigation Control System is what does the majority of the work. It runs under Windows CE using the .NET 2.0 Compact Framework. There are two main languages that are used to program .NET applications, VB.NET and C#. For this system, C# was chosen because it is similar to C, and easier to work with in an environment where C and C++ are the common languages. The other advantage of using .NET 2.0 and Windows CE 6.0 together is that they both use a common environment for writing software and building the OS images – Visual Studio 2005.

Part of the core of the Navigation Control System is based on a simple HTTP server. In order to provide a User Interface that can be used on any platform, the interface was implemented as a web page, which therefore requires a web server. A simple HTTP server that is capable of serving up static pages is fairly simple. HTTP is a text-based protocol, so by reading in lines of text and parsing them, the data sent by and wanted by the client is easy to determine. Replies are also fairly easy to generate since most of the fields needed are static and can be hard-coded into the server. The program flow of the web server is shown in Figure 7 below.

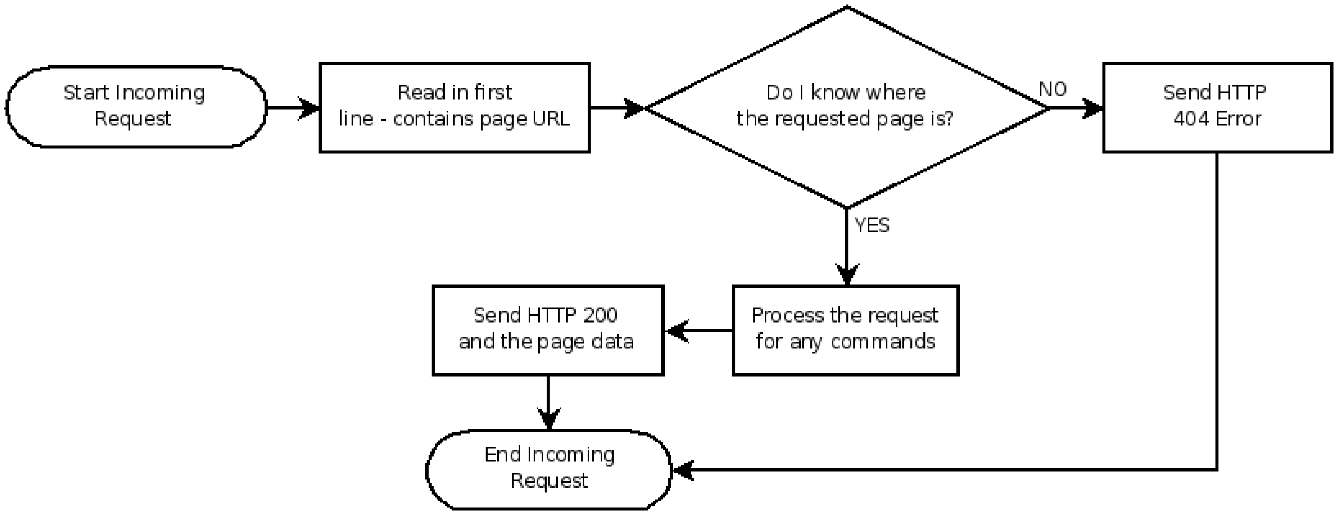

Start Incoming Request
Read in first line - contains page URL
Do I know where the requested page is?
NO
Send HTTP 404 Error
YES
Process the request for any commands
Send HTTP 200 and the page data
End Incoming Request

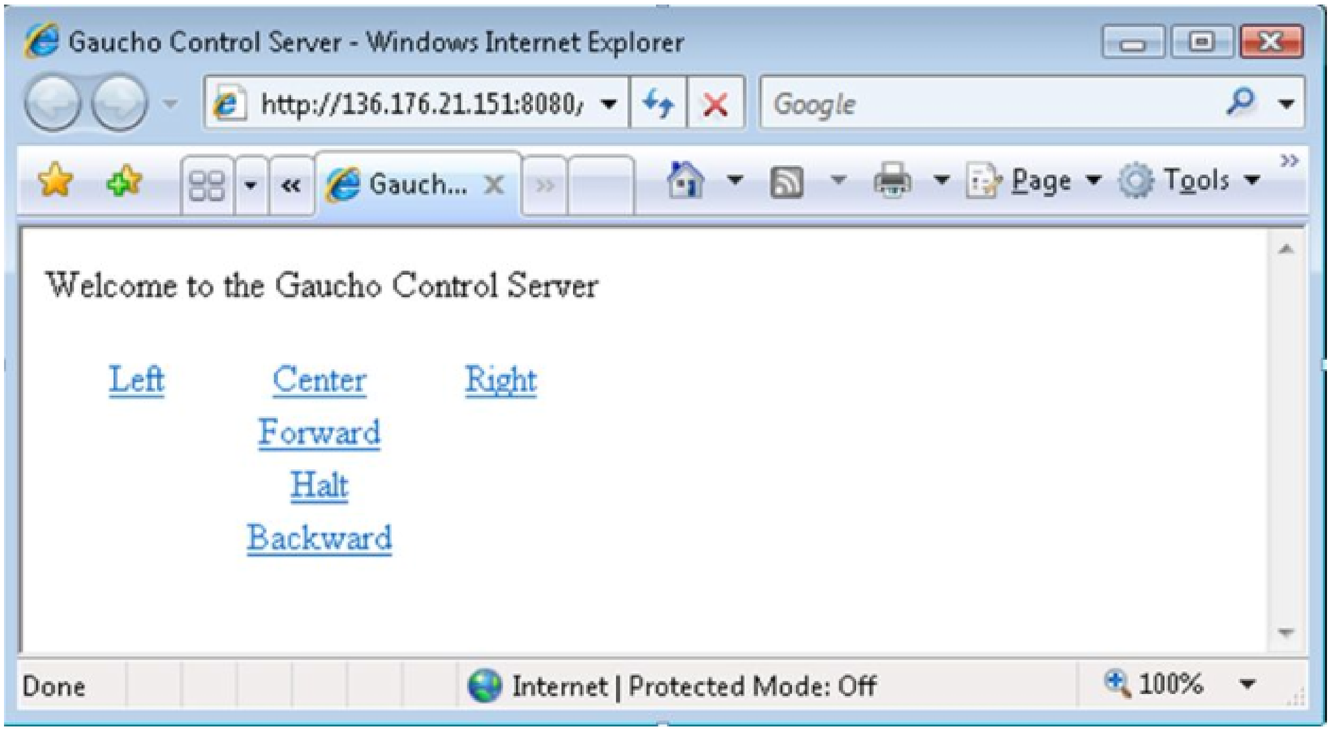
Figure 7 - Flowchart of Incoming Request Process

In order to make things as simple as possible, the web server is designed to use the name of the page being requested as the command from the user. Currently, the only capability that the entire system has is remote navigation. Therefore, the only commands that the user sends are commands to directly control various movement functions of the Gaucho. These commands were implemented based on the URL of the page being requested from the server. The pages and corresponding commands are listed below:

- left.html to turn left
- center.html to center wheels
- right.html to turn right
- forward.html to go forward
- backward.html to go backward
- halt.html to stop
- sequence.html to execute a preprogrammed sequence

There is also an index.html page that is a set of Hyperlinks laid out that execute various commands. This same page is also provided whenever one of the main movement commands is executed. The sequence page is different in that it instead builds and sends the page while the Gaucho is executing commands, which allows the user to be able to tell what the Gaucho is doing while it does it. A screenshot of the web page interface is shown in Figure 8 at the top of the next page.

