Universidad
Politécnica
de Cartagena

industriales
etsii UPCT

GNX Project v.1

Módulo de control programable vía bluetooth para aplicaciones de telemetría, telecontrol y posicionamiento vía satélite.

Titulación: Ingeniero técnico industrial, esp. Electrónica Industrial
Intensificación: Automática/Electrónica
Alumno/a: Daniel Carreres Prieto
Director/a/s: Manuel Sánchez Alonso

Cartagena, 4 de septiembre de 2012

Autor	Daniel Carreres Prieto
E-mail del Autor	dcarrpri@gmail.com
Director(es)	Manuel Sánchez Alonso
E-mail del Director	manuel.sanchez@upct.es
Título del PFC	GNX Project v.1.- Módulo de control programable vía bluetooth para aplicaciones de telemetría, telecontrol y posicionamiento vía satélite.

Resumen

En la actualidad existen muchas situaciones en las que se precisan de sistemas de control a fin automatizar determinados procesos o desempeñar tareas concretas para los cuales se requieren de costosos autómatas programables o conocimientos de electrónica y programación de microcontroladores.

A fin de poder paliar esta situación se plantea el diseño e implementación de un módulo hardware programable por medio de un Smartphone con posibilidad de ser configurado por el usuario a sus necesidades de una manera simple y sin precisar de conocimientos en programación o automática. Dicho módulo estará basado en la plataforma de software y hardware libre Arduino y se podrá orientar a diversas tareas tales como domótica, control y seguridad así como sistema de adquisición de datos y logger GPS.

Titulación	Ingeniero Técnico Industrial especialidad Electrónica industrial
Departamento	Electrónica, Tecnología de Computadoras y Proyectos
Fecha de presentación	Septiembre 2012

Índice General

I. Memoria

II. Planos y especificaciones

III. Apéndices

Índice de Figuras

Índice de Tablas

Agradecimientos

Quiero agradecer a mis padres, Pedro y Matilde todo su apoyo y comprensión, animándome a seguir intentándolo cuando las cosas no salían bien. Sin su ayuda no habría conseguido todo lo que he logrado hasta ahora.

Agradecer a mi director de proyecto, Don Manuel Sánchez Alonso que sin su ayuda este proyecto no podría haber desarrollado. Dándome libertar a la hora de decidir determinados aspectos del proyecto, proporcionándome todo el material que he precisado y echándome un cable muy grande con el tema del papeleo y las fechas.

Y por último, pero no por ello menos importante, quisiera agradecer a David Henarejos Navarro por su aportación vital a este proyecto. Enseñándome con muchísima paciencia a manejar los programas de diseño de placas y explicarme dudas referente a la electrónica.

A todos ellos, gracias.

I. Memoria

1. Introducción

1.1. Enunciado y objetivo del proyecto

Los continuos avances tecnológicos han permitido desarrollar multitud de equipos y dispositivos destinados a realizar tareas concretas de forma cada vez más eficiente. A este hecho hay que sumarle la automatización de los procesos de fabricación, lo que ha traído consigo un abaratamiento de dichos dispositivos permitiendo su acceso a un precio relativamente asequible.

Sin embargo, en multitud de ocasiones, se precisa de conocimientos previos para la correcta utilización y configuración de los equipos, como es el caso de los autómatas programables, los controladores industriales tipo PID, Predictivo, etc… o las FPGA's. Estos equipos están destinados a un público "elitista" por decirlo de algún modo, por lo que se reduce en cierta medida el acceso a la mayoría de usuarios.

Para paliar este hecho, encontramos equipos ya diseñados para tareas concretas, como por ejemplo sistemas de alarma y seguridad, tarjetas de adquisición de datos o controladores de invernaderos por poner algunas ejemplos.

El usuario no puede modificar la funcionabilidad de dichos equipos más allá de lo permitido por el fabricante, siendo necesario adquirir otros equipos para realizar otras tareas; lo que se traduce en un mayor desembolso económico y una peor gestión de los recursos.

A partir de esta situación se decide realizar este proyecto es diseñar e implementar un módulo hardware basado en la plataforma de software y hardware libre Arduino así como el desarrollo de una aplicación para Smartphone de 5º generación con sistema operativo Symbia destinada a controlar y configurar dicho módulo.

El sistema, al que le hemos apodado el sobrenombre de *GNX Project v.1*, consistirá en un módulo configurable vía Bluetooth a las necesidades del usuario por medio de la aplicación para Smartphone anteriormente citada. Mediante una interface de usuario agradable e intuitiva, el usuario podrá "diseñar un producto" acorde a sus necesidades en menos de 1 minuto, pudiendo utilizarlo para desarrollar tareas diferentes de manera simultánea y con la posibilidad de reutilización.

Dicho sistema constará de una placa con varios pines de entradas/salidas a los que se podrá conectar todos aquellos periféricos que desee el usuario, entendiéndose como tales **sensores** (resistivos (LDR, galgas, etc…), pulsadores, interruptores, ultrasonidos, infrarrojos y sensores de temperatura) y **actuadores**, tales como relés, servomotores, motores de corriente continua, pudiendo así mismo variar el ciclo de trabajo o controlar la frecuencia de trabajo de altavoces.

Dicha placa constará de los siguientes elementos incorporados:

- **Módulo transceptor Bluetooth**: El cual permite la comunicación entre la placa y el Smartphone a efectos de controlar y monitorizar el estado de accesorios incorporados a este.

- **Módulo GPS**: Mediante el cual podremos registrar las coordenadas geoestacionarias así como obtener los datos de fecha y hora que serán utilizados en las operaciones de registro de los datos del GPS así como del estado y configuración de cada unos de los pines de la placa a diseñar, tal y como se expondrá en capítulos posteriores.

- **Sistema de almacenamiento**: Sistema de almacenamiento extraíble no volátil destinados a almacenar los registros de posición y datos de interés para el usuario para posterior procesamiento en el PC tal y como se ha expuesto en el punto anterior.

- **Placa controladora Arduino**: Plataforma de software y hardware libre gobernada por un microcontrolador Atmel. Este módulo será el que se conectará a la placa final de diseño, la cual es objeto de este proyecto. Dicho módulo será el que contenga el firmware del sistema. Al igual que la mayoría de componentes de la familia Arduino, incorpora un conector USB (en este caso USB tipo B hembra) por el que se realiza la descarga del firmware a la placa Arduino, sin embargo se destinará para futuras aplicaciones del sistema a fin de poder ser controlado mediante el PC.

- **Indicadores y switchs**: Destinados a proporcionar una interface básica de usuario permitiendo realizar funciones exclusivas como la posibilidad de montar/desmontar la tarjeta MicroSD.

A continuación se exponen los principales objetivos que persigue el proyecto a realizar:

- Diseño e implementación de una placa de circuito impreso sobre la que se conectarán los elementos anteriormente citados. Esta placa se alojará en una caja de prototipado estándar a la que se le efectuarán las modificaciones pertinentes a fin de otorgar una mayor resistencia al sistema.

- Desarrollo de una estructura que permite la configuración y ejecución de todas aquellas acciones definidas por el usuario, pudiendo asociar a cada condición todas las acciones deseadas, así como órdenes inmediatas.

- Desarrollo del firmware que se cargará en la placa Arduino Mega 2560 el cual implementará entre otros, el algoritmo que permite la gestión y configuración del módulo tal y como hemos comentado en el punto anterior.

- Decisión del lenguaje de programación a emplear para el diseño de la aplicación para Smartphone así como la implementación de la misma buscando obtener una interface intuitiva que permita la comunicación entre el usuario y el módulo hardware mediante la tecnología Bluetooth.

1.2. Resumen de la memoria

La memoria se ha dividido en los siguientes capítulos a fin de ofrecer una visión más clara de todas las fases del proyecto:

- **Agradecimientos**

- **Capítulo 1**: Enunciado y objetivos del proyecto y resumen del contenido de la memoria.

- **Capítulo 2**: Base teórica, es decir, los conceptos sobre los que se basa el proyecto.

- **Capítulo 3**: Metodología de desarrollo y características principales de los elementos que componen el conjunto así como las justificaciones de su elección.

- **Capítulo 4**: Diseño y especificaciones del hardware y descripción detallada del funcionamiento interno del firmware del sistema *GNX Project v.1* así como de la aplicación para Smartphone.

- **Capítulo 5**: Descripción del proceso de implementación, abarcando desde el primer diseño del hardware pasando por el proceso de grabación del firmware en la plataforma Arduino para concluir con la compilación de la aplicación para Smartphone.

- **Capítulo 6**: Descripción y resultado de las pruebas realizadas.

- **Capítulo 7**: Conclusiones extraídas y posibles líneas de desarrollo a partir del estado final del proyecto.

- **Capítulo 8**: Esquema eléctrico, planos del PCB y listado de los componentes.

- **Apéndice A**: Código fuente del firmware del sistema *GNX Project v.1*.

- **Apéndice B**: Código fuente de la aplicación para Smartphone.

- **Apéndice C**: Manual de usuario donde se describirán las distintas secciones que las componen, así como uso del módulo hardware.

- **Bibliografía**

2. Base teórica

2.1. Introducción

En este capítulo vamos hablar de los conceptos teóricos sobre los que se sustenta el proyecto estructurándolo en los distintos elementos que lo componen, descritos en el capítulo anterior.

2.2. Comunicación Bluetooth

Los sistemas Bluetooth trabajan en un rango de frecuencias comprendido entre 2400 y 2483.5 MHz, el cual está dividido en diversas subbandas. Su tasa teórica es de 1Mbps. Al igual que en IP, los datos se transmiten en pequeños paquetes, y cada uno de ellos incluye una cabecera que, entre otros datos, indica la frecuencia de la banda en la que se enviará el siguiente paquete; de esta forma, los periféricos no transmiten siempre en una única frecuencia, sino que van saltando de una a otra en función del tráfico de la red y de otros factores.

Las redes Bluetooth están diseñadas para interconectar hasta ocho periféricos entre sí, en lo que se denomina una **red pico o piconet** (ver figura 2.1). Cada periférico se puede configurar como maestro o esclavo; los maestros son los encargados de dirigir el tráfico entre ellos mismos y los esclavos, e incluso entre un esclavo y otro. Además, cada maestro puede estar conectado a dos piconets distintas, y como puede haber varios maestros en una misma red, se pueden interconectar varias piconets entre sí de forma encadenada, hasta un máximo de 10. Esto nos da un máximo de 72 periféricos.

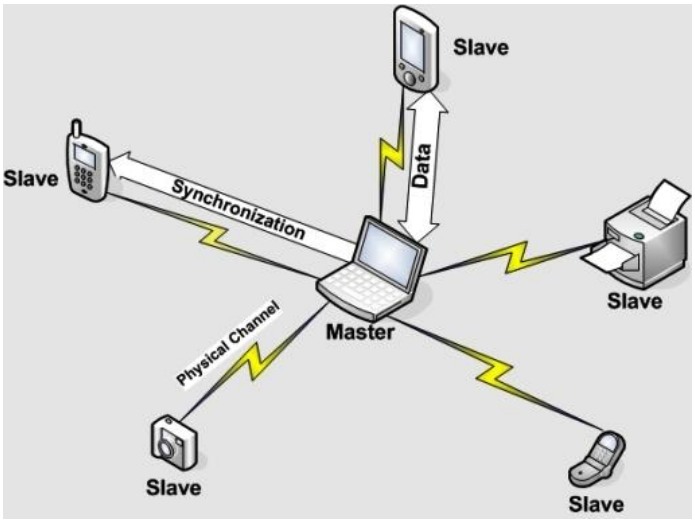


Figura 2.1: Representación gráfica de una red pic (Piconet).

Finalmente, hay 20 perfiles y tres tipos de enlaces distintos. Los perfiles controlan el comportamiento del periférico Bluetooth, y los enlaces asignan los modos de transmisión entre dispositivos. Por ejemplo, para que un PDA se pueda comunicar con un teléfono móvil es preciso que ambos tengan un perfil de modem y un enlace de voz/datos.

En caso contrario, incluso si ambos son dispositivos Bluetooth 1.1, serán incapaces de comunicarse entre ellos.

De los 20 perfiles validados por el (SIG), sólo 13, se usan hoy en día. Algunos son genéricos, como el que permite que los periféricos se reconozcan entre sí, y otros son para usos más específicos y pueden ser, por tanto, opcionales. Es el caso de un perfil de transmisión de sonido, el cual será indispensable para un teléfono móvil pero completamente inútil en un teclado.

La elección del enlace depende de la naturaleza del tráfico. Los enlaces síncronos ofrecen una conexión bidireccional a 432 Kbps en cada sentido, lo que los hace ideales para intercambio de datos entre equipos, tal y como se haría en una red local. Los enlaces asíncronos, por su parte, ofrecen 721 Kbps en un sentido y 57.6 Kbps en el opuesto, lo que los hace más adecuados para periféricos del tipo de una impresora, que recibe una gran cantidad de datos pero envía muy pocos. Finalmente está el enlace de voz/datos, que ofrece un canal bidireccional a 64Kbps pero con la característica de ser garantizados, lo que permite la fluidez necesaria para una transmisión de audio.

2.3. Principio de funcionamiento del GPS

Antes de la aparición de los actuales sistemas de posicionamiento vía satélite, se empleaba en la navegación marítima el principio matemático de triangulación, mediante el cual podemos conocer el punto o lugar donde nos encontramos situados. El sistema GPS utiliza el mismo principio, pero en lugar de emplear círculos o líneas rectas crea esferas virtuales para lograr el mismo objetivo.

Desde el mismo momento que el receptor GPS detecta una señal de radiofrecuencia transmitida por un satélite desde su órbita, se genera una esfera virtual que envuelve al satélite. El propio satélite actuará como centro de la esfera cuya superficie se extenderá hasta el punto o lugar donde se encuentre situada la antena del receptor; por tanto, el radio de la esfera será igual a la distancia que separa al satélite del receptor. A partir de ese instante el receptor GPS medirá las distancias que lo separan como mínimo de dos satélites más. Para ello tendrá que calcular el tiempo que demora cada señal en viajar desde los satélites hasta el punto donde éste se encuentra situado y realizar los correspondientes cálculos matemáticos. La intersección de las tres esferas nos da el punto deseado como podemos ver en la figura 2.2.

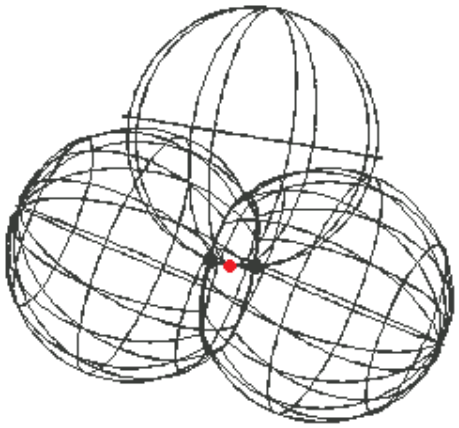


Figura 2.2: Intersección de las esferas imaginarias en un único punto.

Todas las señales de radiofrecuencias están formadas por ondas electromagnéticas que se desplazan por el espacio de forma concéntrica a partir de la antena transmisora, de forma similar a como lo hacen las ondas que se generan en la superficie del agua cuando tiramos una piedra. Debido a esa propiedad las señales de radio se pueden captar desde cualquier punto situado alrededor de una antena transmisora. Las ondas de radio viajan a la velocidad de la luz, es decir, 300 mil kilómetros por segundo medida en el vacío, por lo que es posible calcular la distancia existente entre un transmisor y un receptor si se conoce el tiempo que demora la señal en viajar desde un punto hasta el otro.

Para medir el momento a partir del cual el satélite emite la señal y el receptor GPS la recibe, es necesario que tanto el reloj del satélite como el del receptor estén perfectamente sincronizados. El satélite utiliza un reloj atómico de cesio, extremadamente exacto, pero el receptor GPS posee uno normal de cuarzo, no tan preciso. Para sincronizar con exactitud el reloj del receptor GPS, el satélite emite cada cierto tiempo una señal digital o patrón de control junto con la señal de radiofrecuencia. Esa señal de control llega siempre al receptor GPS con más retraso que la señal normal de radiofrecuencia. El retraso

entre ambas señales será igual al tiempo que demora la señal de radiofrecuencia en viajar del satélite al receptor GPS.

La distancia existente entre cada satélite y el receptor GPS la calcula el propio receptor realizando diferentes operaciones matemáticas. Para hacer este cálculo el receptor GPS multiplica el tiempo de retraso de la señal de control por el valor de la velocidad de la luz. Si la señal ha viajado en línea recta, sin que la haya afectado ninguna interferencia por el camino, el resultado matemático será la distancia exacta que separa al receptor del satélite.

Las ondas de radio que recorren la Tierra lógicamente no viajan por el vacío sino que se desplazan a través de la masa gaseosa que compone la atmósfera; por tanto, su velocidad no será exactamente igual a la de la luz, sino un poco más lenta. Existen también otros factores que pueden influir también algo en el desplazamiento de la señal, como son las condiciones atmosféricas locales, el ángulo existente entre el satélite y el receptor GPS, etc. Para corregir los efectos de todas esas variables, el receptor se sirve de complejos modelos matemáticos que guarda en su memoria. Los resultados de los cálculos los complementa después con la información adicional que recibe también del satélite, lo que permite mostrar la posición con mayor exactitud.

De manera resumida, los pasos que sigue el módulo GPS son:

- Cuando el receptor detecta el primer satélite se genera una esfera virtual o imaginaria, cuyo centro es el propio satélite. El radio de la esfera, es decir, la distancia que existe desde su centro hasta la superficie, será la misma que separa al satélite del receptor. Éste último asume entonces que se encuentra situado en un punto cualquiera de la superficie de la esfera, que aún no puede precisar.

- Al calcular la distancia hasta un segundo satélite, se genera otra esfera virtual. La esfera anteriormente creada se superpone a esta otra y se crea un anillo imaginario que pasa por los dos puntos donde se interceptan ambas esferas. En ese instante ya el receptor reconoce que sólo se puede encontrar situado en uno de ellos.

- El receptor calcula la distancia a un tercer satélite y se genera una tercera esfera virtual. Esa esfera se corta con un extremo del anillo anteriormente creado en un punto en el espacio y con el otro extremo en la superficie de la Tierra. El receptor discrimina como ubicación el punto situado en el espacio utilizando sus recursos matemáticos de posicionamiento y toma como posición correcta el punto situado en la Tierra.

- Una vez que el receptor ejecuta los tres pasos anteriores ya puede mostrar en su pantalla los valores correspondientes a las coordenadas de su posición, es decir, la latitud y la longitud.

- Para detectar también la altura a la que se encuentra situado el receptor GPS sobre el nivel del mar, tendrá que medir adicionalmente la distancia que lo separa de un cuarto satélite y generar otra esfera virtual que permitirá determinar esa medición.

2.4. Sistema de almacenamiento SD

Las tarjetas SD están basadas en las memorias flash tipo NAND. La memoria flash NAND se está convirtiendo en el medio de almacenamiento elegido para aplicaciones de almacenamiento de estado sólido, debido a su gran velocidad de acceso y borrado así como su bajo coste. La naturaleza secuencial de las memorias flash basadas en NAND proporciona notables ventajas para las aplicaciones de almacenajes de datos orientadas a bloques.

Todas las tarjetas de memoria SD y SDIO necesitan soportar el antiguo modo SPI/MMC que soporta la interfaz de serie de cuatro cables ligeramente más lenta (reloj, entrada serial, salida serial y selección de chip) que es compatible con los puertos SPI en muchos microcontroladores. Muchas cámaras digitales, reproductores de audio digital y otros dispositivos portátiles, probablemente utilicen exclusivamente el modo MMC.

El modo MMC no proporciona acceso a las características propietarias de cifrado de las tarjetas SD y la documentación libre de SD no describe dichas características. La información del cifrado es utilizada primordialmente por los productores de medios y no es muy utilizada por los consumidores quienes típicamente utilizan tarjetas SD para almacenar datos no protegidos.

Las tarjetas de baja velocidad soportan tasas de transferencia de 0 a 400 Kbps y modo de trasferencia un-bit SD, mientras que las tarjetas de alta velocidad soportan tasas de transferencia de 0 a 100 Mbps en el modo de cuatro-bit, y de 0 a 25 Mbps en el modo un-bit SD.

Existen tres modos de transferencia soportados por SD:

- **Modo un-bit SD**: separa comandos, canales de datos y un formato propietario de transferencia.
- **Modo cuatro-bit SD**: utiliza terminales extra más algunos terminales reasignados para soportar transferencias paralelas de cuatro bits.
- **Modo SPI**: entrada separada serial y salida serial.

Este último modo es que utilizaremos para comunicar la tarjeta SD con Arduino utilizando el protocolo de comunicación SPI. El **Bus SPI** es un estándar de comunicaciones usado principalmente para la **transferencia de información entre circuitos integrados en equipos electrónicos**. Sirve para controlar casi cualquier dispositivo electrónico digital que acepte un flujo de bits serie regulado por un reloj.

Este protocolo funciona con una **configuración Master-Slave (maestro-esclavo)**, donde en nuestro caso el **Arduino será el maestro**, y la **tarjeta SD será el esclavo**. Ver figura 2.3.

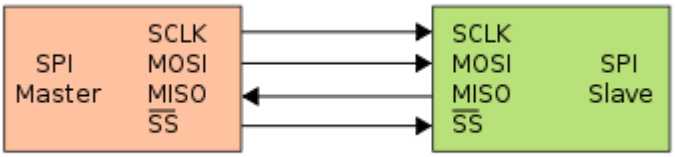


Figura 2.3: Interconexión para comunicación con la SD.

- **CLK (Línea de reloj):** Es la señal de reloj.
- **MOSI (MasterOut-SlaveIn):** Línea por donde el maestro envía y el esclavo recibe.
- **MISO (MasterIn-SlaveOut):** Línea por donde el maestro recibe y el esclavo envía.
- **CS (Chip Select)/ SS (Slave Selector):** Conecta o desconecta la operación del dispositivo con el que comunicamos. Permite la comunicación de varios esclavos a un mismo maestro, multiplexando la señal de reloj.

El maestro configura la velocidad de transmisión a la máxima permitida por el esclavo, y seguidamente pone a nivel bajo la señal "Selector Slave" para indicarle que va a comunicar.

Durante cada ciclo de reloj se produce una comunicación bidireccional, ya que por una parte el maestro va a enviar bits en serie (MOSI) y el esclavo a recibir, mientras que a la vez el esclavo va a enviar bits en serie (MISO) para que el maestro lo reciba. Al terminar la comunicación el maestro coloca a nivel alto la señal SS y para la señal de reloj.

2.5. Sistema operativo Symbian

Este proyecto precisa como se ha mencionada en el capítulo 1 de un sistema de control que posibilite la comunicación con el sistema *GNX Project v.1*, para lo cual será necesario elaborar una aplicación, en este caso para Smartphone. El dispositivo que disponemos para ello es un Nokia 5230 Xpressmusic, cuyo sistema operativo es Symbian. A fin de conocer con más detalle este sistema operativo, vamos a hablar brevemente del mismo:

Symbian es una compañía de software propiedad de Ericsson, Matsushita, Motorola, Nokia y Psion, creada con el propósito de desarrollar una plataforma estándar para teléfonos móviles inteligentes.

El objetivo de Symbian es crear dicha plataforma para los nuevos dispositivos inalámbricos que representan la próxima revolución en comunicación

Las principales características del sistema operativo Symbian son:

- Telefonía móvil "multi-modo" integrada. Symbian OS integra la potencia de computación con la telefonía móvil, aportando servicios avanzados de datos al mercado masivo.

- Entorno abierto de aplicación. Permite a los teléfonos móviles ser una plataforma de aplicaciones y servicios (programas y contenido) que puede ser desarrollada en una amplia gama de lenguajes y formatos.

- Estándares libres e interoperabilidad. Con una implementación flexible y modular, Symbian OS proporciona un sistema esencial de APIs (Application Programming Interface) y de tecnologías compatibles para todos los teléfonos Symbian.

- "Multi-Tarea". Está basado en una arquitectura de "micro-kernel" e implementa funciones de tipo "multi-tarea". Los servicios del sistema como la telefonía, las redes 'middleware' y las aplicaciones, funcionan en sus propios procesos.

- Orientado a objeto y Basado en componente. El sistema operativo está diseñado y pensado exclusivamente para los dispositivos móviles, usando técnicas avanzadas de OO (orientación a objetos), para construir una arquitectura flexible basada en componentes.

- Diseño flexible de la interfaz de usuario. Permitiendo un diseño gráfico flexible de la interfaz de usuario en el sistema y usando el mismo SO como base en diferentes diseños, Symbian OS facilita el proceso de desarrollo de la aplicación.

- Robustez. Symbian OS mantiene el acceso inmediato a los datos del usuario. Asegura la integridad de los datos, incluso en presencia de comunicación no fiable y de falta de recursos como son la memoria, el almacenamiento y la energía.

3. Metodología, análisis de alternativas y materiales elegidos.

3.1. Introducción

Vamos a describir las fases detalladas de desarrollo del proyecto, partiendo desde las especificaciones de requisitos, pasando por el desarrollo del hardware y software para concluir con el diseño, fabricación y posterior comprobación del prototipo.

Así mismo se plantearán las posibilidades de elección de determinados elementos de manera justificada que más se ajusten a los objetivos del proyecto.

3.2. Metodología

En el diseño de sistemas se suelen seguir una serie de pasos que van desde la especificación de requisitos hasta la comercialización del producto final. El presente proyecto tiene como objetivo final la construcción de un prototipo, por lo que el ciclo de diseño no abarcará hasta la fase final de fabricación en serie del producto y posterior comercialización. Este ciclo de diseño se ha adaptado a las características del proyecto y constará de las fases de especificación de requisitos, establecimiento de la arquitectura del sistema, desarrollo del hardware y software, test y debug y, por último, diseño y fabricación del PCB. A continuación se detallan las fases:

- **Especificación de requisitos**. En esta fase se establecen los requisitos que se quiere que cumpla el producto a desarrollar. Así pues, se tendrán que especificar parámetros tales como el coste máximo, el rendimiento mínimo, las ampliaciones futuras, el consumo de energía, tamaño o peso.

- **Establecimiento de la arquitectura del sistema.** Siendo primordial la elección de el modelo de la plataforma Arduino que se ajuste mejor a nuestras necesidades en términos de memoria, velocidad, número de pines (y cuántos de estos pueden funcionar como digitales, analógicos o PWM, es decir, que permiten modulación de la anchura de pulso)

- **Desarrollo del hardware y software**. Una vez se tiene claro lo que se quiere hacer, es el momento de empezar a desarrollarlo y para ello, en primer lugar, se hace una selección de los componentes a utilizar en la placa, realizando también el esquema de la circuitería. Hay que tener en cuenta que en el ciclo de diseño hardware se pueden dar situaciones de diseño-prototipo-testeo-vuelta atrás, con el consiguiente encarecimiento del producto final. Por esta razón, se deja la fase de diseño y fabricación del PCB para el final, desarrollando antes una primera placa de prueba que servirá para verificar en la siguiente fase de testeo tanto el correcto diseño de la circuitería como el firmware del sistema Arduino y la aplicación software para Smartphone.

