ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN

"Control por internet del robot Pololu 3π mediante el modem GSM Narobo DroneCell"

TESINA DE SEMINARIO

Previa la obtención del Título de:

INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

Presentado por:

Davis Vicente Garcés Naranjo

Jorge Luis Gómez Ponce

GUAYAQUIL – ECUADOR

AÑO 2011

AGRADECIMIENTO

A Dios.

A la familia.

A todas las personas que apoyaron en el desarrollo de este trabajo.

A todos quienes fomentan el desarrollo tecnológico en Ecuador.

DEDICATORIA

A Dios por la fortaleza que nos ha brindado al realizar este trabajo, por su infinito amor reflejado en nuestros seres queridos y maestros.

A nuestros padres y seres queridos por su comprensión y apoyo incondicional, quienes siempre fomentaron diligencia y perseverancia con valores éticos, y que nunca nos dejaron decaer ni darnos por vencidos.

TRIBUNAL DE SUSTENTACIÓN

Ing. Carlos Valdivieso

Profesor de Seminario de Graduación

Ing. Hugo Villavicencio V.

Delegado del Decano

DECLARACIÓN EXPRESA

"La responsabilidad del contenido de esta tesina, nos corresponde exclusivamente; y el patrimonio intelectual del mismo a la ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL".

(Reglamento de exámenes y títulos profesionales de la ESPOL)

_____

Davis Vicente Garcés Naranjo

_____

Jorge Luis Gómez Ponce

RESUMEN

El principal objetivo de este proyecto es el implementar las técnicas aprendidas en nuestra vida académica con respecto al uso de los microcontroladores sus características, programación y optimización para poder realizar con la ayuda de algunos elementos, el control del Robot Pololu 3π a través del internet usando chips de celular con tecnología GPRS de 3ra generación para que a través de la misma el robot reciba comandos enviados a través de una terminal remota y así poder mover cuando el usuario lo desee, con el uso de herramientas como el software AVR STUDIO 4 para poder programar el microcontrolador que dirige al robot que es el ATmega328P.

ÍNDICE GENERAL

Contenido

ÍNDICE DE FIGURAS

INTRODUCCIÓN

El objetivo del proyecto es implementar el control del robot Pololu 3π a través del módulo inalámbrico Narobo DroneCell el cual nos permitirá manipular el robot desde una PC o terminal remota que posea conexión a internet, el robot recibirá estos comandos a través del Narobo DroneCell el cual es un dispositivo que necesita un chip de celular con tecnología GPRS para poder recibir, decodificar y transmitir las órdenes del usuario para que el robot las procese y realice.

En el primer capítulo, se menciona una descripción general del proyecto, las partes y funciones del mismo, aplicaciones en el campo industrial y proyectos similares como TEMO el cual es un robot hecho con legos pero es controlado a través de una terminal remota o PC y los comandos son recibidos por un celular el cual también sirve para tomar y enviar fotos.

En el segundo capítulo, se da un detalle sobre las herramientas de hardware: el robot Pololu 3π, el Narobo DroneCell, el microcontrolador ATMEGA328P. Además de las herramientas de software: AVR STUDIO4 con sus 2 compiladores los cuales son AVR ASSEMBLER que se usa para crear archivos en lenguaje ensamblador y el AVR GCC que crea archivos en C, además de las librerías que controlan el robot Pololu 3π y las librerías que sirven para recibir y enviar información al internet.

El tercer capítulo, trata del diseño e implementación del proyecto, el funcionamiento de las librerías del compilador AVR GCC que sirven para controlar el robot y enviar o recibir datos por internet. Se desarrolló un diagrama de bloques que detalla los elementos de nuestro proyecto, y explica de manera independiente cada parte del mismo.

En el cuarto y último capítulo se muestran las pruebas realizadas con el robot Pololu 3π, además de observar cómo se ingresan los comandos a través de una PC y como el robot reacciona a cada comando que le es enviado.

Capítulo 1

1 DESCRIPCIÓN GENERAL DEL PROYECTO

1.1 Antecedentes.

La creciente necesidad de desarrollar prototipos de robots para que cumplan funciones repetitivas o que reemplacen al ser humano por la calidad y precisión de su trabajo, se ha vuelto muy popular en este tiempo; sin embargo la controversia que aún persiste es de cómo controlar a ese robot. Existen actualmente modelos establecidos para el control de estos robots, tales como: IR (Infra Red control), RF (Radio Frecuency control), PC controller (conectados directamente a un computador), Wireless protocols y demás.

Ahora, incluido en la robótica, se observa también el avance en las telecomunicaciones, donde, es cada vez más común observar equipos de todo tipo: domésticos, educacionales, profesionales, entre otros; en los cuales vienen incorporados nuevos y avanzados sistemas de comunicaciones inalámbricas, que se basan desde un control de mando a distancia hasta su control a través de un computador mediante un router wireless.

Muchos de nosotros sabemos qué apariencia tiene una computadora. Usualmente tiene teclado, monitor, CPU (Unidad de Procesamiento Central), impresora y mouse. Este tipo de dispositivos, son diseñados principalmente para comunicarse con humanos.

Manejo de base de datos, análisis financieros o incluso procesadores de textos, se encuentran todos dentro de la "gran caja", que contiene CPU, la memoria, el disco rígido, etc. El verdadero "cómputo", sin embargo, tiene lugar dentro de la CPU.

¿Pero usted sabía que hay computadoras alrededor de nosotros, corriendo programas y haciendo cálculos silenciosamente sin interactuar con ningún humano? Estas computadoras están en su auto, en el transbordador espacial, en un juguete, e incluso robots.

Llamamos a éstos dispositivos "microcontroladores". Micro porque son pequeños, y controladores, porque controlan máquinas o incluso otros controladores. Los Microcontroladores, por definición entonces, son diseñados para ser conectados más a máquinas que a personas. Son muy útiles porque se puede crear el objeto físico, escribir programas para controlarlo, y luego este automáticamente trabaja para usted.

1.2 Descripción del Proyecto.

En este trabajo presentaremos al POLOLU 3π que es un pequeño robot semi-autónomo de alto rendimiento, designado para competiciones de seguimiento de línea y resolución de laberintos; al cual se le enviaran comandos mediante un dispositivo que nos sirve como interfaz de comunicaciones celulares, el NAROBO DroneCell, que hará las veces de receptor de comandos enviados por internet. El POLOLU también cuenta

con un display en el cual se mostraran mensajes indicando cual comando ha sido receptado y si está disponible el robot para recibir otro comando a ejecutar o si se encuentra procesando algún comando. Este comando le indicara al robot la acción que deberá de cumplir, las cuales serán previamente definidas en el código de control del POLOLU. Además, al momento de integrar al NAROBO con el POLOLU, debemos de tomar en cuenta cuales son los comandos propios de estos equipos, ya que si se decide crear funciones para otras acciones del robot y no tomamos en cuenta los nombres de estas funciones, entonces tendríamos problemas con el cumplimiento de nuestras funciones.

Todo el sistema controlador será diseñado para que funcione con el microcontrolador ATMEGA328 mediante la interfaz de programación AVR Studio 4 que nos permite integrar librerías propias como librerías basadas en los componentes de nuestro proyecto.

Este proyecto nos ayudara a plasmar nuestros conocimientos tanto en robótica como en el desarrollo de controladores especializados y dedicados a una cierta labor en específico. Además, se podría establecer su aplicación como un robot de exploración basado en posiciones enviadas por internet.

1.3 Aplicaciones.

La aplicación de este proyecto es básicamente el control de un robot mediante un celular, tomando en cuenta la configuración del equipo dependiendo del proveedor del servicio de telefonía celular que elijamos para la interconexión de nuestro proyecto,

esto gracias al NAROBO lo que nos permite tener todas las funciones que cumplen un celular pero aplicado a un circuito en general, en este caso, al control del POLOLU.

Ya que el robot es controlado por celular, entonces cualquier persona puede enviarle un comando al robot con tan solo conocer la dirección de este robot. Por lo cual, se puede diseñar un robot explorador de zonas urbanas, al cual se le ingresen coordenadas y el robot se moverá hacia ellas.

1.4 Proyectos Similares.

Hemos observado muchos trabajos de robótica donde se usa protocolos de conexión inalámbricos o alámbricos, con los cuales el robot cumple una función dependiendo del comando del controlador. En esos casos, no se ha implementado el funcionamiento de tecnología celular, ya que contar con estos circuitos es costoso, pero gracias al avance tecnológico ahora contamos con módulos que cumplen las veces de una comunicación celular con la cual podemos comunicar a nuestro robot a un celular.

Algunos ejemplos de proyectos similares son mostrados a continuación:

- Robot Plen

  Android es un sistema operativo basado en LINUX específicamente diseñado para celulares inteligentes y tablets los cuales son considerados de celulares 4G.

  Android saco al mercado un robot el cual es manipulado a través de un software diseñado específicamente para celulares con este sistema operativo, el cual es capaz de realizar varios movimientos solo con la interacción entre el usuario y el software.

Ilustración 1 Robot Plen

Este robot tiene 228 mm de altura posee 18 grados de libertad y además posee sensores giroscópicos los cuales le permiten mantener el equilibrio y además tiene un software especial para programarlo específicamente creado por Android para celulares con este sistema operativo.

- Brazo robot controlado por CPU con interfaz creada en JAVA

Este proyecto posee muchas aplicaciones las cuales pueden ir al campo de la industria y la domótica, el cual se basa en usar una interfaz creada en JAVA, que es un lenguaje de altísimo nivel, lo cual nos permitirá controlar el brazo para realizar diferentes acciones.

Ilustración 2 Interfaz creada en JAVA para el control del brazo robot

La operaciones básicas que realiza el robot es moverse en todas las direcciones (arriba, abajo, izquierda y derecha), rotar y además de agarrar objetos o dejarlos en una posición especificada por el usuario.

- Robot TeMo

Este robot está constituido con LEGOS y su funcionamiento lo controla un celular colocado en la parte superior de la estructura donde existe un cable el cual permite conectar la interfaz del celular con el microcontrolador encargado de controlar los motores que le dan movimiento.

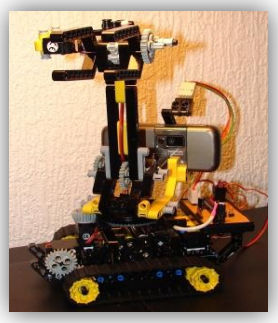

Ilustración 3 Robot TeMo

Este robot posee además un software el cual controla el movimiento del mismo por lo que el celular se convierte en elemento que permite la comunicación entre el usuario y el robot, el interfaz en la computadora le permite al usuario mover al robot en la dirección que él desea, rotar la cabeza del robot y tomar las fotos desde la cámara del celular para saber con exactitud su posición actual.

Capítulo 2

2 FUNDAMENTO TEÓRICO

El proyecto básicamente se divide en 2 partes las cuales son el Hardware y el Software ambos indispensables para la creación del proyecto.

Como podemos observar uniremos la programación en un lenguaje de alto nivel junto con los microprocesadores los cuales son la interfaz entre lo digital y lo real que es el robot en sí.

2.1 Requerimientos para la aplicación del proyecto

El software que se utilizará para programar los diferentes microcontroladores a usar en el proyecto es el AVR Studio 4 el cual nos ofrece usar 2 compiladores los cuales son: el ATMEL AVR Assembler y el AVR GCC los cuales son compiladores de lenguaje Assembler y C respectivamente, estos compiladores son los que nos ayudaran a entender el código base del robot Pololu 3π y nos permitirá implementar una variante al mismo.

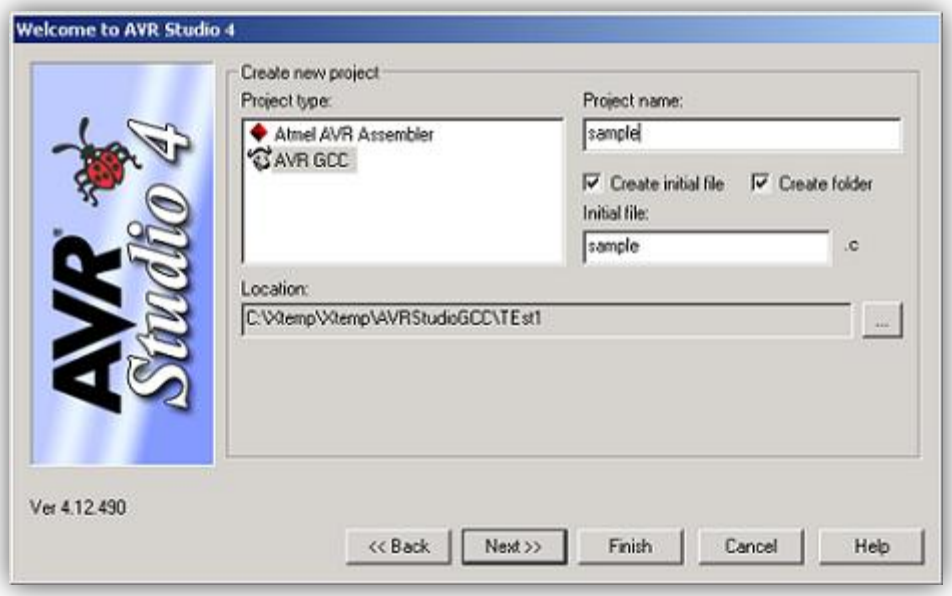

Ilustración 4 Página de inicio del programa AVR STUDIO 4

Además para poder implementar los cambios que se van a realizar en el código principal en el microcontrolador se usará el software de simulación PROTEUS 7.7 SP2, el cual nos permite implementar en forma simulada los códigos hechos en lenguaje C con los integrados y sus conexiones.

Ilustración 5 Software Proteus

El Hardware que se usará en el proyecto principalmente es el robot Pololu 3π, pero dentro de este robot se usan algunos microcontroladores que son los que se van a programar los cuales son: el ATMEGA168 o el ATMEGA328 ambos de la serie AVR.

Ilustración 6 Robot Pololu 3π

2.2 Herramientas del Software

2.2.1 AVR Studio 4

AVR Studio 4 es un software que pertenece a la familia de ATMEL el cual es un entorno estructurado que posee 2 subprogramas los cuales le permiten al usuario crear un archivo en Assembler o en C dependiendo de las necesidades del usuario.

Dentro del AVR STUDIO existe el AVR ASSEMBLER, este un programa el cual nos permite implementar funciones escritas en lenguaje ensamblador, el cual es el lenguaje más cercano al lenguaje máquina y él nos permite optimizar las rutinas que se emplean en la configuración de los microcontroladores ATMEL.

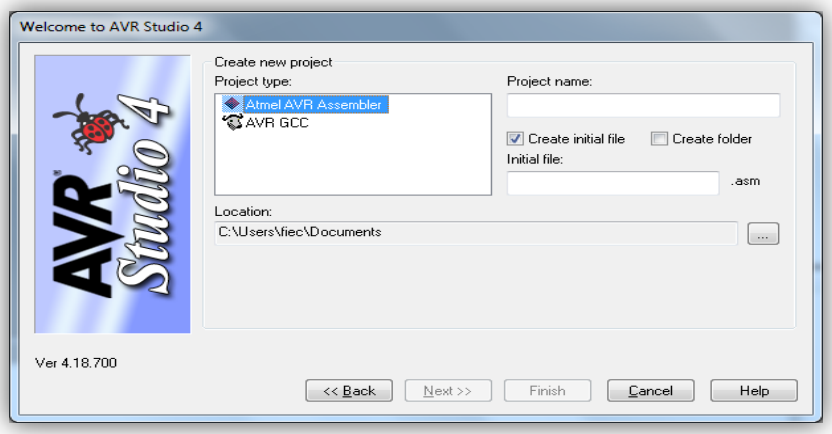

Ilustración 7 Pantalla de Inicio de AVR STUDIO 4

AVR STUDIO 4 permite manejar varias plataformas de depuración pero la usada en este proyecto es el AVR Simulator 2, el cual nos permite elegir entre los muchos integrados que ofrece la familia ATMEL, en nuestro caso el ATmega328P.