The page shown in Figure 8 is a very basic interface to control the Gaucho, but it allows the user to direct the Gaucho anywhere that it needs to go, thus serving it's purpose. This interface was only built to provide a way to test other parts of the system and allow the Gaucho to be navigated away from obstacles in the case that something goes wrong completely. This functionality of the system should remain in existence because of it's usefulness even when the autonomous navigation is functional.


Gaucho Control Server - Windows Internet Explorer
http://136.176.21.151:8080/
Google
Gauch...
Page
Tools
Done
Internet | Protected Mode: Off
100%

Welcome to the Gaucho Control Server

Left Center Right
Forward
Halt
Backward

Figure 8 - Screenshot of User Web Interface

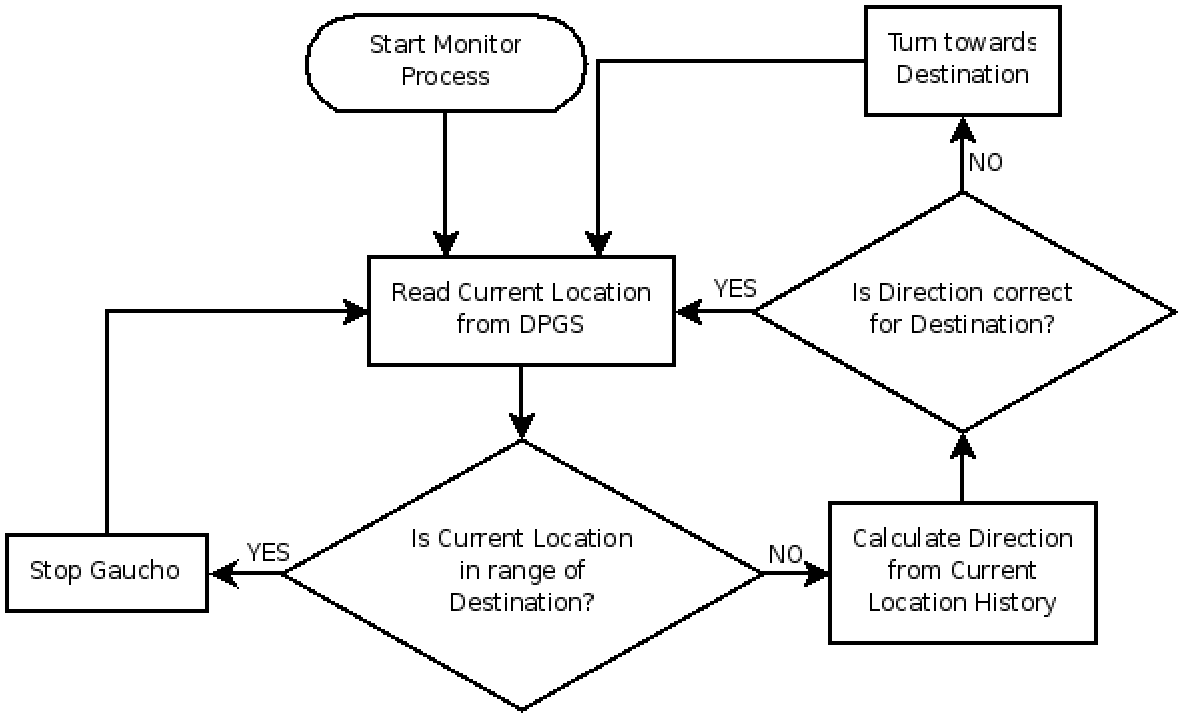

Start Monitor Process
Turn towards Destination
NO
Read Current Location from DPGS
YES
Is Direction correct for Destination?
Stop Gaucho
YES
Is Current Location in range of Destination?
NO
Calculate Direction from Current Location History

Figure 9 - Flowchart of Monitor Process

The other part of the system is a background process that runs constantly and compares the current location from the DGPS to the desired location. It also calculates direction and speed based on the past several DGPS locations, which allows it to tell if the Gaucho is heading towards or away from the target location and how long it is going to take to reach it. If the direction is not correct to get the Gaucho from the Current Location to the Destination, then the system instructs the Gaucho to turn towards the Destination until it is pointing the correct direction. This process flow is shown in Figure 9 on the previous page.

Unfortunately, due to problems encountered in other parts of the Autonomous Vehicle system that caused delays, this part of the system was never implemented. Due to this, beyond the basic structure shown in Figure 9, very little of this part of the GPS Gaucho system has been identified and designed.

5. Results

The goals of the project were not completely met. It was intended that the Gaucho would be capable of driving around the Quad on its own when this project is over, but currently, the only functionality that exists is that to remote control the Gaucho through the web interface. While the DGPS has been researched and implemented, it has not been possible to integrate it with the rest of the system yet. This is because there is only one serial port on the EBox, making it impossible to connect two devices. It was originally intended to overcome this problem through use of a USB-serial dongle, however, the Windows CE drivers for these types of devices do not want to recognize the dongle as being supported.

Another major problem that was encountered was in the Gaucho itself. Last year there was a project to build hardware and software for the Gaucho that could be used by projects without dealing with any of the low-level control systems. That project was only partially successful in that the hardware was well-built and integrated together nicely, but the software was unusable. Much of the early development time of this project was spent first trying to test and make the old software work before eventually deciding that new software needed to be written completely from scratch. This led to the design and building of the system described in the Gaucho Software section above.

6. Conclusion

This project encountered many stumbling blocks which eventually led to it not being completed according to the original goals. This is not to say that the project wasn't successful in some way. Many of the problems that were encountered were due to things outside of our control, the best example of this being the Gaucho. When we started this project in the fall, we were led to believe that the Gaucho's control system was completely finished and functional. However, since it turned out that the only functional part of the Gaucho was the hardware, we ended up being set back considerably, both from trying to get the old code to run on the Gaucho, upon which we learned that the code was unusable, and from development of new code that would work correctly. Because of this problem and the time spent solving it, additional problems were compounded. By the time that we learned that Windows CE was not going to support our USB-serial dongle, it was too close to the end of the semester to try and get hardware that would allow us to bypass this problem. This has led us to the point where it is not possible to implement any form of GPS usage, either for navigation or reporting, at this time. However, replacing the EBox with a newer version that does have multiple serial ports would eliminate this problem and allow for proper continuation of the project.