- **Test y debug**. Una vez acabado el desarrollo del hardware, firmware y software se procede a comprobar su correcto funcionamiento. Para comprobar el sistema, se realizaran "pruebas de sobrecarga", es decir, se someterá al módulo a la programación y gestión de múltiples tareas de manera simultánea, tales como el control de los periféricos, o recopilación de los datos en la tarjeta de memoria MicroSD, por poner algún ejemplo; a efectos de garantizar un correcto

funcionamiento siendo en caso contrario necesario realizar las pertinentes modificaciones.

- **Diseño y fabricación del PCB**. Por último, una vez se ha comprobado el correcto funcionamiento de la placa de prueba, se realiza el diseño del circuito impreso, que previa verificación de todas las conexiones y habiendo incluido como parte del PCB todas aquellas indicaciones necesarias tales como la numeración de los pines. Una vez comprobados, se procederá a su fabricación y posterior montaje.

- **Testeo y experimentación.** Con el prototipo listo se realizarán experimentaciones para validarlo y comprobar que todo funciona acorde a lo establecido.

3.3. Plataformas software de desarrollo

Para el desarrollo del proyecto se han empleado varios paquetes de software, tanto para el diseño del esquemático y PCB como para la creación del firmware del microcontrolador Atmel y la aplicación destinada al control del módulo para Smartphones.

Dichos software son los que se exponen a continuación: Orcad 9.2, Altium Designer Summer 09, IDE Arduino 0.22, Notepad ++. A continuación se va a proceder a describir cada una de estas herramientas indicando su utilidad y principales características:

Orcad 9.2

El paquete Orcad, entre otras muchas funcionalidades, permite el diseño de PCBs multicapa de forma sencilla y rápida. Consta del **Orcad Capture**, con el que se realizan los esquemas del circuito, y del **Orcad Layout**, con el que se realiza el diseño del PCB. El **Orcad Capture** permite la gestión de proyectos de forma jerárquica y la división de grandes diseños en otros más pequeños. Por medio del Orcad Capture se realiza el esquema del circuito incluyendo sus componentes. Aunque posee gran cantidad de librerías de componentes, permite la creación de librerías propias y el diseño de componentes específicos.

Una vez realizado el esquema del diseño se utilizará el **Orcad Layout** para realizar el diseño del PCB. Para mayor facilidad permite la posibilidad de realizar un ruteado automático de las pistas del diseño así como diferentes opciones de ruteado para minimizar los costes de implementación.

Altium Designer Summer 09

Se trata de una aplicación de características muy similares al paquete de herramientas Orcad, que permite crear componentes, crear esquemáticos e implementarlos en tarjetas de circuito impreso (PCB's). Sin embargo, proporciona una mayor libertad que Orcad a la hora de realizar todo tipo de modificaciones del proyecto, pudiendo mover, redimensionar o incluso sustituir elementos por otros de una forma más rápida e intuitivas, debiendo destacar su impecable interfaz gráfica muy útil a la hora de crear nuevos componentes, donde su asistente te solicita los datos para crearlo de forma automática.

Por ello, esta herramienta se utilizó en la medida de corregir "imperfecciones" durante la fase de verificación previa a la fabricación del PCB, de lo contrario nos obligaría a tener que rehacer en determinadas situaciones todo el diseño con el considerable coste de tiempo que ello conlleva.

IDE Arduino 0.22

Arduino es una iniciativa surgida a mediados del año 2005 por un grupo de ingenieros italianos como una herramienta didáctica con la que se pretendía iniciar a un público con pocos conocimientos de programación y electrónica al mundo del prototipado "homebrew".

Consiste básicamente en una placa de circuito impreso de reducidas dimensiones gobernada por un microcontrolador marca Atmel el cual lleva implementado en 2KB de su

memoria un firmware de código abierto (al igual que su hardware) conocido como Bootloader el cual permite que la propia placa se pueda hacer de programador a fin de reprogramarse a sí misma, a demás de permitir interpretar las instrucciones que componen los sketch o programas desarrollados.

Esta plataforma dispone de una interface de desarrollo conocida como IDE, la cual puede ser descargada desde la web oficial del equipo Arduino, y que permite grabar nuestros proyectos directamente en la placa mediante conexión USB o FTDI.

Notepad ++

Notepad++ es un editor de texto y de código fuente libre con soporte para varios lenguajes de programación tales como C, Java o Basic entre muchos otros, el cual sólo está disponible para sistemas operativos Windows. Se distribuye bajo los términos de la Licencia Pública General de GNU.

Se parece al Bloc de notas en cuanto al hecho de que puede editar texto sin formato y de forma simple. No obstante, incluye opciones más avanzadas que pueden ser útiles para usuarios avanzados como desarrolladores y programadores.

Esta herramienta juega un papel decisivo a la hora de desarrollar la aplicación para Smartphone debido a ser multilenguajes, proporcionar una distinción de colores y llaves lo cual es muy útil a la hora realizar bucles anidados.

Todas estas aplicaciones has sido ejecutada en un PC Packard Bell TJ66 con sistema operativo Windows 7 de x64.

3.4. Análisis de alternativas

A continuación se describirán las especificaciones que debe de cumplir el proyecto para posteriormente y en base a estas, analizar las distintas alternativas a la hora de elección los distintos elementos que compondrá el sistema *GNX Project v.1* tanto a nivel de hardware (Modelo de Arduino a emplear, tipo de GPS, etc…) como en lo referente al software (aplicación para Smartphone) seleccionando las opciones que se ajustan mejor al presente proyecto.

3.4.1. Requisitos

Los requisitos a nivel de usuario que debe de cumplir el proyecto se dividen en tres grupos: Requisitos del periférico, requisitos de APP y requisitos generales del proyecto.

El módulo hardware deberá tener unas dimensiones lo más reducidas posibles a fin de poder ubicarlo con facilidad en todas aquellos lugares donde sea preceptiva su instalación. Así mismo deberá tener la posibilidad de ser alimentado de forma independiente dándole así una mayor versatilidad y capacidad de adaptación. Estas formas de alimentación serán:

- Mediante una batería de 9 a 12 voltios removible, con posibilidad de ser desconectada por medio de un interruptor general alojado en la parte frontal de la carcasa.

- Mediante conexión a la red por medio de un transformador externo que proporcione una tensión entre 5 y 20 voltios (siendo esto el rango que soportan todos los modelos de Arduino a excepción de la versión Mini y Micro). Se hace también necesario la posibilidad de cortar la alimentación por medio del interruptor antes mencionado así como la desconexión de la batería en el momento de conectar el Jack de alimentación, lo cual es fácil una clavija de tres terminales.

El módulo deberá de disponer de un conector USB que permita la programación y posibilite el futuro desarrollo de una aplicación para su control y gestión por medio de cualquier equipo que actúe a modo de host, como un PC o una tablet. Por medio del USB se deberá poder alimentar la placa no pudiendo ser desconectada por conmutación del principal a fin de evitar la interrupción del proceso de grabación del firmware del sistema GNX Project v.1, para lo cual sería necesario disponer de indicadores luminosos que nos advirtieran de este hecho.

Así mismo deberá disponer de un número considerable de pines de entrada/salida disponibles para el usuario con posibilidad de ser configurados a sus necesidades; como incorporar un pequeño panel de control en la parte frontal del dispositivo que permita realizar tareas como reiniciar o indicar el estado del sistema.

El proyecto precisa de un sistema de control accesible a la mayoría de usuarios. Por ello se opta por que esté diseñado en una aplicación ejecutable para Smartphone. Esta deberá ser intuitiva y proporcionar una interface clara pero con toda la información que precise el usuario para obtener un control total del módulo sin precisar de conocimientos

previos. De todas formas, se ha elaborado un manual de usuario ubicado en el Anexo C de esta memoria desarrollando todas las funcionalidades que dispone.

La aplicación deberá permitir la programación/configuración de los sensores y actuadores así como constar con un menú de "Favoritos" donde el usuario pueda almacenar aquellas acciones que emplee con mayor regularidad.

En previsión de posibles futuras mejoras dar soporte tanto a nivel de hardware como de software lo que se traduce en un ahorro económico, al no precisar rehacer todo el diseño sino en todo caso pequeños módulos independientes, como podría ser el caso de una tarjeta Ethernet por ejemplo.

3.4.2. Evaluación de alternativas

Una vez establecidas estas premisas se procederá a analizar las distintas alternativas que se nos plantean así como la justificación de las mismas.

1) Modelo de la plataforma Arduino

Antes de empezar a describir las opciones, vamos a proceder a justificar los motivos que nos han conducido desde primera instancia a la elección de la iniciativa Arduino como elemento clave de este proyecto frente a otras opciones como los tradicionales microcontroladores PIC de Microchip o sistemas más "potentes" como es el caso de las FPGA's.

Si los comparamos frente a las FPGA's, estas disponen de una estructura de programación menos intuitiva que la que proporciona Arduino, a demás existe un amplio abanico de librerías destinadas a realizar tareas concretas como por ejemplo el control de servomotores sin la necesidad de conocer el funcionamiento de las mismas. Aunque es cierto que las FPGA's proporcionan una mayor versatilidad en términos de potencia, capacidad y velocidad; su coste y dimensiones es significativamente mayor al de un modelo Arduino, siendo para este proyecto un hecho determinante.

Otro de los aspectos principales de elección de esta plataforma frente a los microcontroladores PIC reside en el importante hecho de que la propia placa Arduino permite reprogramar el microcontrolador Atmel que integra. Esto es posible, como hemos comentado en el punto 3.3 de este capítulo gracias al Bootloader o gestor de arranque alojado en la memoria flash del microcontrolador el cual puede ser editado al estar bajo licencia GNU y es propiamente el kernel de Arduino.

Hay diferentes versiones del gestor de arranque para trabajar sobre dispositivos diferentes o porque han sufrido actualizaciones a lo largo del tiempo. Los gestores de arranque actuales son prácticamente idénticos para la Diecimila y la NG (con ATmega168). Ambos corren a 19200 baudios y ocupan unos 2 KBytes de memoria flash en elATmega168. La única diferencia está en el tiempo que el gestor de arranque espera para que un nuevo programa llegue y el numero de destellos que emite el led del pin 13 cuando arranca. Porque en

el reset automático de la versión Diecimila este gestor de arranque emplea muy poco tiempo de espera, menos de un segundo, y para ahorrar tiempo el led del pin 13 solo destella una vez.

Los comandos "Burn Bootloader" del entorno Arduino utilizan una herramienta open-source, concretamente el programa **Avrdude**, el cual se trata de un programa escrito inicialmente para Linux, que facilita la programación de microcontroladores Atmel AVR en este sistema operativo. No posee interfaz gráfica, aunque la comunidad de software libre y otras personas han puesto a disposición herramientas GUI que facilitan el trabajo con este programa.

Hay cuatro pasos que realiza a la hora de grabar un sketch:

- Desbloquear la sección del gestor de arranque en el chip
- Fijar los fusibles en el chip
- Subir el código del gestor de arranque al chip
- Bloquear la sección del gestor de arranque en el chip.

Esto nos conduce a otro aspecto determinante en su elección, la **posibilidad de clonar esta plataforma** tanto a nivel de hardware, estando disponibles en la web oficial de Arduino (http://arduino.cc/en/Main/Hardware) los esquemáticos y archivos gerber de todos los modelos que se encuentran en el mercado; así como de software, recurriendo a grabadores externos del Bootloader como por ejemplo un programador paralelo (ver figura 3.1). El programador se puede conectar la los pines **ICSP** (ver imagen 3.2), también llamado **Programación Serial en Circuitos**, mediante el cual puede grabarse la memoria de programa, la EEPROM y registros de configuración directamente desde la placa sin necesidad de retirar el microcontrolador del circuito.

La comunicación ICSP requiere cinco señales:

- **ICSPDAT o PGD:** Datos de Programación; es una línea de datos bidireccional sincrónica serial.
- **ICSPCLK o PGC:** Reloj de Programación; es una línea unidireccional sincrónica serial de reloj que va desde el programador hasta el microcontrolador.
- **VPP:** Voltaje de Programación; cuando es aplicado, el microcontrolador entra en el modo Programación.
- **VDD:** Suministro de voltaje positivo.
- **VSS:** Negativo.
- Arduino implementa un pin adicional de **reset**, de ahí este sexto pin.

También debe asegurarse de estar correctamente seleccionada la opción correcta en el menú **Tools/Board**. Por último solo hay que lanzar el comando

apropiado desde el menú Tools/Burn Bootloader del Arduino IDE. El proceso de grabación del gestor de arranque puede tardar unos 15 segundos variando estos tiempos en los modelos Mega y Mega 2560 que llegan a casi el doble.

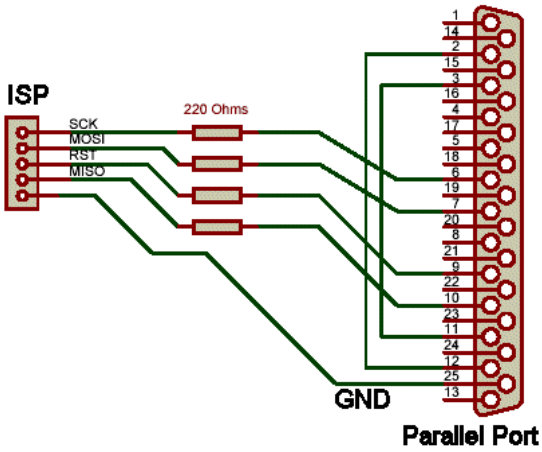


Figura 3.1: Esquema programador paralelo del Bootloader.

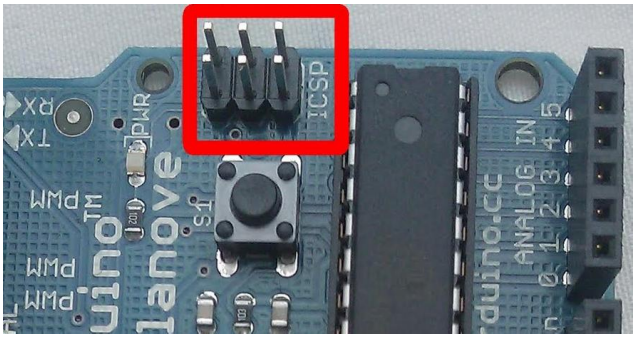


Figura 3.2: Ubicación de los pines ICSP en una placa Arduino.

Ahora, habiendo dejado claro los motivos de selección de esta plataforma, vamos a discutir cual de los principales modelos se ajustan más a nuestras exigencias. Para ellos debemos de fijarnos en aspectos como: Existencia o no de conector USB y tipo, microcontrolador removible o no, número de pines así como los distintos tipos de los mismos, memoria EEPROM, flash y RAM así como el precio. Todo esto queda resumido en la tabla 3.1.

	USB	Micro	Nº Pines	Memoria	Precio
Modelo					
UNO	Tipo B	ATmega328	14 Digital 6 PWM 6 Analog	Flash:31,5KByte Ram:2KByte EEPROM: 1KByte	18€
Mega 2560	Tipo B	Atmega2560	54 Digital 15 PWM 16 Analog	Flash:248KByte Ram:8KByte EEPROM: 4KByte	21€
Nano v3	Mini B	ATmega328	14 Digital 6 PWM 8 Analog	Flash:30KByte Ram:2KByte EEPROM: 1KByte	13€
Mini	No	ATmega328	14 Digital 6 PWM 8 Analog	Flash:30KByte Ram:2KByte EEPROM: 1Kbyte	12€
LiliyPad	No	ATmega168	14 Digital 6 PWM 6 Analog	Flash:14KByte Ram:1KByte EEPROM:512 Byte	20€

Tabla 3.1: Resumen de características de los modelos de Arduino.

Como vemos existe una pequeña variación de precios, siendo los de mayor coste el Arduino Mega 2560 y el Arduino LiliyPad. Teniendo en cuenta el número de pines, capacidades y el hecho de que no precisamos coser el módulo a ninguna prenda (debido a que esta versión es flexible y especialmente diseñada para incorporarse en prendas, como podemos ver en la figura 3.3) parece obvio descartar este último.

Como deseamos constar con un conector USB y precisamos de reducir el número de elementos a incorporar en la placa de circuito impreso final (refiriéndonos a un convertidos USB-Serie como el de la figura 5.11), se descarta la versión Mini.

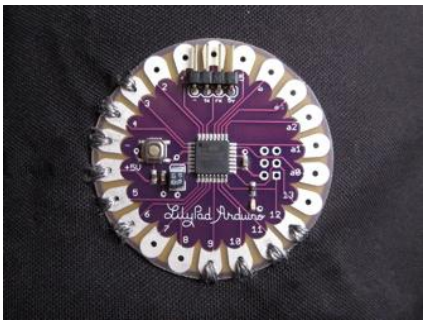


Figura 3.3: Arduino LiliyPad.

Ahora debemos de decidir entre los tres modelos resultantes: Arduino UNO, Arduino Mega 2560 y Arduino Nano v3.

Como podemos observar, tanto la versión UNO como la Nano v3 disponen de las mismas características, diferenciándose únicamente en sus dimensiones, tipo de conector USB y en la carencia de jack de alimentación en la versión Nano v3, dado que este está pensado para ser montado en protoboads. Atendiendo a esto, preferimos elegir la versión Nano v3, al ser menor su precio.

Como deseamos dar soporte al proyecto para futuras mejoras, se debe prever un importante consumo de recursos, tales como RAM y memoria flash así como la posibilidad de incorporar nuevos periféricos al sistema, llegamos a la conclusión que la mejor opción para este proyecto es recurrir al modelo Arduino Mega 2560 (Ver figura 3.4).

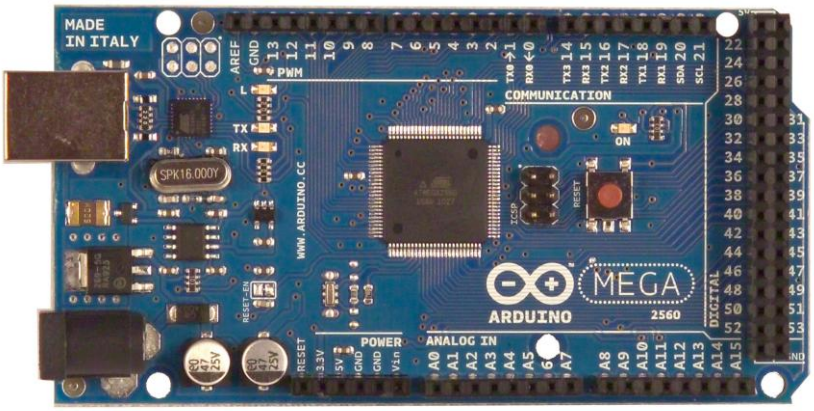


Figura 3.4: Arduino Mega 2560.

De una forma más detallada, las características de este modelo son:

- Voltaje de funcionamiento: 5V
- Voltaje de entrada: 7-12 V
- Corriente de salida de los pines: 40 mA
- Corriente de salida para el pin de 3.3V: 50Ma
- Velocidad del reloj: 16 MHz
- Número de puertos series: 4

2) Selección del lenguaje de programación para el desarrollo de la aplicación para Smartphone

Un aspecto importante a la hora de orienta el desarrollo del proyecto es decidir el lenguaje de programación a emplear para la creación de la aplicación que permitirá la comunicación con el módulo hardware. En nuestro caso, esta se va a instalar en un Nokia 5230 Xpressmusic con sistema operativo Symbian, al no disponer de otro dispositivo con sistema operativo Android por ejemplo.

Los propietarios de licencias Symbian desarrollan teléfonos y software de soporte para estos dispositivos con SDKs (kits de desarrollo software) para la comunidad ISV (vendedores de software independiente). El Symbian OS Customization Kit proporciona las herramientas requeridas por los poseedores de licencias para construir SDKs, que pueden ser introducidas al mercado ISV directamente o bien mediante compañías de herramientas.

Es variado el número de lenguajes soportados, por ello vamos a centrarnos en los tres más utilizados para el desarrollo de aplicaciones para esta plataforma: C++, Java y Python.

C++

C++ es un lenguaje de programación diseñado a mediados de los años 1980 por Bjarne Stroustrup. La intención de su creación fue el extender al exitoso lenguaje de programación C con mecanismos que permitan la manipulación de objetos. En ese sentido, desde el punto de vista de los lenguajes, el C++ es un lenguaje híbrido.

Posteriormente se añadieron facilidades de programación genérica, que se sumó a los otros dos paradigmas que ya estaban admitidos (programación estructurada y la programación orientada a objetos). Por esto se suele decir que el C++ es un lenguaje de programación multiparadigma.

Este lenguaje es uno de los más utilizados para el desarrollo de aplicaciones para todo tipo de plataformas, destacando por su robustez y relativa simplicidad de programación. Sin embargo a no ser un lenguaje interpretado como los siguientes que se expondrán, se precisa estudiar las librerías que se deberán incluir para su ejecución en cada una de las plataformas donde se quiera ejecutar así como la necesidad de compilar la aplicación cada vez que se desea probarla lo que significa una mayor inversión de tiempo.

Java

Java es un lenguaje de programación orientado a objetos, desarrollado por Sun Microsystems en 1995. El lenguaje en sí mismo toma mucha de su sintaxis de C, Cobol y Visual Basic, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria. La memoria es gestionada mediante un recolector de basura.

Las aplicaciones Java están típicamente compiladas en un bytecode, aunque la compilación en código máquina nativo también es posible. En el tiempo de ejecución, el bytecode es normalmente interpretado o compilado a código nativo para la ejecución, aunque la ejecución directa por hardware del bytecode por un procesador Java también es posible.

La implementación original y de referencia del compilador, la máquina virtual y las bibliotecas de clases de Java fueron desarrolladas por Sun Microsystems en 1995. Desde entonces, Sun ha controlado las especificaciones, el desarrollo y evolución del lenguaje a través del Java Community Process, si bien otros han desarrollado también implementaciones alternativas de estas tecnologías de Sun, algunas incluso bajo licencias de software libre.

Entre diciembre de 2006 y mayo de 2007, Sun Microsystems liberó la mayor parte de sus tecnologías Java bajo la licencia GNU GPL, de acuerdo con las especificaciones del Java Community Process, de tal forma que prácticamente todo el Java de Sun es ahora software libre aunque la biblioteca de clases de páginas web comprendidas en las librerías para ser compilados como aplicaciones comprimidas no están totalmente acopladas de acuerdo con Sun que dice que se requiere un intérprete para ejecutar los programas de Java.

La liberación del código fuente junto con el requerimiento de emplear un intérprete ha potenciado su divulgación y desarrollo de infinidad de aplicaciones para todo tipo de plataformas y sistemas operativos, como es el caso de Symbian.

Pese a proporcionar una gran potencia de desarrollo y existir abundante documentación por la red, se requiere invertir un considerable tiempo en estudiar este lenguaje así como el manejo de las herramientas de desarrollo precisamente orientadas a Smartphone, siendo este el principal motivo de emplear este lenguaje.

Python

Python es un lenguaje de programación de alto nivel cuya filosofía hace hincapié en una sintaxis muy limpia y que favorezca un código legible.

Se trata de un lenguaje de programación multiparadigma ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico, es fuertemente tipado y multiplataforma.

Es administrado por la Python Software Foundation. Posee una licencia de código abierto, denominada Python Software Foundation License, que es compatible con la Licencia pública general de GNU a partir de la versión 2.1.1, e incompatible en ciertas versiones anteriores.

Este lenguaje al igual que Java, precisa de ser ejecutado bajo una máquina virtual que debe estar presente en el equipo. Esto potencia sus posibilidades de expansión hacia infinidad de plataformas al no requerir

modificar el código del programa desarrollado en una plataforma para que se pueda ejecutar en otra.

Frente a los anteriores lenguajes mencionados, debemos destacar que la versión Python S60 proporciona un recurso de desarrollo rápido de interface gráfica de usuario (GUI) lo cual nos permite reducir el tiempo de creación de la aplicación al no precisar diseñar gráficos bastándonos con los que nos proporcionan las librerías implementadas.

Así mismo su lenguaje simple y estructurado permite su estudio y comprensión en un periodo corto de tiempo, gracias a los tutoriales existentes por la red destacando el hecho de que no precisa de ninguna herramienta de desarrollo más allá de un blog de notas así como al implementación en el propio intérprete para Symbian (**PythonScriptShell**) funciones orientadas especialmente para Smartphones como el envío de SMS o uso de la cámara.

Por todo ello hemos llegado a la conclusión de que la mejor opción para el desarrollo de este proyecto es emplear

3) Sistema de almacenamiento extraíble

El sistema *GNX Project v.1* pretende ser destinado a realizar operaciones de adquisición de datos procedentes de los pines de entrada/salida que incorpora pudiendo registrar por ejemplo la variación de temperatura en una sala o las veces que una puerta ha permanecido abierta o cerrada. Así mismo también deseamos almacenar otros datos como las coordenadas satélites de lo cual hablaremos más adelante.

A fin de recuperar esos datos, una exigencia de este sistema de almacenamiento es que los datos permanezcan al ser desconectado de la alimentación, es decir, que no sea volátil.

Una buena opción sería emplear un Pendrive de poca capacidad (dado que solo se van a almacenar archivos de texto) para realizar esta función. Sin embargo, para poder emplearlo se precisa de un suplemento de hardware específicamente diseñado para esta plataforma, se trata del **Arduino Host Shield** de Sparkfun, que como podemos observar en la figura 3.5, consta básicamente de un MAX3421E conectado mediante el bus SPI y al conector USB hembra proporciona la posibilidad de funcionar como host, pudiendo conectar ratones, teclados y pendrives. Sin embargo, esto supone un mayor desembolso económico y una mayor complejidad a la hora de realizar el diseño.

Figura 3.5: Arduino Host Shield.

Otra buena opción reside en emplear tarjetas de almacenamiento tipo SD (o cualquiera de sus subtipos), las cuales pueden conseguirse a un precio muy asequible y están presentes cada vez más en multitud de aparatos. A demás no precisan de circuitería externa para su funcionamiento (salvo aquella que permita adaptar los niveles lógicos de 0 a 5 V que proporciona Arduino a los 0 a 3.3V que precisa) y existen multitud de librerías para su manejo, como es el caso de "SD.H", convirtiéndolas en la mejor elección.

4) Sistema de geoposicionamiento vía GPS

La forma más fácil y eficiente de obtener las coordenadas es a través de un módulo GPS, los cuales se pueden obtener por menos de 25€ en determinadas webs. Entre los modelos considerados nos hemos decidido por los de la familia GlobaltSat debido a haber realizado pequeños proyectos de aeromodelismo empleando sus productos con anterioridad, obteniendo magníficos resultados. Concretamente, hemos decidido emplear el modelo EM-411 (ver figura 3.6) que pertenece a la gama intermedia de sus productos.

Figura 3.6: GPS EM-411.