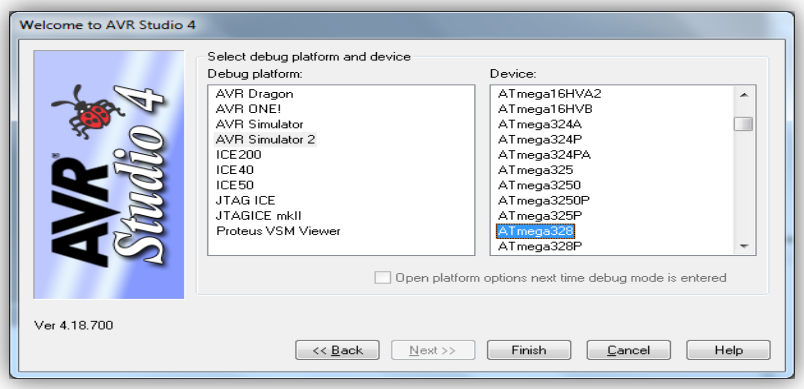

Figura 2.5 Pantalla de selección de plataforma de debug y el dispositivo a usar

Al igual el AVR GCC nos ofrece las mismas características que el AVR ASSEMBLER en cuestión de selección de plataforma y de integrados pero la diferencia entre ambas son los archivos que se generan después de la compilación.

Los archivos que se generan en la creación de un código en assembler son (***.asm***) y en C el archivo que se crea en un (***.c***) los cuales son los que guardan la programación que se va a colocar en el microcontroladores.

2.2.1.1 ¿Qué es un Compilador?

Un **compilador** es un programa informático que traduce un programa escrito en un lenguaje de programación a otro lenguaje de programación, generando un programa equivalente que la máquina será capaz de interpretar. Usualmente el segundo lenguaje es lenguaje de máquina, pero también puede ser simplemente texto. Este proceso de traducción se conoce como compilación.

Un compilador es un programa que permite traducir el código fuente de un programa en lenguaje de alto nivel, a otro lenguaje de nivel inferior (típicamente lenguaje de máquina). De esta manera un programador puede diseñar un programa en un lenguaje mucho más cercano a cómo piensa un ser humano, para luego *compilarlo* a un programa más manejable por una computadora.

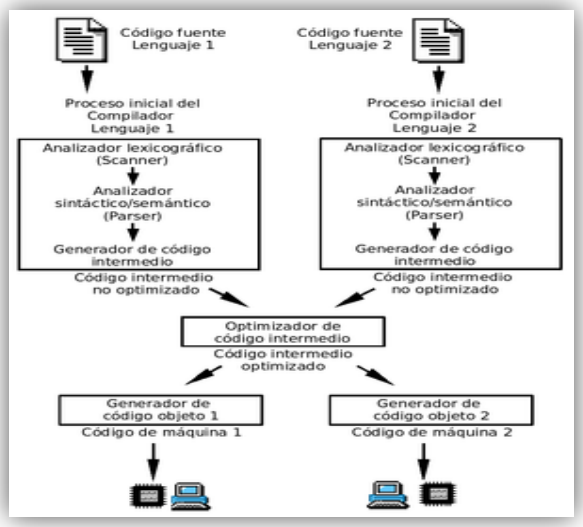

Ilustración 8Esquema algorítmico de un compilador

2.2.2 PROTEUS

PROTEUS es una herramienta software que permite la simulación de circuitos electrónicos con microcontroladores. Sus reconocidas prestaciones lo han convertido en el más popular simulador software para microcontroladores.

Esta herramienta permite simular circuitos electrónicos complejos integrando inclusive desarrollos realizados con microcontroladores de varios tipos, en una herramienta de alto desempeño con unas capacidades graficas impresionantes.

Presenta una filosofía de trabajo semejante al SPICE, arrastrando componentes de una barra e incrustándolos en la aplicación, es muy sencillo de manejar y presenta una interfaz gráfica amigable para un mejor manejo de las herramientas proporcionadas por el Proteus.

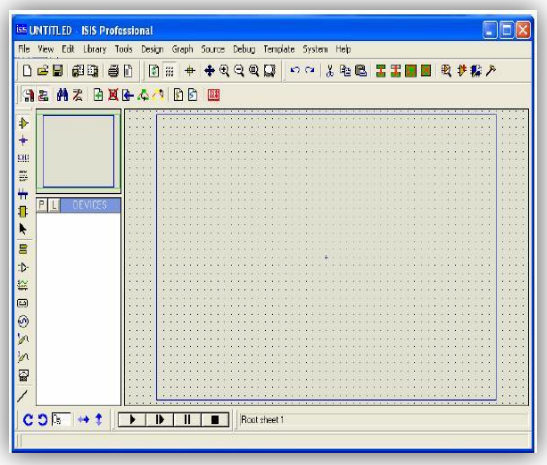

Ilustración 9 Interfaz Gráfica Proteus

2.2.3 Hercules Setup

Este es un programa el cual permite que cualquier computador o laptop pueda convertirse en un servidor TCP/IP, cliente TCP/IP usando por default el puerto 23,

además de también poder ser utilizado como hiperterminal con la capacidad de conectarse a cualquier puerto COM desde el 01 hasta el 100, control de paridad par, impar y mascara, 7 u 8 bits de datos y con velocidades seleccionables desde 4800 hasta 115200 baudios.

En nuestro proyecto se tuvo la necesidad de usar este programa debido a que nos permite enviar mensajes desde una computadora hasta cualquier otro destino conociendo su IP, lo cual es vital para el proyecto ya que nos permite enlazar robot con el usuario directamente desde el internet con una comunicación entre 2 IPs.

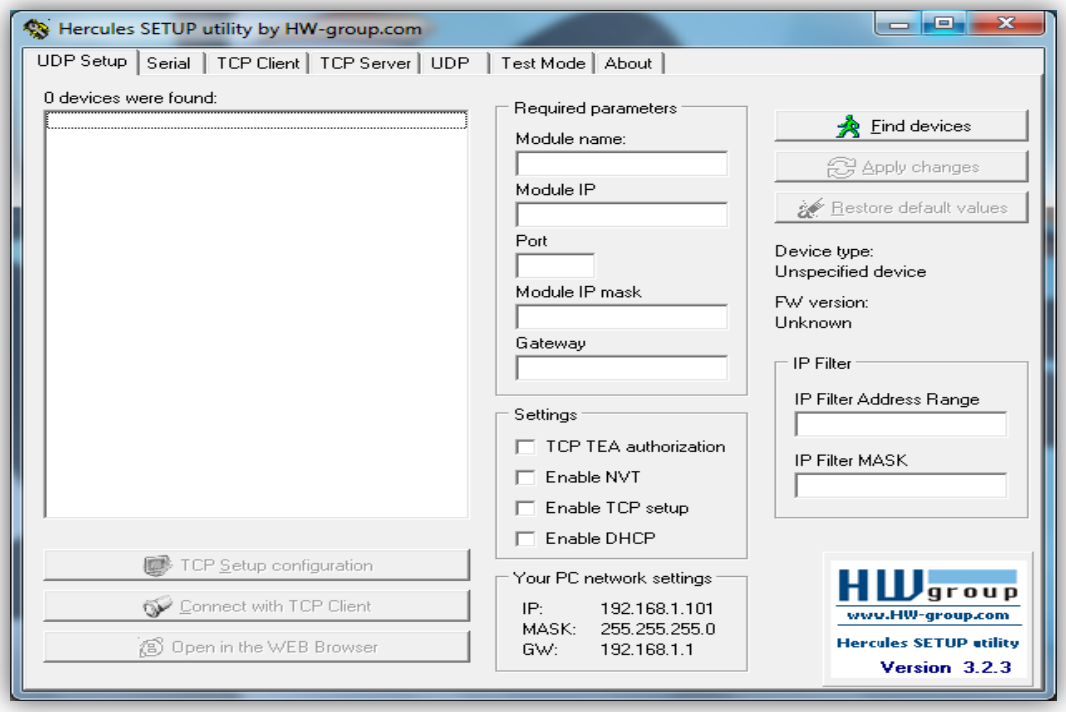

Ilustración 10 Pantalla de Inicio del programa Hercules Setup

2.3 Herramientas de Hardware

2.3.1 Robot Pololu 3π

El robot Pololu 3π es un robot el cual fue diseñado para ser un seguidor de línea y es un robot capaz de resolver laberintos de forma autónoma. Como fue posee una alimentación de 4 baterías AAA los cuales alimentan a los motores con 9.25 V, este robot es capaz de alcanzar velocidades de 100cm/s.

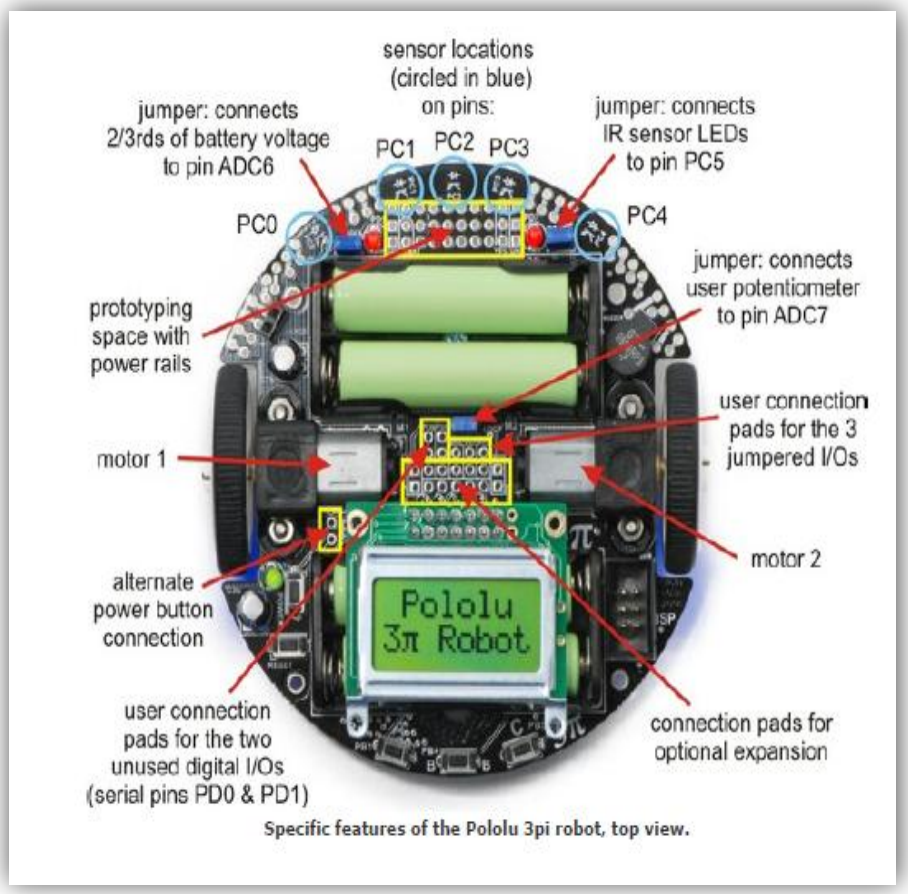

Ilustración 11 Vista Superior del robot Pololu 3π

A continuación se mencionarán las características del robot:

- Posee 4 sensores reflectantes QTR-RC
- Microcontrolador ATmega328P

- Un sensor de luz
- Un display 8 x2
- 2 Motores
- Leds para marcar alimentación de 5v al microcontrolador y 9.25v para los motores.

Dentro del robot Pololu 3π se encuentra el microcontrolador ATmega328P, el cual es el encargado de controlar las acciones del motor.

2.3.1.1 ATmega328P

ATmega328P es un microcontrolador de 32 KB que trabaja con reloj de 20MHz además posee una memoria Flash de 2KB y memoria de datos, a continuación se dará una descripción más detallada del microcontrolador:

- Microcontrolador de 8 bits de alto rendimiento
- Posee arquitectura RISC
- Posee 131 instrucciones
- Puede desarrollar 20MIPS A 20Mhz
- Posee 1KB de memoria EEPROM
- Posee 2KB de memoria SRAM
- Se puede regrabar 10000 veces la memoria Flash y 100000 veces la EEPROM
- Posee 2 Timer/Counter de 8 bytes
- Posee 1 Timer/Counter de 16 bytes
- 6 canales PWM
- 8 canales ADC con 10 bits de precisión

- Comunicación USART, I2C

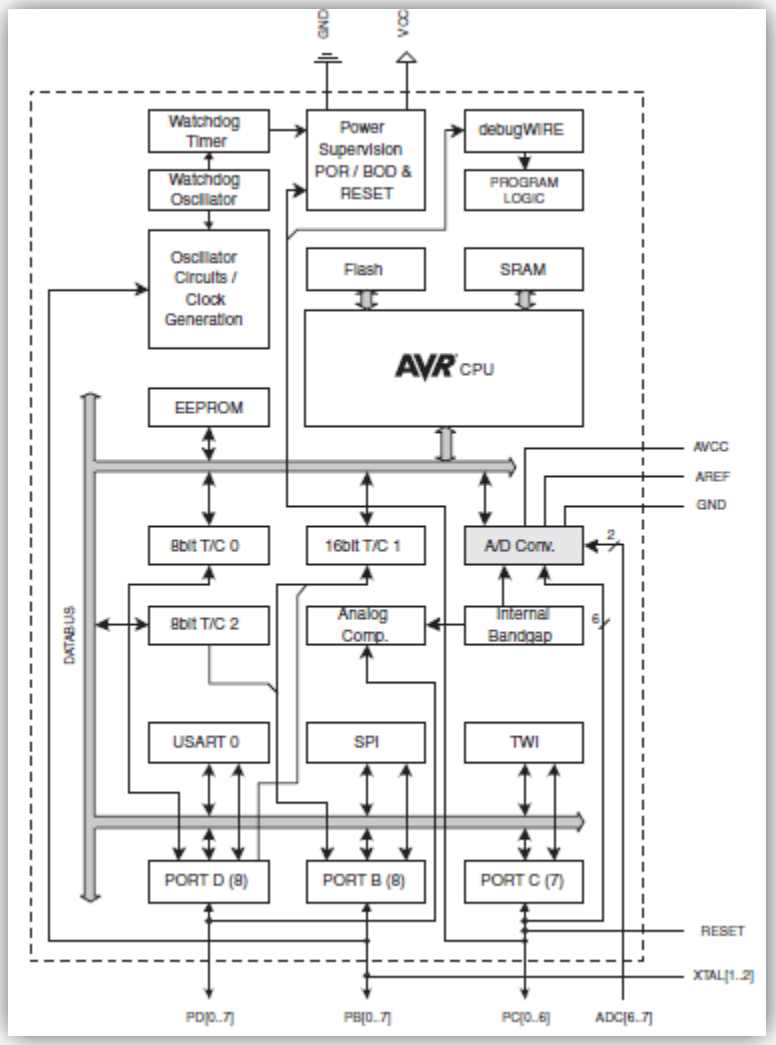

Ilustración 12 Diagrama de bloques del microcontrolador ATmega328P

Para nuestro caso es necesario recalcar unos aspectos muy importantes de este microcontrolador como son los timers 0 y 1 los cuales permiten el uso del PWM que controlan los motores y el módulo USART.

2.3.1.1.1 Timer Counter 0

El timer counter 0 es un temporizador de 8 bits el cual permite multiples funcionalidades, este posee multiples registros los cuales le permiten la gran

cantidad de modos en los que puede manejarse, además posee 3 tipos diferentes e independientes de generación de interrupciones, las cuales son: TOV0, OCRF0A y OCRFB.