7. Future Work

This project has plenty of work that could still be accomplished on it. Future projects will be able to make use of and improve the Gaucho Control System in addition to now having a functional Differential GPS system. Some work that could easily be continued on this project includes:

- Replacement of the EBox with a better platform and development of the navigation system
- Differential GPS mapping capabilities
- Improvements to the Gaucho control system, such as obstacle detection, direction from a compass, and distance sensing and calculations

References/Data Sheets

1. 78M09A 9V Regulator Data Sheet

2. ICOP Windows Embedded CE 6.0 Jump Start Guide
   http://www.dsl-ltd.co.uk/software/ebox2300/eBox2300_CE6_Guide.pdf

3. Configuring a registry file to run an application at startup
   http://msdn2.microsoft.com/en-us/library/aa909369.aspx

4. Registry Types
   http://msdn2.microsoft.com/en-us/library/aa910532.aspx

5. File Systems and Storage Management Catalog Items
   http://msdn2.microsoft.com/en-us/library/aa916309.aspx

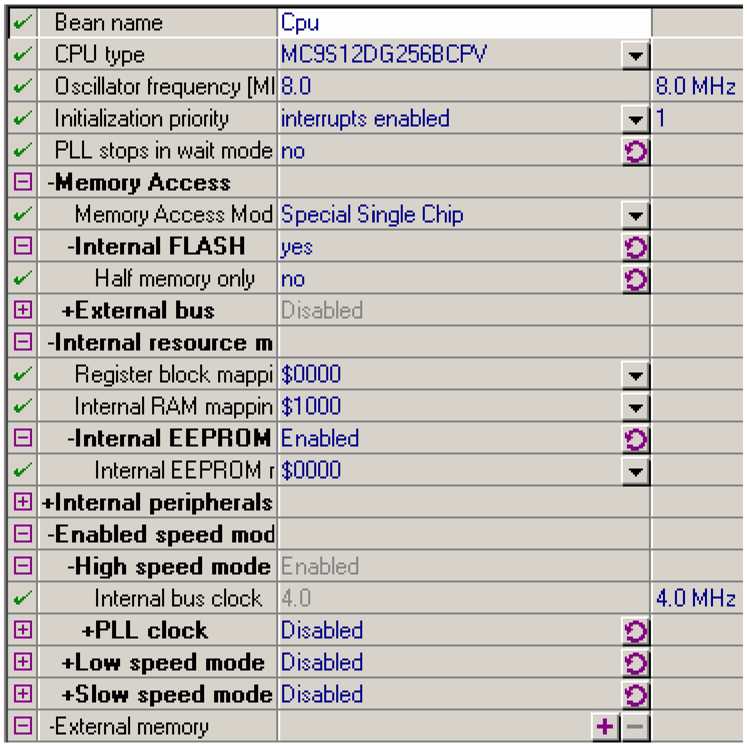
6. GPSCard Command Descriptions Manual
   http://www.novatel.com/Documents/Manuals/om-20000008.pdf

7. GPSCard Installation and Operating Manual
   http://www.novatel.com/Documents/Manuals/om-20000007.pdf

8. ProPak User Manual
   http://www.novatel.com/Documents/Manuals/om-20000011.pdf

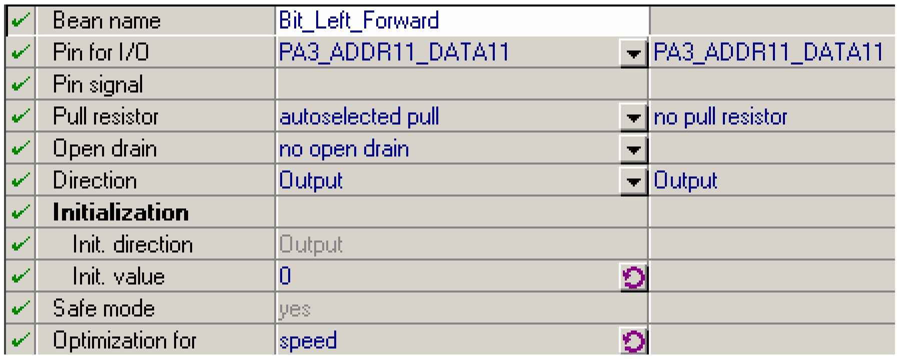
Appendix A: Gaucho Control System Beans Settings

CPU Bean:



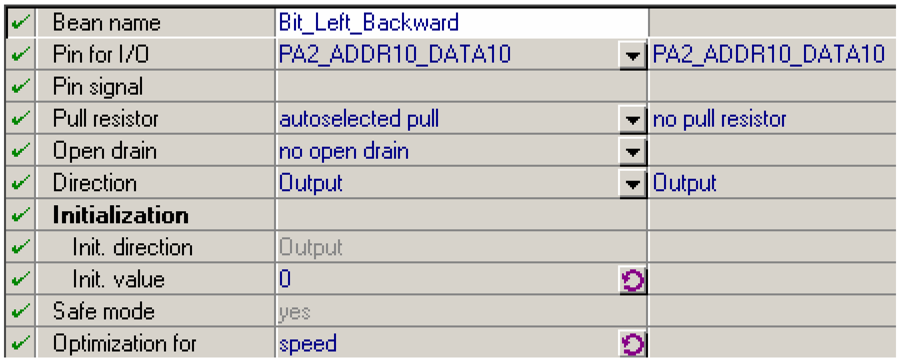
✓	Bean name	Cpu	
✓	CPU type	MC9S12DG256BCPV	
✓	Oscillator frequency [MI	8.0	8.0 MHz
✓	Initialization priority	interrupts enabled	1
✓	PLL stops in wait mode	no	
⊟	-Memory Access		
✓	Memory Access Mod	Special Single Chip	
⊟	-Internal FLASH	yes	
✓	Half memory only	no	
⊞	+External bus	Disabled	
⊟	-Internal resource m		
✓	Register block mappi	$0000	
✓	Internal RAM mappin	$1000	
⊟	-Internal EEPROM	Enabled	
✓	Internal EEPROM r	$0000	
⊞	+Internal peripherals		
⊟	-Enabled speed mod		
⊟	-High speed mode	Enabled	
✓	Internal bus clock	4.0	4.0 MHz
⊞	+PLL clock	Disabled	
⊞	+Low speed mode	Disabled	
⊞	+Slow speed mode	Disabled	
⊟	-External memory		

Left Motor Forward Pin:



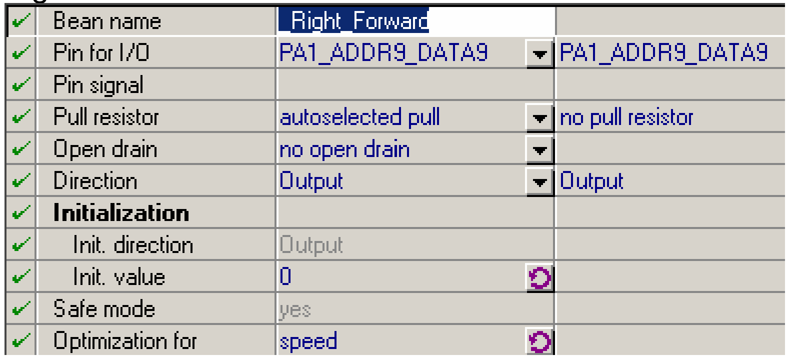
✓	Bean name	Bit_Left_Forward	
✓	Pin for I/O	PA3_ADDR11_DATA11	PA3_ADDR11_DATA11
✓	Pin signal		
✓	Pull resistor	autoselected pull	no pull resistor
✓	Open drain	no open drain	
✓	Direction	Output	Output
✓	**Initialization**		
✓	Init. direction	Output	
✓	Init. value	0	
✓	Safe mode	yes	
✓	Optimization for	speed	