Entre sus características debemos destacar las siguientes:

- Chipset: SiRF StarⅢ
- Fecuencia: 1575.42 MHz
- C/A code: 1.023 MHz
- Canales: 20
- Sensibilidad: -159 dBm
- Readquisición: Cada 0.1 segundo promedio
- Readquisición en caliente : Cada 1 segundo promedio
- Readquisición en frio: Cada 42 segundo promedio
- Altitud: Máximo 18,000 metros
- Velocidad: Máximo 515 ms /seg
- Tensiones de entrada: 4.5V-6.5V DC
- Consumo: 60mA
- Niveles lógicos: Nivel TTL, entre 0V-2.85V.
- Baudio: 4800 bps
- Dimensiones: 30mm x 30mm x 10.5mm
- Temperatura de trabajo: -40℃ a +85℃

5) Utilización de la tecnología Bluetooth

Para completar este capítulo debemos hablar de los aspectos que nos han hecho decantarnos desde primera instancia por esta tecnología.

Debemos destacar su bajo coste (menos de 3€) junto al hecho de estar presente en prácticamente cualquier sistema, tales como PCs, teléfono, tablet, etc…, lo que dota de un mayor abanico de posibilidades de desarrollo de aplicaciones para el control del sistema *GNX Project v.1*. Este modelo es JY-MCU.

Pese a su reducido tamaño, como observamos en la figura 3.7, proporciona un enlace de unos 5 metros en abierto, lo cual es más que suficiente para este proyecto. A demás su rango de velocidades varían entre los 1200 hasta los 115200 proporcionando una conexión estable durante el flujo de información. Su consumo es muy reducido, de unos 20-30mA cuando está buscando dispositivo al que vincularse, y de 8mA cuando se encuentra establecido el enlace.

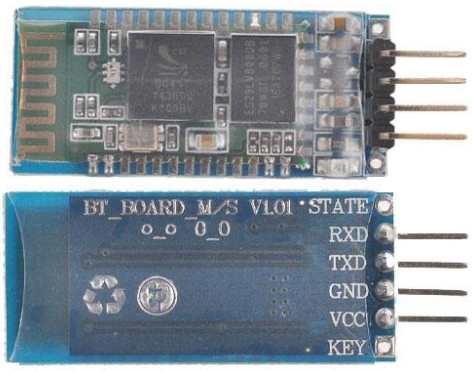


Figura 3.7: Módulo transceptor Bluetooth.

4. Diseño

4.1. Introducción

El presente capítulo se expone la explicación detallada de las características del circuito impreso diseñado e implementado así como un análisis en profundidad de los algoritmos que componente el firmware diseñado para la placa Arduino y la estructura de gestión/configuración sobre la que se sustenta todo el proyecto.

4.2. Diseño del hardware

Ya mencionamos en el capítulo 3 que elementos principales compondrían el hardware del sistema *GNX Project v.1*. Sin embargo, se nos plantea un problema referente al lector de tarjetas SD. Este reside en el elevado coste de los módulos comercializados por la empresa Sparkfun los cuales rondan los 20€ incluyendo los gastos de envío, junto con el hecho de que la mayoría de estos están diseñados para el Arduino UNO, al ser considerado el estándar de esta plataforma.

Por ello hemos decidido construir un lector de tarjetas SD de forma casera mediante el uso de una placa de topos y de un adaptador de tarjetas MicroSD a SD (de ahí que empleemos una MicroSD en lugar de una SD). Debemos observar el esquema dispuesto en la figura 2.3 del capítulo 2 donde para la comunicación Maestro-Esclavo, necesitamos cuatro líneas de conexión para establecer la comunicación SPI. Para ello hemos resumido en la figura 4.1 los pines que conforman el adaptador SD indicando los pines del Arduino.

Figura 4.1: Distribución de pines SD.

Hay un aspecto importante a resaltar antes de realizar las conexiones. Existen multitud de librerías implementadas en la IDE de Arduino que permiten utilizar los sistemas de almacenamiento SD. Las más importantes son: **SDFat32.h, SDFat16.h y SD.h**.

El sistema *GNX Project v.1* cuenta con la posibilidad ser configurada para guardar en la tarjeta de memoria datos procedentes de los sensores o incluso del GPS. Para desarrollar dichas funcionalidades que en el apartado 4.3 de este capítulo se describirán con más detalle, se ha empleado la librería SD.h, al ser la única que soporta las tarjetas MicroSD HD (en nuestro caso hemos empleado una de 2GB marca Verbatim). Todas las

librerías anteriormente citadas precisan de unos pines concretos para acceder a la tarjeta. Estos son:

- CS/SS (Pin 4)
- MOSI (Pin 11)
- MISO (Pin 12)
- CLK (Pin 13)

Sin embargo en los modelos Mega y Mega 2560 de Arduino, estos pines cambian de la siguiente forma:

- CS/SS (Pin 53)
- MOSI (Pin 51)
- MISO (Pin 50)
- CLK (Pin 52)

La librería SD nos proporciona, por un lado seleccionar el pin CS/SS lo que permite tener varios sistemas de almacenamiento compartiendo los mismo pines MOSI, MISO, CLK y por otro lado, a fin de simplificar los algoritmos del sistema de control, gestión y monitorización, podemos cambiar estos pines 50,51,52 por los pines 11,12,13 de una forma simple.

Para ello deberemos hacer una modificación en la librería SD.h, concretamente al archivo "**Sd2Card.h**" que encontramos en la carpeta "**utility**" de la librería. Abriendo este archivo con cualquier editor de texto, en nuestro caso el Notepad++ se tiene que **localizar la línea 42,** y cambiar donde pone: #define MEGA_SOFT_SPI 0 **sustituir este cero del final por un uno** quedando como vemos en la figura 4.2, así los pines que utilizará para acceder a la SD serán los deseados.

```
38  * MEGA_SOFT_SPI allows an unmodified Adafruit GPS Shield to be used
39  * on Mega Arduinos.  Software SPI works well with GPS Shield V1.1
40  * but many SD cards will fail with GPS Shield V1.0.
41  */
42  #define MEGA_SOFT_SPI 1
43  //------------------------------------------------------------------------------
44  #if MEGA_SOFT_SPI && (defined(__AVR_ATmega1280__)||defin
45  #define SOFTWARE_SPI
46  #endif  // MEGA_SOFT_SPI
47  //------------------------------------------------------------------------------
48  // SPI pin definitions
49  //
50  #ifndef SOFTWARE_SPI
51  // hardware pin defs
52  /**
53  * SD Chip Select pin
```

Figura 4.2: Modificación de la librería SD para cambiar los pines SPI.

Una vez realizado este cambio, ya estamos en las condiciones de realizar el montaje del lector de tarjetas SD. Para ello debemos tener en cuenta que los pines 8 y 9 de la figura 4.1 no precisan ser conectados, ya que únicamente son necesarios si empleamos el protocolo BUS SD que emplea cuatro bits en paralelo para comunicarse.

Otro aspecto de diseño importante es el hecho de que las tarjetas SD funcionan a un voltaje comprendido entre los 2.7 y 3.3V, por lo que no podemos conectar la tarjeta directamente al Arduino ya que sus pines digitales funcionan a 5 V y quemaríamos la SD.

Considerando esto existen varias opciones; por un lado emplear un divisor resistivo o utilizar un 74HC4050 que regula el voltaje a 3.3V. Como nuestro objetivo es la simplicidad y menor coste, hemos optado por realizar el montaje mediante tres divisores resistivos. Para obtener el valor de las resistencias no debemos hacer más que aplicar la ley de ohm considerando un voltaje de 3.2V en lugar de 3.3V teniendo en cuenta el efecto de las tolerancias:

$$V_{out} = \frac{R_2}{R_1 + R_2} \cdot V_{in} \Rightarrow 3.2 = \frac{R_2}{R_1 + R_2} \cdot 5 \Rightarrow$$

$$R_2 = 3.3K\Omega; R_1 = 1.8K\Omega;$$

A continuación solo debemos realizar el siguiente montaje (ver imagen 4.3):

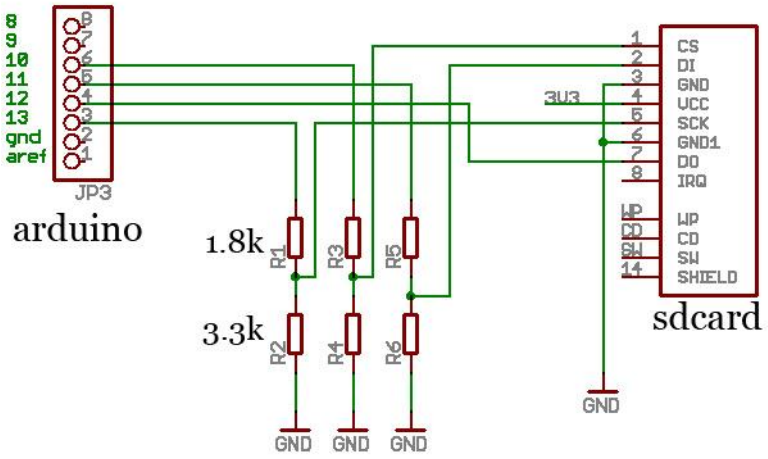


Figura 4.3: Esquema de montaje del lector de tarjetas SD.

Para montarla debemos desmontar el adaptador a fin de poder soldar trozos de alambre de unos 5cm de largo a cada una de ellas sin riesgo a derretir el plástico de la cubierta. Una vez soldados, realizamos el montaje dispuesto en la figura 4.3 e insertamos el adaptador con los alambres en la tarjeta de topos ya soldada (ver figura 4.4 A) tal y como se observa en la figura 4.4 B, siendo ahora el momento de finalizar todas las conexiones eléctricas. Ahora con la ayuda de un alicates de punta plana debemos envolver el trozo de placa (ver figura 4.4 C) procurando que no sufran los pines ya que se romperán con facilidad al aplicarle un poco de presión. Ahora lo único que debemos hacer es fijar el adaptador mediante pegamento de contacto y dejarlo pegar durante 24 horas.

(A) **(B)** **(C)**

Figura 4.4: Proceso de construcción del lector de tarjetas MicroSD.

A objeto de poder realizar un primer diseño del esquema eléctrico, una vez estando en posesión de todos los elementos descritos en este capítulo, procedemos a realizar su interconexión en una placa de prototipado, donde incluiremos el GPS, el lector de tarjetas MicroSD, el Arduino Mega 2560, los indicadores led, los pulsadores y el altavoz.

Dada la densidad de conexiones requeridas por el proyecto (debido sobre todo a los 50 conectores de alimentación, masa y datos alojados en la periferia de la placa de circuito impreso) como podemos observar en el esquemático del capítulo 9, únicamente hemos realizados en la protoboard las conexiones a los elementos anteriormente citados sin incluir los jumpers, el pulsador de reset ni los 50 conectores.

El GPS adquirido se trata como ya hemos comentado de un EM 411 el cual tiene el mismo conector que cualquiera de los de la familia EM. Se trata de un conector vertical SMD hembra, debido a que el módulo nos lo proporcionan con un cable doble macho de 5 cm de longitud, como el que podemos observar en la figura 4.5.

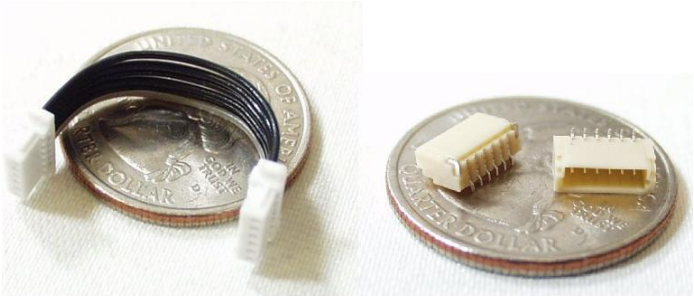


Figura 4.5: Cable y conector del módulo GPS.

Al no tener la posibilidad de conseguir dicho conector y no pudiendo asumir la demora que llevaría obtenerlo por otros medios, se decidió realizar un corte en el cable que nos proporcionaba el fabricante soldándose una extensión a la cual se le colocaría a modo de conector definitivo pines doble macho torneados con un espaciado entre ellos de 2.54 mm. Este cambio de conector repercutió en una mayor facilidad a la hora de diseño del esquema eléctrico y montaje en la placa de prototipado al disponer este de los mismos

milímetros de separación. Esto se puede observar en la figura 4.7. Para realizar el conexionado del GPS se ha seguido el esquema de la figura 9.7 del capítulo 9 conectando el RX del GPS (pin 4) con el TX2 (Pin 17) del Arduino Mega 2560 y el TX del GPS con el RX2 (Pin 16).

En lo referente al conexionado del módulo transceptor Bluetooth únicamente indicar que el conexionado debe realizarse cruzando los pines de flujo de datos RX, TX, es decir, RX con TX3 y TX con RX3. Estos pines vienen indicados en la parte trasera del módulo como podemos apreciar en la figura 3.7.

En previsión de futura ampliaciones, se ha decidido incorporar un conector adicional de cinco pines que se conectan a los terminales del puerto Serial1, TX1 (Pin 18) y RX1 (Pin 19) además de los pines de alimentación (5V) y masa y un pin adicional conectado al pin 49. Todo ello en previsión de futuros módulo que precisen de algunos de estos pines.

Pese a lo comentado en el capítulo 3 en lo referente a la elección del modelo de Arduino a emplear para la elaboración de este proyecto; en un principio se optó por utilizar una versión inferior, que combinaba las características del Arduino UNO, con el reducido tamaño de la versión Mini Pro, concretamente el Arduino Nano v3 que podemos observar en la figura 4.6.

Figura 4.6: Arduino Nano v3.

Con este modelo se desarrolló gran parte del proyecto, sin embargo atendiendo al potencial que el sistema *GNX Project v.1* podría llegar a tener, se optó por añadir nuevas funcionalidades como por ejemplo la adquisición y almacenamiento de datos en la tarjeta de memoria MicroSD así como aumentar el número de pines a disposición del usuario, aumentando estos de 10 a 50. En un principio se pensó en mantener este modelo Nano, tratando de cumplir estas nuevas premisas marcadas mediante modificaciones en el diseño, empleando multiplexores 74HC4051 que permiten conmutar entre ocho señales analógicas o digitales mediante tres señales de control. Este aumento de pines precisa de un aumento de la memoria RAM y no ROM como se podría pensar, debido a que el algoritmo implementado está pensado para gestionar un número indefinido de pines, sin embargo consta de varios arrays cuya longitud varía de forma proporcional a estos, no siendo necesario ampliar la memoria EEPROM de 2KByte al ser más que suficiente para el almacenamiento de más de 1000 configuraciones, concretamente 1365 debido a la distribución de las posiciones de la EEPROM que sólo almacenan valores enteros entre 0 y 255. Pero como hemos comentado, lo hemos limitado a un valor de 150 para garantizar el perfecto funcionamiento de todo el sistema.

Como se puede observar, estas adiciones complican el esquemático, aumentan el coste del proyecto y las dimensiones del mismo. Por ello, se vio pertinente sustituir la placa controladora por la gama más altas de la familia Arduino, así pues se recurrió a la versión Arduino Mega 2560 gobernada con por un Atmel Mega 2560 del que procede su nombre. Este módulo, como se puede observar en la tabla 3.1 del capítulo 3, permite logar los nuevos objetivos impuestos, no siendo necesario realizar modificación del software más allá de la ampliación del listado de pines disponibles en la aplicación para Smartphone y ligera alteración del algoritmo de grabación y recuperación de instrucciones en la memoria EEPROM debido a que el máximo valor almacenado en las posiciones de la EEPROM es 255.

Como podemos observar en la figura 4.7, la densidad de conexiones es baja frente a las que se precisaría si decidiésemos a implementar las alternativas anteriormente mencionadas.

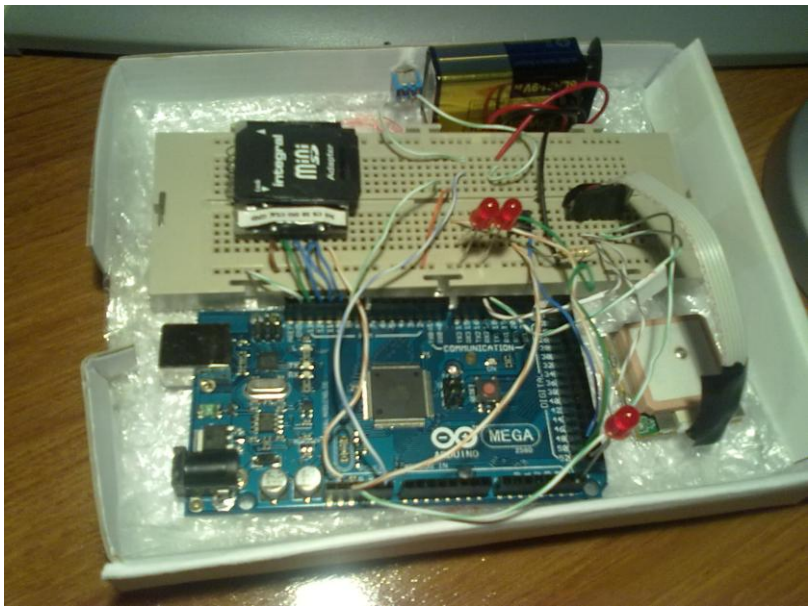


Figura 4.7: Primer montaje de todo el conjunto con el Arduino Mega 2560.

Durante la fase de verificación, que se comentará con más detalle en los posteriores capítulos, se observó el inconveniente de que la tarjeta de memoria MicroSD perdía en ocasiones su formato, es decir, que se corrompía su contenido cuando se desconectaba la alimentación o se extraía para ver los datos grabados en el monitor. Esto era debido a que la extracción coincidía con el momento en el que se transcurría el proceso de escritura, no ocurriendo como es lógico durante la lectura. Para solucionar esta contingencia, se decidió instalar junto con el led de estado del pin 51 (State) un segundo indicador luminoso conectado al pin 52 del Arduino (Check) que actuara de forma conjunta con el primero indicando mediante su parpadeo alternativo que se está realizando una operación de escritura a efectos de no remover la tarjeta durante su transcurso. A efectos de complementar esta medida, se decidió instalar un pulsador en la placa de prototipado conectado al pin 50 cuya función era la de poner un flag a 1 impidiendo la grabación en ella cuando se deseara desmontar la tarjeta según el esquema mostrado en la figura 4.8. Esto se reflejaba mediante el encendido del led Chek, siendo el mismo botón el encargado de montar/desmontar el cual puede ser ejecutado durante los espacios de tiempo entra grabaciones los cuales son configurables con un mínimo de 20 segundos lo que también

permite poder enviar órdenes desde el Smartphones durante ese tiempo, de lo contrario, al estar constantemente grabando las órdenes no llegarían a ser nunca ejecutadas.

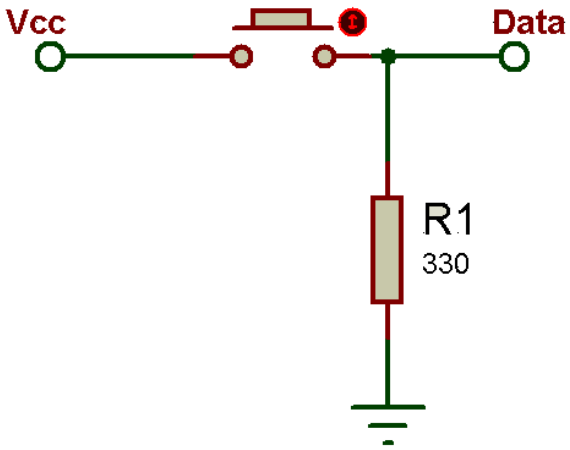


Figura 4.8: Esquema de conexión del pulsador realizado con Protheus.

Como lo que deseamos es obtener un módulo al que podamos conectar todos aquellos sensores/actuadores que veamos necesarios según la tarea a realizar y habiendo empleado el modelo Mega 2560, únicamente hemos reservado para fines propios del proyecto un total de 20 pines de los 70 disponibles. Los 50 restantes se encuentran a disposición del usuario, para lo cual se hace preciso realizar 50 bloques de tres conectores cada uno: Datos, Alimentación (5V) y Masa. Así podemos conectar directamente al módulo hardware servos, relés y sensores tales como ultrasonidos o infrarrojos.

Todos los modelos de Arduino disponen tanto de un botón de reinicio integrado en la placa como de un pin de reset activo a nivel bajo. El fin de todo este proyecto es elaborar un PCB el cual se conectará encima del Arduino Mega 2560, quedando el pulsador inaccesible, por ello se ha decidido incorporar en el panel de control que se colocará en el frontal de la caja donde se alojará la placa (ver figura 4.11) y todos sus elemento, un pulsador adicional donde uno de sus terminales lo conectamos a masa y el otro al pin de reset. Así mismo también precisamos de poder reiniciar la placa de forma remota, es decir, por medio del Smartphone. Esto no puede realizarse por únicamente por software dado que precisa de un componente adicional, un transistor. El que hemos empleado es un NPN, concretamente el BC547C, donde lo único que debemos hacer es añadir una resistencia de base de unos 330Ω y conectar este al correspondiente pin que vamos a destinar para realizar reinicios, en nuestro caso el pin 53; por último conectar el colector a masa y el emisor al pin de reset de Arduino como vemos en la figura 4.9.

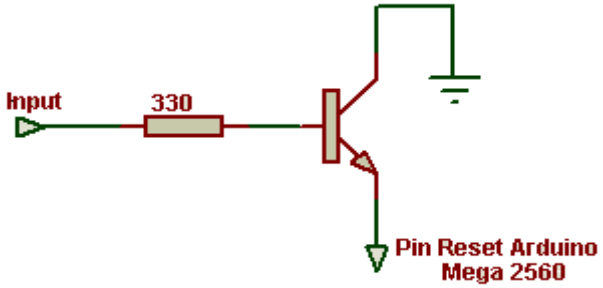


Figura 4.9: Esquema para reiniciar la placa por software.

Debido a que la placa debe tener total independencia, se ha decidido alimentarla mediante una pila de 9V, aunque también podría haberse empleado una de hasta 12V y debe ser posible cortar la alimentación mediante el interruptor principal alojado en el panel de control (ver figura 4.11) A sí mismo la placa dispondrá de un jack de alimentación a fin de poder alimentarlo por medio de una fuente externa, por lo que nos interesa que cuando conectemos el jack la pila se desconecte a fin de ahorrar carga, este montaje se resumen en la figura 4.10 y puede verse implementado en la figura 4.12. Igualmente se precisa que se pueda desconectar mediante el interruptor ya citado (el que hemos empleado es de 250V y 3 A). La placa también deberá poder alimentarse por USB siendo en este caso no posible desconectar la alimentación por medio del interruptor a fin de garantizar que no se interrumpa la descarga de datos a la placa durante el proceso de grabación.

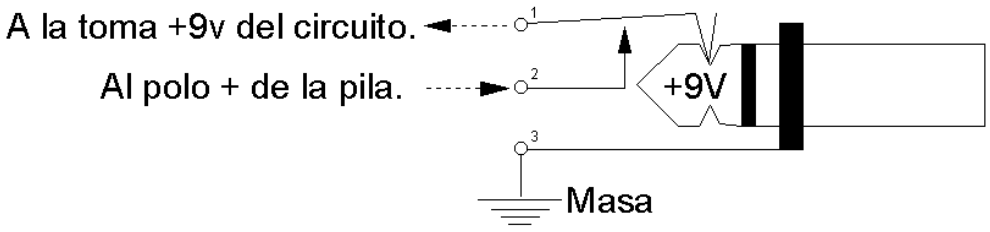


Figura 4.10: Jack de alimentación con desconexión de batería.

Se ha decidido incluir en el diseño un altavoz de 8Ω y 0.2W a fin de dotar de un elemento indicador más "contundente" por decirlo de algún modo. La forma de conexión es muy simple; uno de los dos pines se coloca a masa y el otro a uno de los pines de Arduino que permitan PWM (modulación de anchura de pulso) por medio de la función **Tone** que incorpora de serie Arduino, la cual toma por parámetros la el pin sobre el que actuar y el valor de la frecuencia a la que deseamos que suene. Simplemente lo hace es generar una onda cuadrada de la frecuencia especificada y un 50% de ciclo de trabajo sobre el pin especificado durante un tiempo infinito o hasta el valor especificado, debiendo mencionar que la placa sólo puede generar un tono cada vez, así si ya existe un pin donde esté sonando una frecuencia y deseamos otra, debemos parar de emitir mediante el comando **noTone**.

Para completar este diseño se han incluido además un par de pulsadores adicionales ubicados en la parte posterior de la caja donde se emplazará una vez construido el PCB (ver figura 4.12) destinados a ser utilizados por el usuario conectando el terminal de la placa con uno de los 50 pines disponibles y luego realizar la configuración por medio del Smartphone. Por último se han incorporado un par de jumpers que realizan las siguientes conexiones: Pin 0 con el pin 14 y el pin 1 con el pin 15. Con esto hemos cruzado los terminales del puerto **Serial 0** que es el que emplea la IDE de Arduino, con el puerto **Serial 3** que es donde se encuentra conectado el módulo Bluetooth. Gracias a ello logramos dos cosas al estar los jumpers conectados:

- Al ser el puerto **Serial 3** por el que se reciben las instrucciones del Smartphone, podemos mediante el Serial Monitor que incorpora la IDE de Arduino transmitir esas mismas órdenes así como visualizar los datos que transmita el sistema (realizando un cambio en la programación del firmware) por medio del USB.

- Por otro lado, también sirve para proteger el código ante modificación del mismo debido a que la IDE lanzará un error al encontrarse el puerto **Serial 0** conectado al módulo Bluetooth.

Una vez definidas todas las conexiones teniendo en cuenta las características y consideraciones expuestas en los capítulos 2 y 3 se procede a la creación de la placa de circuito impreso según los esquemas de las figuras 9.1 y 9.2, mediante las herramientas de software indicadas en el capítulo 3. Una vez construida la placa y realizada todas las soldaduras se procederá conectar todos los dispositivos y a emplazarla en una caja de prototipado al que le hemos practicado los orificios pertinentes para poder conectar los pulsadores, el interruptor de encendido y apagado, los led, el jack de alimentación y los pequeños orificios para el altavoz. Todo ellos colocados bajo una carátula con las indicaciones de cada uno de estos elementos como podemos a preciar en las figuras 9.8 y 4.11.

Figura 4.11: Frontal conjunto GNX Project v.1.

Figura 4.12: Parte trasera del conjunto GNX Project v.1.

4.3. Diseño del firmware

El sistema *GNX Project v.1*, precisa del desarrollo de un firmware que se instalará en el microcontrolador Atmel que incorpora el Arduino Mega 2560. Los pasos para instalarlo se describen en el capítulo 5.

A fin de comprender de una forma más profunda el funcionamiento interno del proyecto, se decide ir comentando cada uno de los bloques principales que lo conforman:

4.3.1. Detección y procesado de las instrucciones recibidas

Para lograr un control del sistema lo más versátil posible, se decidió idear un sistema de tramas de 15 caracteres que incorporan todos los datos que precisa el proyecto para configurar o ejecutar cada una de las acciones implementadas. Esta trama se resume en la tabla 4.1.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Contraseña				**E/S**	**Modo**	**Pin**		**Condición**	**Valor 1**			**Valor 2**		

Tabla 4.1: Descripción de las tramas de control.