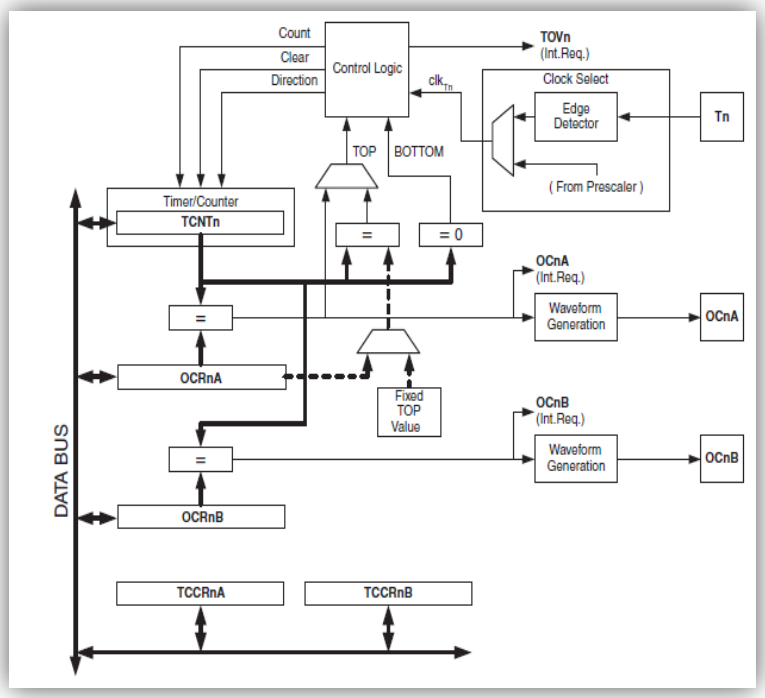

Ilustración 13 Diagrama de bloques de los timer counter del ATmega328P

Posee 4 modos de operación los cuales son importantes de recalcar, los cuales son:

- **Modo Normal.-** Este modo permite que el temporizador cuente de manera normal en forma creciente, cuando este llega a su valor de tope o TOP para que este se vuelva a cargar y cuente de nuevo, dependiendo de cómo este configurado el microcontrolador puede generar un '1', '0', o negar el valor que está actualmente en la salida.

- **Modo CTC.**- En esta modalidad el timer 0 encera el valor del contador cada vez que llega a su máximo valor, es decir cada vez que ocurre una interrupción, cuando ocurre eso la salida del comparador cambia su valor

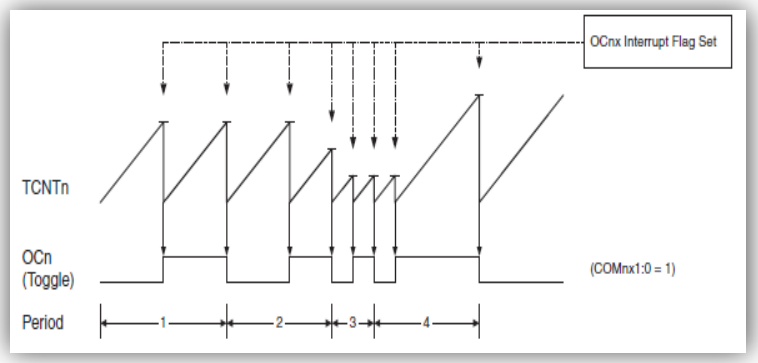

Ilustración 14 Diagrama de tiempo del modo CTC

.

- **Modo Fast PWM.-** Este modo genera PWM de tal manera que la frecuencia puede alcanzar valores muy grandes, esto se debe a que el PWM se realiza en un periodo del diente de sierra, cuando ocurre la interrupción debido al OCR0A o OCR0B el pulso se va a un valor de '0' y cuando el contador llegue a su máximo valor este se vuelve a colocar en '1'.

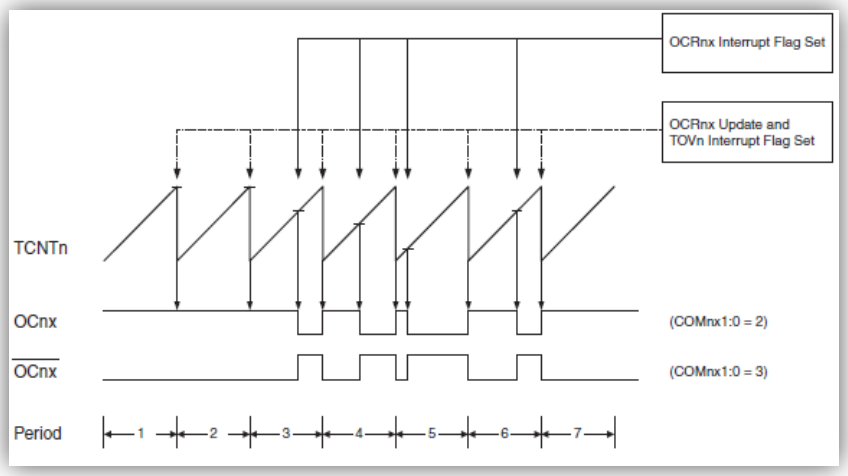

Ilustración 15 Diagrama de tiempo de la generación del Fast PWM

- **Modo Phase Correct PWM.-** En este modo se pierde un poco la velocidad pero como su nombre lo indica se usa en aplicaciones que ameriten una gran precisión en el duty cycle de la señal, en este caso para generar este PWM se necesita una señal triangular completa para que se genere la misma, como el caso anterior se genera las interrupciones con el OCR0A o el OCR0B, para generar la señal triangular completa el contador debe aumentar al máximo y luego disminuir hasta el mínimo, lo que lo hace demorarse más tiempo.

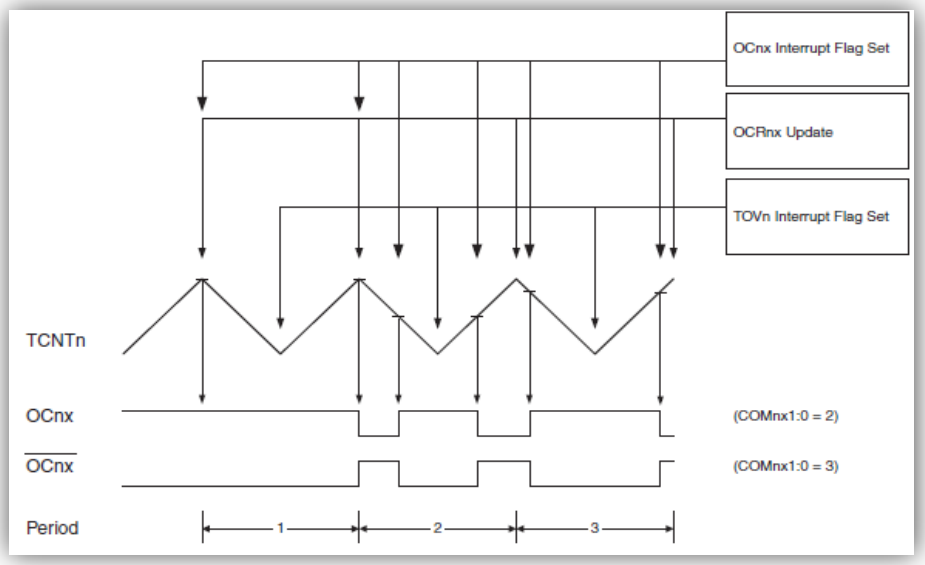

Ilustración 16 Generación de Phase Correct PWM

2.3.1.1.2 Timer Counter 1

El timer counter 1 a diferencia del 0 es un temporizador de 16 bits el cual me permite mejorar la precisión de la temporización que se desea además de proveer más valores de duty cycle para la generación del PWM, al igual que el timer counter

0 posee una arquitectura similar la cual nos permite decir que posee las mismas características.

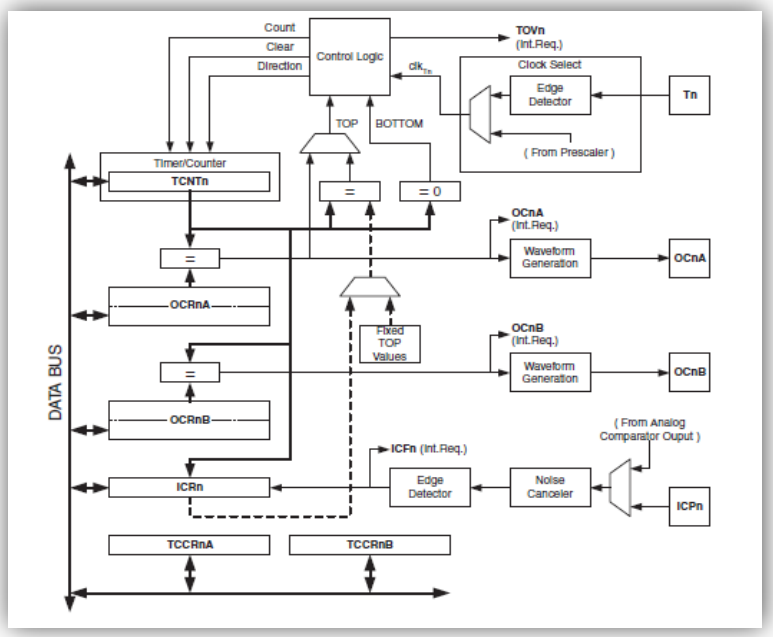

Ilustración 17 Arquitectura del Timer Counter 1

Al igual que el timer counter 0 este posee la capacidad de cambiar los límites del conteo, Bottom y Top, lo cual nos permite cambiar la temporización, además posee los mismos modos de operación que el timer 0, los cuales son:

- **Modo Normal.-** De la misma forma que el timer 0 en este modo el contador solo incrementa hasta llegar a su valor de tope y luego genera una interrupción, dependiendo de cómo este configurado el microcontrolador este puede generar un '1', '0' o negar la salida del pin OC1A.
- **Modo CTC.**- En este modo al igual que en el anterior temporizador lo que se genera son señales diente de sierra que simula el aumento del valor del contador del timer 1 hasta cuando llegue a su valor de tope. Cuando esto

ocurra, el timer se resetea y luego dependiendo de cómo se configure el microcontrolador puede dar a la salida un ‘1’, ‘0’ o negar la salida al igual que el modo anterior.

- **Modo Fast PWM.**- Como se describió anteriormente en este modo se genera dientes de sierra los cuales permiten generar la señal PWM la cual se dice que es rápida debido a que solo necesita una pendiente para poder generar el PWM.
- **Modo Phase Correct PWM.-** Este modo sirve para poder generar PWM con la cualidad de mover motores de alta precisión lo cual se realiza mediante la generación de señales triangulares completa que es nada más que la visualización de como el contador el contador del timer 1 se incrementa hasta llegar al tope y luego disminuye su valor hasta llegar al fondo lo cual genera la señal triangular, al igual que en el modo anterior, cada vez que ocurre una interrupción se genera un ‘0’, en la salida y luego cuando el contador haya llegado al Bottom o sea se haya terminado la generación de la onda triangular completa la salida vuelve a tomar el valor de ‘1’.

2.3.1.1.3 USART

El módulo USART es aquel que permite comunicar 2 dispositivos de forma sincrónica y asincrónica dependiendo de la configuración previa a los elementos, es necesario tener en cuenta el manejo y funcionamiento del mismo porque este módulo es una pieza clave para la comunicación entre varios dispositivos, en nuestro el Narobo DroneCell y el robot Pololu 3π, para entender un poco más el funcionamiento del mismo a continuación se muestra un diagrama el cual especifica la funcionalidad del mismo.

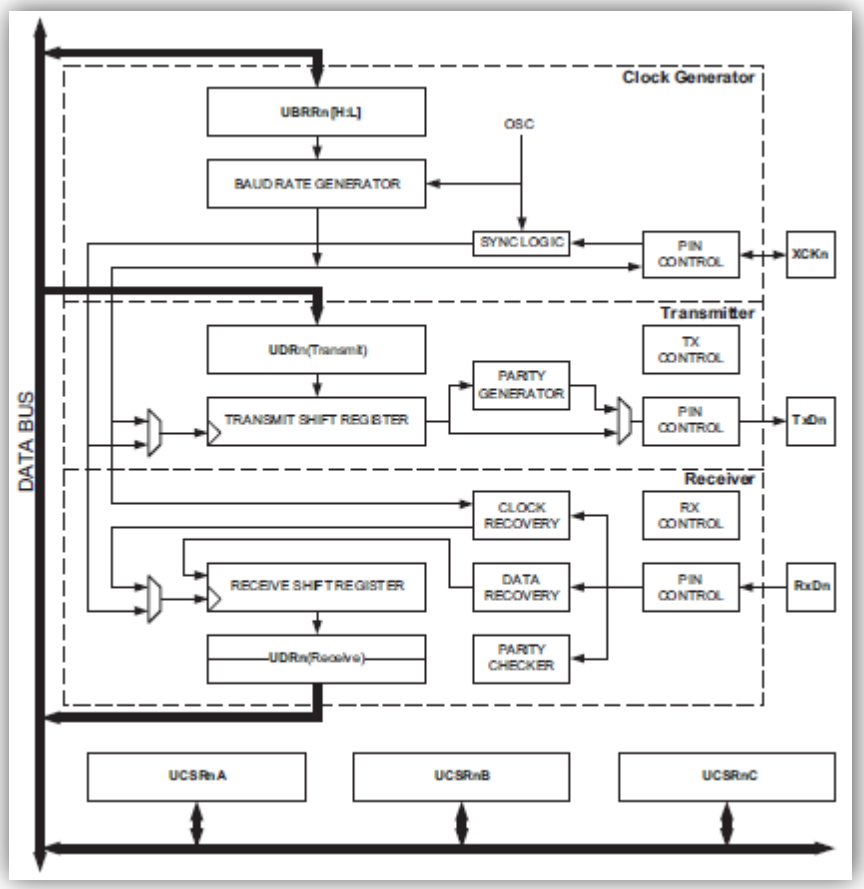

Ilustración 18 Diagrama del módulo del ATmega328P

Este módulo permite la capacidad de cambiar el formato de la trama de bits a enviar que puede ser:

- 1 bit de inicio
- 5, 6, 7, 8 o 9 bits de datos
- Con o sin bits paridad par o impar
- 1 o 2 bits de stop

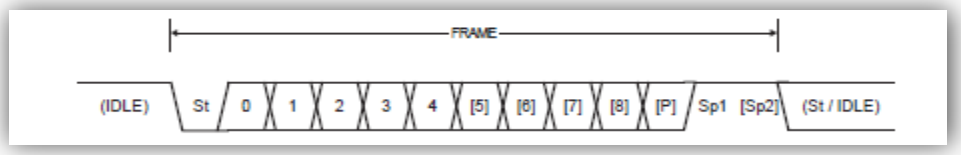

Ilustración 19 Formato de la trama de bits del módulo USART

El módulo USART es muy flexible con respecto a la generación del reloj para poder transmitir los datos permite velocidades de 2400 bps hasta 1Mbps, además este microcontrolador posee la peculiaridad de poder utilizar USART dentro del modo SPI, lo cual ofrece una gran ventaja alta resolución en el generador de baudios.

2.3.2 Narobo DroneCell

El DroneCell es un teléfono celular para nuestro proyecto de electrónica, robótica, minicomputadores o algún otro proyecto. Este es un dispositivo de comunicaciones todo-en-uno: comunicación vía mensajes de texto, llamadas telefónicas, incluso usando internet. Cualquier dispositivo con protocolo TTL UART se podrá comunicar con el DroneCell, además de poseer dentro de sí un microcontrolador que es el SIM 340DZ, el cual es el que se conecta con el chip de celular y con ayuda de los comandos AT (***Véase Apéndice A***), es capaz de realizar la comunicación con otras terminales GSM, enviar SMS, conectarse a internet y otras cosas más.