Left Motor Backward Pin:



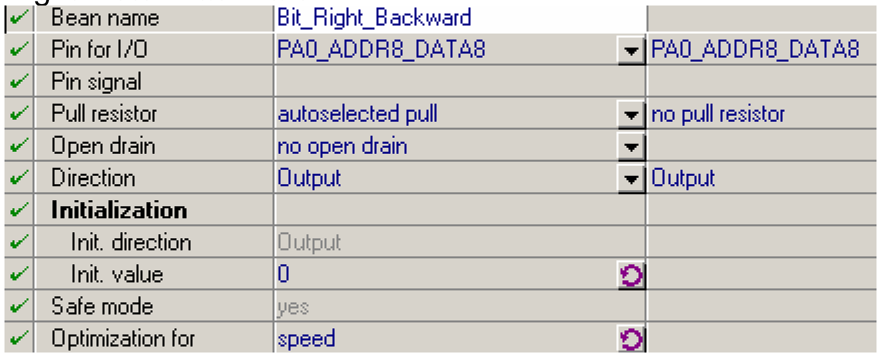
✓	Bean name	Bit_Left_Backward	
✓	Pin for I/O	PA2_ADDR10_DATA10	PA2_ADDR10_DATA10
✓	Pin signal		
✓	Pull resistor	autoselected pull	no pull resistor
✓	Open drain	no open drain	
✓	Direction	Output	Output
✓	**Initialization**		
✓	Init. direction	Output	
✓	Init. value	0	
✓	Safe mode	yes	
✓	Optimization for	speed	

Right Motor Forward Pin:



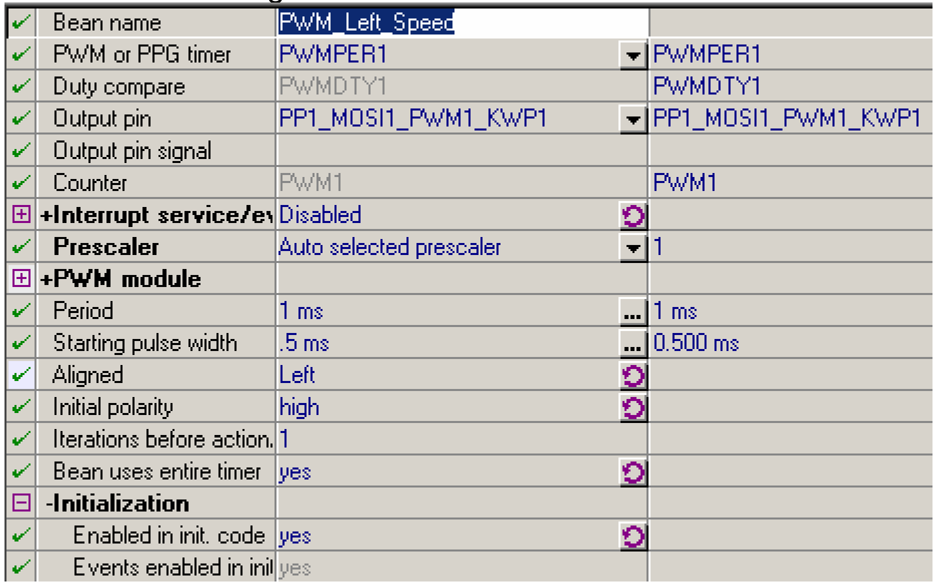
✓	Bean name	Right_Forward	
✓	Pin for I/O	PA1_ADDR9_DATA9	PA1_ADDR9_DATA9
✓	Pin signal		
✓	Pull resistor	autoselected pull	no pull resistor
✓	Open drain	no open drain	
✓	Direction	Output	Output
✓	**Initialization**		
✓	Init. direction	Output	
✓	Init. value	0	
✓	Safe mode	yes	
✓	Optimization for	speed	

Right Motor Backward Pin:



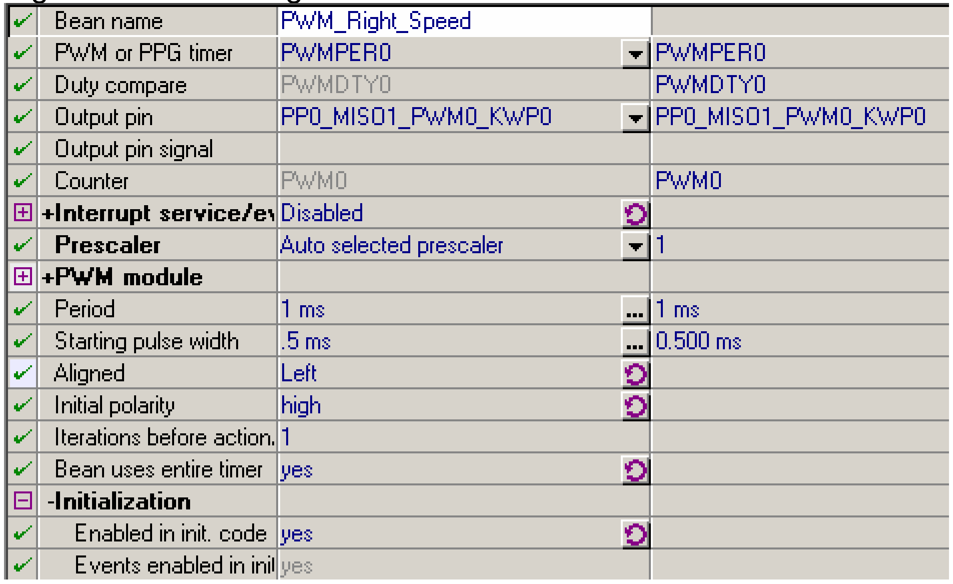
✓	Bean name	Bit_Right_Backward	
✓	Pin for I/O	PA0_ADDR8_DATA8	PA0_ADDR8_DATA8
✓	Pin signal		
✓	Pull resistor	autoselected pull	no pull resistor
✓	Open drain	no open drain	
✓	Direction	Output	Output
✓	**Initialization**		
✓	Init. direction	Output	
✓	Init. value	0	
✓	Safe mode	yes	
✓	Optimization for	speed	

Left Motor PWM Signal:



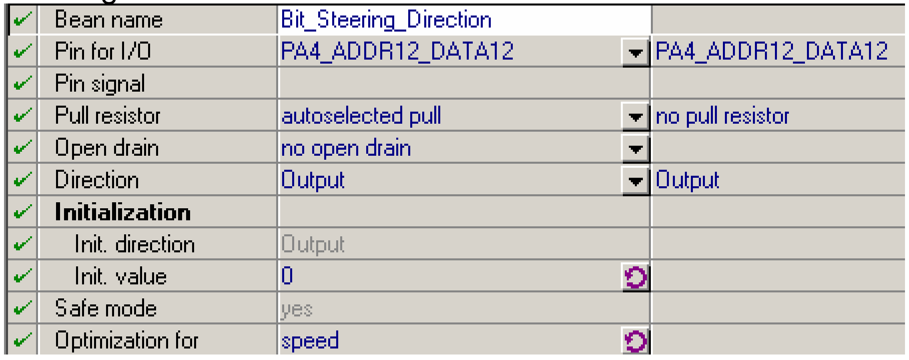
✓	Bean name	PWM_Left_Speed	
✓	PWM or PPG timer	PWMPER1	PWMPER1
✓	Duty compare	PWMDTY1	PWMDTY1
✓	Output pin	PP1_MOSI1_PWM1_KWP1	PP1_MOSI1_PWM1_KWP1
✓	Output pin signal		
✓	Counter	PWM1	PWM1
⊞	**+Interrupt service/ev**	Disabled	
✓	**Prescaler**	Auto selected prescaler	1
⊞	**+PWM module**		
✓	Period	1 ms	1 ms
✓	Starting pulse width	.5 ms	0.500 ms
✓	Aligned	Left	
✓	Initial polarity	high	
✓	Iterations before action	1	
✓	Bean uses entire timer	yes	
⊟	**-Initialization**		
✓	Enabled in init. code	yes	
✓	Events enabled in init	yes	