Varios elementos de la trama se combinan para dar lugar a bloques concretos cuya finalidad puede cambiar del variar los bloques 4 y/o 5: De forma general se describe cada uno de estos 7 bloques resultantes:

- **Contraseña**: Consiste en cuatro caracteres que sirven para evitar el uso no autorizado del sistema. Si la contraseña no coincide con la del sistema esta trama no es ejecutada.

- **E/S**: Puede tomar 4 valores, (siendo uno de ellos para un fin reservado) permitiendo indicarle al sistema si el resto de los bloques hacen referencia a una entrada, salida o acción especial del usuario, como por ejemplo el registro de los datos del GPS.

- **Modo**: Puede tomar un valor de 0 a 9 y básicamente indica que es lo que hay conectado en un pin concreto (este se indica en el siguiente bloque) a objeto de que el sistema sepa cuál es el tratamiento que debe darle. Estos pueden ser desde un servo (si E/S=0) hasta un sensor de temperatura (si E/S=1).

- **Pin**: Formado por dos caracteres dado que los pines del proyecto van desde el 1 al 50. Este pin es el real, es decir, el de la placa Arduino Mega 2560 y no el del GNX Project v.1. Para conocer la correlación entre los pines ver la tabla 9.2.

- **Condición**: Empleado principalmente en el caso de sensores. Sirve para establecer la condición que se debe de cumplir para que se ejecuten las acciones asociadas a dicho sensor en función del valor de este bloque y de los dos siguientes (Valor 1 y Valor 2),es decir:

    - **Condición =0**→ Si Entrada = Valor1….
    - **Condición =1**→ Si Entrada > Valor1….
    - **Condición =2**→ Si Valor1< Entrada < Valor2….
    - **Condición =3**→ Poner la Entrada en la salida, esto lo que permite es, por ejemplo, hacer que se mueva el servo en correlación directa con el valor de un sensor resistivo como por ejemplo una LDR.

- **Valor 1**: Bloque de tres caracteres empleado tanto en las condiciones de los sensores como para indicar el valor a poner en la salida, es decir, cuanto deseamos que gire el servo o a que frecuencia debe sonar el altavoz.

- **Valor 2**: Se componen de tres caracteres al igual que el anterior. Este se emplea para dos fines concretos: Por un lado colaborar en las condiciones como ya se ha comentado y por otro lado sirve para establecer la temporización de desconexión, es decir, que por ejemplo un relé se desconecte pasado un tiempo desde el momento en el que dejó de cumplirse la condición que lo activó. Esto es útil en situaciones como el aviso de intrusos, donde si deseo que si se abre la puerta suene el altavoz, este deberá estar funcionando un tiempo adicional desde el momento en que la puerta vuelva a cerrarse, de lo contrario el usuario podría no percatarse de este hecho. Este es seleccionable por el usuario entre unos valores tabulados, siendo en este caso el valor “000” indicativo de que la desconexión se deberá realizar de manera automática al dejar de cumplirse la condición.

Las combinaciones de estos bloques quedan expuestas en el cuadro resumen de la tabla 4.2:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	Explicación
Contraseña				*E/S*	*Modo*	*Pin*		*Cond*	*Valor 1*			*Valor 2*			
				0 [Out]	0=Activación Digital				X	X	X	Temporización			Accionar Relés
					1=Variar Corriente				Valor (0-255)			X	X	X	Variar Corriente
					2=Servo				Ángulo(0-180°)			X	X	X	Servo
					3=Altavoz				Frecuencia(0-255)			Temporización			Altavoz incorporado o uno externo
					4=Motor				Velocidad (0-255)			Temporización			Motor de corriente continua
				1 [In]	0=Sensor Resistivo			0 (=) Entrada = Valor1				X	X	X	**Nota** En el pulsador Cond. indica el numero de pulsaciones deseadas. Valor1 indica el tipo de flanco Valor2 indica temporización
					1=Pulsadores A/D			1 (>) Entrada > Valor1				X	X	X	
					2=Interruptor			2 (<) Entrada < Valor1				X	X	X	
					3=Sensor de Temp.			3 (< x <) Valor1<Entrada < Valor2							
					4=Ultrasonido										
					5 = Infrarrojo			4 (->) Valor1 → Salida				X	X	X	
					6=Independiente										
				2 [User]	0	0	0	X	X	X	X	X	X	X	Reiniciar Placa
						0	1	Nueva Contraseña				X	X	X	Configuración
						0	2	T.GPS	X	X	X	X	X	X	
						0	3	T.SD	X	X	X	X	X	X	
						1	0	X	X	X	X	X	X	X	Borrar EEPROM
					1	X	X	X	X	X	X	X	X	X	Logger GPS
					2	X	X	X	X	X	X	X	X	X	Telemetría (El pin 0, indica que se registren todos los pines)
				3	**2**	**X**	**X**	**X**	**X**		**X**	**X**	**X**	**X**	Fin de acciones programadas

Tabla 4.2: Cuadro resumen de interpretación de tramas.

Ahora ya estamos en condiciones de poder explicar el funcionamiento de todos los algoritmos que componen el firmware. Continuando con la detección de las tramas, vamos a analizar el código de la figura 4.13:

```
int Detectar_Codigo(int n) {
        if(Serial3.available()){
         if(i<n){
           Codigo[i]=Serial3.read();
           i++;
           Serial3.flush(); //Vaciamo el buffer de entrada de datos serie
          }
        }
       if(i==n){//Hacemos la conversión a decimal
          for(i=0; i<n;i++){
           Codigo[i]=Codigo[i]-48;}
          i=0;
       }

       ES = Codigo[4];
       Modo = Codigo[5];
       Pin = 10*Codigo[6]+Codigo[7];
       Condicion = Codigo[8];
       Valor1 = 100*Codigo[9]+10*Codigo[10]+Codigo[11];
       Valor2 = 100*Codigo[12]+10*Codigo[13]+Codigo[14];

   return Codigo[0]*1000+Codigo[1]*100+Codigo[2]*10+Codigo[3];
}
```

Figura 4.13: Algoritmo encargado de detectar las tramas y descomponerlas.

Como hemos comentado anteriormente, el puerto **Serial 3** es el utilizado para el flujo de datos por medio del módulo Bluetooth. Es por este donde se reciben las tramas antes descritas desde la aplicación para Smartphone. Lo que hace esta función es precisamente esto, verificar en cada ciclo de reloj si hay datos disponibles en el buffer de entrada de este puerto. Si es así, la función **Serial3.available()** tomará el valor 1 y comenzará a ir almacenando el contenido de cada carácter en una posición del array "**Codigo[15]"**, cuyas dimensiones se definen por el número de caracteres que componen la trama, en este caso 15. Debido a que una vez leído un carácter este aun permanece en al buffer de recepción es preciso eliminarlo de lo contrario estaría siempre considerando el mismo carácter. Por ello se emplea la función **Serial3.flush()** que lo que haces es borrar el buffer de entrada. Esto lo hace a 16 MHz, por lo que cuando este es borrado aun no le han llegado el resto de caracteres y por lo tanto no se pierde información útil. Una vez almacenados, se realiza su conversión a decimal restándole 48, dado que el valor que nos proporciona los número que componen la trama corresponde con su equivalencia como caracteres en la tabla ASCII como podemos apreciar en la figura 4.14, donde el "0" es el 48 en ASCII, el "1" el 49 y así sucesivamente.

ASCII	Hex	Símbolo	ASCII	Hex	Símbolo	ASCII	Hex	Símbolo	ASCII	Hex	Símbolo
0	0	NUL	16	10	DLE	32	20	(espacio)	48	30	0
1	1	SOH	17	11	DC1	33	21	!	49	31	1
2	2	STX	18	12	DC2	34	22	"	50	32	2
3	3	ETX	19	13	DC3	35	23	#	51	33	3
4	4	EOT	20	14	DC4	36	24	$	52	34	4
5	5	ENQ	21	15	NAK	37	25	%	53	35	5
6	6	ACK	22	16	SYN	38	26	&	54	36	6
7	7	BEL	23	17	ETB	39	27	'	55	37	7
8	8	BS	24	18	CAN	40	28	(	56	38	8
9	9	TAB	25	19	EM	41	29	)	57	39	9
10	A	LF	26	1A	SUB	42	2A	*	58	3A	:
11	B	VT	27	1B	ESC	43	2B	+	59	3B	;
12	C	FF	28	1C	FS	44	2C	,	60	3C	<
13	D	CR	29	1D	GS	45	2D	-	61	3D	=
14	E	SO	30	1E	RS	46	2E	.	62	3E	>
15	F	SI	31	1F	US	47	2F	/	63	3F	?

ASCII	Hex	Símbolo	ASCII	Hex	Símbolo	ASCII	Hex	Símbolo	ASCII	Hex	Símbolo
64	40	@	80	50	P	96	60	`	112	70	p
65	41	A	81	51	Q	97	61	a	113	71	q
66	42	B	82	52	R	98	62	b	114	72	r
67	43	C	83	53	S	99	63	c	115	73	s
68	44	D	84	54	T	100	64	d	116	74	t
69	45	E	85	55	U	101	65	e	117	75	u
70	46	F	86	56	V	102	66	f	118	76	v
71	47	G	87	57	W	103	67	g	119	77	w
72	48	H	88	58	X	104	68	h	120	78	x
73	49	I	89	59	Y	105	69	i	121	79	y
74	4A	J	90	5A	Z	106	6A	j	122	7A	z
75	4B	K	91	5B	[	107	6B	k	123	7B	{
76	4C	L	92	5C	\	108	6C	l	124	7C	|
77	4D	M	93	5D	]	109	6D	m	125	7D	}
78	4E	N	94	5E	^	110	6E	n	126	7E	~
79	4F	O	95	5F	_	111	6F	o	127	7F	

Figura 4.14: Tabla ASCII

Una vez realizada la conversión, se deben de agrupar estos caracteres a fin de reconstruir los valores, dando lugar a seis nuevas variables que serán empleadas en los sucesivos algoritmos.

Para evitar problemas relacionados con tramas de longitud superior a 15 caracteres, el algoritmo elimina automáticamente este excedente para garantizar el correcto funcionamiento del dispositivo.

4.3.2. Almacenamiento, gestión y ejecución de las tramas.

El sistema puede ser configurado para realizar tareas concretas, pudiendo asociar a determinadas condiciones que deben de cumplir los sensores todas las acciones que se deseen e incluso establecer diferentes condiciones de ejecución para un mismo sensor.

Esto es posible gracias a la estructura de gestión y almacenamiento de las tramas, las cuales son almacenadas en la memoria EEPROM de Arduino, que en la versión Mega 2560 es de 8Kbyte, es decir 8192 posiciones de memoria para almacenamiento de los datos. Sin embargo, en cada posición únicamente es posible almacenar un entero comprendido entre 0 y 255, por lo que no es posible almacenar una trama de 15 dígitos en cada posición. Por ello el algoritmo de gestión, una vez recibido los datos de la trama, comprobada la contraseña (y siempre que se trate de la programación de una tarea) descompondrá cada uno de los bloques de la trama en seis valores decimales que sí se puedan almacenar en seis posiciones de la EEPROM, descartando el campo de la contraseña. Esto queda resumido en la tabla 4.3, donde apreciamos la forma con la que el algoritmo fracciona el código de una forma gráfica apurándolos según colores, para ello hemos considera la trama: **123402020100000** que permite mover el servo conectado en el pin 2 de la placa Arduino (Pin 1 de nuestro diseño) un ángulo de 100º. La contraseña del sistema es en este caso "1234" la cual es posible cambiar como se verá más adelante.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Contraseña				E/S	Modo	Pin		Condición	Valor 1			Valor 2		
1	2	3	4	0	2	0	2	0	1	0	0	0	0	0

Tabla 4.3: Descripción de las tramas de control.

Esta descomposición se realiza únicamente cuando se va grabar en la EEPROM, es decir, cuando configuramos un sensor, en cuyo la primera trama recibida debe ser la que describa el funcionamiento de dicho sensor, donde se indicará tipo, pin donde está conectado y condición a cumplir; para que a continuación se vallan recibiendo las tramas de todas las acciones asociadas a dicha condición concluyendo con la trama "Fin de acciones programadas" (Contraseña+ "32000000000") que indica al sistema que las siguientes instrucciones que reciba no deben ser guardadas en la EEPROM sino ejecutadas de manera inmediata.

La grabación se realiza a partir de la posición 0 la primera vez que configuramos algo y el resto en la posición libre justamente siguiente. En el caso de que la memoria se llenase, el sistema no sobrescribiría ninguna acción, sino que indicaría de dicha incidencia mediante el parpadeo de los dos led del panel de control de forma conjunta.

En caso de que se trate de una acción inmediata por parte del usuario realizado por ejemplo desde el menú "Favoritos" de la aplicación para Smartphone, esta no precisará ser almacenada sino que será directamente ejecutada mediante la llamada de la función **Acciones()** que se describirá más adelante. El algoritmo lo podemos observar en la figura 4.15.

Existe un tipo especial de "sensor" al que le hemos designado por el nombre de "**Sensor Independiente**", al cual se le asignan todas aquellas acciones que se deseen

ejecutar desde el momento que encendemos el módulo hardware. Esta opción está pensada principalmente para las tareas de Logger GPS o Logger Sensor, es decir, la adquisición de los datos de los elementos conectados a los pines, ya sea de forma individual o todos juntos en la tarjeta de memoria SD.

```
if(Detectar_Codigo(ncaracteres)==Password){
    if(ES==0 || ES==2){//Ejecutar una accion por demanda del usuario
      if(Introduciendo_Conf_Sensor==0){//No hay ninguna configuracion de sensores
         Acciones(ES, Modo, Pin, Condicion, Valor1, Valor2);
       }
      else{//Deseo programar las acciones del sensor
          Posicion_EEPROM();

           EEPROM.write(6*A+0,Codigo[4]*10+Codigo[5]);
           delay(50);
           EEPROM.write(6*A+1,Codigo[6]);
           delay(50);
           EEPROM.write(6*A+2,Codigo[7]*10+Codigo[8]);
           delay(50);
           EEPROM.write(6*A+3,Codigo[9]*10+Codigo[10]);
           delay(50);
           EEPROM.write(6*A+4,Codigo[11]*10+Codigo[12]);
           delay(50);
           EEPROM.write(6*A+5,Codigo[13]*10+Codigo[14]);
           delay(50);
      }
      delay(100);
    }
else{//Deseo configurar un sensor
 if(Inicio_Conf_Sensor==0){//Aun NO le he dicho que acciones asocio al sensor
   Posicion_EEPROM();

       EEPROM.write(6*A+0,Codigo[4]*10+Codigo[5]);
       delay(50);
       EEPROM.write(6*A+1,Codigo[6]);
       delay(50);
       EEPROM.write(6*A+2,Codigo[7]*10+Codigo[8]);
       delay(50);
       EEPROM.write(6*A+3,Codigo[9]*10+Codigo[10]);
       delay(50);
       EEPROM.write(6*A+4,Codigo[11]*10+Codigo[12]);
       delay(50);
       EEPROM.write(6*A+5,Codigo[13]*10+Codigo[14]);
       delay(50);


   Inicio_Conf_Sensor=1;
   Introduciendo_Conf_Sensor=1;
}
 else if(Codigo[4]==3 && Codigo[5]==2){
       Introduciendo_Conf_Sensor=0;
       Inicio_Conf_Sensor=0;

       delay(100);

  }
 }

  Borrar_Buffer_Codigo();
```

Figura 4.15: Algoritmo de gestión y almacenamiento de las tramas.

Una vez almacenado los datos, debemos de recuperarlos para poder realizar las tareas asignadas. Para ello, debemos conocer en primer lugar, cuantas acciones se encuentran configuradas, para lo cual se recurre a la función **Posicion()**. Esta va recuperando tramas, es decir, va leyendo seis posiciones de memoria cada vez. Una vez recuperada una trama comprueba si todas sus posiciones son igual a 0; de ser así indicará que ya no hay más instrucciones disponibles. Durante cada lectura se ha ido incrementado una variable que es la que nos da el número de instrucciones programadas. Esto es posible gracias a que la EEPROM parte con todas sus posiciones de memoria puestas a 0, lo cual se ha realizado con antes de cargar el firmware en la placa.

Ahora para ejecutar las acciones de forma correcta, debemos en primer lugar leer empezando por la posición 0 de la EEPROM las tramas hasta encontrar aquella que haga referencia a los sensores (incluido el Sensor Independiente), esta es almacenada en un array llamado "**Buffer_SRAM_Sensor[6]"** de seis posiciones, dado que ahí se guardarán los seis bloques que componen toda instrucción como hemos comentado. Ahora debemos de analizar esta trama almacenada y ejecutar las acciones asociadas cuando se cumpla la condición establecida.

Cuando se cumple la condición, se pone la variable **Ejecución a 1**, indicando que deseo que se ejecuten las acciones. Esto nos lleva a un bucle **while** que va leyendo las posiciones justamente siguientes a las del sensor y ejecutándolas, hasta encontrarse con otra trama que describa otro sensor (u otra condición del mismo) o llegue hasta el final de instrucciones grabadas. Tras ejecutarlas todas, ponemos el flag **Ejecución a 0** y volvemos a esperar a que se cumpla alguna condición.

En caso de que deje de cumplirse alguna condición en curso, se pondrá **Ejecución a 2**, indicando que desactive aquellas salidas que hayan quedado activas al cumplirse la condición antes definida. Tras esto se volverá a poner la variable **Ejecución a 0**.

Todo esto queda expuesto en la tabla 4.4 y el código fuente correspondiente se expone en la figura 4.16.

	0	1	2	3	4	5
	E/S	Modo	Pin	Cond	Valor 1	Valor 2
Bloque 1	1	0	0	0	900	0
	0	11	0	0	0	0
	2	0	0	0	2	0
	0	12	0	0	0	0
Bloque 2	1	2	1	1	28	0
	0	8	0	0	0	0
	0	10	0	0	0	0
	2	1	0	0	100	0
	0	0	0	0	0	0
	0	0	0	0	0	0

Tabla 4.4: Representación estructurada de las acciones programadas en la EEPROM.

```
if(Ejecucion==1){
  while(Read_SRAM(0,k)==0){
   if(Buffer_SRAM_Accion[1]==9 && Buffer_SRAM_Accion[2]==99 && Buffer_SRAM_Accion[3]==9 && Buffer_SRAM_Accion[4]==999 && Buffer_SRAM_Accion[5]==999){
        k++;
    }
    else{
    if(Buffer_SRAM_Accion[0]==2){///Si es una accion avanzada quiero que se ejecute mientras se siga cumpliendo la accion pero espacio x minutos


       Acciones(Buffer_SRAM_Accion[0], Buffer_SRAM_Accion[1], Buffer_SRAM_Accion[2], Buffer_SRAM_Accion[3], Buffer_SRAM_Accion[4], Buffer_SRAM_Accion[5]);

    }
    else if(Buffer_SRAM_Accion[0]==0 && Buffer_SRAM_Accion[1]==0 ){
        if(Once[k]==0){//Quiero que se ejecuta 1 sola vez
        Once[k]=1;
        if(digitalRead(Buffer_SRAM_Accion[2])==LOW)
          Acciones(Buffer_SRAM_Accion[0], Buffer_SRAM_Accion[1], Buffer_SRAM_Accion[2], Buffer_SRAM_Accion[3], Buffer_SRAM_Accion[4], Buffer_SRAM_Accion[5]);
       }
    }
   else if(Buffer_SRAM_Accion[0]==0 && (Buffer_SRAM_Accion[1]!=0)){//Si no es un rele
      int vMin,vMax=0;
      if(Buffer_SRAM_Sensor[3]==4){//Deseo que la salida=entrada
        if(Buffer_SRAM_Sensor[1]==0){//El sensor es un sensor resistivo
          vMin=0;
          vMax=1023;
        }
        else if(Buffer_SRAM_Sensor[1]==3){//El sensor es de Temperatura
          vMin=-55;
          vMax=105;
        }
        else if(Buffer_SRAM_Sensor[1]==4){//El sensor es un sensor Ultrasonido
          vMin=15;
          vMax=645;
        }
        else if(Buffer_SRAM_Sensor[1]==5){//El sensor es un sensor Infrarrojo
          vMin=5;
          vMax=150;
        }
        //Ahora ajustamos la salida segun el lo que tenemos conectado
          if(Buffer_SRAM_Accion[1]==1)//Modular pulso
            Buffer_SRAM_Accion[4]=map(Value_Sensor, vMin, vMax, 0, 255);
          else if(Buffer_SRAM_Accion[1]==2)//Es un servo
            Buffer_SRAM_Accion[4]=map(Value_Sensor, vMin, vMax, 0, 180);
          else if(Buffer_SRAM_Accion[1]==3)//Alarma
            Buffer_SRAM_Accion[4]=map(Value_Sensor, vMin, vMax, 20, 900);
          else if(Buffer_SRAM_Accion[1]==4)//Motor
            Buffer_SRAM_Accion[4]=map(Value_Sensor, vMin, vMax, 0, 255);


         if(millis()-Temoirizaciones[k] >=0.009*56850){
          Temoirizaciones[k]=millis();
          Acciones(Buffer_SRAM_Accion[0], Buffer_SRAM_Accion[1], Buffer_SRAM_Accion[2], Buffer_SRAM_Accion[3], Buffer_SRAM_Accion[4], Buffer_SRAM_Accion[5]);
         }
     }


     if(Once[k]==0 && Buffer_SRAM_Sensor[3]!=4){//Quiero que se ejecuta 1 sola vez
        Once[k]=1;
        Acciones(Buffer_SRAM_Accion[0], Buffer_SRAM_Accion[1], Buffer_SRAM_Accion[2], Buffer_SRAM_Accion[3], Buffer_SRAM_Accion[4], Buffer_SRAM_Accion[5]);
      }
    }

   k++;
  }
  }
  Ejecucion=0;
}
```

Figura 4.16: Algoritmo ejecución de las instrucciones asociadas a un sensor.

4.3.3. Proceso adquisición de los datos del GPS

El módulo GPS EM-411 transmite constantemente unas tramas por el puerto **Serial 2** de Arduino. Estas son enviadas aun cuando no hay datos disponibles o estos son erróneos. Los datos que recibiremos siguen el protocolo NMEA siglas de National Marine Electronics Asociation. Tienen esta estructura:

$GPRMC,044235.000,A,4322.0289,N,00824.5210,W,0.39,65.46,020911,,,A*44

Empiezan por "$" y acaban en un "A*" seguido de dos números, éste es el checksum para poder saber si el dato recibido es correcto. Los datos se separan por comas y el primero es el tipo de transmisión, en este caso el GPRMC (o RMC), uno de los más usados, que por ejemplo no incluye el valor de altitud, que si se incluye en el GPGGA. El fabricante nos proporciona una plantilla para poder interpretar dichos datos, donde si analizamos el ejemplo tenemos que:

- **044235.000** es el es la hora GMT (04:42:35).
- **A:** es la indicación de que el dato de posición está fijado y es correcto. V seria no válido.
- **4322.0289** es la longitud (43º 22.0289´).
- **N** Norte
- **00824.5210** es la latitud (8º 24.5210´).
- **W** Oeste.
- **0.39** velocidad en nudos.
- **65.46** orientación en grados.
- **050712** fecha (5 de julio del 2012).

La adquisición se realiza mediante la librería TinyGPS, la cual nos devuelve ya procesado cada uno de los parámetros del GPS. Su funcionamiento es sencillo; en primer lugar debe de almacenar los 64 dígitos que componen normalmente una trama en un array de al menos el doble de posiciones a para evitar problemas ocasionadas por algunas tramas.

Una vez almacenados, se comprueba el carácter de estado, el cual puede tomar el valor de “A” si la trama es correcta, es decir, que su checksum es correcto o “V” si no es válida, en cuyo caso se desecha dicha trama almacenada.

Si los datos almacenados son válidos, únicamente debemos concadenar todos aquellos caracteres comprendidos entre las comas (“,”) y luego convertirlos de string a tipo float. Los datos que almacenamos son: Fecha, hora, longitud, latitud, altitud, velocidad y curso.

El sistema *GNX Project v.1*, está constantemente actualizando los datos procedentes del satélite a fin de disponer siempre de una posición válida y de unos valores de fecha y hora que luego se utilizarán para hacer el registro en la MicroSD, tanto de las coordenadas como del estado de los pines conectados. El GPS dispone de un led que

informa con su parpadeo de que está recibiendo datos. Sin embargo al encontrarse la caja cubierta con su tapa no es posible ver este hecho. Por ello, cuando le programamos que funcione como Logger GPS, los dos led del panel frontal parpadean de forma alternativa cuando se está grabando dichos datos. Estas grabaciones se realizan con un espaciado de tiempo de unos 20 seg como mínimo configurable por el usuario. En cada sesión de encendido, es decir, cada vez que apagamos y encendemos el sistema *GNX Project v.1* crea un archivo distinto donde guardar los datos, a fin de poder realizar varios registros distintos. Estos archivos .txt se alojan en la carpeta "**GPS**" que se crea automáticamente en la raíz de la tarjeta con el nombre: GPS_00, GPS_01,….

Como podemos observar en la figura 4.17, los datos son precedidos de una cabecera que será empleada a la hora de pasar el archivo .txt en el que se encuentran alojados a un archivo .kmz que es el soportado por Google Earth a fin de poder ver la ruta realizada directamente sobre el mapa. Para ello, debemos recurrir a una web llamada **GPSVisualizer** (http://www.gpsvisualizer.com/map_input?form=googleearth) donde simplemente cargaremos los el archivo .txt con las coordenadas y configuraremos los parámetros visuales.

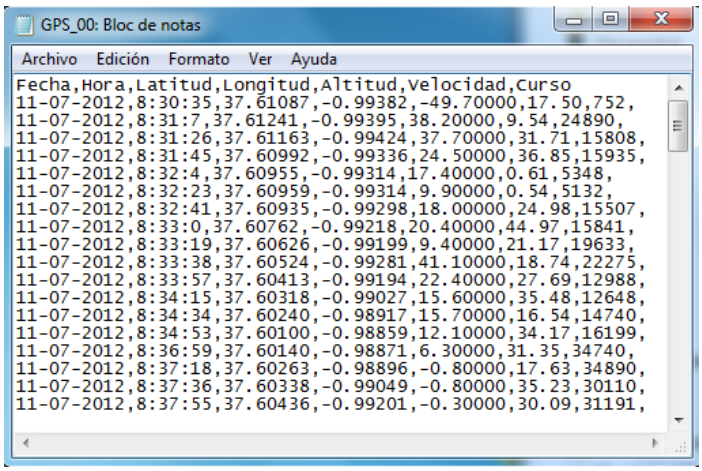


Figura 4.17: Ejemplo de registro de coordenadas GPS en la MicroSD.