El Narobo DroneCell posee algunas características que son detalladas a continuación:

- Entrada de voltaje de 5-16 V
- Interfaz de UART de 3.3 a 5 V

- Alta velocidad de transmisión de datos (hasta 115200 bps)
- Compatibilidad con GPRS (velocidades de hasta 86.5 kbps en downlink)
- Entrega y recibe datos sobre los protocolos TCP y UDP

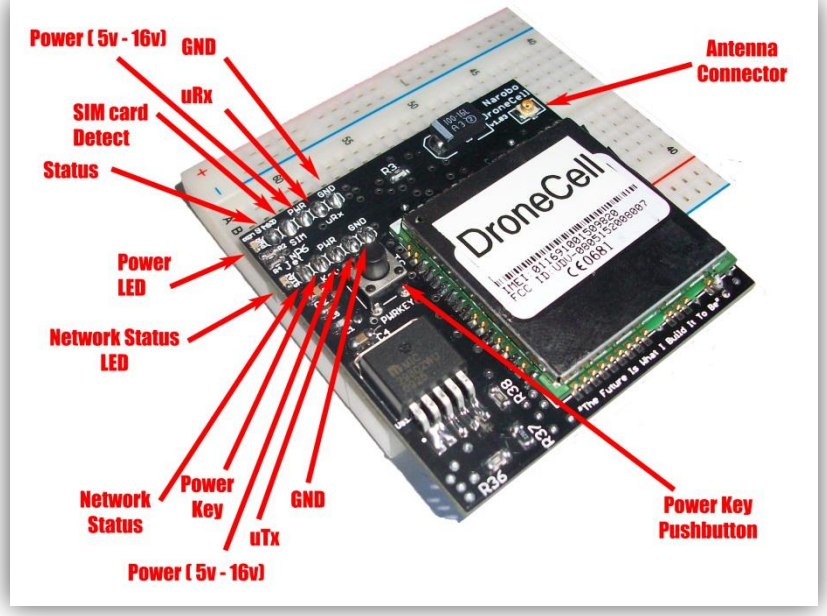

Ilustración 20 Vista superior del Narobo DroneCell

Como el modulo con el cual estamos trabajando tiene un chip GSM es bueno tener una pequeña perspectiva de lo que es GSM:

2.3.3 GSM

Es un estándar de 2da generación el cual posee varias características que la hacen actualmente uno de los estándares más usados, sus siglas significan Global System for Mobile Communications y consiste en un sistema el cual se basa en la modulación GMSK para realizar sus funciones.

Este sistema trabaja en las bandas de 800, 900 y 1800 Mhz lo cual la hace muy versátil para su uso, dependiendo de esta puede tener diferentes separaciones entre la frecuencia transmisora y receptora.

A continuación se hablara sobre las características de la red GSM:

La red GSM consta de varias componentes los cuales poseen funciones únicas para poder realizar el enlace.

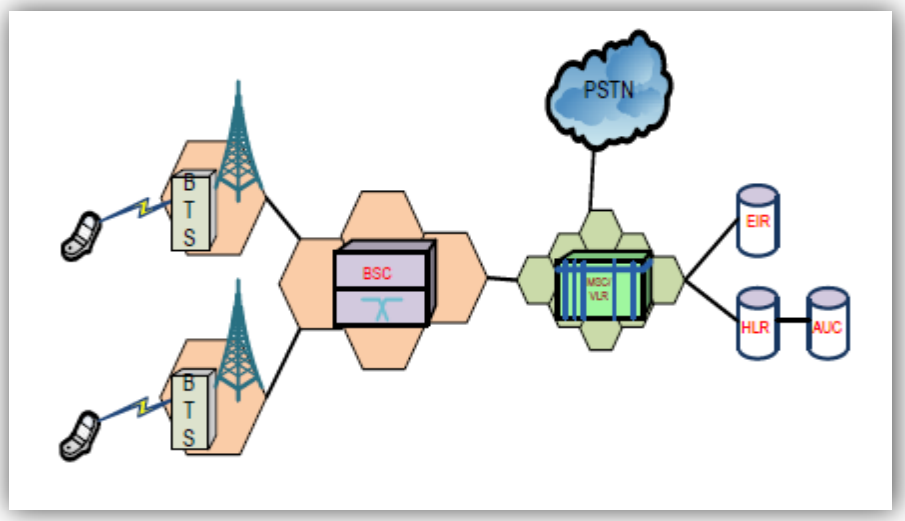

Ilustración 21 Estructura de una red GSM

Como observamos en el grafico esta red posee varios elementos entre los cuales podemos mencionar:

- **MSS**: El cual se encarga de comunicar la terminal móvil con la red inalámbrica, este se compone de los siguientes elementos:
  - **MS**: El cual es en si el equipo físico con la capacidad de acceder a la red GSM.

- **SIM**: El cual contiene la información del usuario y le permite ingresar a la red sin importar el equipo que esté usando.

- **BS**: Es la encargada de controlar las comunicaciones de radio de los móviles dentro del área especificada
- **NSS**: Se encarga de controlar la movilidad e información y la localización de los usuarios.

El principio de funcionamiento de la red GSM se basa en las comunicaciones de voz, por esta razón se detalla a continuación los pasos que ejecuta el móvil para la realización de una llamada.

- Primero se realiza un escaneo para determinar el canal con la señal más fuerte y seleccionar la estación base.
- Se realiza un handshake entre la estación móvil y el MSC.
- Se identifica el móvil y se registra la ubicación del usuario, la red necesita saber en qué ubicación se encuentra el móvil, para poder recibir los servicios de red.
- Una vez realizado el scan y el registro, se espera por mensajes de paging.
- El móvil realiza la llamada, la BTS envía un mensaje de paging en busca del MIN para establecer la comunicación.
- Si no se encuentra en la misma BTS; la MSC envía mensaje de paging en forma de broadcast a todas la BTS en busca del MIN.
- Por último, las BSTs realizan un broadcast en mensajes de paging, el móvil compara con su propio MIN (Mobile Identification Number), y si concuerda se identifica estableciéndose la comunicación.

2.3.4 GPRS

El sistema GPRS (General Packet Radio Service), es un complemento de la red GSM para darle mayor velocidad a la transmisión de datos y acceso a internet por lo que se lo conoce como tecnología 2.5G.

GPRS es una tecnología que une varios aspectos y está basada en la conmutación de paquetes lo que le da integración a protocolos como TCP/IP entre otros.

Entre las principales características que presenta la red GPRS están las siguientes:

- Requiere de poca inversión de red ya que utiliza recursos de la red GSM. Esto permite una rápida implementación y proporciona una cobertura geográfica similar a los niveles actuales alcanzados por GSM.
- El sistema de conexión GPRS, no necesita de un canal dedicado para cada usuario como se realiza en la red GSM, de esta manera nace el concepto de facturación por utilización del canal y no por tiempo, además el canal puede ser compartido por otros usuarios.
- GPRS permite a los usuarios recibir o realizar llamadas mientras se están ejecutando aplicaciones, los datos de las aplicaciones se interrumpen momentáneamente, hasta terminar la comunicación y luego se reanudan automáticamente.
- Separa el tráfico de voz con el de datos; el tráfico de voz se lo realiza por conmutación de circuitos, reservando un canal de transmisión dedicado hasta que la comunicación se termine. El tráfico de datos se lo realiza por conmutación de paquetes, la información es fraccionada en pequeños bloques siendo reconstruida en el destino.

Para que la red GSM pueda dar servicios de conmutación de paquetes, que ofrece GPRS se añaden nuevos nodos y elementos de red.

- **MS (Mobile Station) :** En el mercado se pueden encontrar diferentes tipos de terminales, que trabajan en la red GSM como en la red GPRS. Entre los siguientes terminales tenemos:
    - **Clase A:** Los terminales de clase A tienen la característica de soportar en forma simultánea, el servicio que presta GSM y GPRS. Está en la capacidad de recibir llamadas en los dos servicios simultáneamente, son los menos difundidos en el mercado por su costo.
    - **Clase B:** Este tipo de terminal puede estar registrado en los dos servicios simultáneamente, GSM y GPRS pero solo trabaja en uno de ellos a la vez. Solo soporta tráfico en forma secuencial, son los más difundidos en el mercado.
    - **Clase C:** Este tipo de terminales no soporta servicios de GSM y GPRS de forma simultánea, están diseñados para trabajar solo en GSM o en GPRS, o realizar la conmutación de servicios de forma manual.

- **BSS (Base Station Subsystem) :**La red GPRS utiliza el mismo interfaz de radio que la red GSM, reutiliza la BSC y la BTS, con algunas modificaciones en cada una de ellas como se describe:
    - La BTS requiere un nuevo software para implementación de los nuevos esquemas de codificación de GPRS.

- La BSC requiere un nuevo Hardware y software para la transmisión de paquetes, dentro de la cual se incorpora una nueva unidad denominada PCU (Packet Control Unit).

La PCU se encuentra trabajando con la BSC, se encarga de gestionar conexiones hacia el SGSN (Gateway GPRS Support Node), así como los recursos de radio para los usuarios GPRS, además multiplexa el tráfico GSM y GPRS.

- **NSS (Network Switching Subsystem):** Los elementos de red de GSM en el subsistema de red, que son reutilizados para la red GPRS necesitan cumplir las siguientes funciones: La HLR debe Contener información de los usuarios GPRS como es: Dirección SGSN y GGSN y la VLR debe poseer la capacidad de almacenar información del área de encaminamiento del SGSN. Los nuevos elementos que se encuentran en la NSS son:
  - **GGSN (Gateway GPRS Support Node):** Es el interfaz lógico hacia redes externas de paquetes, como la Internet y X.25. Se encarga de encaminar los datos que envía el móvil y los datos que recibe, comprueba la dirección del usuario al cual se quiere enviar los datos y luego se envía los datos al SGSN.
  - **SGSN (Serving GPRS Support Node):** Este nodo se encuentra al mismo nivel jerárquico que la MSC y su función principal es la conmutación de paquetes. Se encarga del trasporte de datos desde y hacia la BTS, detecta nuevos terminales GPRS y registra su localización interactuando con la HLR, realiza funciones como: control de acceso, seguridad realizando encriptación y compresión de datos.

- **BG (Border Gateway):** Cumple la función de interfaz entre Backbone de distintas operadoras GPRS.

.

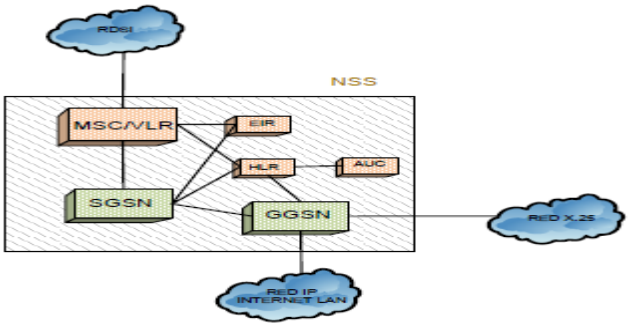

Ilustración 22 Elementos de la red NSS

Capítulo 3

3 DISEÑO E IMPLEMENTACIÓN DEL PROYECTO

A continuación se detalla el proceso del diseño y la implementación del proyecto. Nuestra prueba inicial nos ayudó a entender el funcionamiento de las funciones del robot Pololu 3π y el funcionamiento del módulo inalámbrico Narobo DroneCell.

Teniendo en cuenta el funcionamiento del Pololu 3π y de las librerias que posee el programa AVR STUDIO 4 con su compilador AVR GCC podemos implementar el programa en cuestión; en este capítulo se detallarán los fundamentos e ideas en los cuales se basa el proyecto, primero se describirá un diagrama de bloques general explicando las entradas y salidas que necesita el proyecto, luego se detallará más especificamente el algoritmo usado para el controlador, que es el ATmega328P, el cual permitirá comprender aún más como funciona el proyecto y al final se detallará el código que se uso para el proyecto en sí.

Pero antes de unir todas las partes del proyecto se configura previamente al Narobo DroneCell para que pueda recibir la información desde el internet con la ayuda de un chip con tecnología GSM de 3ra generación (GPRS).

3.1 Prueba inicial

Como primer paso se realizó una prueba a parte para poder conocer las funcionalidades del robot Pololu 3π y de las librerias del AVR GCC para poder controlar al robot, recordando que el microcontrolador ATmega328P es el encargado de recibir y enviar las órdenes del usuario al microcontrolador.

El programa en sí hace que el robot muestre el nombre “Pololu 3π Robot”, en la pantalla LCD que se encuentra en la parte superior del robot y suena una nota musical a través del buzzer que este posee.

3.2 Descripción del proyecto final

3.2.1 Diagrama de bloques

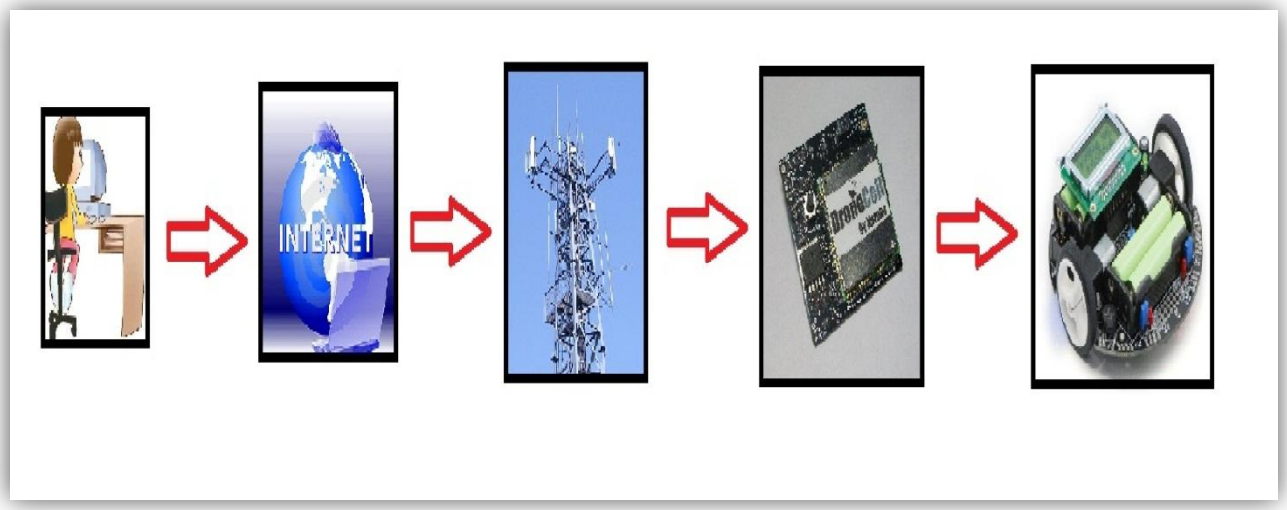

Ilustración 23 Diagrama de bloques del proyecto

Se puede observar que la comunicación que existe entre la computadora y el robot Pololu es full dúplex debido a que se envían los comandos a través del CPU y el robot

retorna señales de haber recibido correctamente el código y otro tipo de mensaje lo cual hace que la comunicación sea de 2 vías, además luego de ser recibida la instrucción a través de los módulos inalámbricos este se comunica con el microcontrolador el cual es el encargado de decodificar el comando y realizar la acción.