Right Motor PWM Signal:



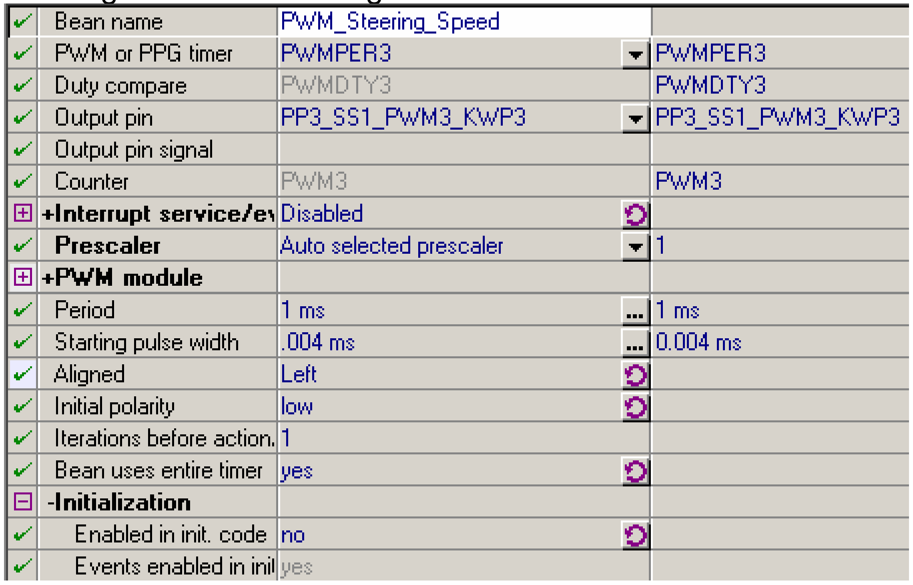
✓	Bean name	PWM_Right_Speed	
✓	PWM or PPG timer	PWMPER0	PWMPER0
✓	Duty compare	PWMDTY0	PWMDTY0
✓	Output pin	PP0_MISO1_PWM0_KWP0	PP0_MISO1_PWM0_KWP0
✓	Output pin signal		
✓	Counter	PWM0	PWM0
⊞	**+Interrupt service/ev**	Disabled	
✓	**Prescaler**	Auto selected prescaler	1
⊞	**+PWM module**		
✓	Period	1 ms	1 ms
✓	Starting pulse width	.5 ms	0.500 ms
✓	Aligned	Left	
✓	Initial polarity	high	
✓	Iterations before action.	1	
✓	Bean uses entire timer	yes	
⊟	**-Initialization**		
✓	Enabled in init. code	yes	
✓	Events enabled in init	yes	

Steering Actuator Direction Pin:



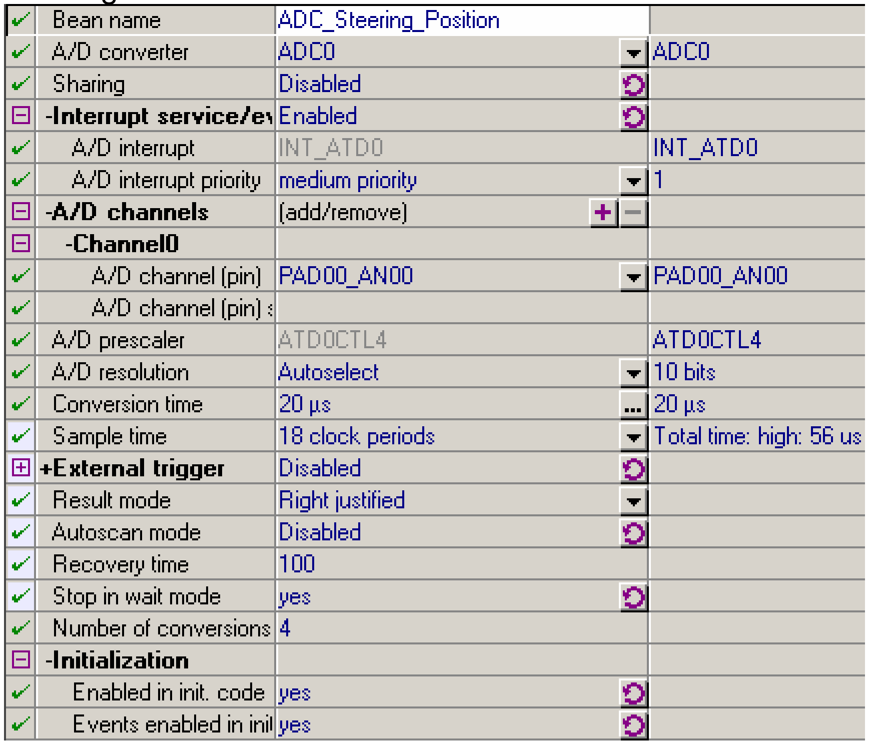
✓	Bean name	Bit_Steering_Direction	
✓	Pin for I/O	PA4_ADDR12_DATA12	PA4_ADDR12_DATA12
✓	Pin signal		
✓	Pull resistor	autoselected pull	no pull resistor
✓	Open drain	no open drain	
✓	Direction	Output	Output
✓	**Initialization**		
✓	Init. direction	Output	
✓	Init. value	0	
✓	Safe mode	yes	
✓	Optimization for	speed	

Steering Actuator PWM Signal:



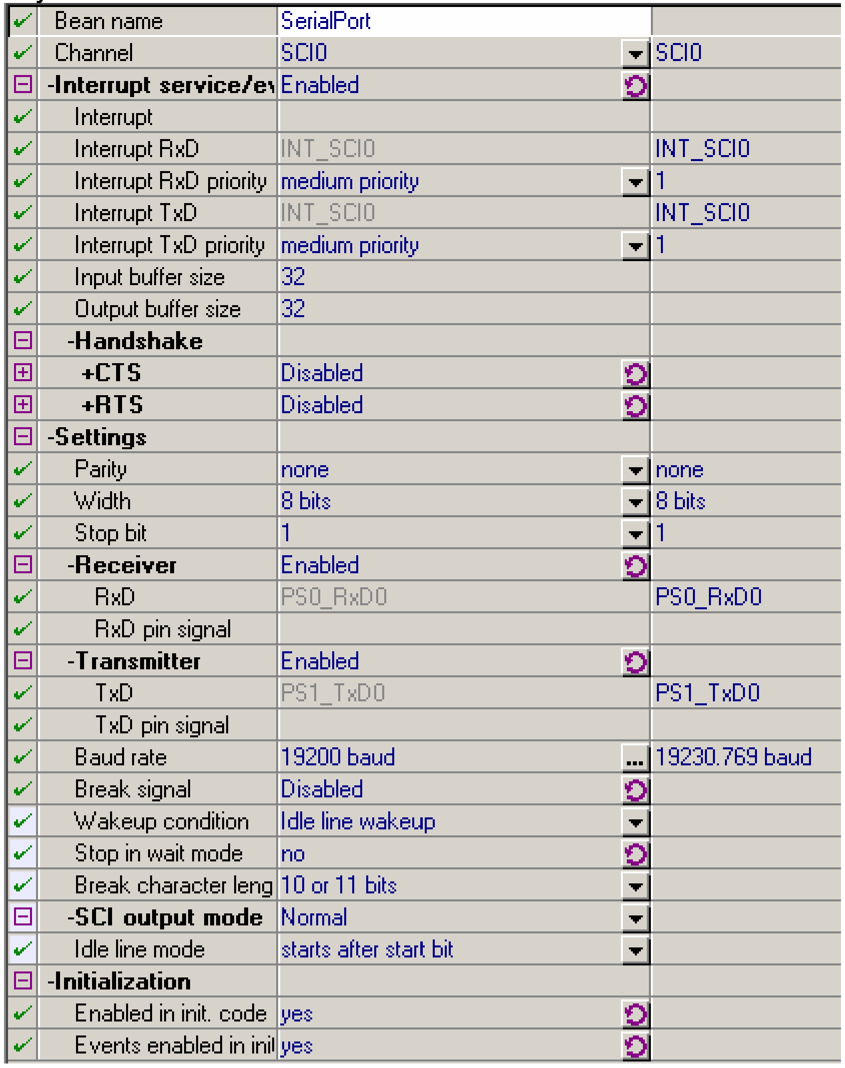
✓	Bean name	PWM_Steering_Speed	
✓	PWM or PPG timer	PWMPER3	PWMPER3
✓	Duty compare	PWMDTY3	PWMDTY3
✓	Output pin	PP3_SS1_PWM3_KWP3	PP3_SS1_PWM3_KWP3
✓	Output pin signal		
✓	Counter	PWM3	PWM3
⊞	**+Interrupt service/ev**	Disabled	
✓	**Prescaler**	Auto selected prescaler	1
⊞	**+PWM module**		
✓	Period	1 ms	1 ms
✓	Starting pulse width	.004 ms	0.004 ms
✓	Aligned	Left	
✓	Initial polarity	low	
✓	Iterations before action.	1	
✓	Bean uses entire timer	yes	
⊟	**-Initialization**		
✓	Enabled in init. code	no	
✓	Events enabled in init	yes	

Steering Actuator A/D Converter:



✓	Bean name	ADC_Steering_Position	
✓	A/D converter	ADC0	ADC0
✓	Sharing	Disabled	
⊟	-Interrupt service/ev	Enabled	
✓	A/D interrupt	INT_ATD0	INT_ATD0
✓	A/D interrupt priority	medium priority	1
⊟	-A/D channels	(add/remove)	
⊟	-Channel0		
✓	A/D channel (pin)	PAD00_AN00	PAD00_AN00
✓	A/D channel (pin) s		
✓	A/D prescaler	ATD0CTL4	ATD0CTL4
✓	A/D resolution	Autoselect	10 bits
✓	Conversion time	20 µs	20 µs
✓	Sample time	18 clock periods	Total time: high: 56 us
⊞	+External trigger	Disabled	
✓	Result mode	Right justified	
✓	Autoscan mode	Disabled	
✓	Recovery time	100	
✓	Stop in wait mode	yes	
✓	Number of conversions	4	
⊟	-Initialization		
✓	Enabled in init. code	yes	
✓	Events enabled in init	yes	

Asynchronous Serial Interface:



✓	Bean name	SerialPort	
✓	Channel	SCI0	SCI0
⊟	**-Interrupt service/ev**	Enabled	
✓	Interrupt		
✓	Interrupt RxD	INT_SCI0	INT_SCI0
✓	Interrupt RxD priority	medium priority	1
✓	Interrupt TxD	INT_SCI0	INT_SCI0
✓	Interrupt TxD priority	medium priority	1
✓	Input buffer size	32	
✓	Output buffer size	32	
⊟	**-Handshake**		
⊞	**+CTS**	Disabled	
⊞	**+RTS**	Disabled	
⊟	**-Settings**		
✓	Parity	none	none
✓	Width	8 bits	8 bits
✓	Stop bit	1	1
⊟	**-Receiver**	Enabled	
✓	RxD	PS0_RxD0	PS0_RxD0
✓	RxD pin signal		
⊟	**-Transmitter**	Enabled	
✓	TxD	PS1_TxD0	PS1_TxD0
✓	TxD pin signal		
✓	Baud rate	19200 baud	19230.769 baud
✓	Break signal	Disabled	
✓	Wakeup condition	Idle line wakeup	
✓	Stop in wait mode	no	
✓	Break character leng	10 or 11 bits	
⊟	**-SCI output mode**	Normal	
✓	Idle line mode	starts after start bit	
⊟	**-Initialization**		
✓	Enabled in init. code	yes	
✓	Events enabled in init	yes	