En la figura 4.18 podemos apreciar la configuración empleada para la visualización de los datos, donde hemos empleado una plantilla (seleccionando en Draw as way point: Yes, using a custom template) que nos permite organizar estos datos a la hora de mostrarlo en cada uno de los globos como vemos en la figura 4.19. La plantilla empleada es:

"Fecha: {Fecha} Hora: {Hora} Latitud: {Latitud}º Longitud{Longitud}º Altitud{Altitud}m Velocidad:{Velocidad}Km/h Curso: {Curso}"

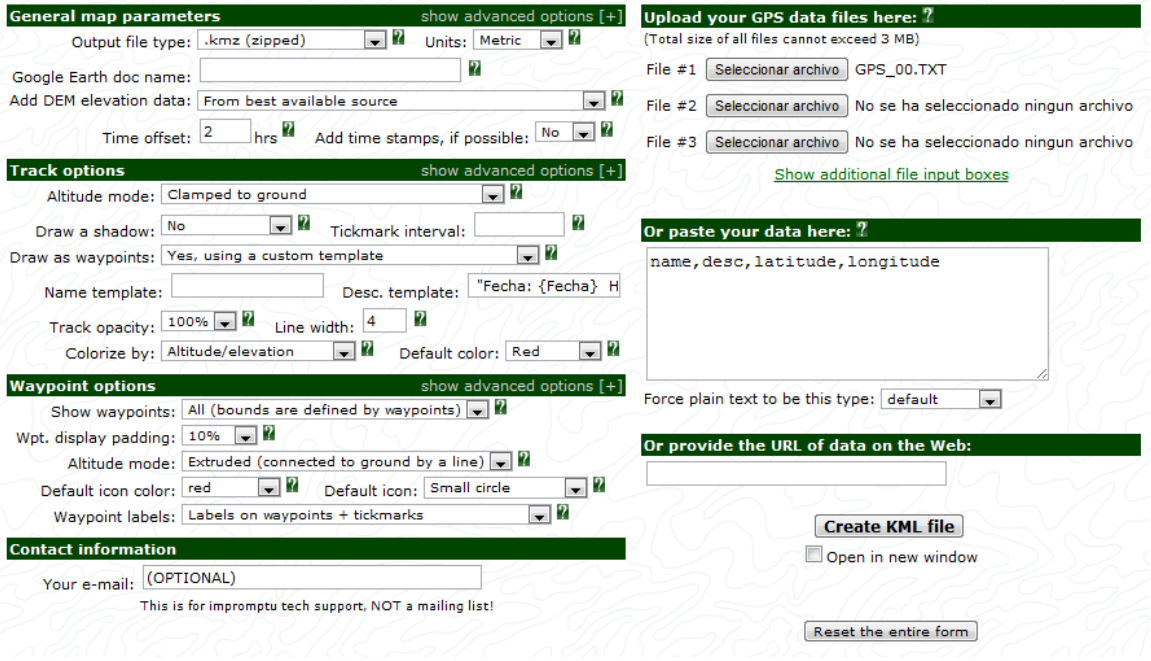


Figura 4.18: Parámetros de configuración de GPSVisualizer.

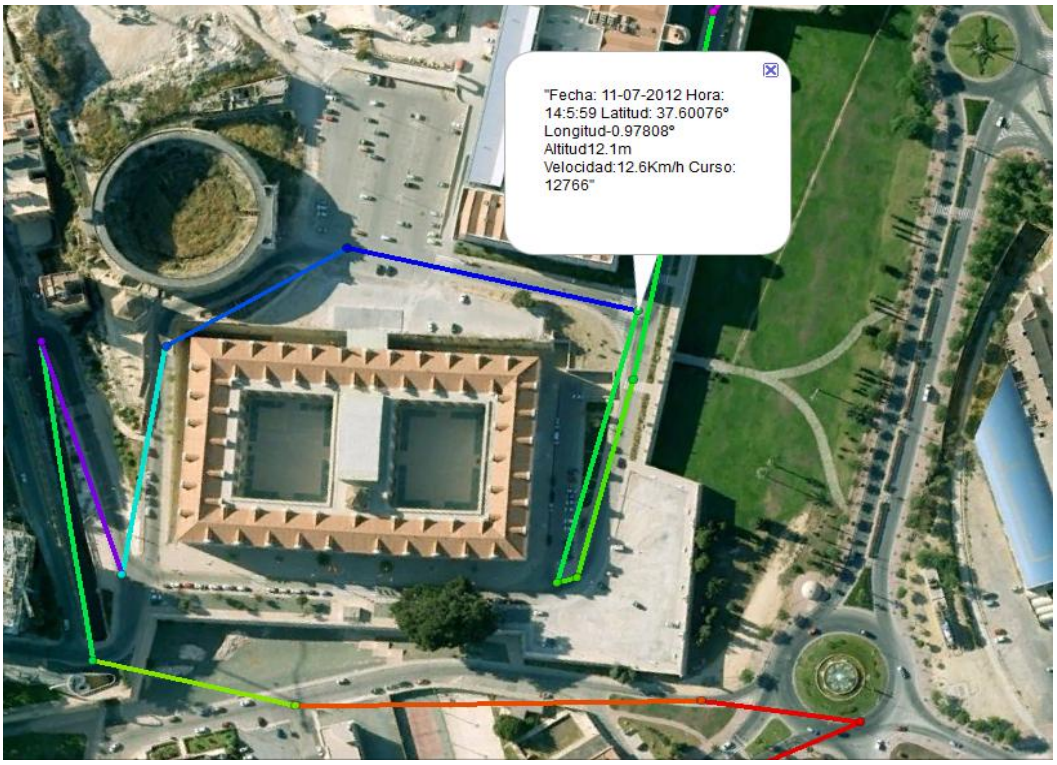


Figura 4.19 Resultado de la ruta en Google Earth.

4.3.4. Adquisición de datos en la tarjeta de memoria

Como hemos comentado en capítulos anteriores, una de las finalidades a las que podemos destinar el proyecto es la de recolectar los datos procedentes de los pines de entrada/salida. La adquisición sólo se realizará de aquellos pines que se encuentren configurados.

Existen dos posibilidades: Almacenar todos los pines configurados cada vez o almacenar un pin concreto. El sistema distingue entre un modo u otro por medio del valor que tome el las posiciones 4 y 5 de la tabla 4.3 que conforman el pin. Si este es 0 indicará que deben guardarse todos los pines en un archivo llamado "**SENSORES.txt**", de lo contrario se registrará uno concreto, creando un archivo .txt con el nombre del pin a registrar. Estos datos se guardan en la carpeta "**SENSORES**" ubicada en la raíz de la tarjeta, la cual (al igual que la carpeta "**GPS**" se crea automáticamente).

A la hora de grabar los datos, estos se tabulan añadiendo una cabecera consistente en: **Pin; Sensor/Actuador, Valor, Fecha y Hora**. Los valores de Fecha y Hora son obtenidos del GPS tal y como hemos comentado antes, debiendo de aplicar una corrección del meridiano dado que nos proporciona la hora central. En el campo "Pin" se imprime el número del pin del sistema GNX Project v.1 y no su correspondencia con los de Arduino.

Todo esto se puede apreciar en la figura 4.20:

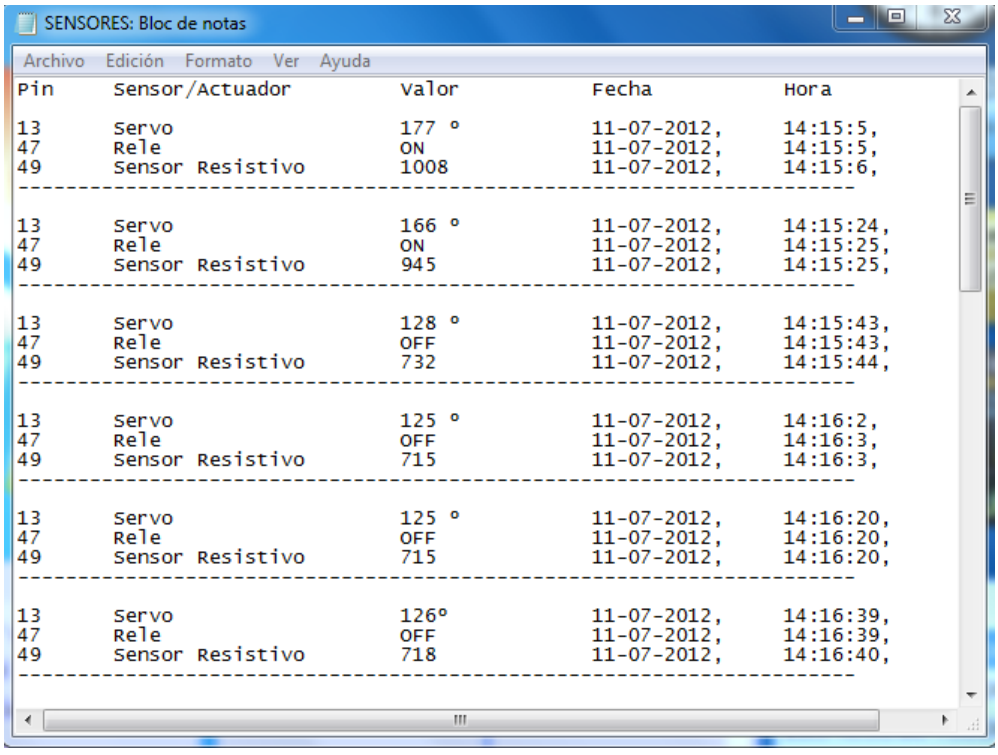


Figura 4.20 Ejemplo de registro de pines en la MicroSD.

Al igual que sucedía con el GPS, estos datos se graban con un espaciado mínimo de 20 segundos, parpadeando los dos led del panel frontal de forma alternativa durante el proceso. Mediante el pulsador "SD" podemos montar/desmontar la tarjeta, de forma que

cuando el led “Check” esté encendido indicará que podemos extraer la tarjeta dado a que hemos impedimos la escritura en su interior.

Cabe mencionar que los led también indican si hay algún fallo en la tarjeta, mediante las siguientes combinaciones:

- **Led verde (State) fijo y led rojo (Check) apagado**: Se ha podido acceder a la tarjeta sin ningún problema.
- **State apagado y Check intermitente**: No se ha podido acceder al dispositivo. Los motivos pueden ir desde que la tarjeta carece del formato adecuado (FAT) hasta que esta no esté insertado.

4.3.5. Envío de datos por Bluetooth al Smartphone

Una de las funcionalidades implementadas en la aplicación para Smartphone es la posibilidad de ver el estado de todos los pines que tengan alguna configuración ya sea por asociación con sensores o por alguna orden inmediata.

Aparte de los datos sobre los pines, también se transmite las coordenadas GPS de forma que podamos visualizarla en la pantalla del Smartphone en tiempo real sin precisar que estas estén siendo grabadas en la tarjeta de memoria.

Para enviar los datos, se recurre a la función **Serial3.print()** que implementa la IDE de Arduino que permite escribir por el puerto serie especificado (en esta caso es el **Serial 3** que es donde se encuentra conectado el módulo Bluetooth) los datos indicados dentro del paréntesis.

Estos datos son separados por ";" y se transmiten todos los pines incluso los que no se encuentran configurados, dado que el número de elementos a enviar debe ser conocido por la aplicación del Smartphone, a fin de poder interpretar correctamente estos datos. En este proyecto el número de datos a enviar es de 55 (50 Pines + 5 datos del GPS).

Cada vez que la placa transmite los datos a la aplicación, el led State (el verde) parpadea tres veces indicando que se produce el tránsito de datos.

4.3.6. Periféricos soportados

Se han dado soporte para utilizar determinados sensores y actuadores conectados a los pines de entrada y salida que se encuentran formando una "U" en el módulo hardware. Estos elementos son:

- Relés
- Servos
- Modular anchura de pulso
- Altavoces
- Motores de CC.
- Sensores resistivos, tales como LDR, potenciómetro, etc…
- Pulsadores
- Interruptores
- Sensor de temperatura LM35
- Sensor de infrarrojo
- Sensor de ultrasonido

El tratamiento de los cuatro primeros elementos es muy simple. En el caso del relé, simplemente, empleamos la función **digitalWrite()** implementada en la IDE de Arduino, mediante la cual podemos activar un pin concreto proporcionando una tensión a la salida de 5V. A fin de simplificar el control se ha implementado en el firmware la posibilidad de que la misma instrucción permita activa y desactivar una salida, esto es aplicable únicamente al relé y al altavoz.

Para el control del motor y la modulación de la anchura de pulso se han empleado la función **analogWrite()** que permite poner a la salida una onda cuadrada de un % del ciclo de trabajo, de esta forma podemos controlar la velocidad del motor. Para el caso del servo, el principio es el mismo, sabemos que este es simplemente un motor de continua con una reductora, por lo que podríamos aplicar la misma función que antes, sin embargo esta está limitada a únicamente los pines que permiten modulación de anchura de pulso (PWM), sin embargo si empleamos la librería Servo.H incluida de serie en la IDE de Arduino podemos controlar casi 40 servos desde una misma placa. Su funcionamiento es muy simple, generar una onda cuadrada poniendo la salida a 5V mediante **digitalWrite(PIN, HIGH)** durante X milisegundos y luego poner la salida a LOW.

Si atendemos a los sensores, Arduino tiene una resolución que va desde 0 hasta 1023, de ahí que se empleara tres caracteres para los campos **Valor 1** y **Valor 2** de las tramas. Para los sensores resistivos sólo se debía leer valor presente a la entrada mediante la función **analogRead()**, sin embargo para los sensores de temperatura, infrarrojo y ultrasonido debía de procesar esta valor a fin de expresarlo en grados centígrados o en cm. Para los pulsadores al igual que los interruptores se emplea la función **digitalRead()** que nos devuelve el 1 si está cerrado ó 0 en caso de que esté abierto. En ambos casos el sistema elimina los rebotes de los pulsadores realizando dos lecturas espaciadas 100 milisegundos ya que al presionarse, no se alcanza la referencia de forma inmediata, sino que se producen sobrepicos que pueden ser detectado como varias pulsaciones, los cuales tienes un tiempo de establecimiento de 10 milisegundos, de ahí que se lean cada 100. En el caso de los pulsadores se emplean como condición el número de veces que son pulsados a diferencia de los interruptores que se emplean el utilizan su estado abierto/cerrado.

4.4. Diseño de la aplicación para Smartphone

Para concluir este capítulo debemos de realizar algunas consideraciones referentes al funcionamiento de la aplicación para Smartphone.

Esta debe de proporcionar una interfaz agradable e intuitiva, que ofrezca al usuario todos los datos y funcionalidades requeridas para un correcto control del dispositivo desarrollado. Se compondrá de dos pestañas:

- **Home**: Que contendrá las opciones del menú principal tales como:
    - **Conectar**: Permite localizar el sistema GNX Project v.1 de la lista dispositivos disponibles y conectarse con él.
    - **Favoritos**: Submenú donde el usuario podrá definir aquellas acciones que ejecute con mayor frecuencia, permaneciendo estas almacenadas en la raíz de la tarjeta de memoria del Smartphone. Así mismo tendrá la posibilidad de editar, renombrar o eliminar dicho favorito.
    - **Monitorizar**: En este apartado se procesan los datos procedentes de la placa tal y como se ha explicado en el apartado 4.3.5 del presente capítulo. El usuario podrá visualizar únicamente los pines configurados y tendrá la opción de manipular las salidas sin precisar salir al menú principal. Así mismo constará con una opción llamada "Mostrar/Ocultar GPS" que permitirá mostrar/ocultar los datos relativos a la posición, velocidad y curso.
    - **Panel de Control**: Es en esta sección donde el usuario podrá realizar la configuración del dispositivo, distinguiendo las siguientes opciones:
        - **Configurar Sensor**: Donde definiremos la trama que configura el sensor así como la condición que debe de cumplir.
        - **Programar Acciones**: Estableceremos una a una las acciones que se desean asociar al sensor definido en el punto anterior.
        - **Ver acciones programadas**: Buffer de salida de las instrucciones a enviar, que son las definidas en los dos anteriores puntos. Así mismo todas ellas será modificables, pudiendo, editarlas, eliminarlas o incluso enviar cada una (salvo la referente al sensor) como acción inmediata.

- **Eliminar acciones programadas**: Permite realizar el borrado de la EEPROM del Arduino.

- **Enviar**: Transmite cada una de las instrucciones almacenadas en el apartado: "Ver acciones programadas" incluyendo al final la instrucción "**fin de acciones programadas**" (Contraseña + 32000000000).

- **Volver**: Permite regresar al menú principal.

❖ **Opciones avanzadas:** Proporciona al usuario algunas opciones adicionales como:

- **Modo manual:** Permite enviar una trama introducida manualmente.
- **Altavoz:** Permite encender/apagar el altavoz integrado en la placa.
- **Reiniciar sistema:** Provoca el reset por medio del transistor descrito en la figura 4.9.
- **Eliminar acciones programadas:** Permite borrar todas las EEPROM al igual que la disponible en el menú "**Panel de Control**".

❖ **Salir**

- **<u>Configuración</u>:** Permitirá configurar parámetros como:

❖ Contraseña.

❖ Cada cuanto se grabarán los datos del GPS.

❖ Espacio de tiempo entre dos grabaciones en la SD en lo referido a la adquisición de datos.

Como apunte debemos indicar una funcionalidad a las que dotaremos a la aplicación la cual no han quedado recogidas en los puntos anteriores:

En el menú de Monitorizar, al mostrar la información del GPS será posible reenviar dicha información por SMS al teléfono que se especifique en la interfaz que aparecerá en pantalla. Esto será muy útil para realizar un registro manual del recorrido trazado así como que el destinatario conozca nuestra situación de una forma rápida y precisa.

5. Implementación

5.1. Introducción

En este capítulo se describe el proceso completo de implementación del proyecto, partiendo de la fabricación de la placa del circuito impreso para luego continuar con el proceso de configuración de la IDE de Arduino y la grabación del firmware desarrollado, el cual se ha descrito en el capítulo anterior. Así mismo se hará mención al proceso de compilación del código fuente desarrollado en Python a fin de crear un fichero ejecutable .SIS para plataformas Symbian, el cual incluirá el intérprete y las librerías a objeto de no precisar la instalación de estos elementos de forma individual.

5.2. Implementación del hardware

Una vez que se han definido en el esquemático todos aquellos elementos que precisa el proyecto y probado su correcto funcionamiento mediante la ayuda de un voltímetro, se procederá a realizar el portado desde el esquemático de Orcad Capture hasta la interface de diseño de Orcad Layout donde se realizará un PCB inicial, el cual será editado con posterioridad con la herramienta Altium Designer Summer 09 a objeto de realizar cambios por motivos que se descubran en posteriores revisiones del diseño de una forma mucho más rápida de la que proporciona Orcad.

El tamaño del PCB es de 12,5 x 10 cm, siendo estas las dimensiones óptimas para que se pueda introducir sin ningún problema en la caja de prototipado cuyas dimensiones generales son: 15,5 x 10 x 4,5 cm. A efectos de aprovechar este espacio libre en la caja, se ha desplazado la placa y todos sus elementos a la zona derecha de la misma, a fin de ubicar en ese espacio libre el jack de alimentación así como la pila, como podemos apreciar en la figura 5.1.

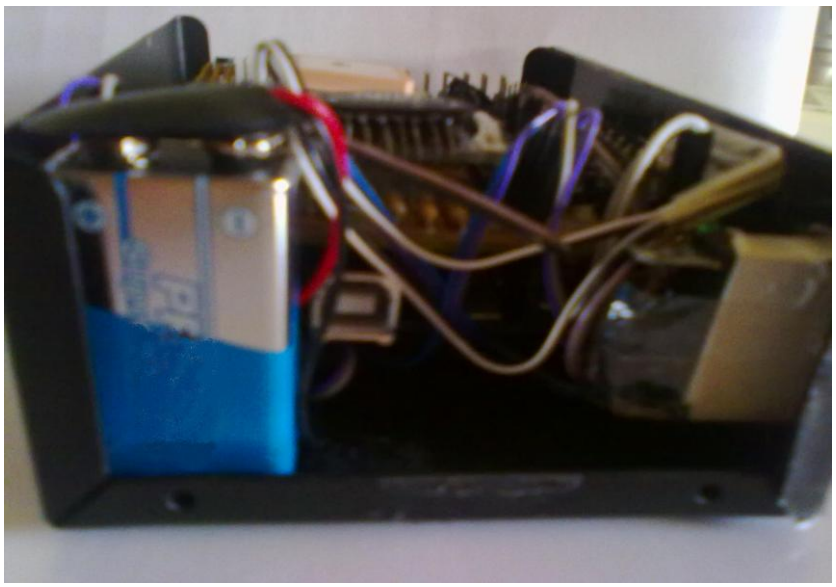


Figura 5.1: Emplazamiento de la pila y la placa dentro del conjunto.

Tanto la colocación de los componentes como el ruteado de las pistas se realizarán de forma que se puedan lograr los siguientes objetivos:

- Acortar la longitud de las pistas.
- Disminuir el número de pistas.
- Minimizar el número de vías.
- Facilitar el soldado de todos los componentes
- Permitir el acceso al puesto USB de la placa Arduino así como los conectores de los pulsadores adicionales, que se alojarán bajo la placa en la zona izquierda de la misma.

El elemento principal del diseño es el Arduino, el emplazamiento de este elemento condicionará la ubicación de todos los demás, por ello se ha optado por colocarlo a 1.25 cm del centro de la placa, tal y como podemos observar en el PCB de la figura 9.3 del capítulo 9. Esto permite que el conector USB se encuentre disponible en el lado izquierdo sobresaliendo unos pocos milímetros respecto al perímetro de la placa.

La placa de circuito impreso se conectará de forma vertical a los pines que dispone el Arduino Mega 2560 de forma que pueda desmontarse con facilidad realizar modificaciones o inspecciones. Sobre esta placa se colocarán todos los elementos quedando como se aprecia en las figuras 5.2 y 5.3.

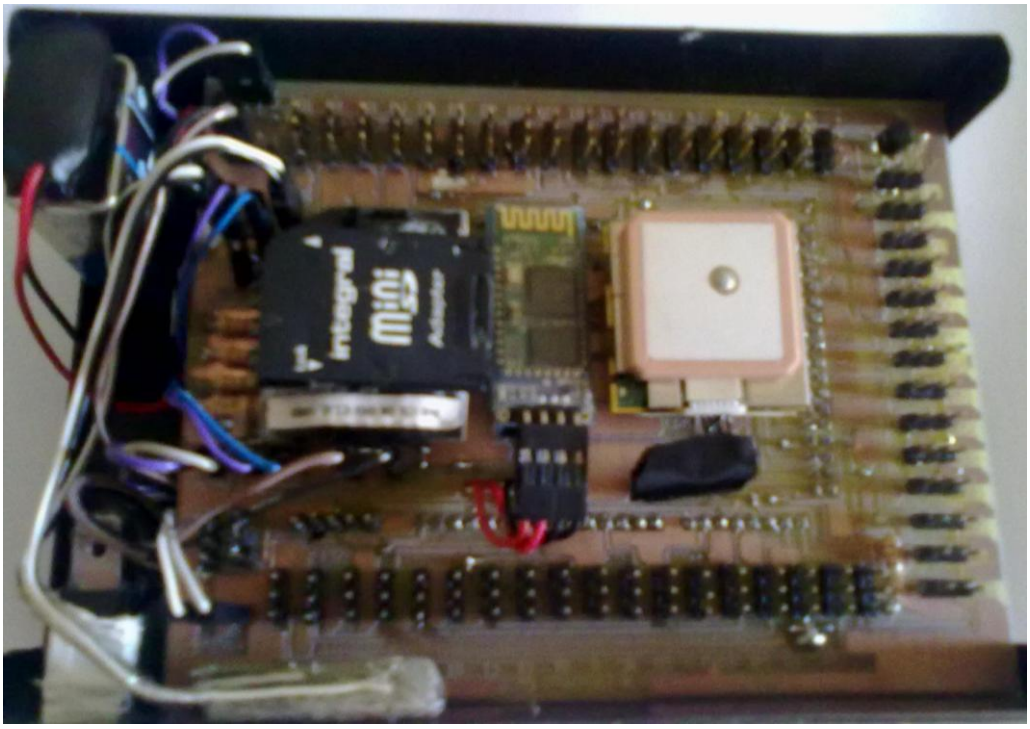


Figura 5.2: Montaje de todo los elementos del sistema GNX Project v.1.

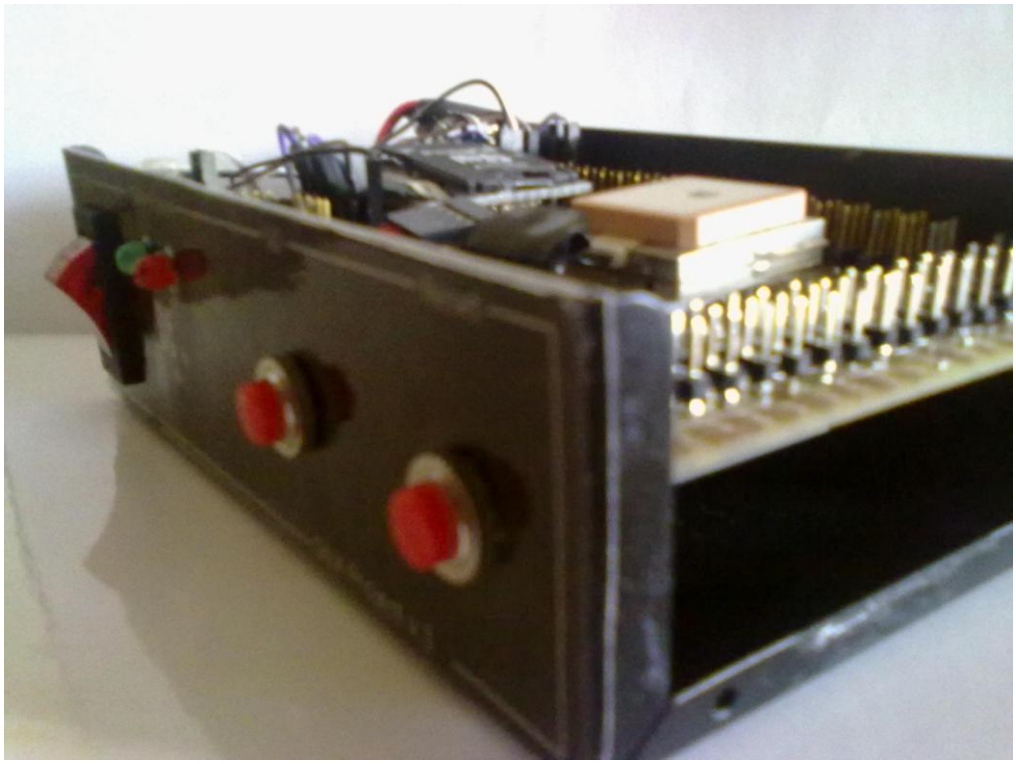


Figura 5.3: Vista trasversal del conjunto.