3.3 Algoritmo del controlador

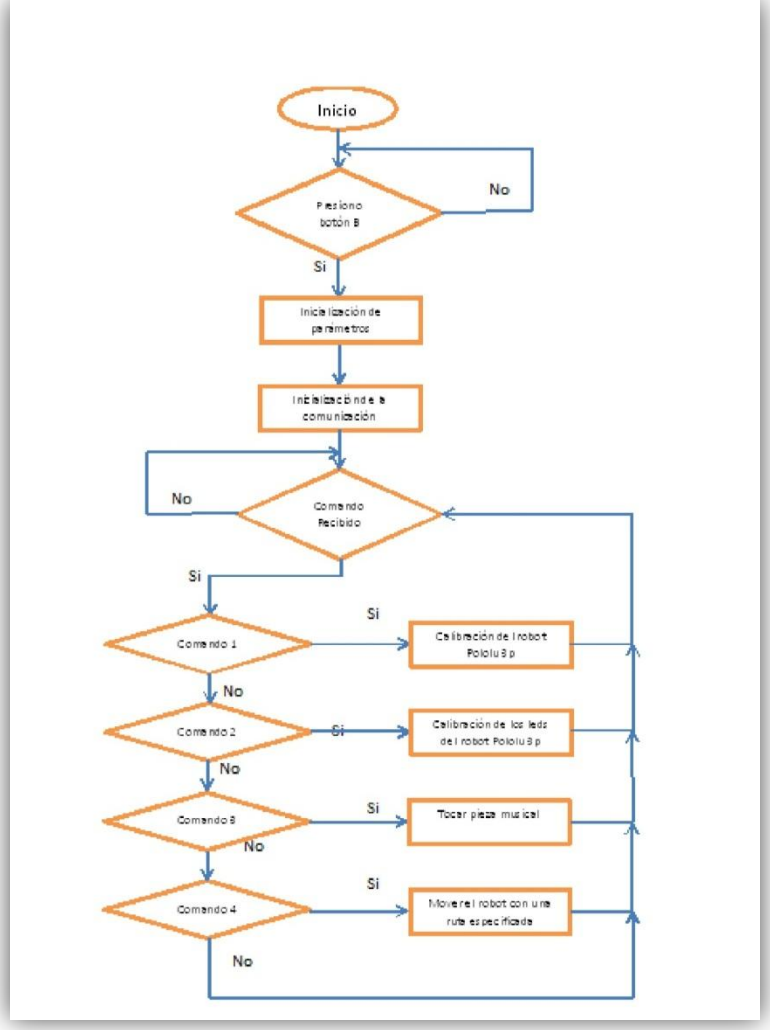

Ilustración 24 Algoritmo del controlador

Con este algoritmo lo que podemos darnos cuenta es que básicamente el programa principal se puede ver con una unión de subprogramas los cuales son invocados cada vez que se teclee un botón los cuales son A, B y C, dependiendo del botón presionado este puede realizar múltiples funciones, como la de funcionar el cronómetro, tocar una melodía moverse de manera predeterminada o encender los Leds.

3.4 Programa principal del microcontrolador

```
/*
 * 3pi-demo-program - demo code for the Pololu 3pi Robot
 *
 *
 * http://www.pololu.com/docs/0J20
 * http://www.pololu.com
 * http://forum.pololu.com
*
 */

// The 3pi include file must be at the beginning of any program that
// uses the Pololu AVR library and 3pi.
//#include "dronecell.c"

#include <pololu/3pi.h>
#include <string.h>

//POWER KEY PIN LOW PARA ENCENDER NAROBO

// This include file allows data to be stored in program space.  The
// ATmega168 has 16k of program space compared to 1k of RAM, so large
// pieces of static data should be stored in program space.
#include <avr/pgmspace.h>

// Introductory messages.  The "PROGMEM" identifier causes the data to
// go into program space.
const char welcome_line1[] PROGMEM = " Pololu";
const char welcome_line2[] PROGMEM = "3\xf7 Robot"; // \xf7 is a greek

            // pi character
const char title_project1[] PROGMEM = "Proyecto";
const char title_project2[] PROGMEM = "Graduacion";

const char project_name1[] PROGMEM = "Comunicacion";
const char project_name2[] PROGMEM = "Inalambrica";
const char project_name3[] PROGMEM = "NAROBO";
const char project_name4[] PROGMEM = "DRONECELL";

const char integrantes_1_1[] PROGMEM = "Davis";
```

```
const char integrantes_1_2[] PROGMEM = "Garces";
const char integrantes_2_1[] PROGMEM = "Jorge";
const char integrantes_2_2[] PROGMEM = "Gomez";

const char          wait_comand1[] PROGMEM = "Pres. B";
const char          wait_comand2[] PROGMEM = "para Conect";

const char connecting_line1[] PROGMEM = "Conectando";
const char connecting_line2[] PROGMEM = "Narobo...";

const char main_menu_intro_line1[] PROGMEM = "  Main";
const char main_menu_intro_line2[] PROGMEM = "  Menu";

const char menu_bat_test[] PROGMEM = "Battery";
const char menu_led_test[] PROGMEM = "LEDs";
const char menu_lcd_test[] PROGMEM = "LCD";
const char menu_ir_test[] PROGMEM = "Sensors";
const char menu_motor_test[] PROGMEM = "Motors";
const char menu_music_test[] PROGMEM = "Music";
const char menu_time_test[] PROGMEM = "Timer";
const char menu_auto_calibrate[] PROGMEM = "Calibration";
const char menu_adv_coordinates[] PROGMEM = "Adv. Coord.";
const char menu_er_exit[] PROGMEM = "Salir";

const char tipo_trabajo1[] PROGMEM = "A. Manual.";
const char tipo_trabajo2[] PROGMEM = "B. Internet.";
const char menu_line2[] PROGMEM = "\x7f" "A \xa5" "B C\x7e";
const char back_line2[] PROGMEM = "\6B";

int position_x, position_y, i=0, salir=1;         //Actual Position Indicators
static int menu_index = 0;

void auto_calibrate();
void bat_test();
void led_test();
void lcd_test();
void ir_test();
void motor_test();
void music_test();
void time_test();
void adv_coordinates();
void manual();
void er_exit();
void Estableshing_Connection();
long tiempo(char seleccionar);
typedef void (*function)();
const function main_menu_functions[] = { auto_calibrate, led_test, music_test, motor_test, adv_coordinates,
bat_test, er_exit};
const char *main_menu_options[] = { menu_auto_calibrate, menu_led_test, menu_music_test,
menu_motor_test, menu_adv_coordinates, menu_bat_test, menu_er_exit};
const char main_menu_length = sizeof(main_menu_options)/sizeof(main_menu_options[0]);

// A couple of simple tunes, stored in program space.
const char welcome[] PROGMEM = ">g32>>c32";
const char connecting_music[] PROGMEM = ">>c32>g32";
const char beep_button_a[] PROGMEM = "!c32";
const char beep_button_b[] PROGMEM = "!e32";
```

```
const char beep_button_c[] PROGMEM = "!g32";
const char timer_tick[] PROGMEM = "!v8>>c32";

// Data for generating the characters used in load_custom_characters
// and display_readings.  By reading levels[] starting at various
// offsets, we can generate all of the 7 extra characters needed for a
// bargraph.  This is also stored in program space.
const char levels[] PROGMEM = {
        0b00000,
        0b00000,
        0b00000,
        0b00000,
        0b00000,
        0b00000,
        0b00000,
        0b11111,
        0b11111,
        0b11111,
        0b11111,
        0b11111,
        0b11111,
        0b11111
};

// This character is a musical note.
const prog_char note[] PROGMEM = {
        0b00100,
        0b00110,
        0b00101,
        0b00101,
        0b00100,
        0b11100,
        0b11100,
        0b00000,
};

// This character is a back arrow.
const prog_char back_arrow[] PROGMEM = {
        0b00000,
        0b00010,
        0b00001,
        0b00101,
        0b01001,
        0b11110,
        0b01000,
        0b00100,
};

// This function loads custom characters into the LCD.  Up to 8
// characters can be loaded; we use them for 6 levels of a bar graph
// plus a back arrow and a musical note character.
void load_custom_characters()
{
        lcd_load_custom_character(levels+0,0); // no offset, e.g. one bar
        lcd_load_custom_character(levels+1,1); // two bars
        lcd_load_custom_character(levels+2,2); // etc...
        lcd_load_custom_character(levels+4,3); // skip level 3
```

```
        lcd_load_custom_character(levels+5,4);
        lcd_load_custom_character(levels+6,5);
        lcd_load_custom_character(back_arrow,6);
        lcd_load_custom_character(note,7);
        clear(); // the LCD must be cleared for the characters to take effect
}

// 10 levels of bar graph characters
const char bar_graph_characters[10] = {' ',0,0,1,2,3,3,4,5,255};

//UART Code Lines

// receive_buffer: A ring buffer that we will use to receive bytes on PD0/RXD.
// The OrangutanSerial library will put received bytes in to
// the buffer starting at the beginning (receiveBuffer[0]).
// After the buffer has been filled, the library will automatically
// start over at the beginning.
char receive_buffer[32];
char comando[8];
char ejecutar;

// receive_buffer_position: This variable will keep track of which bytes in the receive buffer
// we have already processed.  It is the offset (0-31) of the next byte
// in the buffer to process.
unsigned char receive_buffer_position = 0;

// send_buffer: A buffer for sending bytes on PD1/TXD.
char send_buffer[32];

// wait_for_sending_to_finish:  Waits for the bytes in the send buffer to
// finish transmitting on PD1/TXD.  We must call this before modifying
// send_buffer or trying to send more bytes, because otherwise we could
// corrupt an existing transmission.
void wait_for_sending_to_finish()
{
while(!serial_send_buffer_empty());
}
//Finish UART Code Lines

//Code lines to auto_calibrate 3pi
void auto_calibrate()
{
print("Auto");
  lcd_goto_xy(0,1);
print("Calibracion");
  time_reset();
  set_motors(60, -60);
while(get_ms() < 250)
    calibrate_line_sensors(IR_EMITTERS_ON);
  set_motors(-60, 60);
while(get_ms() < 750)
    calibrate_line_sensors(IR_EMITTERS_ON);
  set_motors(60, -60);
while(get_ms() < 1000)
    calibrate_line_sensors(IR_EMITTERS_ON);
  set_motors(0, 0);
  green_led(1); red_led(0);
```

```
  delay_ms(250);
  green_led(0); red_led(1);
  delay_ms(250);
  green_led(0); red_led(0);
  play_from_program_space(beep_button_a);
  delay_ms(300);

  //serial_send_blocking("c",1);
}

void er_exit()
{
        salir=0;
}

// Displays the battery voltage.
void bat_test()
{
        int bat = read_battery_millivolts();

        print_long(bat);
        print("mV");

        delay_ms(2000);
}

char wait_for_250_ms_or_button_b()
{
        int i;
        for(i=0;i<25;i++)
        {
                delay_ms(10);
                if(button_is_pressed(BUTTON_B))
                        return 1;
        }
        return 0;
}

// Blinks the LEDs
void led_test()
{
        static int veces=0;
        while (!(veces==3))
        {
                lcd_goto_xy(0,0);
                play("c32");
                print("Red  ");

                red_led(1);
                if(wait_for_250_ms_or_button_b())
                        return;
                red_led(0);
                if(wait_for_250_ms_or_button_b())
                        return;

                play(">c32");
                lcd_goto_xy(0,0);
```

```
                print("Green");

                green_led(1);
                if(wait_for_250_ms_or_button_b())
                        return;
                green_led(0);
                if(wait_for_250_ms_or_button_b())
                        return;
                veces++;
        }
}

void ir_test()
{
        unsigned int sensors[5]; // an array to hold sensor values

        if(button_is_pressed(BUTTON_C))
                read_line_sensors(sensors, IR_EMITTERS_OFF);
        else
                read_line_sensors(sensors,IR_EMITTERS_ON);

        unsigned char i;

        for(i=0;i<5;i++) {
                // Initialize the array of characters that we will use for the
                // graph.  Using the space, an extra copy of the one-bar
                // character, and character 255 (a full black box), we get 10
                // characters in the array.

                // The variable c will have values from 0 to 9, since
                // values are in the range of 0 to 2000, and 2000/201 is 9
                // with integer math.
                char c = bar_graph_characters[sensors[i]/201];

                // Display the bar graph characters.
                print_character(c);

        }

        // Display an indicator of whether IR is on or off
        if(button_is_pressed(BUTTON_C))
                print("IR-");
        else
                print("  C");

        delay_ms(100);
}

int m1_speed = 0;
int m2_speed = 0;

void motor_test()
{
        static char m1_back = 0, m2_back = 0;
        char m1_char, m2_char;

        delay_ms(1000);
```

```
	for (i=0;i<=51;i++)
	{
		if(i<25)
		{
			m1_speed += 10;
			m2_speed += 10;
		}
		else if (i==25)
		{
			m1_back = !m1_back;
			m2_back = !m2_back;
			m1_speed=0;
			m2_speed=0;
			set_motors(0,0);
			delay_ms(1000);
		}
		else if (i>25)
		{
			m1_speed += 10;
			m2_speed += 10;
		}

		if(m1_speed < 0)
			m1_speed = 0;

		if(m1_speed > 255)
			m1_speed = 255;

		if(m2_speed < 0)
			m2_speed = 0;

		if(m2_speed > 255)
			m2_speed = 255;

		delay_ms(10);
		clear();
		// 255/26 = 9, so this gives values in the range of 0 to 9
		m1_char = bar_graph_characters[m1_speed / 26];
		print_character(m1_char);
		print_character(m1_back ? 'B' : 'F');
		print_character(m1_char);
		lcd_goto_xy(5,0);
		m2_char = bar_graph_characters[m2_speed / 26];
		print_character(m2_char);
		print_character(m2_back ? 'B' : 'F');
		print_character(m2_char);

		set_motors(m1_speed * (m1_back ? -1 : 1), m2_speed * (m2_back ? -1 : 1));
		delay_ms(50);
	}
}

const char fugue[] PROGMEM =
  "! T120O5L16agafaea dac+adaea fa<aa<bac#a dac#adaea f"
  "O6dcd<b-d<ad<g d<f+d<gd<ad<b- d<dd<ed<f+d<g d<f+d<gd<ad"
  "L8MS<b-d<b-d MLe-<ge-<g MSc<ac<a MLd<fd<f O5MSb-gb-g"
"ML>c#e>c#e MS afaf ML gc#gc# MS fdfd ML e<b-e<b-"
```

```
  "O6L16ragafaea dac#adaea fa<aa<bac#a dac#adaea faeadaca"
  "<b-acadg<b-g egdgcg<b-g <ag<b-gcf<af dfcf<b-f<af"
  "<gf<af<b-e<ge c#e<b-e<ae<ge <fe<ge<ad<fd"
  "O5e>ee>ef>df>d b->c#b->c#a>df>d e>ee>ef>df>d"
"e>d>c#>db>d>c#b >c#agaegfe fO6dc#dfdc#<b c#4";

const char fugue_title1[] PROGMEM = "\7 Fugue";
const char fugue_title2[] PROGMEM = "J.S.Bach \7";

void music_test()
{
	static char fugue_title_pos = 0;
	static long last_shift = 0;

	print_two_lines_delay_1s(fugue_title1,fugue_title2);
	play_from_program_space(fugue);
	while(is_playing());
}

void time_test()
{
	static long elapsed_time = 0;
	static long last_read = 0;
	static long is_ticking = 0;
	static char a_is_pressed = 0;
	static char c_is_pressed = 0;
	static char last_seconds = 0;

	while(!button_is_pressed(BUTTON_B));
	{
		long current_time = get_ms();
		if(is_ticking)
			elapsed_time += current_time - last_read;

		last_read = current_time;

		if(button_is_pressed(BUTTON_A) && !a_is_pressed)
		{
			// reset
			a_is_pressed = 1;
			is_ticking = 0;
			elapsed_time = 0;
			if(!is_playing()) // only play once
				play_from_program_space(beep_button_a);
		}

		// find the end of the button press without stopping
		if(!button_is_pressed(BUTTON_A))
			a_is_pressed = 0;

		if(button_is_pressed(BUTTON_C) && !c_is_pressed)
		{
			// start/stop
			c_is_pressed = 1;
			is_ticking = !is_ticking;
			play_from_program_space(beep_button_c);
		}
```

```
                // find the end of the button press without stopping
                if(!button_is_pressed(BUTTON_C))
                        c_is_pressed = 0;

                print_long((elapsed_time/1000/60/10)%10); // tens of minutes
                print_long((elapsed_time/1000/60)%10); // minutes
                print_character(':');
                print_long((elapsed_time/1000)%60/10); // tens of seconds
                char seconds = ((elapsed_time/1000)%60)%10;
                print_long(seconds); // seconds
                print_character('.');
                print_long((elapsed_time/100)%10); // tenths of seconds
                print_long((elapsed_time/10)%10); // hundredths of seconds

                // beep every second
                if(seconds != last_seconds && elapsed_time != 0 && !is_playing())
                        play_from_program_space(timer_tick);
                last_seconds = seconds;
        }
}

void adv_coordinates()
{
        long mov;
        int m1vel,m2vel;

        clear();

        i=0;

        while(i<8)
        {
                switch(comando[i])
                {
                        case 'U':               i++;
                                                mov=tiempo(comando[i]);
                                                print_character(comando[i]);
                                                set_motors(50, 50);
                                                mov=mov*500;
                                                delay_ms(mov);
                                                break;

                        case 'D':               i++;
                                                mov=tiempo(comando[i]);
                                                print_character(comando[i]);
                                                set_motors(-50, -50);
                                                mov=mov*500;
                                                delay_ms(mov);
                                                break;

                        case 'L':               i++;
                                                set_motors(-50,50);
                                                delay_ms(310);
                                                mov=tiempo(comando[i]);
                                                print_character(comando[i]);
                                                set_motors(50, 50);
```

```
                                                mov=mov*500;
                                                delay_ms(mov);
                                                break;

                        case 'R':               i++;
                                                set_motors(50,-50);
                                                delay_ms(310);
                                                mov=tiempo(comando[i]);
                                                print_character(comando[i]);
                                                set_motors(50, 50);
                                                mov=mov*500;
                                                delay_ms(mov);
                                                break;
                }
                i++;
        }
}

long tiempo(char seleccionar)
{
        switch(seleccionar)
        {
                case '1': return 1; break;
                case '2': return 2; break;
                case '3': return 3; break;
                case '4': return 4; break;
                case '5': return 5; break;
                case '6': return 6; break;
                case '7': return 7; break;
                case '8': return 8; break;
                case '9': return 9; break;
                case '0': return 0; break;
        }
}

void print_two_lines_delay_1s(const char *line1, const char *line2)
{
        // Play welcome music and display a message
        clear();
        print_from_program_space(line1);
        lcd_goto_xy(0,1);
        print_from_program_space(line2);
        delay_ms(1000);
}

// waits for a button, plays the appropriate beep, and returns the
// button or buttons that were pressed
char wait_for_button_and_beep()
{
        char button = wait_for_button_press(ANY_BUTTON);

        if(button & BUTTON_A)
                play_from_program_space(beep_button_a);
        else if(button & BUTTON_B)
                play_from_program_space(beep_button_b);
        else
                play_from_program_space(beep_button_c);
```

```
	wait_for_button_release(button);
	return button;
}

// Initializes the 3pi, displays a welcome message, calibrates, and
// plays the initial music.
void initialize()
{
	// This must be called at the beginning of 3pi code, to set up the
	// sensors.  We use a value of 2000 for the timeout, which
	// corresponds to 2000*0.4 us = 0.8 ms on our 20 MHz processor.
	pololu_3pi_init(2000);
	load_custom_characters(); // load the custom characters

	//Calibrate the motors
	//auto_calibrate();

	//Setting the Baud Rate
	serial_set_baud_rate(115200);


	serial_receive_ring(receive_buffer, sizeof(receive_buffer));

	play_from_program_space(welcome);
	print_two_lines_delay_1s(welcome_line1,welcome_line2);
	print_two_lines_delay_1s(title_project1,title_project2);
	print_two_lines_delay_1s(integrantes_1_1,integrantes_1_2);
	print_two_lines_delay_1s(integrantes_2_1,integrantes_2_2);
	print_two_lines_delay_1s(project_name1,project_name2);
	print_two_lines_delay_1s(project_name3,project_name4);
}


void Estableshing_Connection()
{
	//UART Communication with NAROBO to set parameters

	//Commands for internet connection
	const char AT[] PROGMEM = "AT\x0d";
	const char CSTT[] PROGMEM = "AT+CSTT=\"internet.porta.com.ec\",\"porta\",\"porta\"\x0d";
	const char CIICR[] PROGMEM = "AT+CIICR\x0d";
	const char CIFSR[] PROGMEM = "AT+CIFSR\x0d";
	const char CIPSTART[] PROGMEM = "AT+CIPSTART=\"TCP\",\"200.126.14.165\",\"80\"\x0d";
	const char CIPSEND[] PROGMEM = "AT+CIPSEND\x0d";
	const char estado[] PROGMEM = "OK\n\r\n\r\x1A";
	const char CIPSHUT[] PROGMEM = "AT+CIPSHUT\x0d";

	//Begin Connection Music
	play_from_program_space(connecting_music);

	//LCD Message
	print_two_lines_delay_1s(connecting_line1,connecting_line2);

	//Read ring buffer
	serial_receive_blocking(receive_buffer,1, 1000);
```

```
//Cleaning the buffer
for (i=0;i<=32;i++)
{
        receive_buffer[i]=0;
}

//AT to get ready the NAROBO
serial_send(AT,sizeof(AT));
while(serial_receive_blocking(receive_buffer,1, 1000));
clear();
print("AT");
lcd_goto_xy(0,1);
print(receive_buffer);
delay_ms(2000);

//Set the connections settings
serial_send(CSTT,sizeof(CSTT));
while(serial_receive_blocking(receive_buffer,1, 1000));
clear();
print("AT+CSTT");
lcd_goto_xy(0,1);
print(receive_buffer);
delay_ms(2000);

//Set the connection start
serial_send(CIICR,sizeof(CIICR));
while(serial_receive_blocking(receive_buffer,1, 1000));
clear();
print("AT+CIICR");
lcd_goto_xy(0,1);
print(receive_buffer);
delay_ms(2000);

//Retreive IP asigned
serial_send(CIFSR,sizeof(CIFSR));
while(serial_receive_blocking(receive_buffer,15, 1000));
clear();
print("AT+CIFSR");
lcd_goto_xy(0,1);
print(receive_buffer);
delay_ms(2000);

//Cleaning the buffer
for (i=0;i<=32;i++)
{
        receive_buffer[i]=0;
}

//Start Connection
serial_send(CIPSTART,sizeof(CIPSTART));
while(serial_receive_blocking(receive_buffer,5, 1000));
clear();
print("AT+CIPSTART");
lcd_goto_xy(0,1);
print(receive_buffer);
delay_ms(2000);
```

```
//Cleaning the buffer
for (i=0;i<=32;i++)
{
        receive_buffer[i]=0;
}

//Prepare to send message
serial_send(CIPSEND,sizeof(CIPSEND));
while(serial_receive_blocking(receive_buffer,1, 1000));
clear();
print("AT+CIPSEND");
lcd_goto_xy(0,1);
print(receive_buffer);
delay_ms(1000);

//Cleaning the buffer
for (i=0;i<=32;i++)
{
        receive_buffer[i]=0;
}

//Sending verification message to the server
serial_send(estado,sizeof(estado));
while(serial_receive_blocking(receive_buffer,7, 1000));
clear();
print("Receive");
lcd_goto_xy(0,1);
print(receive_buffer);
delay_ms(2000);

//Cleaning the buffer
for (i=0;i<=32;i++)
{
        receive_buffer[i]=0;
}

while(serial_receive_blocking(receive_buffer,8, 1000));

//Command
for (i=0;i<=7;i++)
{
        comando[i]=receive_buffer[i];
}
ejecutar=comando[7];
//menu_index=2;

switch(ejecutar)
{
        case '1':menu_index=0; break;
        case '2':menu_index=1; break;
        case '3':menu_index=2; break;
        case '4':menu_index=3; break;
        case '5':menu_index=4; break;
        case '6':menu_index=5; break;
}

clear();
```

```
	print("Comando");
	lcd_goto_xy(0,1);
	print(comando);
	delay_ms(2000);

	//Cleaning the buffer
	for (i=0;i<=32;i++)
	{
		receive_buffer[i]=0;
	}

	//Close Connection
	serial_send(CIPSHUT,sizeof(CIPSHUT));
	while(serial_receive_blocking(receive_buffer,3, 1000));
	clear();
	print("AT+CIPSHUT");
	lcd_goto_xy(0,1);
	print(receive_buffer);
	delay_ms(2000);
}

void manual()
{

	print_two_lines_delay_1s(main_menu_intro_line1,main_menu_intro_line2);

	while(salir)
	{
		clear();
		lcd_goto_xy(0,1);
		print_from_program_space(menu_line2);
		lcd_goto_xy(0,0);
		print_from_program_space(main_menu_options[menu_index]);
		lcd_show_cursor(CURSOR_BLINKING);
		// the cursor will be blinking at the end of the option name

		// wait for all buttons to be released, then a press
		while(button_is_pressed(ANY_BUTTON));
		char button = wait_for_button_press(ANY_BUTTON);

		if(button & BUTTON_A)
		{
			play_from_program_space(beep_button_a);
			menu_index --;
		}
		else
		if(button & BUTTON_B)
		{
			lcd_hide_cursor();
			clear();

			play_from_program_space(beep_button_b);
			wait_for_button_release(button);

			if (menu_index == 0)
			{
				main_menu_functions[menu_index]();
```

```
                        menu_index++;
                }
                else
                {
                        lcd_goto_xy(0,1);
                        print_from_program_space(back_line2);
                        lcd_goto_xy(0,0);
                        main_menu_functions[menu_index]();
                }

                set_motors(0,0);
                stop_playing();
                m1_speed = 0;
                m2_speed = 0;
                red_led(0);
                green_led(0);
                play_from_program_space(beep_button_b);
                wait_for_button_release(button);

        }
        else if(button & BUTTON_C)
        {
                play_from_program_space(beep_button_c);
                menu_index ++;
        }

        if(menu_index < 0)
                menu_index = main_menu_length-1;
        if(menu_index >= main_menu_length)
                menu_index = 0;
    }
}

void main()
{
    initialize();

    while(1)
    {
        clear();
        print_from_program_space(tipo_trabajo1);
        lcd_goto_xy(0,1);
        print_from_program_space(tipo_trabajo2);
        lcd_show_cursor(CURSOR_BLINKING);
        // the cursor will be blinking at the end of the option name

        // wait for all buttons to be released, then a press
        while(button_is_pressed(ANY_BUTTON));
        char button = wait_for_button_press(ANY_BUTTON);

        if(button & BUTTON_A)
        {
                salir=1;

                manual();

                set_motors(0,0);
```

```
                stop_playing();
                m1_speed = 0;
                m2_speed = 0;
                red_led(0);
                green_led(0);
                play_from_program_space(beep_button_b);
        }
        else
        if(button & BUTTON_B)
        {

                while(salir)
                {
                        lcd_hide_cursor();
                        clear();
                        play_from_program_space(beep_button_b);
                        wait_for_button_release(button);

                        //Estableshing Connection
                        Estableshing_Connection();

                        lcd_goto_xy(0,1);
                        print_from_program_space(back_line2);
                        lcd_goto_xy(0,0);
                        main_menu_functions[menu_index]();

                        set_motors(0,0);
                        stop_playing();
                        m1_speed = 0;
                        m2_speed = 0;
                        red_led(0);
                        green_led(0);
                        play_from_program_space(beep_button_b);
                }
        }
    }
}
```