Appendix B: Gaucho Control System User Code

GauchoCommander.C

```c
/* MODULE GauchoCommander */

/* Including used modules for compiling procedure */
#include "Cpu.h"
#include "Events.h"
#include "SerialPort.h"
#include "PWM_Right_Speed.h"
#include "PWM_Left_Speed.h"
#include "Bit_Right_Forward.h"
#include "Bit_Right_Backward.h"
#include "Bit_Left_Forward.h"
#include "Bit_Left_Backward.h"
#include "PWM_Steering_Speed.h"
#include "Bit_Steering_Direction.h"
#include "ADC_Steering_Position.h"
/* Include shared modules, which are used for whole project */
#include "PE_Types.h"
#include "PE_Error.h"
#include "PE_Const.h"
#include "IO_Map.h"

#include "myVars.h"

// Sends a reply back to tell the host that it got the command
void sendAck(byte* message, word length) {
  word count;
  if (SerialPort_SendBlock("ACK:", 4, &count) == ERR_OK) {
    SerialPort_SendBlock(message, length, &count);
  }
}

// Sends a reply back to tell the host that it didn't understand the
command
void sendUnknown(byte* message, word length) {
  word count;
  if (SerialPort_SendBlock("UKN:", 4, &count) == ERR_OK) {
    SerialPort_SendBlock(message, length, &count);
  }
}

// Reads up to a newline character or max length
// ERR_OK returned if found newline
// ERR_OVERRUN returned if maxLength reached
// line with newline is in line, length is in count
byte readLine(byte line[], word maxLength, word* count) {
  byte character = 0;
  *count = 0;
  while (character != '\r') {   // The carriage return character code
    if (SerialPort_RecvChar(&character) == ERR_OK) {
      line[*count] = character;
      (*count)++;
      if (*count == maxLength) {
        return ERR_OVERRUN;
      }
    }
  }
  return ERR_OK;
}

// Drive the Gaucho forwards
void doForward(byte message[], word length) {
   sendAck("Forward\r", 8);
   Bit_Right_Backward_ClrVal();
   Bit_Right_Forward_SetVal();
   Bit_Left_Backward_ClrVal();
   Bit_Left_Forward_SetVal();
}

// Drive the Gaucho backwards
void doBackward(byte message[], word length) {
  sendAck("Backward\r", 9);
  Bit_Right_Forward_ClrVal();
  Bit_Right_Backward_SetVal();
  Bit_Left_Forward_ClrVal();
  Bit_Left_Backward_SetVal();
}

// Turn the wheels to the right
```

```
void doRight(byte message[], word length) {
  word meas_val = 0;

  sendAck("Right\r", 6);
  Bit_Steering_Direction_PutVal(0);
  PWM_Steering_Speed_SetRatio16(0x4000);
  PWM_Steering_Speed_Enable();

  TurningDirection = Turning_RIGHT;
}

// Turn the wheels to the left
void doLeft(byte message[], word length) {
  word meas_val = 0xFFFF;

  sendAck("Left\r", 5);
  Bit_Steering_Direction_PutVal(1);
  PWM_Steering_Speed_SetRatio16(0x4000);
  PWM_Steering_Speed_Enable();

  TurningDirection = Turning_LEFT;
}

// Center the wheels
void doCenter(byte message[], word length) {
  word meas_val;
  sendAck("Center\r", 7);

  ADC_Steering_Position_Stop();
  do {
    ADC_Steering_Position_MeasureChan(TRUE, 0);
  } while (ADC_Steering_Position_GetChanValue(0, &meas_val) != ERR_OK);

  if (meas_val == 749) {
    // Do nothing
  } else if (meas_val < 749) { // Turn to the right
    Bit_Steering_Direction_PutVal(0);
  } else {      // Turn to the left
    Bit_Steering_Direction_PutVal(1);
  }
  PWM_Steering_Speed_SetRatio16(0x4000);
  TurningDirection = Turning_STRAIGHT;
  PWM_Steering_Speed_Enable();

  ADC_Steering_Position_Start();
}

// Halt all actions, stops wheels, centers steering
// This tends to coast to a stop
void doHalt(byte message[], word length) {
  sendAck("Stop\r", 5);
  Bit_Right_Forward_ClrVal();
  Bit_Right_Backward_ClrVal();

  Bit_Left_Forward_ClrVal();
  Bit_Left_Backward_ClrVal();

  // For now, need to center wheels later
  PWM_Steering_Speed_Disable();
  PWM_Steering_Speed_SetRatio16(0);
  TurningDirection = Turning_NONE;
}

// Kills all actions, no reset, meant to be an emergency stop, so Ack at
end
// Setting all bits so that it brakes the wheels
void doKill(byte message[], word length) {
  Bit_Right_Forward_SetVal();
  Bit_Right_Backward_SetVal();
  Bit_Left_Forward_SetVal();
  Bit_Left_Backward_SetVal();
  PWM_Steering_Speed_Disable();
  PWM_Steering_Speed_SetRatio16(0);
  TurningDirection = Turning_NONE;
  sendAck("Killed\r", 7);
}

void main(void)
{
/***Processor Expert internal initialization. DON'T REMOVE THIS CODE!!!***/
  PE_low_level_init();
```

```
/*** End of Processor Expert internal initialization. ***/

/*Write your code here*/

// My Init
TurningDirection = Turning_NONE;
ADC_Steering_Position_Start();

for(;;) {
  byte message[32];
  word length = 0;
  if (readLine(&message, 32, &length) == ERR_OK) {
    if (length < 2) {     // We should have at least two characters,
                          //command and newline
      continue;
    }
    switch (message[0]) {
      case 'F':
      case 'f': doForward(message, length); break;
      case 'B':
      case 'b': doBackward(message, length); break;
      case 'R':
      case 'r': doRight(message, length); break;
      case 'L':
      case 'l': doLeft(message, length); break;
      case 'C':
      case 'c': doCenter(message, length); break;
      case 'H':
      case 'h': doHalt(message, length); break;
      case 'X':
      case 'x': doKill(message, length); break;
      default: sendUnknown(&message, length); break;
    }
  }
}

/*** Processor Expert end of main routine. DON'T MODIFY THIS CODE!!! ***/
  for(;;);
/*** Processor Expert end of main routine. DON'T WRITE CODE BELOW!!! ***/
```

```
} /*** End of main routine. DO NOT MODIFY THIS TEXT!!! ***/
/* END GauchoCommander */
```

Events.C
Note: Edited to remove unmodified generated methods

```
/* MODULE Events */

/*Including used modules for compilling procedure*/
#include "Cpu.h"
#include "Events.h"
#include "SerialPort.h"
#include "PWM_Right_Speed.h"
#include "PWM_Left_Speed.h"
#include "Bit_Right_Forward.h"
#include "Bit_Right_Backward.h"
#include "Bit_Left_Forward.h"
#include "Bit_Left_Backward.h"
#include "PWM_Steering_Speed.h"
#include "Bit_Steering_Direction.h"
#include "ADC_Steering_Position.h"

/*Include shared modules, which are used for whole project*/
#include "PE_Types.h"
#include "PE_Error.h"
#include "PE_Const.h"
#include "IO_Map.h"

#include "myVars.h"
/*
** ===================================================================
**     Event       :  ADC_Steering_Position_OnEnd (module Events)
**
**     From bean   :  ADC_Steering_Position [ADC]
**     Description :
**         This event is called after a measurements (which consists
**         of 1 or more conversions) is finished.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
```

```c
void ADC_Steering_Position_OnEnd(void) {
  word meas_val;
  if (ADC_Steering_Position_GetChanValue(0, &meas_val) == ERR_OK) {
    switch (TurningDirection) {
      case Turning_LEFT:
        if (meas_val < 455) {
            PWM_Steering_Speed_Disable();
            TurningDirection = Turning_NONE;
        }
        break;
      case Turning_RIGHT:
        if (meas_val > 1000) {
          PWM_Steering_Speed_Disable();
          TurningDirection = Turning_NONE;
        }
        break;
      case Turning_STRAIGHT:
        if (meas_val == 749) {
          PWM_Steering_Speed_Disable();
          TurningDirection = Turning_NONE;
        }
        break;
      default: break;
    }
  }
}

/* END Events */
```

Appendix C: Navigation Control System Code

GauchoSerial.cs

```
using System;
using System.Collections.Generic;
using System.IO.Ports;
using System.Text;

namespace GPSGaucho {
    class GauchoSerial {
        private SerialPort sPort = null;
        public GauchoSerial(string Port) {
            sPort = new SerialPort(Port, 19200, Parity.None, 8,
                StopBits.One);
            sPort.Open();
        }
        public void Left() {
            sPort.Write("L\r");
        }
        public void Right() {
            sPort.Write("R\r");
        }
        public void Center() {
            sPort.Write("C\r");
        }
        public void Halt() {
            sPort.Write("H\r");
        }
        public void Forward() {
            sPort.Write("F\r");
        }
        public void Backward() {
            sPort.Write("B\r");
        }
        ~GauchoSerial() {
            sPort.Close();
        }
    }
}
```