5.3. Implementación del firmware en la plataforma Arduino

Como se ha comentado con anterioridad, la iniciativa Arduino se creó como una herramienta didáctica destinada a motivar e iniciar a los usuarios carentes de conocimientos previos de programación y electrónica al mundo del desarrollo "homebrew", es decir, la creación de pequeños dispositivos más o menos elaborados por el usuario a partir de reducidos recursos y costes.

En los últimos años se ha popularizado esta plataforma, haciendo que distintos desarrolladores creen nuevas aplicaciones destinadas a simplificar el proceso de desarrollo, como es el caso de la herramienta "**Minibloq**" que como se puede apreciar en la figura 5.4 propone una programación gráfica basada en piezas de puzle que sustituyen a las instrucciones de código teniendo también la posibilidad de complementarlo mediante un editor anexada a la ventana que permite el desarrollo de forma escrita.

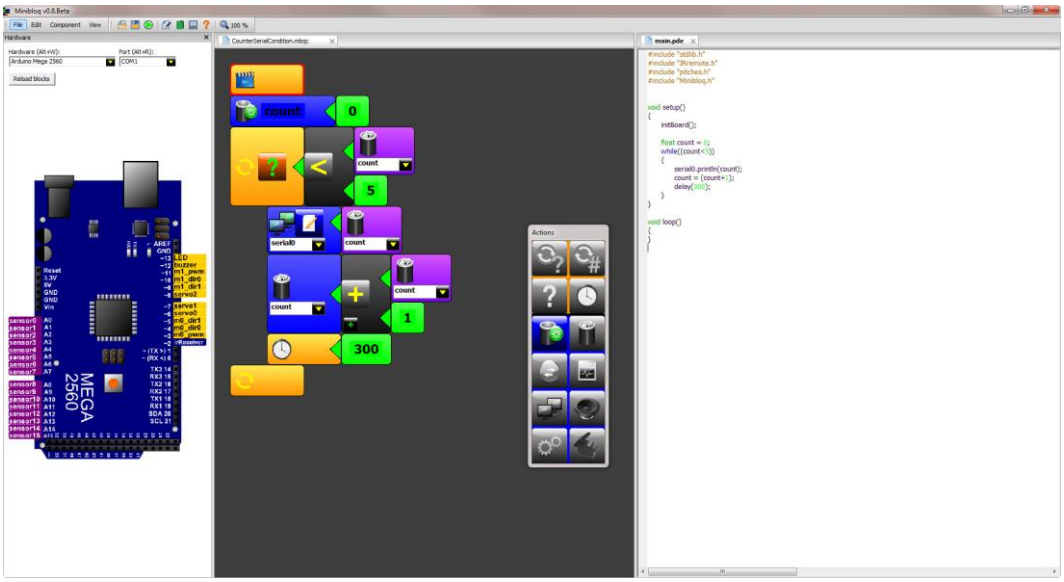


Figura 5.4: Menú principal del software de desarrollo Miniblg.

Pese a esto, la aplicación aun se encuentra en sus primeras fases de desarrollo presentando algunos fallos a la hora de compilar, por lo que se decidió emplear la plataforma software de desarrollo de los creadores de Arduino. Esta IDE que así es como se la conoce comúnmente, se puede descargar des la web oficial de Arduino en la siguiente dirección http://arduino.cc/en/Main/Software.

Para poder realizar este proyecto hemos empleado la versión 0.23 en lugar de la actual versión 1.0.1; debido a que a partir de la 1.0, el equipo de desarrollo de la plataforma realizaron cambios muy profundos en la IDE por lo que toda la documentación y ejemplos procedente de libros oficiales de Arduino dejaron de funciona, obligando a reescribirlos.

A parte de este hecho, la nueva versión 1.0 no incluía ninguna mejora relevante más allá de mejoras gráficas como cambio de iconos o una barra de proceso de grabación como podemos observar en la figura 5.5.

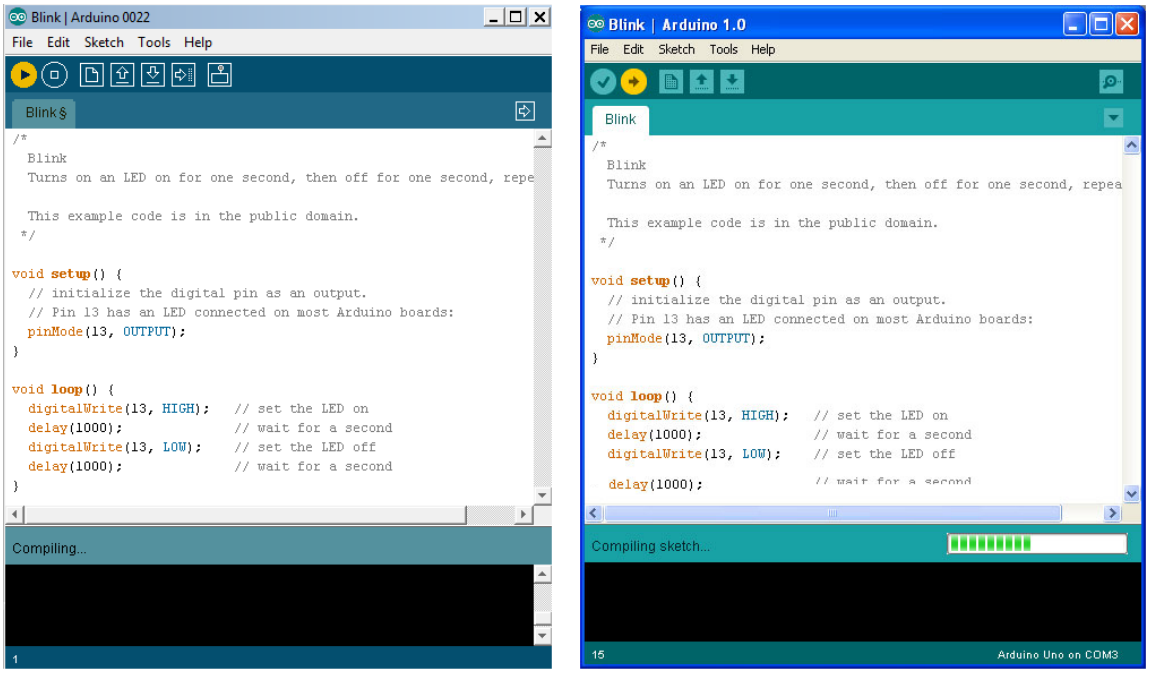


Figura 5.5: Comparación del menú principal de la IDE Arduino 0.22 y 1.0.

La IDE permite realizar determinadas tareas destinadas al desarrollo de proyectos. A continuación vamos a exponer cada una de estas tareas que están asociadas a las opciones de la barra de herramientas en la parte superior de la pantalla. En la tabla 5.1, vamos a presentar cada unas de ellas (las de la versión 0.22) acompañadas de una breve descripción.

Icono	Nombre	Descripción
	*Verify/Compile*	Chequea el código en busca de errores.
	*Stop*	Finaliza la monitorización serie y oculta otros botones
	*New*	Crea un nuevo *sketch*.
	*Open*	Presenta un menú de todos los programas *sketch* de su "sketchbook", (*librería de sketch*). Un click sobre uno de ellos lo abrirá en la ventana actual.
	*Save*	Salva el programa *sketch*.
	*Upload to I/O Board*	Compila el código y lo vuelca en la placa E/S de Arduino.
	*Serial Monitor*	Inicia la monitorización serie del puerto serie principal de la placa

Tabla 5.1: Descripción de las opciones de la IDE de Arduino.

Como hemos mencionado con anterioridad, la base del proyecto reside en el Arduino Mega 2560. Para poder utilizar esta placa con la IDE de Arduino se precisa indicarle en el menú **Tools/Board** el tipo de placa, en nuestro caso Arduino Mega 2560, como vemos en la figura 5.6.

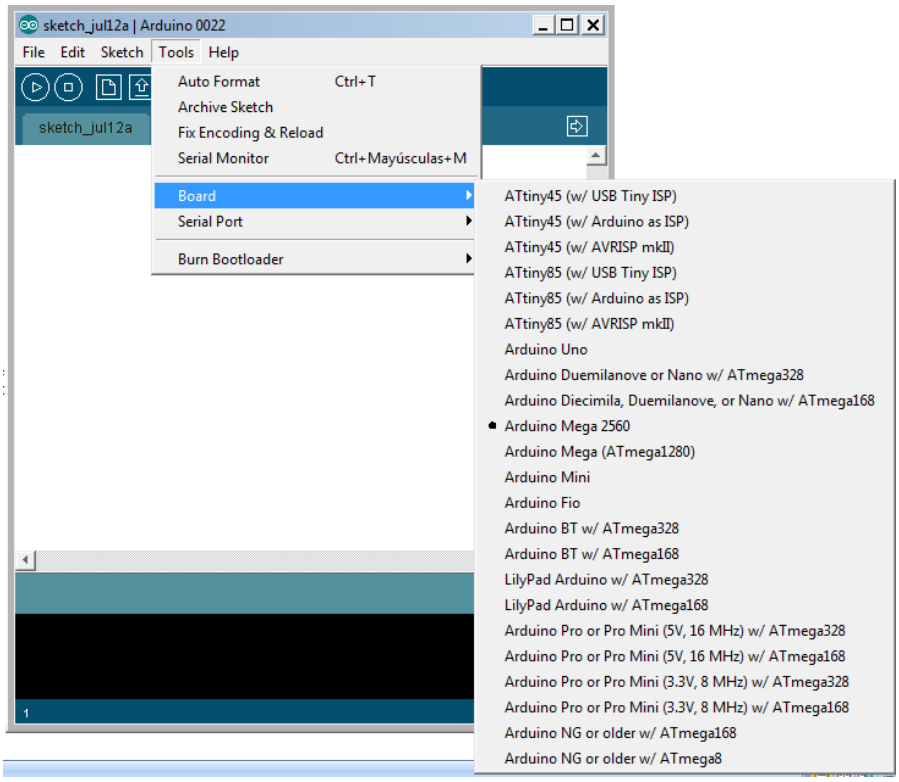


Figura 5.6: Selección de modelo de placa en la IDE.

Esta operación sería suficiente para cualquier otro modelo (a excepción de la versión UNO), sin embargo, al conectar el Arduino Mega 2560 el PC no es capaz de reconocerlo; debido a que precisa de un driver específico para gestionar este dispositivo, el cual se encuentra disponible en la descarga de la IDE, concretamente en la carpeta "**Drivers**".

Para asociar este driver con el dispositivo USB Arduino, debemos ir al administrador de dispositivos de Windows, en el caso de Windows 7 debemos escribir en la barra de búsqueda del menú de inicio: "**Administrador de Dispositivos**" (figura 5.7).

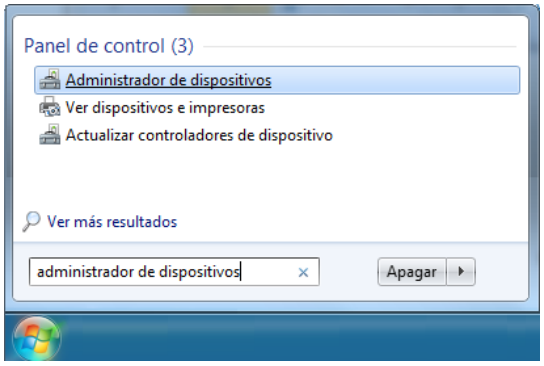


Figura 5.7: Acceso al Administrador de Dispositivos.

Una vez en el Administrador de dispositivos, debemos dirigirnos a la pestaña "**Otros dispositivos**" donde observaremos nuestro periférico acompañado de un signo de exclamación. Debemos hacer doble clic sobre él a fin de abrir un segundo menú donde observamos que Windows no ha podido encontrar el controlador para su funcionamiento. Para arreglarlo, debemos pulsar sobre la opción "**Actualizar Controlador**" y posteriormente a "**Búsqueda de software de controlador en el equipo**" donde deberemos seleccionar la carpeta "Drives" contenida en el directorio de la IDE. Todos estos pasos se quedan resumidos en la figura 5.8.

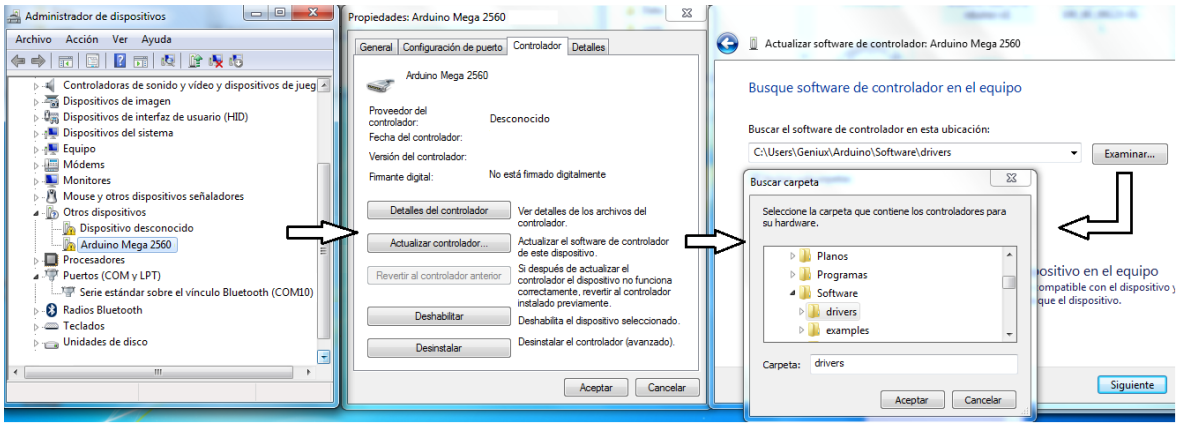


Figura 5.8: Pasos para instalar drivers.

Ahora ya estamos en condiciones de iniciar la grabación del código fuente de nuestro programa o *sketch* en nuestra placa. Para ello además de seleccionar el tipo de placa, debemos establecer el puerto de comunicación asignado por Windows en el menú **Tools/Serial port**, aunque esto es realizado automáticamente por la IDE al detectar el dispositivo.

Una vez realizado el código o parte de este, se procederá a grabarlo siempre y cuando la IDE no detecte errores sintácticos o lógicos. Durante el proceso de grabación, los led TX (pin 1) y RX (pin 0) de todas los modelos de Arduino parpadean indicando el este hecho. Cuando se vuelca un "sketch", está utilizando el Bootloader de Arduino, un pequeño programa que ha sido cargado en el microcontrolador el cual permite el volcado del código sin utilizar hardware adicional. El Bootloader está activo durante unos segundos cuando la placa es reseteada; después se inicia el sketch presente en el microcontrolador. Es este Bootloader el que produce un parpadeo en el LED de la placa (pin 13) cuando se inicia así como cuando son reiniciadas.

Recordando brevemente lo comentado en el capítulo 4 en lo referente a los jumpers, el puerto serie principal de la versión Mega y Mega 2560 (ya que estos disponen de un total de cuatro puertos series por defectos, aunque este número puede aumentarse mediante el uso de las librerías: **SoftwareSerial** o **NewSoftSerial**) al puerto serie principal (**Serial 0**) no puede haber conectado nada, debido a que daría lugar a un error de compilación (ver figura 5.9) e incluso dañar el dispositivo que tengamos conectado a ese puerto, como por ejemplo el módulo transceptor Bluetooth.

Figura 5.9: Error de grabación por ocupación del puerto *Serial 0*.

Estos puertos son adicionales que hemos comentado antes son:

- **Serial 1** en los pines 19 (RX) y 18 (TX)
- **Serial 2** en los pines 17 (RX) y 16 (TX)
- **Serial 3** en los pines 15 (RX) y 14 (TX)

Para utilizar estos pines para comunicarse con el ordenador personal, se precisará un adaptador USB adicional a serie como el de la figura 5.10 usarlos para comunicarse con un dispositivo serie externo TTL, es preciso conectar el pin TX al pin RX del dispositivo, el RX al pin TX del dispositivo, así como las masas de ambos.

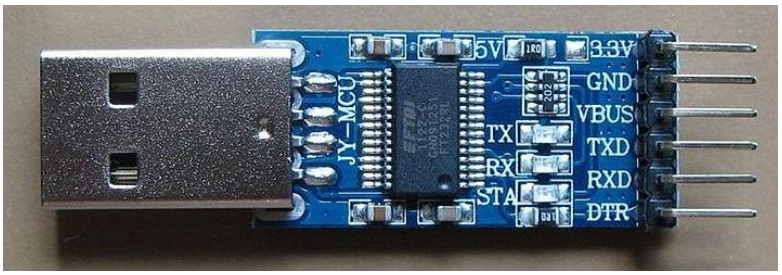


Figura 5.10: Adaptador USB-Serie.

5.4. Implementación de la aplicación para Smartphone

En este apartado nos vamos a central en el proceso de compilación del código fuente de la aplicación para Smartphone desarrollado en Python. Debido a la no disponibilidad de un dispositivo Android donde desarrollar la aplicación, se decidió realizarla para un Smartphone de 5º generación con sistema operativo Symbian, concretamente en un **Nokia 5230 Xpressmusic** con versión de software 3.00.959.

Como otros muchos lenguajes como Java o Lua, requiere ser interpretado en tiempo de ejecución por lo que se precisa de una máquina virtual que cumpla esa función instalada en el dispositivo. Esto hace que se precise instalar un total de tres archivos en el Smartphone:

- **Python_2.0.0.sis**: Es el kernel del lenguaje Python necesario para poder ejecutar cualquier aplicación desarrollada en este lenguaje.
- **PythonScriptShell_2.0.0_3_2.si**s: Es el intérprete en sí mismo, el cual contiene los archivos esenciales como librerías y plugin destinadas a realizar la adaptación del lenguaje a la plataforma Symbian. También incluye algunos ejemplos de programas.
- **Archivo .py**: Archivo que contiene el código fuente del programa desarrollado con extensión .py el cual debe alojarse a la carpeta PYTHON que se crea al instalar los archivos anteriores. Esta deberá ser cargada de manera manual en el intérprete haciendo: **Option/Run Script** y seleccionándolo de la lista que nos aparece en pantalla.

Con objeto de simplificar el proceso de instalación, se decidió compilar la aplicación en un único archivo ejecutable .SIS que contuviera el intérprete, el kernel del Python y el código fuente del programa desarrollado, donde este fuera autoejecutable, es decir, no se precisara cargarlo de forma manual.

Para ello, encontramos en una sección de la web de Python un compilador para Symbian que nos proporcionaba los requisitos exigidos y que además permitía personalizar el .SIS en términos de nombre, icono o versión entre otros, su nombre es **Py60 Application Packager**. Este puede ser descargado desde el siguiente enlace: http://www.python.org/download/releases/2.5.4/

Una vez instalado el compilador en nuestro equipo, procedemos a comentar cada unas de las partes que conforman el programa (referido a las que aparecen en la figura 5.11 B) para luego exponer del proceso de compilación y posterior instalación en el Smartphone:

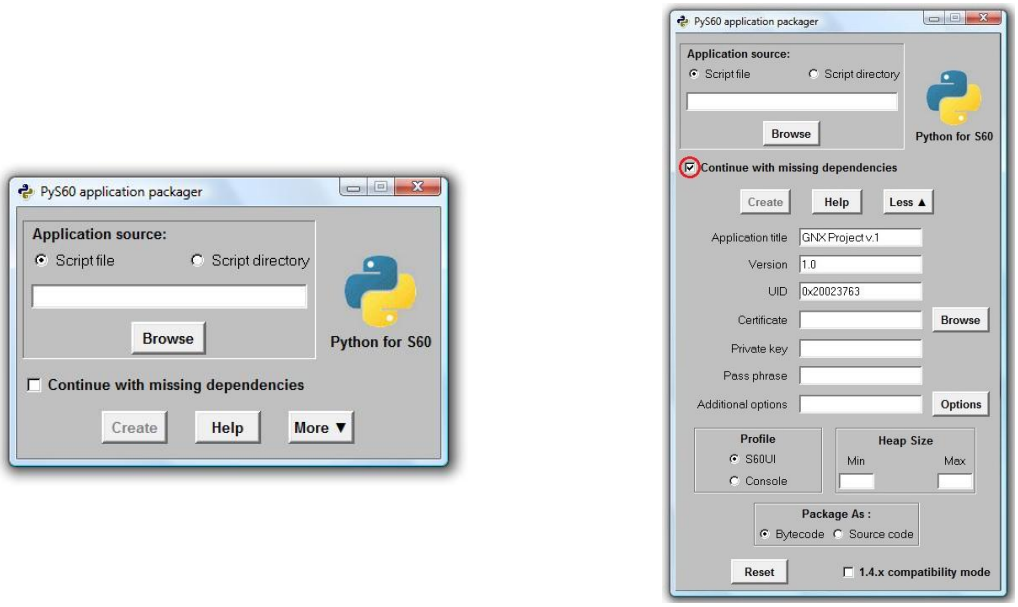
- **Applitation title**: Nombre que aparecerá en el menú principal del Smartphone el cual no pude contener caracteres como la “ñ” o signos de puntuación.
- **Versión**: Número de versión que aparecerá en la interface de instalación del programa.

- **UID**: Cada aplicación independiente del intérprete Python tiene un identificador único (UID) que define el área protegida del sistema de archivos donde se pueden almacenar con seguridad sus datos. Por defecto es asignado por el compilador se el valor 0xE. En la tabla 5.2 podemos observar los valores UID de algunos teléfonos de la marca Nokia de 3º y 5º generación, así como el que se emplea para el desarrollo de este proyecto.

- **Certificate y Private Key**: Son archivos utilizados para firmar las aplicaciones a fin de poder ser ejecutadas en dispositivos sin precisar modificar el firmware de estos. En nuestro caso, no precisamos de estos campos.

- **Pass phrase**: Permite establecer una contraseña para la ejecución del .SIS.

- **Additional options**: Nos permite seleccionar el icono y nivel de seguridad para el control parental entre otros pero mediante el uso de comandos como: ***--icon=\icon_GNX.svg***.



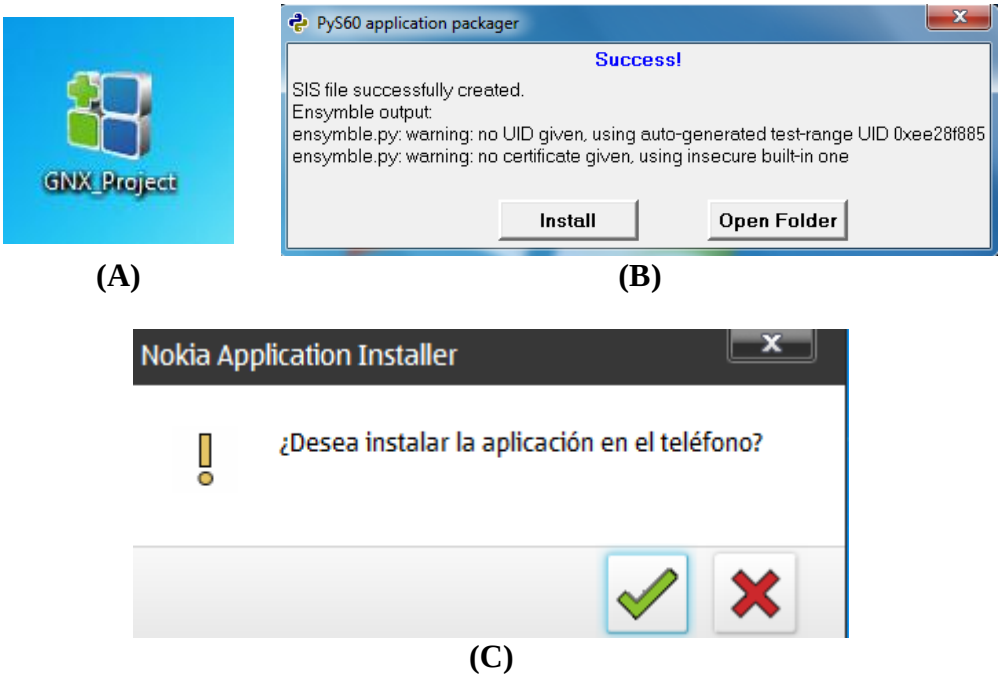
(A) **(B)**

Figura 5.11: Interface del compilador del Py60 Application Packager.

Generación	ID del producto	UID	Dispositivo
5º	0x2002376B	0x2001DE9D	Nokia 5530 XpressMusic
5º	0x20023763	0x20023763	Nokia 5230 & Nokia 5235
5º	0x2002376B	0x2002376B	Nokia 5230 Nuron
5º	0x2002376B	0x2001DE9E	Nokia 5530 XpressMusic
5º	0x2000DA56	0x2000DA56	Nokia 5800 XM
3º	0x20023766	0x20023766	Nokia N97 mini
3º	0x200227DD	0x200227DD	Nokia X6-00
3º	0x200005F8	0x200005F8	Nokia 3250
3º	0x20000602	0x20000602	Nokia 5500 Sport
3º	0x20002495	0x20002495	Nokia E50
3º	0x20001856	0x20001856	Nokia E60
3º	0x20001858	0x20001858	Nokia E61
3º	0x200005FA	0x200005FA	Nokia N92

Tabla 5.2: Algunos ejemplos de UID de dispositivos Nokia.

Una vez conocido el cometido de estas opciones que resultan de pulsar la opción "**More"** de la interface básica de la figura 5.11 A, simplemente debemos rellenar los campos que veamos necesarios, debiendo marcar la casilla "**Continue with missing dependencies**", si no rellenamos todas las opciones, de lo contrario saltaría un error que abortaría la compilación. Si no se precisa de ninguno de estos campos, bastará con la versión básica donde solo deberemos examinar el archivo .py creado el cual no disponga de errores o warning dado que esto también abortaría la compilación. Cuando tengamos definidos los parámetros simplemente debemos pulsar la opción "**Create"**. Esto generará el archivo (ver imagen 5.12 A) y nos dará la posibilidad de instalarlo en el dispositivo (ver imagen 5.12 B), para lo cual será necesario disponer de la aplicación: **Nokia PC Suite** que se puede descargar de manera gratuita desde la web de Nokia (ver imagen 5.12 C).



(A) **(B)**

(C)

Figura 5.12: Proceso de descarga en el Smartphone.

Una vez iniciada la instalación simplemente se deberá seguir los pasos que aparecen en la pantalla del dispositivo, los cuales quedan resumidos en la figura 5.13.

Figura 5.13: Proceso de instalación del .SIS.

6. Experimentación

6.1. Introducción

Una vez finalizada la implementación, se deben realiza una serie de pruebas para evaluar el correcto funcionamiento tanto de periférico como de la aplicación software.

Las pruebas a realizar son:

- Verificación del correcto funcionamiento del GPS y la SD.
- Comprobar el correcto funcionamiento de la aplicación software.

6.2. Comprobación del módulo GPS y la tarjeta SD

Para que el GPS incorporado al sistema *GNX Project v.1* obtenga una posición válida la primera vez es preciso que se encuentre en abierto durante un periodo de un minuto y medio frente a los 42 segundos indicado por el fabricante. Una vez establecido el enlace con el satélite, el led comienza a parpadear, indicador de que ya se pueden realizar las tareas de adquisición y procesado de la señal.