3.5 Funciones implementadas en el Microcontrolador

3.5.1 Inicialización

Primero se inicializa las constantes que se van a usar en la presentación del menú del display, al igual que las palabras que se usan en el menú y las posiciones de memoria.

```
#include <avr/pgmspace.h>

const char welcome_line1[] PROGMEM = " Pololu";
const char welcome_line2[] PROGMEM = "3\xf7 Robot"; // \xf7 is a greek
```

```
const char title_project1[] PROGMEM = "Proyecto";
const char title_project2[] PROGMEM = "Graduacion";

const char project_name1[] PROGMEM = "Comunicacion";
const char project_name2[] PROGMEM = "Inalambrica";
const char project_name3[] PROGMEM = "NAROBO";
const char project_name4[] PROGMEM = "DRONECELL";

const char          wait_comand1[] PROGMEM = "Esperando";
const char          wait_comand2[] PROGMEM = "Comando";

const char thank_you_line1[] PROGMEM = " Thank ";
const char thank_you_line2[] PROGMEM = "  you!";

const char main_menu_intro_line1[] PROGMEM = "  Main";
const char main_menu_intro_line2[] PROGMEM = "  Menu";

const char menu_bat_test[] PROGMEM = "Battery";
const char menu_led_test[] PROGMEM = "LEDs";
const char menu_lcd_test[] PROGMEM = "LCD";
const char menu_ir_test[] PROGMEM = "Sensors";
const char menu_motor_test[] PROGMEM = "Motors";
const char menu_music_test[] PROGMEM = "Music";
const char menu_pot_test[] PROGMEM = "Trimpot";
const char menu_time_test[] PROGMEM = "Timer";
const char menu_metter_test[] PROGMEM = "Metter";

const char menu_line2[] PROGMEM = "\x7f" "A \xa5" "B C\x7e";
const char back_line2[] PROGMEM = "\6B";

int position_x,position_y; //Actual Position Indicators

void bat_test();
void led_test();
void lcd_test();
void ir_test();
void motor_test();
void music_test();
void time_test();
void adv_coordinates();
typedef void (*function)();
const function main_menu_functions[] = { bat_test, led_test, ir_test, motor_test, music_test, time_test,
adv_coordinates};
const char *main_menu_options[] = { menu_bat_test, menu_led_test, menu_ir_test, menu_motor_test,
menu_music_test, menu_time_test, menu_metter_test };
const char main_menu_length = sizeof(main_menu_options)/sizeof(main_menu_options[0]);

// A couple of simple tunes, stored in program space.
const char welcome[] PROGMEM = ">g32>>c32";
const char thank_you_music[] PROGMEM = ">>c32>g32";
const char beep_button_a[] PROGMEM = "!c32";
const char beep_button_b[] PROGMEM = "!e32";
const char beep_button_c[] PROGMEM = "!g32";
const char timer_tick[] PROGMEM = "!v8>>c32";

// Data for generating the characters used in load_custom_characters
// and display_readings.  By reading levels[] starting at various
```

```
// offsets, we can generate all of the 7 extra characters needed for a
// bargraph.  This is also stored in program space.
const char levels[] PROGMEM = {
        0b00000,
        0b00000,
        0b00000,
        0b00000,
        0b00000,
        0b00000,
        0b00000,
        0b11111,
        0b11111,
        0b11111,
        0b11111,
        0b11111,
        0b11111,
        0b11111
};

// This character is a musical note.
const prog_char note[] PROGMEM = {
        0b00100,
        0b00110,
        0b00101,
        0b00101,
        0b00100,
        0b11100,
        0b11100,
        0b00000,
};

// This character is a back arrow.
const prog_char back_arrow[] PROGMEM = {
        0b00000,
        0b00010,
        0b00001,
        0b00101,
        0b01001,
        0b11110,
        0b01000,
        0b00100,
};
```