HTTPServer.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Threading;

namespace GPSGaucho {
    public class HTTPServer {

        private Socket listeningSock = null;
        private GauchoSerial gaucho = null;

        public HTTPServer() {
            this.gaucho = new GauchoSerial("COM1");
            this.listeningSock = new Socket(AddressFamily.InterNetwork,
                SocketType.Stream, ProtocolType.Tcp);
            // Listen on all interfaces, HTTP port
            this.listeningSock.Bind(new IPEndPoint(IPAddress.Any, 8080));
        }

        public void Start() {
            this.listeningSock.Listen(10);

            this.listeningSock.BeginAccept(
                new AsyncCallback(this.listeningSock_AcceptAsyncCallback),
                null);
        }

        public void Stop() {
            this.listeningSock.Close();
        }

        private void listeningSock_AcceptAsyncCallback(IAsyncResult ar) {
            Socket clientSock = this.listeningSock.EndAccept(ar);
            if (clientSock == null) {
```

```
            // This means that there is no client,
            //probably because the connection was closed
            return;
        }
        Console.WriteLine("Client connected");
        processRequest(clientSock);
        clientSock.Close();

        // Look for the next connection
        this.listeningSock.BeginAccept(
            new AsyncCallback(this.listeningSock_AcceptAsyncCallback),
            null);
    }

    private void processRequest(Socket sock) {
        NetworkStream stream = new NetworkStream(sock);
        StreamReader reader = new StreamReader(stream);
        List<string> requestLines = new List<string>();
        string line = String.Empty;
        try {
            while ((line = reader.ReadLine()) != String.Empty) {
                Console.WriteLine(line);
                requestLines.Add(line);
            }
        }
        catch (IOException) {
            try {
                reader.Close();
                stream.Close();
            }
            catch (IOException) { }
        }
        string method = requestLines[0].Split(' ')[0];
        string file = requestLines[0].Split(' ')[1];
        switch (method) {
            case "GET": this.process_GET(stream, file); break;
            default: break;
        }
        try {
            reader.Close();
            stream.Close();
        }
        catch (IOException) { }
    }

    private void process_GET(NetworkStream stream, string file) {
        StringBuilder returnData = new StringBuilder();
        // Write the headers
        // Insert the HTTP code line later on
        returnData.Append("Server: Gaucho Control Server\r\n");
        returnData.Append("Connection: close\r\n");
        returnData.Append("Cache-Control: no-cache\r\n");
        returnData.Append("Expires: 0\r\n");
        returnData.Append("ETag: \"\"\r\n");
        returnData.Append(
            "Content-Type: text/html; charset=ISO-8859-1\r\n");
        returnData.Append("\r\n");

        if (file == "/" || file == "/index.html") {
            returnData.Insert(0, "HTTP/1.1 200\r\n");
            returnData.Append(Properties.Resources.Index);
        }
        else if (file == "/left.html") {
            // Turn left
            this.gaucho.Left();
            returnData.Insert(0, "HTTP/1.1 200\r\n");
            returnData.Append(Properties.Resources.Index);
        }
        else if (file == "/right.html") {
            // Turn right
            this.gaucho.Right();
            returnData.Insert(0, "HTTP/1.1 200\r\n");
            returnData.Append(Properties.Resources.Index);
        }
        else if (file == "/center.html") {
            // Center
            this.gaucho.Center();
            returnData.Insert(0, "HTTP/1.1 200\r\n");
            returnData.Append(Properties.Resources.Index);
        }
```

```
else if (file == "/halt.html") {
    // Halt
    this.gaucho.Halt();
    returnData.Insert(0, "HTTP/1.1 200\r\n");
    returnData.Append(Properties.Resources.Index);
}
else if (file == "/forward.html") {
    // Forward
    this.gaucho.Forward();
    returnData.Insert(0, "HTTP/1.1 200\r\n");
    returnData.Append(Properties.Resources.Index);
}
else if (file == "/backward.html") {
    // Backward
    this.gaucho.Backward();
    returnData.Insert(0, "HTTP/1.1 200\r\n");
    returnData.Append(Properties.Resources.Index);
}
else if (file == "/sequence.html") {
    returnData.Insert(0, "HTTP/1.1 200\r\n");
    byte[] buf =
        Encoding.ASCII.GetBytes(returnData.ToString());
    stream.Write(buf, 0, buf.Length);
    stream.Flush();
    returnData = new StringBuilder();
    returnData.Append(this.executeSequence(stream));
}
else {
    returnData.Insert(0, "HTTP/1.1 404\r\n");
}
try {
    // We try/catch nothing just to make sure
    // the stream doesn't crash
    // if it was closed by the browser
    byte[] buffer =
        Encoding.ASCII.GetBytes(returnData.ToString());
    stream.Write(buffer, 0, buffer.Length);
    stream.Flush();
}
catch (IOException) { }
```

```
    }

    private string executeSequence(NetworkStream stream) {
        try {
            byte[] buffer = Encoding.ASCII.GetBytes(
"<html><head><title>Pre-Programmed Sequence</title></head><body>");
            stream.Write(buffer, 0, buffer.Length);
            stream.Flush();
        }
        catch (IOException) {
            return "";
        }
        try {
            byte[] buffer = Encoding.ASCII.GetBytes(
                "<p>Centering Wheels...<br/>\n");
            stream.Write(buffer, 0, buffer.Length);
            stream.Flush();
            this.gaucho.Center();

            buffer = Encoding.ASCII.GetBytes(
                "<p>Going Forward, 5 seconds...<br/>\n");
            stream.Write(buffer, 0, buffer.Length);
            stream.Flush();
            this.gaucho.Forward();
            Thread.Sleep(5000);

            buffer = Encoding.ASCII.GetBytes(
                "<p>Turning Right, 3 seconds...<br/>\n");
            stream.Write(buffer, 0, buffer.Length);
            stream.Flush();
            this.gaucho.Right();
            Thread.Sleep(3000);

            buffer = Encoding.ASCII.GetBytes(
                "<p>Centering Wheels...<br/>\n");
            stream.Write(buffer, 0, buffer.Length);
            stream.Flush();
            this.gaucho.Center();

            buffer = Encoding.ASCII.GetBytes(
                        "<p>Going Forward, 5 seconds...<br/>\n");
                    stream.Write(buffer, 0, buffer.Length);
                    stream.Flush();
                    // Already moving forward
                    Thread.Sleep(5000);

                    buffer = Encoding.ASCII.GetBytes("<p>Stopping...<br/>\n");
                    stream.Write(buffer, 0, buffer.Length);
                    stream.Flush();
                    this.gaucho.Halt();
                }
                catch (IOException) {
                    this.gaucho.Halt();
                }
                return "</p></body></html>";
            }
        }
}
```

Program.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace GPSGaucho {
    class Program {
        static void Main(string[] args) {
            HTTPServer server = new HTTPServer();
            server.Start();
            Console.ReadLine();
            server.Stop();
        }
    }
}
```