El problema surge cuando pasamos de una zona con cobertura a otra donde esta es errática, como por ejemplo un garaje. En estos casos, el receptor GPS no es capaz de triangular correctamente la posición haciendo que esta se desvíe considerablemente dando lugar a algo como lo que se aprecia en la figura 6.1.

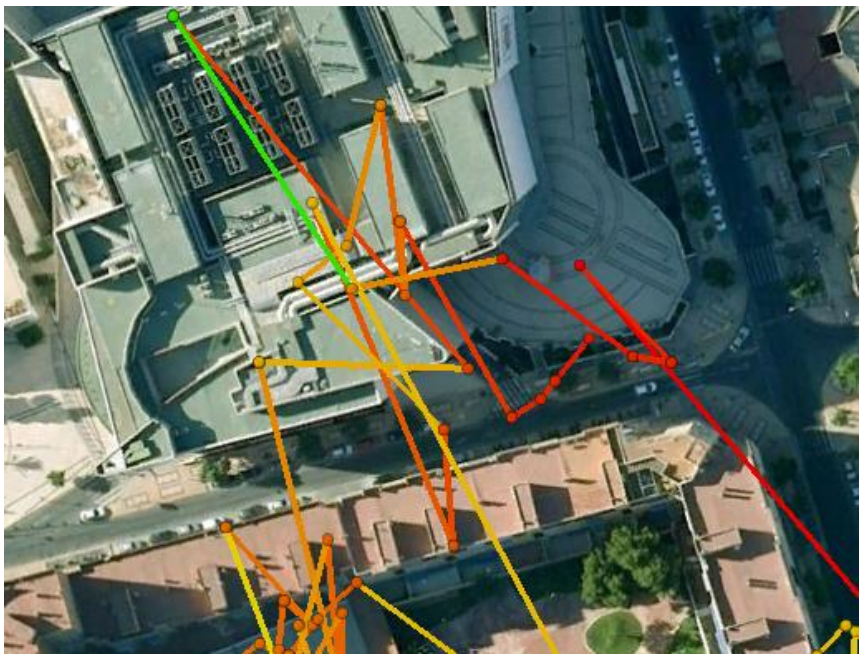


Figura 6.1: Desviación de las coordenadas satélite.

Este aspecto no es posible solucionarse por software de una forma simple dado que habría que tener en cuenta la dispersión de los puntos y en base a eso fijar un perímetro donde estos se consideraran válidos en función si se encontraran comprendidos o no en

dicho perímetro, sin embargo la complejidad reside en que estos valores no siguen un patrón definido, variando en función de la velocidad, la cual tiene una excelente precisión para velocidades a partir de los 5 km/h, fluctuando mucho cuando el sistema se encuentra detenido.

Pese a este hecho, el resto de las ruta es registrada sin ningún problema proporcionando una gran fiabilidad contractada gracias a la exportación de los datos e Google Earth, tal y como se expuso en el capítulo 4.

El dispositivo crea correctamente un archivo distinto donde almacenar los datos del GPS cada vez que es encendido, permitiendo realizar varios registros independientes lo cual es útil cuando se desea por ejemplo evaluar distintas ruta en términos de velocidad.

En general, el sistema de almacenamiento que dispone el proyecto ha respondido correctamente, sólo presentando problemas cuando se extraía la tarjeta durante el proceso de escritura, en cuyo caso está perdía el formato y por lo tanto todo su contenido.

6.3. Comprobación del software

Es preciso verificar que la aplicación para Smartphone cumpla con su cometido de configurar el módulo hardware pero además debe realizar de forma correcta el reto de funciones que se indicaron en el capítulo de diseño.

En primer lugar, hemos verificado si la aplicación era capaz de vincularse con el sistema *GNX Project v.1* vía Bluetooth de forma correcta. Para ello nos hemos valido del led que incorpora el módulo transceptor Bluetooth el cual parpadea mientras está a la espera de conectarse con algún dispositivo y se permanece fijo cuando la comunicación se establece. También se ha probado la estabilidad de dicho enlace cuando el sistema se encuentra montado junto con el control en vehículo en movimiento, no habiendo obtenido ningún problema. Una vez establecida la comunicación la aplicación debe de mostrar un mensaje indicando este hecho, de lo contrario saldría un error interno en el intérprete de Python.

En el menú "Panel de control" se ha realizado varias programaciones de sensores y actuadores, las cuales has sido sometidas ediciones desde la propia interface a fin de probar su correcto funcionamiento. Tras ello, se han enviado todo lo programada de forma correcta, haciendo que la placa las ejecuten tal y como hemos descrito.

Analizando el proceso de monitorización, se ha observado que este se ejecuta de una forma fluida y sin errores, pudiendo apreciar la variación de los datos del GPS en la pantalla del Smartphone así como los demás elementos conectados al proyecto.

7. Conclusiones y trabajos futuros

7.1. Conclusiones generales del proyecto

Tras haber realizado las pruebas descritas en el capítulo anterior, se puede extraer las conclusiones a partir de los resultados obtenidos.

Como hemos podido comprobar, el proyecto cumple el objetivo principal, es decir, puede ser configurado a las necesidades del usuario; desde orientarlo para controlar pequeños electrodomésticos hasta realizar las funciones de Looger GPS pudiendo reflejar en Google Map la ruta realizada con puntos que indican datos como velocidad o curso entre otros.

Esta programación se realiza tanto por medio de la aplicación desarrollada para Smartphone como por el puesto USB que incorpora Arduino por medio de la opción "Serial Monitor" que incorpora la IDE de Arduino.

Como se ha comentado como anterioridad, el desarrollo de una aplicación para PC destinada a suplir las funciones de la aplicación para Smartphone no es objeto de este proyecto, se han sentado las bases para permitir, entre otras, su implementación en un futuro.

Tras haber sometido al proyecto a esfuerzos de "sobrecarga", entendiendo como tales el haberle asignado un número considerables de configuraciones a ejecutar de forma continuada y habiendo verificado el correcto funcionamiento de las mismas, podemos confirmar el cumplimiento de todos los objetivos del proyecto.

El módulo transceptor Bluetooth no ha presentado ningún problema a la hora de establecer comunicación ni en el tránsito de datos, siendo quizás su único inconveniente el hecho de que sólo puede estar vinculado a un único dispositivo maestro, dado a que como hemos mencionado en el apartado 2.2 sobre la **red pico o piconet** sólo podría hacerlo si se tratase de un maestro comunicándose a varios esclavos. Recordando que la tecnología Bluetooth emplea un rango de frecuencia sobre las que se va moviendo para optimizar el flujo de datos; estas se podrían ser multiplexadas para permitir varios maestros (Smartphone en este caso) puedan controlar el módulo hardware.

En lo referente al sistema de almacenamiento SD (en nuestro caso se ha empleado una tarjeta MicroSD montada sobre un adaptador MiniSD-SD) debemos destacar la necesidad de estar formateada en Fat 16 o Fat 32, no soportando el formato NTFS. A demás y como dato a tener en cuenta, si se extrae o desconecta la alimentación durante el proceso de escritura (siendo totalmente indiferente durante el de lectura), la tarjeta perderá su formato y con ello todo el contenido almacenado en su interior siendo necesario formatearla desde el PC. La solución a este problema se expuso en el capítulo 4, donde al ser preciso sacar la tarjeta para procesar los datos, se decidió instalar un pulsador en el panel frontal destinado a montar/desmontar la SD, impidiendo o no que las funciones escriban en ellas, así mismo se estableció una indicación luminosa que alertaba de cuando se había producido un error en la tarjeta, apagando el led verde y haciendo parpadear el rojo o cuando se está escribiendo en su interior, en cuyo caso ambos led parpadean alternativamente.

Como apunte negativo remarcar el problema (ajeno a nuestra programación), referido al módulo GPS EM-411, que como hemos comentado en el capítulo anterior, se produce una desviación de las coordenadas recibidas al pasar de una zona con cobertura a otra donde esta es significativamente menor.

Pese a este hecho, el proyecto ha superado todas las expectativas considerándose apto para su aplicación en diversos entornos y tareas.

7.2. Futuras ampliaciones y mejoras

Debido al gran potencial que posee el sistema *GNX Project v.1*, se plantean algunas futuras mejoras que dotarán de un abanico de posibilidades aun mayor.

Como futuras mejoras se proponen:

- Acoplar al conector adicional que dispone la placa desarrollada en previsión de estos fines, un módulo Ethernet como el de la figura 7.1 a fin de controlar y monitorizar el módulo a través de internet lo que traería consigo el desarrollo de una aplicación web alojada en un servidor remoto.

- Ampliar el número de sensores/actuadores que el sistema es capaz de soportar, como por ejemplo un sensor de humedad, útil para control automático de un invernadero.

- Portar la aplicación para Smartphone al sistema Android dotándola de mayores funcionalidades.

- Modificar la estructura de configuración detallada en el capítulo 4 a fin de poder establecer relaciones entre varios sensores diferentes.

- Añadir un display LCD de 16x2 con una pequeña consola a fin de sustituir el panel básico que incorpora el módulo, pueden realizar desde este las mismas funciones que se pueden realizar desde el Smartphone.

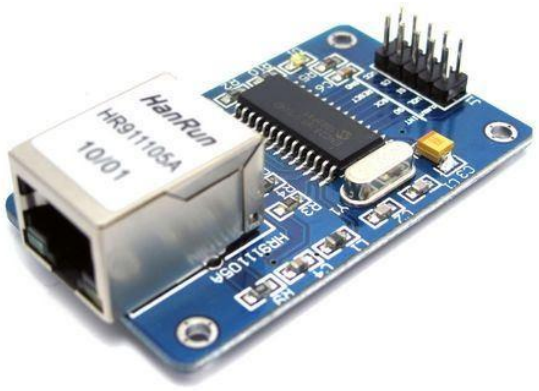


Figura 7.1: Módulo Ethernet.

II. PLANOS Y ESPECIFICACIONES

8. Planos

8.1. Esquemas eléctricos

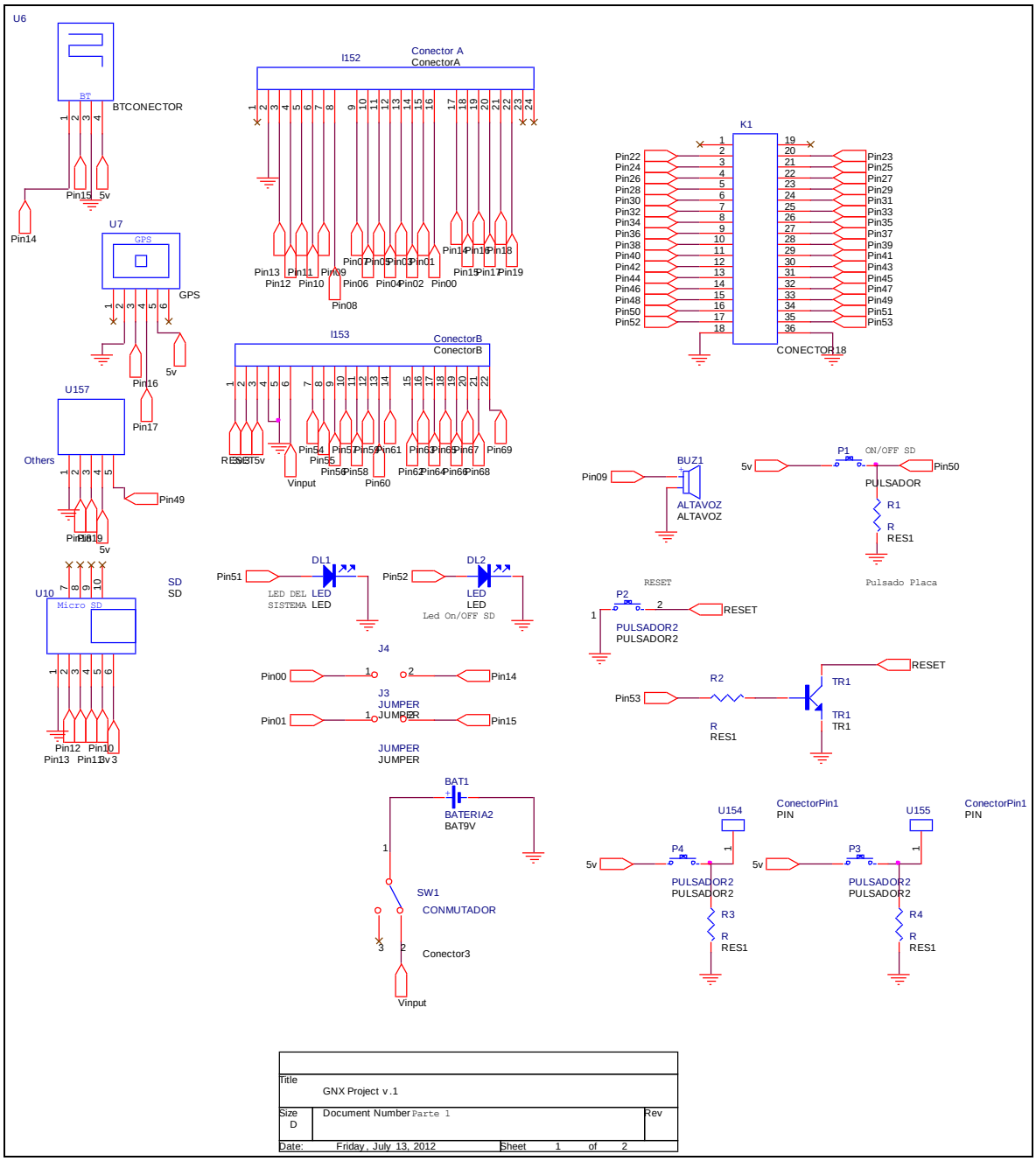


Figura 8.1: Parte 1 del esquemático.

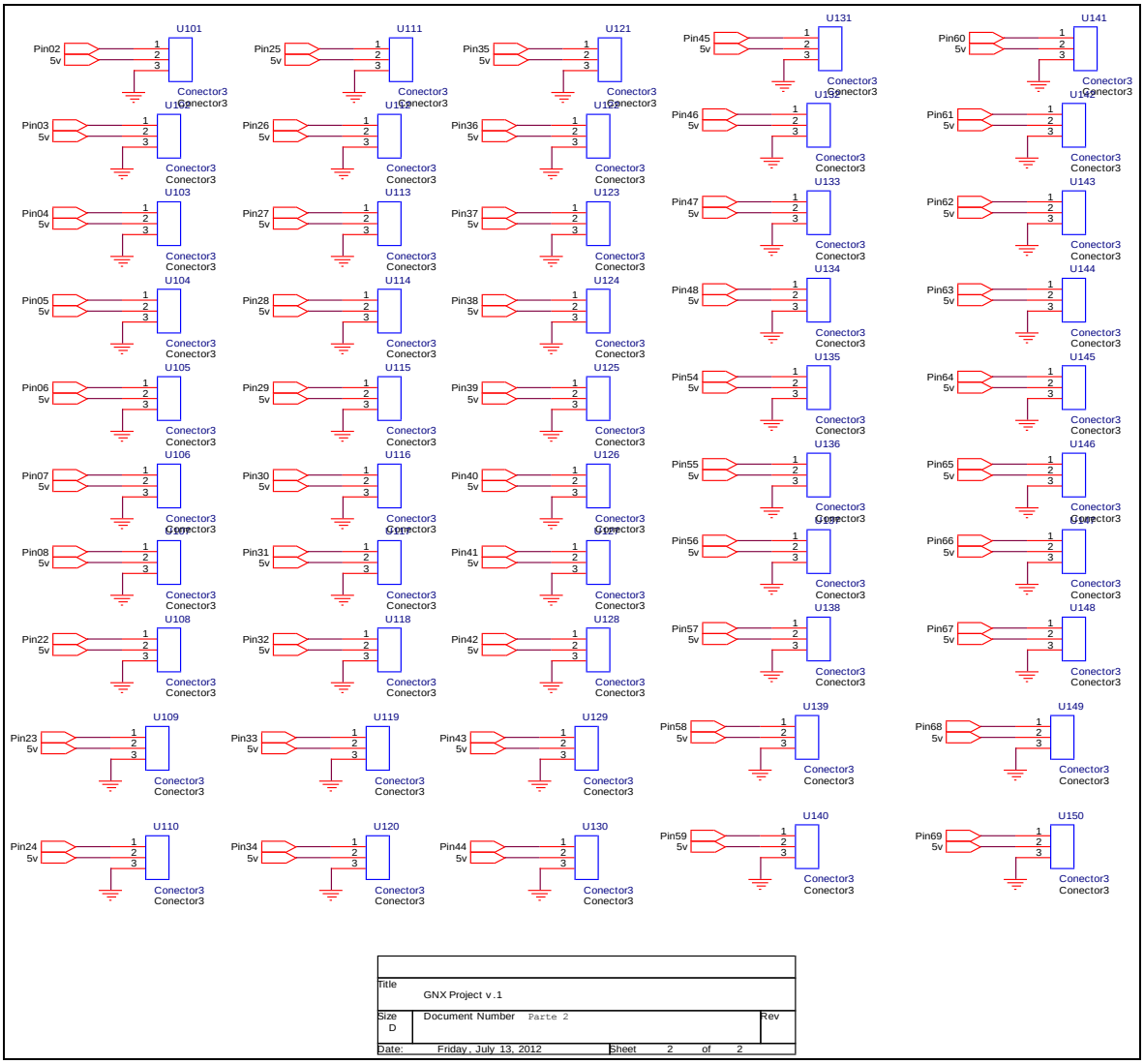


Figura 8.2: Parte 2 del esquemático.

8.2. Placa de circuito impreso (PCB)

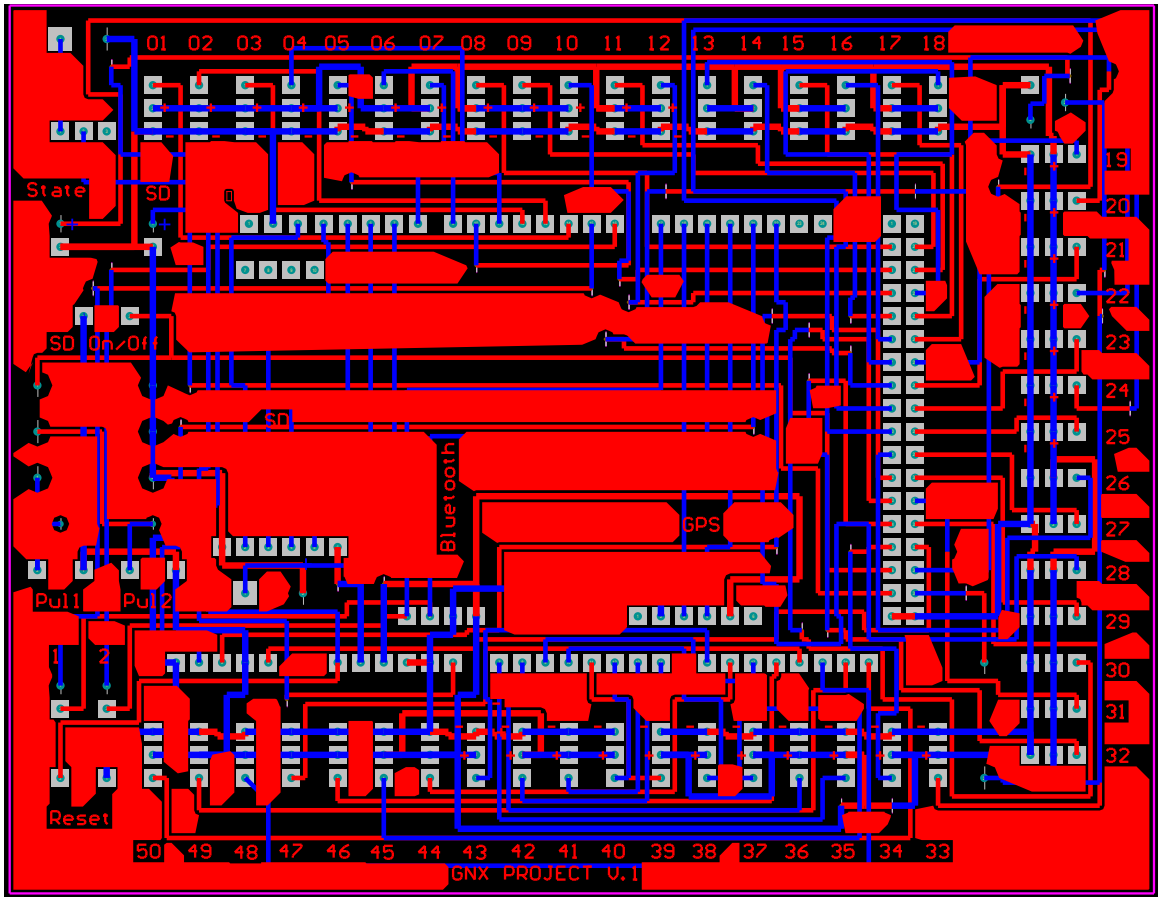


Figura 8.3: PCB del sistema GNX Project v.1.

8.3. Listado de componentes

Número	Referencia	Descripción
1	U1,2,3,4,5	Arduino Mega 2560
1	-	Módulo GPS EM-411
1	-	Transceptor Bluetooth
1	-	Tarjeta de memoria MicroSD
1	-	Lector de tarjetas MicroSD
2	J4,J5	Jumpers
1	BUZ1	Altavoz
1	U11	Conmutador de 250V y 3 A
4	P1,P2,P3,P4	Pulsadores
1	TR1	Transistor NPN BC547C
1	BATERIA2	Porta pilas 6LR61
4	R1,R2,R3,R4	Resistencias 330 Ω
50	U	Conector de 3 pines doble macho
1	U157	Conector de 5 pines doble macho
1	-	Jack de alimentación de tres terminales
1	SW1	Interruptor de 250V y 3 A

Tabla 8.1: Listado de componentes.

8.4. Pinout

Pin	Correlación	Pin	Correlación
AREF	Ninguna	36	Pin 22
GND	GNX	37	Pin 23
RESET	RESET	38	Pin 24
3.3V	3.3V	39	Pin 25
5V	5V	40	Pin 26
VIN	VIN	41	Pin 27
0-TX0	Jumper	42	Pin 28
1-RX0	Jumper	43	Pin 29
2	Pin 1	44	Pin 30
3	Pin 2	45	Pin 31
4	Pin 3	46	Pin 32
5	Pin 4	47	Pin 33
6	Pin 5	48	Pin 34
7	Pin 6	49	Pin conector Módulo adicional
8	Pin 7	50	Interruptor SD
9	Altavoz	51	Led State
10	SS SD	52	Led Check
11	MOSI SD	53	PinReset
12	MISO SD	54-A0	Pin 35
13	CLK SD	55-A1	Pin 36
14-TX3	RX Bluetooth	56-A2	Pin 37
15-RX3	TX Bluetooth	57-A3	Pin 38
16-TX2	RX GPS	58-A4	Pin 39
17-RX2	TX GPS	59-A5	Pin 40
18-TX1	RX Módulo adicional	60-A6	Pin 41
19-RX1	TX Módulo adicional	61-A7	Pin 42
20-SDA	Ninguna	62-A8	Pin 43
21-SCL	Ninguna	63-A9	Pin 44
22	Pin 8	64-A10	Pin 45
23	Pin 9	65-A11	Pin 46
24	Pin 10	66-A12	Pin 47
25	Pin 11	67-A13	Pin 48
26	Pin 12	68-A14	Pin 49
27	Pin 13	69-A15	Pin 50
28	Pin 14		
29	Pin 15		
30	Pin 16		
31	Pin 17		
32	Pin 18		
33	Pin 19		
34	Pin 20		
35	Pin 21		

Tabla 8.2: Pinout Arduino Mega 2560 y correlación con el sistema GNX Project.

8.5. Arduino Mega 2560

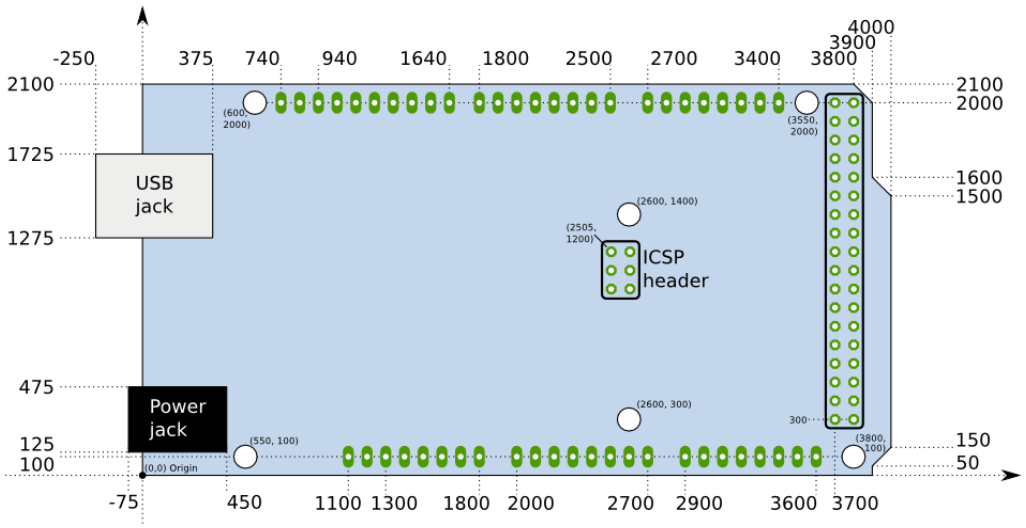


Figura 8.4: Dimensiones generales del Arduino Mega 2560.