3.5.2 Funciones

- **LOAD_CUSTOM_CHARACTERS:** Carga los mensajes o símbolos que posteriormente se verán en la pantalla LCD.

```
void load_custom_characters()
```

```
{
    lcd_load_custom_character(levels+0,0); // no offset, e.g. one bar
    lcd_load_custom_character(levels+1,1); // two bars
    lcd_load_custom_character(levels+2,2); // etc...
    lcd_load_custom_character(levels+4,3); // skip level 3
    lcd_load_custom_character(levels+5,4);
    lcd_load_custom_character(levels+6,5);
    lcd_load_custom_character(back_arrow,6);
    lcd_load_custom_character(note,7);
    clear(); // the LCD must be cleared for the characters to take effect
}
```

- **AUTO_CALIBRATE:** Auto-calibra los elementos del robot Pololu 3π, y al final manda un mensaje a través del módulo serial.

```
void auto_calibrate()
{
  time_reset();
  set_motors(60, -60);
  while(get_ms() < 250)
    calibrate_line_sensors(IR_EMITTERS_ON);
  set_motors(-60, 60);
  while(get_ms() < 750)
    calibrate_line_sensors(IR_EMITTERS_ON);
  set_motors(60, -60);
  while(get_ms() < 1000)
    calibrate_line_sensors(IR_EMITTERS_ON);
  set_motors(0, 0);
  green_led(1); red_led(0);
  delay_ms(250);
  green_led(0); red_led(1);
  delay_ms(250);
  green_led(0); red_led(0);
  play_from_program_space(beep_button_a);
  delay_ms(300);

  //serial_send_blocking("c",1);
}
```

- **BAT_TEST:** Esta función permite verificar el estado de la batería, mostrando el valor de carga que esta posee.

```
void bat_test()
{
    int bat = read_battery_millivolts();

    print_long(bat);
    print("mV");

    delay_ms(100);
}
```

- **CHAR WAIT_FOR_250_MS_OR_BUTTON_B:** Permite esperar 250 ms milisegundos al microcontrolador para presionar el botón B, si esto sucede devuelve 1, caso contrario 0.

```
char wait_for_250_ms_or_button_b()
{
    int i;
    for(i=0;i<25;i++)
    {
            delay_ms(10);
            if(button_is_pressed(BUTTON_B))
                    return 1;
    }
    return 0;
}
```

- **LED_TEST:** Esta función permite probar el estado de los Leds conectados al Pololu 3π, si se presiona el botón B se sale de esta rutina.

```
void led_test()
{
    play("c32");
    print("Red  ");

    red_led(1);
    if(wait_for_250_ms_or_button_b())
            return;
    red_led(0);
    if(wait_for_250_ms_or_button_b())
            return;

    play(">c32");
    lcd_goto_xy(0,0);
    print("Green");

    green_led(1);
    if(wait_for_250_ms_or_button_b())
            return;
    green_led(0);
    if(wait_for_250_ms_or_button_b())
            return;
}
```

- **IR_TEST**: Esta función permiteimplementar el seguidor de línea el cual permite al robot de forma autónoma seguir una trayectoria previamente establecida.

```
void ir_test()
```

```
{
    unsigned int sensors[5]; // an array to hold sensor values

    if(button_is_pressed(BUTTON_C))
        read_line_sensors(sensors, IR_EMITTERS_OFF);
    else
        read_line_sensors(sensors,IR_EMITTERS_ON);

    unsigned char i;

    for(i=0;i<5;i++) {
        // Initialize the array of characters that we will use for the
        // graph.  Using the space, an extra copy of the one-bar
        // character, and character 255 (a full black box), we get 10
        // characters in the array.

        // The variable c will have values from 0 to 9, since
        // values are in the range of 0 to 2000, and 2000/201 is 9
        // with integer math.
        char c = bar_graph_characters[sensors[i]/201];

        // Display the bar graph characters.
        print_character(c);

    }

    // Display an indicator of whether IR is on or off
    if(button_is_pressed(BUTTON_C))
        print("IR-");
    else
        print("  C");

    delay_ms(100);
}
```

- **MOTOR_TEST:** Permite comprobar el funcionamiento correcto de los motores.

```
void motor_test()
{
    static char m1_back = 0, m2_back = 0;
    char m1_char, m2_char;

    if(button_is_pressed(BUTTON_A))
    {
        if(m1_speed == 0)
        {
            delay_ms(200);

            // If the button is pressed quickly when the motor is off,
            // reverse direction.
            if(!button_is_pressed(BUTTON_A))
                m1_back = !m1_back;
        }
```

```
            m1_speed += 10;
        }
        else
            m1_speed -= 20;

        if(button_is_pressed(BUTTON_C))
        {
            if(m2_speed == 0)
            {
                delay_ms(200);

                // If the button is pressed quickly when the motor is off,
                // reverse direction.
                if(!button_is_pressed(BUTTON_C))
                    m2_back = !m2_back;
            }

            m2_speed += 10;
        }
        else
            m2_speed -= 20;

        if(m1_speed < 0)
            m1_speed = 0;

        if(m1_speed > 255)
            m1_speed = 255;

        if(m2_speed < 0)
            m2_speed = 0;

        if(m2_speed > 255)
            m2_speed = 255;

        // 255/26 = 9, so this gives values in the range of 0 to 9
        m1_char = bar_graph_characters[m1_speed / 26];
        m2_char = bar_graph_characters[m2_speed / 26];
        print_character(m1_char);
        print_character(m1_back ? 'a' : 'A');
        print_character(m1_char);
        lcd_goto_xy(5,0);
        print_character(m2_char);
        print_character(m2_back ? 'c' : 'C');
        print_character(m2_char);

        set_motors(m1_speed * (m1_back ? -1 : 1), m2_speed * (m2_back ? -1 : 1));
        delay_ms(50);
    }
```

- **MUSIC_TEST:** Esta función permite comprobar el funcionamiento del buzzer tocando una pieza musical.

```
void music_test()
{
        static char fugue_title_pos = 0;
```

```
	static long last_shift = 0;
	char c,i;

	if(get_ms() - last_shift > 250)
	{
		for(i=0;i<8;i++)
		{
			c = pgm_read_byte(fugue_title + fugue_title_pos + i);
			print_character(c);
		}
		last_shift = get_ms();

		fugue_title_pos ++;
		if(fugue_title_pos + 8 >= sizeof(fugue_title))
			fugue_title_pos = 0;
	}

	if(!is_playing())
	{
		play_from_program_space(fugue);
	}

	delay_ms(100);
}
```

- **TIME_TEST**: Esta función se encarga de comprobar la funcionalidad del tiempo realizando la simulación de un cronómetro en el cual se muestran horas minutos y segundos.

```
void time_test()
{
	static long elapsed_time = 0;
	static long last_read = 0;
	static long is_ticking = 0;
	static char a_is_pressed = 0;
	static char c_is_pressed = 0;
	static char last_seconds = 0;

	long current_time = get_ms();
	if(is_ticking)
		elapsed_time += current_time - last_read;

	last_read = current_time;

	if(button_is_pressed(BUTTON_A) && !a_is_pressed)
	{
		// reset
		a_is_pressed = 1;
		is_ticking = 0;
		elapsed_time = 0;
		if(!is_playing()) // only play once
			play_from_program_space(beep_button_a);
	}
```

```
        // find the end of the button press without stopping
        if(!button_is_pressed(BUTTON_A))
                a_is_pressed = 0;

        if(button_is_pressed(BUTTON_C) && !c_is_pressed)
        {
                // start/stop
                c_is_pressed = 1;
                is_ticking = !is_ticking;
                play_from_program_space(beep_button_c);
        }

        // find the end of the button press without stopping
        if(!button_is_pressed(BUTTON_C))
                c_is_pressed = 0;

        print_long((elapsed_time/1000/60/10)%10); // tens of minutes
        print_long((elapsed_time/1000/60)%10); // minutes
        print_character(':');
        print_long((elapsed_time/1000)%60/10); // tens of seconds
        char seconds = ((elapsed_time/1000)%60)%10;
        print_long(seconds); // seconds
        print_character('.');
        print_long((elapsed_time/100)%10); // tenths of seconds
        print_long((elapsed_time/10)%10); // hundredths of seconds

        // beep every second
        if(seconds != last_seconds && elapsed_time != 0 && !is_playing())
                play_from_program_space(timer_tick);
        last_seconds = seconds;
}
```

- **ADV_COORDINATES:** Esta función se encarga de movilizar el robot Pololu en una ruta especificada la cual le indica la dirección y cuantos segundos se desea que se movilice en esa dirección con un máximo de 4 instrucciones por ruta y con un valor en dígitos de 0 a 9 segundos.

```
void adv_coordinates()
{
        long mov;
        int m1vel,m2vel;

        clear();

        i=0;

        while(i<8)
        {
                switch(comando[i])
                {
```

```
                        case 'U':       i++;
                                        mov=tiempo(comando[i]);
                                        print_character(comando[i]);
                                        set_motors(50, 50);
                                        mov=mov*500;
                                        delay_ms(mov);
                                        break;

                        case 'D':       i++;
                                        mov=tiempo(comando[i]);
                                        print_character(comando[i]);
                                        set_motors(-50, -50);
                                        mov=mov*500;
                                        delay_ms(mov);
                                        break;

                        case 'L':       i++;
                                        set_motors(-50,50);
                                        delay_ms(310);
                                        mov=tiempo(comando[i]);
                                        print_character(comando[i]);
                                        set_motors(50, 50);
                                        mov=mov*500;
                                        delay_ms(mov);
                                        break;

                        case 'R':       i++;
                                        set_motors(50,-50);
                                        delay_ms(310);
                                        mov=tiempo(comando[i]);
                                        print_character(comando[i]);
                                        set_motors(50, 50);
                                        mov=mov*500;
                                        delay_ms(mov);
                                        break;
                }
                i++;
        }
}
```

- **TIEMPO:** Esta función se encarga de evaluar cuantos segundos se va a movilizar el robot en una dirección especificada.

```
long tiempo(char seleccionar)
{
        switch(seleccionar)
        {
                case '1': return 1; break;
                case '2': return 2; break;
                case '3': return 3; break;
                case '4': return 4; break;
                case '5': return 5; break;
                case '6': return 6; break;
                case '7': return 7; break;
```

```
                case '8': return 8; break;
                case '9': return 9; break;
                case '0': return 0; break;
            }
        }
```

- **PRINT_TWO_LINES_DELAY_1S:** Se le ingresa 2 líneas de código las cuales son mostradas a través del LCD, luego espera un segundo y luego dependiendo del botón que se presione toca una melodía (beep).

```
void print_two_lines_delay_1s(const char *line1, const char *line2)
{
    // Play welcome music and display a message
    clear();
    print_from_program_space(line1);
    lcd_goto_xy(0,1);
    print_from_program_space(line2);
    delay_ms(1000);
}
```

- **WAIT_FOR_BUTTON_AND_BEEP:** Esta función espera hasta que una tecla sea presionada y luego devuelve el carácter representando el botón que se ha presionado.

```
char wait_for_button_and_beep()
{
    char button = wait_for_button_press(ANY_BUTTON);

    if(button & BUTTON_A)
        play_from_program_space(beep_button_a);
    else if(button & BUTTON_B)
        play_from_program_space(beep_button_b);
    else
        play_from_program_space(beep_button_c);

    wait_for_button_release(button);
    return button;
}
```

- **INITIALIZE:** Es una función la cual permite configurar inicialmente los elementos que conforman el robot Pololu 3π.

```
void initialize()
{
    // This must be called at the beginning of 3pi code, to set up the
```

```
    // sensors.  We use a value of 2000 for the timeout, which
    // corresponds to 2000*0.4 us = 0.8 ms on our 20 MHz processor.
    pololu_3pi_init(2000);
    load_custom_characters(); // load the custom characters

    //Calibrate the motors
    auto_calibrate();

    play_from_program_space(welcome);
    print_two_lines_delay_1s(welcome_line1,welcome_line2);
    print_two_lines_delay_1s(title_project1,title_project2);
    print_two_lines_delay_1s(project_name1,project_name2);
    print_two_lines_delay_1s(project_name3,project_name4);

    clear();
    print_from_program_space(wait_comand1);
    lcd_goto_xy(0,1);
    print_from_program_space(wait_comand2);

    while(!(wait_for_button_and_beep() & BUTTON_B));

    play_from_program_space(thank_you_music);

    print_two_lines_delay_1s(thank_you_line1,thank_you_line2);
}
```

- **MENU_SELECT:** Es la función encargada de mostrar los mensajes a través del LCD para que dependiendo del botón que se presione realice una función en específico.

```
void menu_select()
{
    static int menu_index = 0;

    print_two_lines_delay_1s(main_menu_intro_line1,main_menu_intro_line2);

    while(1)
    {
            clear();
            lcd_goto_xy(0,1);
            print_from_program_space(menu_line2);
            lcd_goto_xy(0,0);
            print_from_program_space(main_menu_options[menu_index]);
            lcd_show_cursor(CURSOR_BLINKING);
            // the cursor will be blinking at the end of the option name

            // wait for all buttons to be released, then a press
            while(button_is_pressed(ANY_BUTTON));
            char button = wait_for_button_press(ANY_BUTTON);

            if(button & BUTTON_A)
            {
                    play_from_program_space(beep_button_a);
```

```
                menu_index --;
        }
        else if(button & BUTTON_B)
        {
                lcd_hide_cursor();
                clear();

                play_from_program_space(beep_button_b);
                wait_for_button_release(button);

                while(!button_is_pressed(BUTTON_B))
                {
                        lcd_goto_xy(0,1);
                        print_from_program_space(back_line2);
                        lcd_goto_xy(0,0);
                        main_menu_functions[menu_index]();
                }

                set_motors(0,0);
                stop_playing();
                m1_speed = 0;
                m2_speed = 0;
                red_led(0);
                green_led(0);
                play_from_program_space(beep_button_b);

                return;
        }
        else if(button & BUTTON_C)
        {
                play_from_program_space(beep_button_c);
                menu_index ++;
        }

        if(menu_index < 0)
                menu_index = main_menu_length-1;
        if(menu_index >= main_menu_length)
                menu_index = 0;
    }
}
```

Capítulo 4

4 SIMULACIONES Y PRUEBAS

En este capítulo podemos observar las simulaciones que se realizaron previo a la implementación del proyecto en físico y nos ayudó a tener una mejor idea de que es lo que esperamos observar cuando ya se encuentre armado el esquema.

Al final observando detenidamente el proyecto funcionando detallaremos nuestras conclusiones y recomendaciones con referencia al mismo y conclusiones los cuales indicarán puntos a tener en cuenta para cambios del proyecto a futuro o algún nuevo punto de vista que se le quiera dar.

4.1 Simulación en PROTEUS

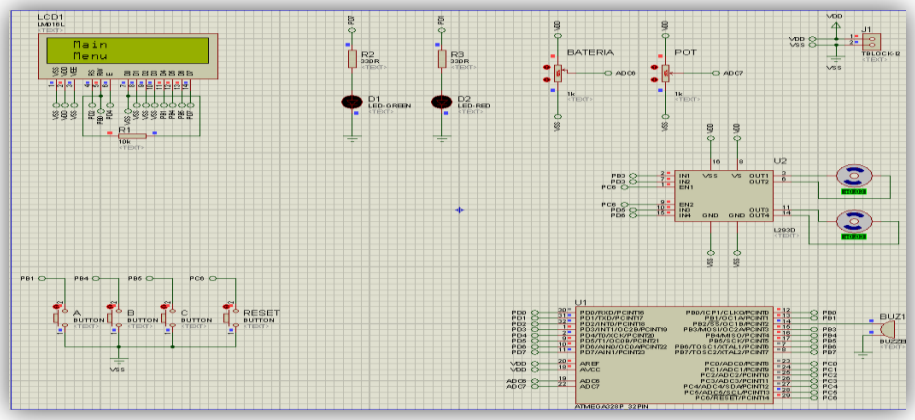

Ilustración 25 Menú de Inicio

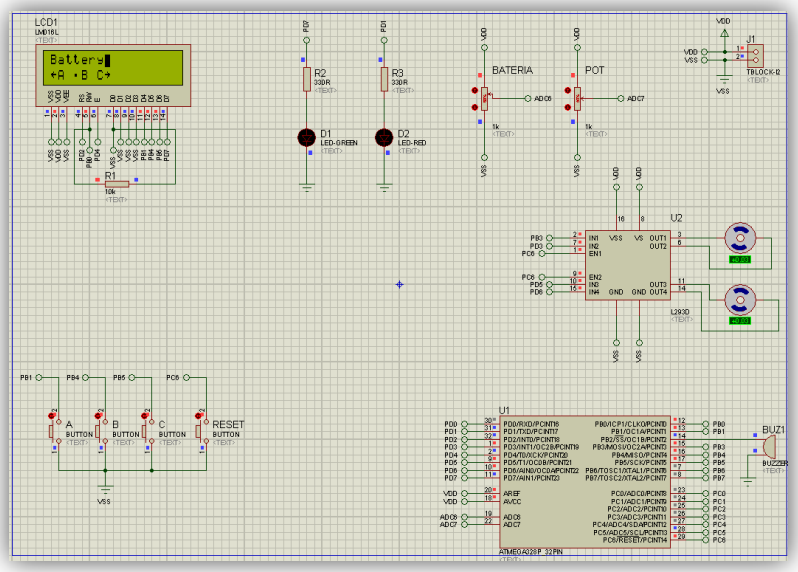

Ilustración 26 Primera Opción del Menú

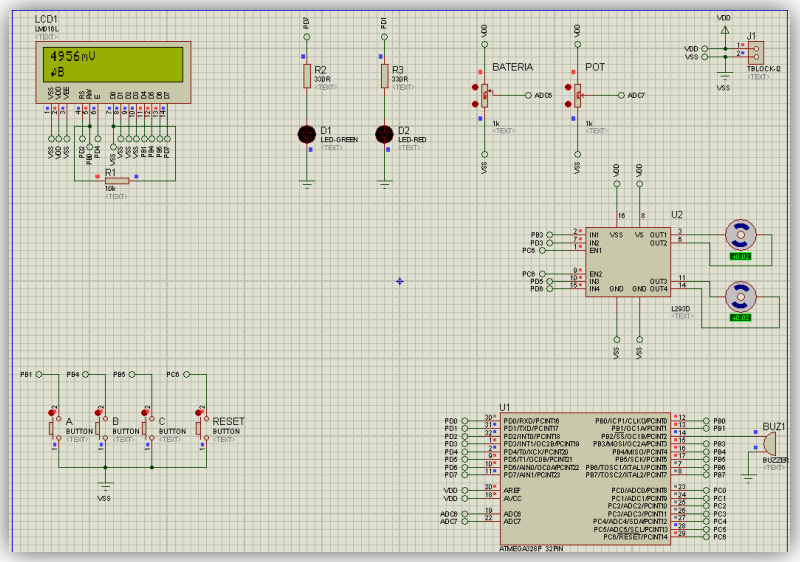

Ilustración 27 Estado de la Batería

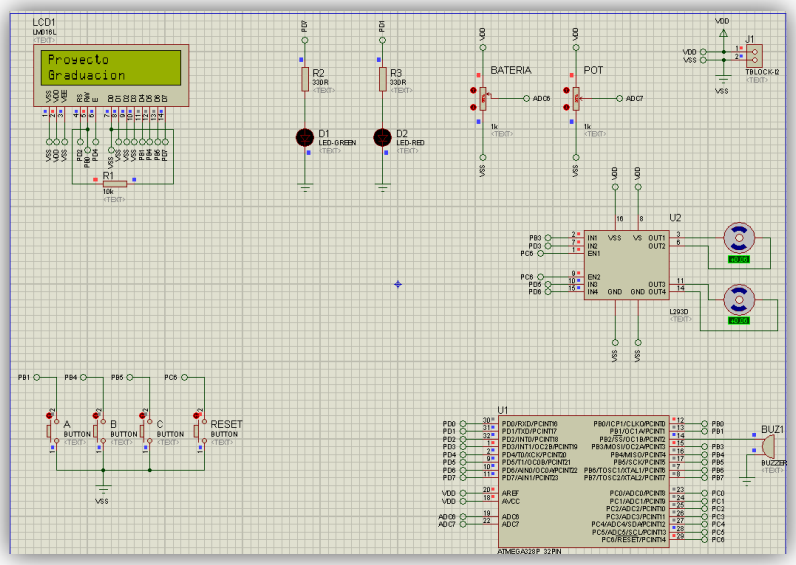

Ilustración 28 Mansaje de Inicio

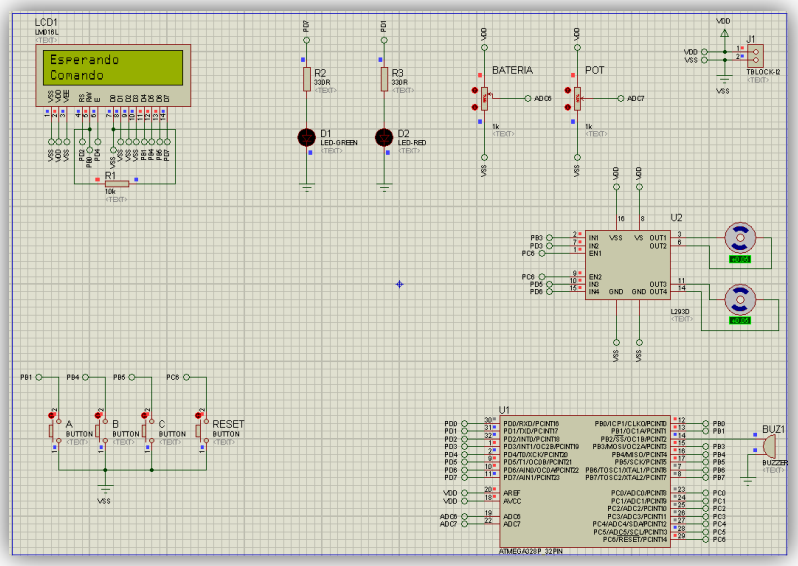

Ilustración 29 Esperando comandos del Narobo DroneCell

4.2 Pruebas Físicas

En este capítulo observaremos el funcionamiento del proyecto con todas las partes en funcionamiento, lo cual nos indicara como se desenvolvió y que conclusiones podemos obtener del mismo.

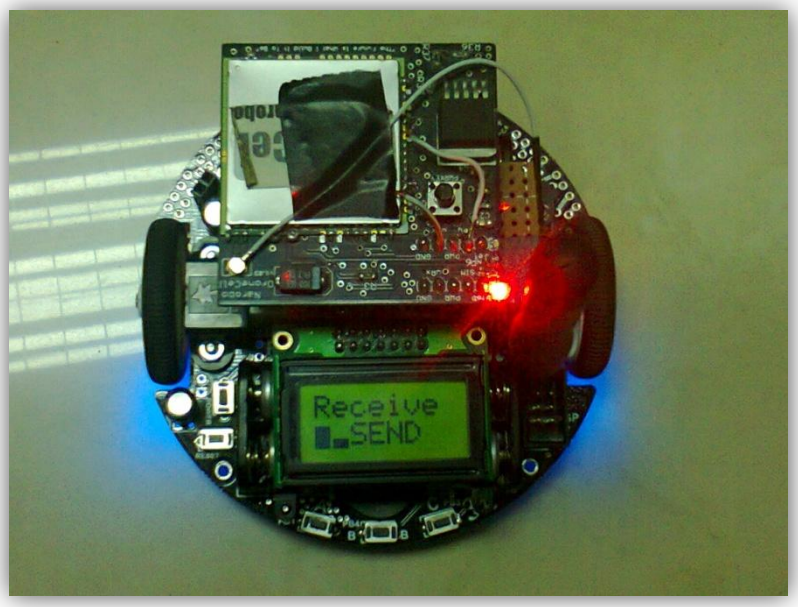

Ilustración 30Pololu conectado a Internet mediante NAROBO DroneCell

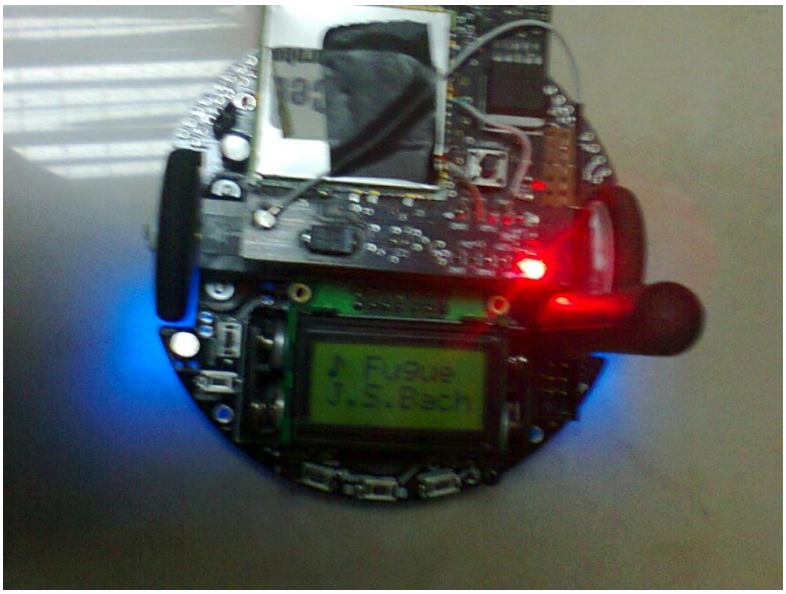

Ilustración 31Ejecutando instrucción enviada desde Internet

Algunos aspectos a tener en cuenta son:

- La conexión del Narobo DroneCell a internet es un poco lenta se demora aproximadamente unos 10 segundos para lo cual el Narobo debe estar en un modo de espera en el cual está a la expectativa para saber si entro o no al internet.
- La comunicación entre la terminal remota y el robot solo se puede dar siempre y cuando la terminal remota se comporte como un servidor TCP, debido a que cuando el Narobo DroneCell se conecta el internet necesita engancharse a una dirección IP fija la cual este a la expectativa de un mensaje directo a esa IP.
- La comunicación entre el robot y el servidor TCP es Half Duplex lo cual indica que mientras uno está enviando datos el otro solo está a la expectativa de recibirlas y viceversa.
- Para evitar problemas es necesario que cada vez que se recibe un dato a través del Narobo DroneCell se cierre la comunicación y luego de realizar la acción escogida se la vuelva a conectar.

Conclusiones y Recomendaciones

- Las configuraciones previas que se hicieron con el módulo Narobo y con el Pololu 3π son debido a que se necesita ajustar o calibrar estos para que reciban los datos o tramas de bits que son nada más que códigos hexadecimales de los cuales se extraen el comando que se le envía al robot debido a que el mensaje vine incluido una gran cantidad de datos que indica información del destinatario y del remitente.
- Debido a que la configuración del Narobo es DHCP debido a eso es necesario calibrar al dispositivo porque cada vez que se enciende el dispositivo este se carga con una nueva dirección IP o sea la asignación del mismo es dinámica.
- Es necesario tener en cuenta que el Narobo DroneCell actúa como un modem GSM/GPRS por lo que es necesario contar con un chip el cual nos permita navegar en internet sin ningún problema, pero hay que tener en consideración que si el chip no tiene la configuración correcta el Narobo no funcionara correctamente, es decir que no se podrá ingresar a internet.
- Debemos tomar en cuenta la alimentación de los elementos debido a que estos poseen ciertas limitaciones cuando las baterías están un poco agotadas, por ejemplo si se graba el Robot Pololu cuando la batería esta baja puede ocasionar que el programador se quema y que el Pololu ya no vuelva a ser capaz de regrabarse,

además las baterías del Narobo deben tener más del 70% para poder trabajar caso contrario no funciona.

- Para este proyecto se tuvo la necesidad de utilizar una aplicación que convierta a la computadora en un servidor TCP, para que esta sea la encargada de enviar los comandos a través de internet para que el robot pueda realizar alguna acción, debido a que se necesita constantemente enviar comandos al robot desde una dirección IP fija porque el Narobo debe estar siempre en espera de un comando enviado desde la IP que hace las veces de servidor.
- Debemos tener en cuenta que el buffer de recepción del Narobo debe estar configurado para poder recibir toda la trama de bits que se le envía desde la página web, dado que si el buffer es muy pequeño se eliminaran bytes importantes de los cuales se extraerá el comando que el robot deberá seguir, debido a que siempre el comando que se le envía al robot se encuentra al de la trama enviada.
- Es necesario tener en cuenta que el Narobo DroneCell actúa como un modem GSM/GPRS por lo que es necesario contar con un chip el cual nos permita navegar en internet sin ningún problema, pero hay que tener en consideración que si el chip no tiene la configuración correcta el Narobo no funcionara correctamente, es decir que no se podrá ingresar a internet.
- A través de la conexión realizada con el programa HERCULES SETUP y el Narobo se podrá controlar al Pololu enviándole comandos para ser ejecutados por el robot. El robot estará en modo de espera hasta que reciba un comando y mientras se esté ejecutando la acción no habrá enlace activo con el robot.

Anexos

Anexo A: Comandos AT

Los comandos AT son el protocolo usado para la comunicación con los módems, ahora también es utilizado en la telefonía celular GSM para que se logre la comunicación entre 2 o más terminales, en otras palabras se adoptaron a los comandos AT como el estándar para telefonía GSM en adelante.Los comandos AT permiten realizar llamadas de datos y voz, leer y escribir en la agenda de contactos, enviar mensajes SMS, conectarse a internet a través de los estándares GSM o GPRS, enviar mensajes por internet a una dirección IP específica entre otras cosas, y en este caso nos centraremos en las sentencias para conexión a internet:

- **AT**: Comando usado para comprobar el estado del elemento.
- **AT+CSTT**: Setea el APN (GPRS Access Point Name), el nombre del usuario y la contraseña.
- **AT+CIICR**: Genera la comunicación inalámbrica GPRS (General Packet Radio Service) o CSD (Circuit Switched Data).
- **AT+CIFSR**: Obtiene la dirección IP local.
- **AT+CIPSTART**: Comienza la conexión TCP (Transmission Control Protocol) o UDP(User Datagram Protocol).
- **AT+CIPSEND**: Envía información a través de la comunicación TCP o UDP.
- **AT+CIPSHUT**: Desactiva e contenido PDP GPRS.
- **AT+CIPCLOSE**: Cierra la conexión TCP o UDP.

Bibliografía

1. Hoja de Datos del microcontrolador ATMega328P.

   http://www.atmel.com/dyn/resources/prod_documents/doc8025.pdf

   Fecha de Consulta: 10/02/2011

2. Hoja de Datos del microcontrolador SIM340DZ

   http://narobo.com/products/DroneCell/Datasheets/SIM340DZ_ATC_V1.02.pdf

   Fecha de Consulta: 11/02/2011

3. Especificaciones del Narobo DroneCell

   www.Narobo.com

   Fecha de Consulta: 15/02/2011

4. Manual del robot Pololu 3π (Robot Pololu 3π Guia de Usuario)

   Fecha de Consulta: 17/02/2011

5. Planificación de red GSM/GPRS/EDGE Capitulo 3

   Fecha de Consulta: 19/02/2011

6. Gua de usuario del Narobo DroneCell

   http://narobo.com/products/DroneCell/Getting_Started.pdf

   Fecha de Consulta: 22/02/2011

7. http://en.wikipedia.org/wiki/GSM

   Fecha de Consulta: 25/02/2011

8. http://en.wikipedia.org/wiki/General_Packet_Radio_Service

   Fecha de Consulta: 25/02/2011