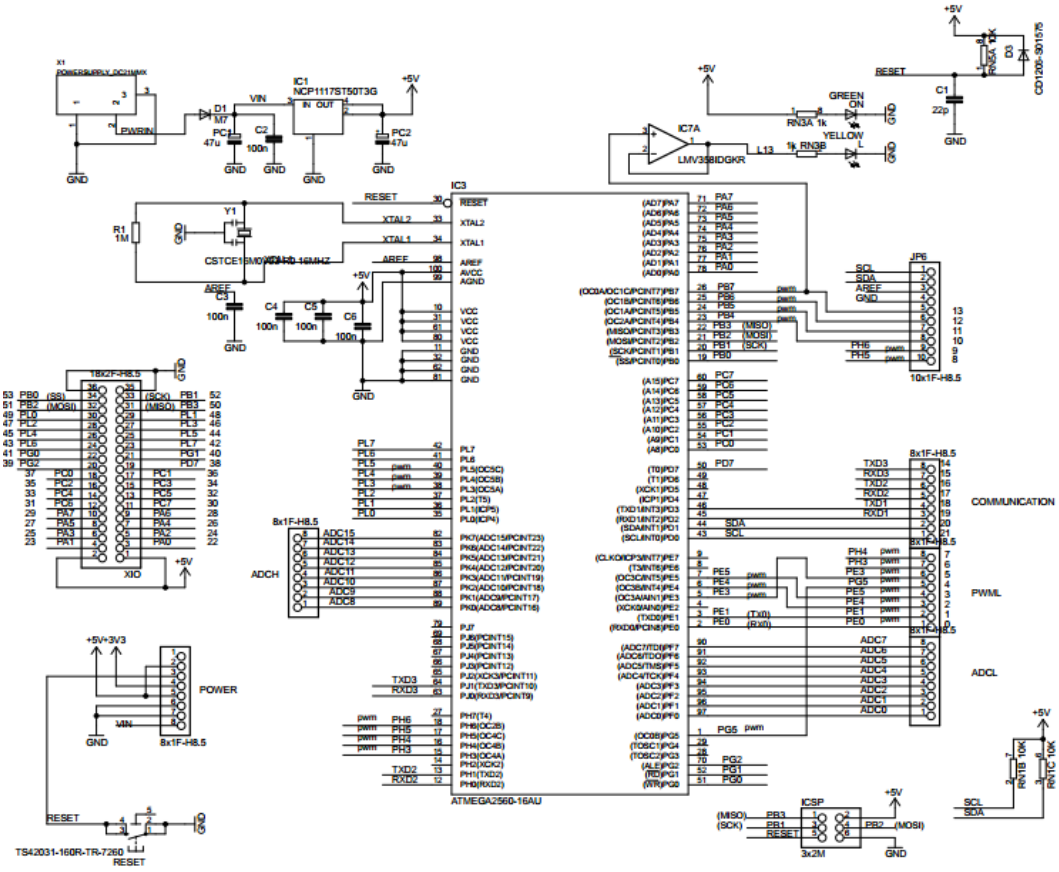


Figura 8.5: Esquemático Arduino Mega 2560.

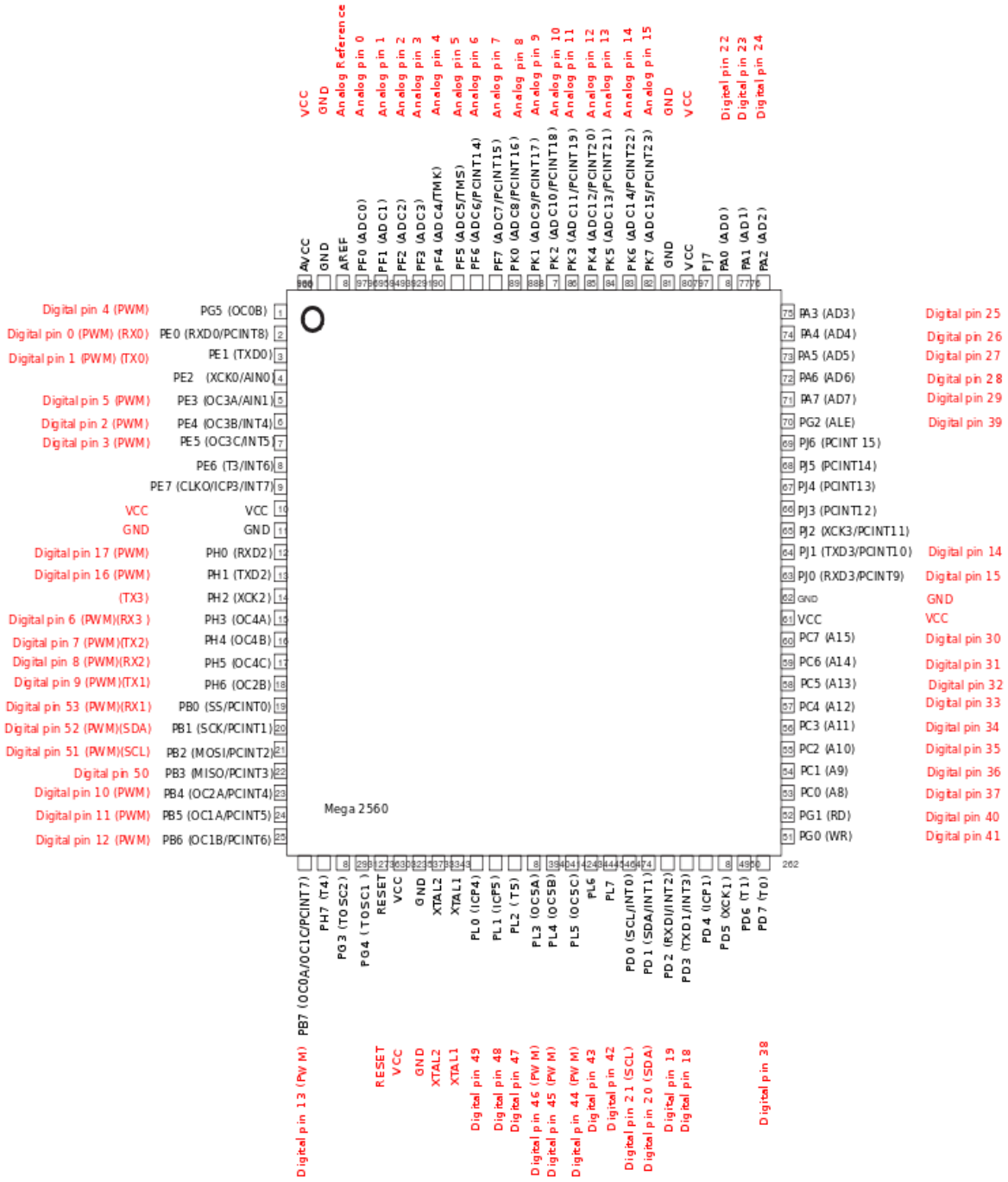


Figura 8.6: Pinout Atmel Mega 2560 y correspondencia con los pines de Arduino.

8.6. Módulo GPS EM-411

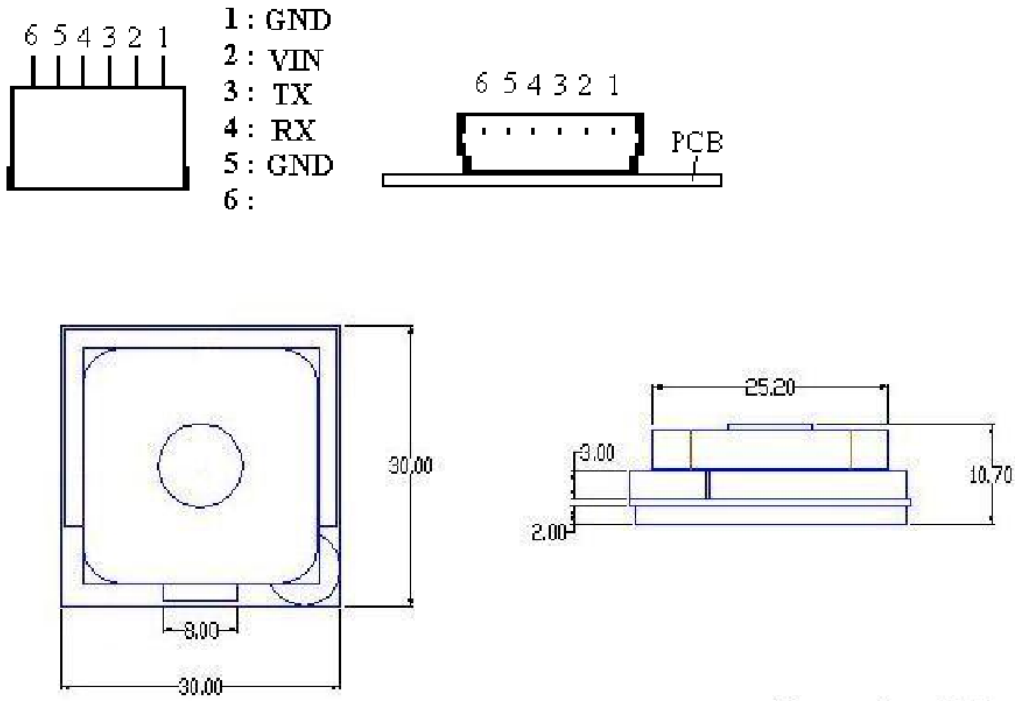


Figura 8.7: Dimensiones generales y distribución de los pines del EM-411.

8.7. Carátula del panel frontal de control

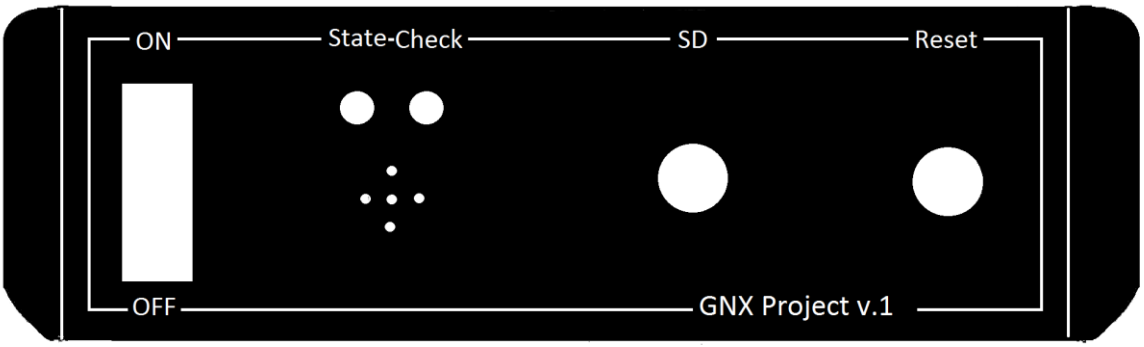


Figura 8.8: Carátula panel frontal.

III. APÉNDICES

Apéndice A

Firmware del Arduino Mega 2560

```
/*
**********************PFC*************************** *
* Proyecto: GNX Project v.1.- Módulo de control programable    *
*vía Bluetooth para aplicaciones de telemetría, telecontrol     *
*y posicionamiento vía satélite.                                *
*                                                               *
* Autor: Daniel Carreres Prieto                                 *
*                                                               *
*****************************************************
*/


//Librerias
  #include <Servo.h>  //Para controlar los Servos
  #include <EEPROM.h>
  #include <TinyGPS.h>
  #include <MemoryFree.h>
  #include <SD.h>
   TinyGPS gps;
   Servo Servo_Device;


//Literales
  #define NAcciones  150
  #define ncaracteres  15
  #define Altavoz 9


  #define TX1 14 //Puerto de envio de datos que hemos habilitado como puerto serie para el GPS
  #define RX1 15

  #define TX2 16 //Puerto de envio de datos que hemos habilitado como puerto serie para el GPS
  #define RX2 17 //Puerto de recepción de datos que hemos habilitado como puerto serie para el GPS

  #define TX3 18 //Puerto de envio de datos que hemos habilitado como puerto serie para el GSM
  #define RX3 19 //Puerto de recepción de datos que hemos habilitado como puerto serie para el GSM
```

```
#define PinModuleOthers  49
#define InterruptorSD 50
#define Led 51
#define LedDesmontarSD 52
#define PinReset 53


//Variables
  File myFile;
   int Password = EEPROM.read(7000)*100+EEPROM.read(7001); //Contraseña predeterminada .

  int RemoveSD=0;

  int Tiempo_SMS_Llamadas =1;// Min por defecto

  float Time_GPS,Time_SD=0;
 // unsigned long Timer[73]={0};//La posicion 25 se usara para los sms y llamadas y la posicion 26 como variable auxiliar en los sms
  unsigned long Temoirizaciones[NAcciones+7]={0};
 char Codigo[ncaracteres] = {0}; // Numero de caracteres que compone el codigo de instruccion.
  int Estado[NAcciones]={0};
  byte Once[NAcciones+1] = {0};
 // int Vinculado_Bluetooth=0;//Por defecto NO hay conexion
 int AuxSMS[70]={0};

  int Buffer_SRAM_Accion[6]={0};  //Donde se almacenaran los datos recuperados de la SRAM Esclava
  int Buffer_SRAM_Sensor[6]={0};  //Donde se almacenaran los datos del sensor recuperados de la SRAM Esclava 23k256
  byte Estado_SMS=0;
  char* Tipo_Pin[70]={0};//Se inicializa a un numero que NUNCA va a tener para saber que si aparece ese numero, el pin NO esta configurado
  int Aux_Buffer_Pin_Sensor=0;
  int Ejecucion=0;
  int Value_Sensor = 0;
  byte Introduciendo_Conf_Sensor=0;
  byte Inicio_Conf_Sensor=0;

  int Editar_Instruccion=0;
  int Posicion_Editar_Instruccion = 7500;
  int Error_sd =0;
  float latitude, longitude,altitude,speed;
  unsigned long course,fix_age, Hora, Fecha;
  unsigned long chars;
  int year;
  //byte month, day, hour, minutes, second, hundredths;
  //unsigned long fix_age;
```

```
  int Cabecera_GPS=0;
  int Meridiano_GPS=2;
  int i = 0;
  int k = 0;//Para los sensores
  int A=0;//Indica el bloque de accion que podemos utilizar
  int u = 0;
     int ES = 0;
     int Modo = 0;
     int Pin = 0;
     int Condicion = 0;
     int Valor1 = 0;
     int Valor2 = 0;
     //Variables usuadas para los sensores
  int val = 0;                    // Variable auxiliar
  int ultrasoundValue = 0;          // Valor del senros de ultrasonidos
  int timecount = 0;                // Contador del eco

  int previous = LOW;    // lectura anterior del pinX de entrada
  int state = LOW;       // estado actual del pin de salida
  int reading;           // lectura actual del pin de entrada

 // las siguientes variables son largas por el tiempo medido en miliseconds,
 // rápidamente se hará un número más grande que puede ser almacenado en un int.
  long time = 0;         // la ultima vez que el pin de salida fue basculado
  long debounce = 500;   // tiempo de rebote, se aumenta si la salida parpadea


//Funciones



  int Detectar_Codigo(int n) { //Detecta el código enviado por bluetooth o por GSM y
transforma de "caracteres individuales" a un numero entero
        if(Serial3.available()){
         if(i<n){
           Codigo[i]=Serial3.read();
           i++;
           Serial3.flush(); //Vaciamo el buffer de entrada de datos serie
          }
        }
       if(i==n){//Hacemos la conversión a decimal
          for(i=0; i<n;i++){
           Codigo[i]=Codigo[i]-48;}
          i=0;
       }

       ES = Codigo[4];
       Modo = Codigo[5];
       Pin = 10*Codigo[6]+Codigo[7];
       Condicion = Codigo[8];
```

```
        Valor1 = 100*Codigo[9]+10*Codigo[10]+Codigo[11];
        Valor2 = 100*Codigo[12]+10*Codigo[13]+Codigo[14];

    return Codigo[0]*1000+Codigo[1]*100+Codigo[2]*10+Codigo[3]; //Devuelve la
contraseña enviada en el codigo
  }//FIN

  void Borrar_Buffer_Codigo(){ //Borra el codigo captado por el puerto serie
       for(u=0;u<15+1;u++){
         Codigo[u] = 0;
       }
      ES = 0;
      Modo = 0;
      Pin = 0;
      Condicion = 0;
      Valor1 = 0;
      Valor2 = 0;
      Serial3.flush(); //Vaciamo el buffer de entrada de datos serie

  }

  void Acciones(int AuxES, int AuxModo, int AuxPin, int AuxCondicion, int AuxValor1,
int AuxValor2){

    if(AuxES==0){//Salida
      pinMode(AuxPin, OUTPUT);//Ponemos el PIN como salida
      switch (AuxModo){
         case 0: //Activación digital
              Tipo_Pin[AuxPin-1]="RE";
              if(Estado[AuxPin-1]==0)//Miramos la variable ESTADO que se encuentra en 0,0
de cada pin
               Estado[AuxPin-1]=1;
               else
               Estado[AuxPin-1]=0;
             digitalWrite(AuxPin,Estado[AuxPin-1]);
           break;
         case 1: //Variar corriente
             Tipo_Pin[AuxPin-1]="MP";
             if(AuxValor1>255)
             AuxValor1=255;
             Estado[AuxPin-1]=AuxValor1;
             analogWrite(AuxPin, AuxValor1);
             delay(30);
           break;
         case 2: //Servo
             Tipo_Pin[AuxPin-1]="SV";
             Servo_Device.attach(AuxPin);
             delay(15);
             Servo_Device.write(AuxValor1);
             delay(15);
```

```
          Estado[AuxPin-1]=Servo_Device.read();
          delay(15);
         break;
        case 3: //Alarma
          Tipo_Pin[AuxPin-1]="AL";
          if(Estado[AuxPin-1]==0){//Miramos la variable ESTADO que se encuentra en 0,0 de cada pin
            Estado[AuxPin-1]=1;
            tone(AuxPin, AuxValor1);
            delay(200);
          }
          else{
            Estado[AuxPin-1]=0;
            noTone(AuxPin);
            delay(200);
          }
         break;
        case 4: //Motor
          Tipo_Pin[AuxPin-1]="MO";
          if(AuxValor1>0){
            analogWrite(AuxPin, AuxValor1);
          }
          Estado[AuxPin-1]= map(AuxValor1,0,255,0,3000);//Revoluciones por minuto
          delay(30);

        break;
    }


  }
  else if(AuxES==2){//Acciones del usuario
    switch (AuxModo){
      case 0: //Configuracion, Reinicio y Borrado de EEPROM
       if(AuxPin==00){// Reinicio Placa
          Inicio_Conf_Sensor=1;//Evitamos que se ejecute cualquier accion programada
           digitalWrite(Led,HIGH);
           digitalWrite(LedDesmontarSD,HIGH);
           for (int q=0; q<10; q++){
            digitalWrite(Led,!digitalRead(Led));
            digitalWrite(LedDesmontarSD,!digitalRead(LedDesmontarSD));//Hacemos parpadear los led frontales 10 veces
            delay(200);
           }
           digitalWrite(PinReset,HIGH);//Reiniciamos la placa

        }
       else if (AuxPin==10){// Borrado de EEPROM
          digitalWrite(Led,HIGH);
          digitalWrite(LedDesmontarSD,LOW);
         Inicio_Conf_Sensor=1;//Evitamos que se ejecute cualquier accion programada
```

```
          for (int m = 0; m < NAcciones; m++){
            EEPROM.write(m, 0);
            delay(50);
            if(m%2==0){//Para reducir un poco la velocidad de parpadeo
              digitalWrite(Led,!digitalRead(Led));//Hacemos que parpadeen de forma
alternativa
              digitalWrite(LedDesmontarSD,!digitalRead(LedDesmontarSD));
            }
          }
          digitalWrite(PinReset,HIGH);//Reiniciamos la placa
        }

        //Opciones para gestionar la ordenes enviadas
         else if (AuxPin==11){// Elimiar una instrucción
            EEPROM.write(5*AuxValor1+0,99);//Introducimos todo 9 debido a que si fuera
0 la placa creería que
            delay(50);                  //ya ha terminado la lista de acciones a ejecutar.
            EEPROM.write(5*AuxValor1+1,999);
            delay(50);
            EEPROM.write(5*AuxValor1+2,99);
            delay(50);
            EEPROM.write(5*AuxValor1+3,99);
            delay(50);
            EEPROM.write(5*AuxValor1+4,99);
            delay(50);
          }
          else if (AuxPin==12){// Modificar una instruccion
            Editar_Instruccion=1;
            Posicion_Editar_Instruccion = AuxValor1;
          }




        //Configuracion
          else if (AuxPin==1){// Cambiar Contraseña
            EEPROM.write(7000,(1000*AuxCondicion+AuxValor1)/100);
            delay(50);
            EEPROM.write(7001,(1000*AuxCondicion+AuxValor1)%100);
            delay(50);
            //Serial3.print("Password cambiada con exito");
            //Serial.print(EEPROM.read(7000)*100+EEPROM.read(7001)); //Contraseña
predeterminada
            digitalWrite(PinReset,HIGH);//Reiniciamos la placa
          }

         else if (AuxPin==2){// Tiempo Looger GPS
            EEPROM.write(7002,AuxCondicion);
            delay(50);
          }
```

```
        else if (AuxPin==3){// Tiempo Escritura SD
           EEPROM.write(7003,AuxCondicion);
           delay(50);
        }

        break;

    case 1: //Enviar SMS
         Serial.println("Cabecera");
         Serial1.println("AT+CMGF=1"); //Pone el teléfono en modo Envio de SMS
         delay(1000); // Esperamos 1 seg para que el teléfono nos dé una contestación
         Serial1.print("AT+CMGS="); //Creas un nuevo mensaje
         Serial1.println("646538737"); //Envías dicho mensaje al telefono indicado
         delay(1500);
         Serial1.println("Alertas: "); //Cabecera del SMS para saber quien lo envia
         Borrar_Buffer_Codigo();
        break;

      case 2: //Realizar una llamada perdida
         Serial.println("Realizar una llamada perdida");
         Borrar_Buffer_Codigo();
        break;

      case 3: //Obtener Posicion GPS y envio por sms
        if(latitude!=0 && longitude!=0){
         digitalWrite(PinModuleOthers,HIGH);//Reiniciamos Modulo antes de trabajar
con el
         delay(1000);
         digitalWrite(PinModuleOthers,LOW);
         delay(1000);
          Serial1.println("AT+CMGF=1");
          Serial1.print("AT+CMGS=");
          Serial1.print(34,BYTE);
          Serial1.print("650532660");
          Serial1.println(34,BYTE);
          delay(500);
          Serial1.print("Lat: ");
          Serial1.print(latitude, 6);
          Serial1.println("");
          Serial1.print("Long: ");
          Serial1.print(longitude, 6);
          Serial1.println("");
          Serial1.print("Alt: ");
          Serial1.print(altitude, 6);
          Serial1.println("");
          Serial1.print("Curso: ");
          Serial1.print(course, 6);
          Serial1.println("");
          Serial1.print("Velocidad: ");
          Serial1.print(speed, 6);
```

```
            Serial1.print("Km/h");
            Serial1.println(char(26));
            Serial.println("AT*PSCPOF");//Desconecta el modulo
           }
           Borrar_Buffer_Codigo();
         break;

        case 4: //Trasnmision de datos
         if(AuxPin==0){//Telemetria
           Telemetria();
         }
         else if(AuxPin==1){//Transmision de las instrucciones programadas<<--------
          int Buffer_Trans_Instrucc[12]={0};
          Posicion_EEPROM();
          Serial.print(A);
          Serial.print(";");
          for(int re=0;re<A;re++){



            Serial.print(EEPROM.read(6*re+0));
            Serial.print(EEPROM.read(6*re+1));
            Serial.print(EEPROM.read(6*re+2));
            Serial.print(EEPROM.read(6*re+3));
            Serial.print(EEPROM.read(6*re+4));
            Serial.print(EEPROM.read(6*re+5));



            Serial.print(";");
          }

         }
          digitalWrite(Led,HIGH);
          for (int q=0; q<6; q++){
            digitalWrite(Led,!digitalRead(Led));
            delay(200);
           }

        break;

       case 5: // Logger GPS
        if(RemoveSD==0 && millis()-Temoirizaciones[NAcciones+3]
>=Time_GPS*56850){
          Temoirizaciones[NAcciones+3]=millis();
          char name[] = "GPS/GPS_00.txt";
          if(Cabecera_GPS==0){
           for (int y=0; y<100; y++){
            name[8] = y/10 + '0';
            name[9] = y%10 + '0';
```

```
            if(!SD.exists(name)){
              break;
            }
           }
        }
          if(latitude!=0 && longitude!=0){
             myFile = SD.open(name, FILE_WRITE);

            if(speed<1000 && speed>3,5){//Para evitar que grabe una velocidad y curso
erroneo que se produce en los primeros registros.
             if (myFile) {
               if (Cabecera_GPS==0){

myFile.println("Fecha,Hora,Latitud,Longitud,Altitud,Velocidad,Curso");//Cabecera
                Cabecera_GPS=1;
              }
               digitalWrite(Led,HIGH);
               digitalWrite(LedDesmontarSD,LOW);

                digitalWrite(Led,!digitalRead(Led));
                digitalWrite(LedDesmontarSD,!digitalRead(LedDesmontarSD));//Hacemos
parpadear los led frontales 10 veces
                delay(50);

               if(Fecha/10000<10)
                 myFile.print("0");
               myFile.print(Fecha/10000);
               myFile.print("-");
               if((Fecha%10000)/100<10)
                myFile.print("0");
               myFile.print((Fecha%10000)/100);
               myFile.print("-");
               myFile.print("20");
               myFile.print((Fecha%10000)%100 );
               myFile.print(",");

                digitalWrite(Led,!digitalRead(Led));
                digitalWrite(LedDesmontarSD,!digitalRead(LedDesmontarSD));//Hacemos
parpadear los led frontales 10 veces
                delay(50);

               if((Meridiano_GPS+Hora/1000000)==24){
                myFile.print("00");
               }
               else if((Meridiano_GPS+Hora/1000000)>24){
                myFile.print("0");
                myFile.print((Hora/1000000)%10);
               }
               else
                myFile.print(Meridiano_GPS+(Hora/1000000));
```

```
                myFile.print(":");
                myFile.print((Hora%1000000)/10000);
                myFile.print(":");
                myFile.print(((Hora%1000000)%10000)/100);
                myFile.print(",");

                 digitalWrite(Led,!digitalRead(Led));
                 digitalWrite(LedDesmontarSD,!digitalRead(LedDesmontarSD));//Hacemos
parpadear los led frontales 10 veces
                 delay(50);

               myFile.print(latitude,5);
               myFile.print(",");

                 digitalWrite(Led,!digitalRead(Led));
                 digitalWrite(LedDesmontarSD,!digitalRead(LedDesmontarSD));//Hacemos
parpadear los led frontales 10 veces
                 delay(50);

               myFile.print(longitude,5);
               myFile.print(",");

                 digitalWrite(Led,!digitalRead(Led));
                 digitalWrite(LedDesmontarSD,!digitalRead(LedDesmontarSD));//Hacemos
parpadear los led frontales 10 veces
                 delay(50);

               myFile.print(altitude,5);
               myFile.print(",");

                 digitalWrite(Led,!digitalRead(Led));
                 digitalWrite(LedDesmontarSD,!digitalRead(LedDesmontarSD));//Hacemos
parpadear los led frontales 10 veces
                 delay(50);

               myFile.print(speed);
               myFile.print(",");

                 digitalWrite(Led,!digitalRead(Led));
                 digitalWrite(LedDesmontarSD,!digitalRead(LedDesmontarSD));//Hacemos
parpadear los led frontales 10 veces
                 delay(50);

               myFile.print(course);
               myFile.println(",");

               digitalWrite(Led,HIGH);
               digitalWrite(LedDesmontarSD,RemoveSD);
```

```
        myFile.close();
        delay(100);
      }
     }
    }
  }
  break;
 case 6: //Looger Sensor
  int inicio_reg, fin_reg, Aux_Auxpin=0;
  if(RemoveSD==0 && (millis()-Temoirizaciones[k] >=Time_SD*56850)){
   Temoirizaciones[k]=millis();
  int Name_Pin_Telemetria=0;
  if(AuxPin==0){//Registro todos los pines
    inicio_reg=1;
    fin_reg=69;
     if(!SD.exists("SENSORES/Sensores.txt")){
       myFile = SD.open("SENSORES/Sensores.txt", FILE_WRITE);
        myFile.println("Pin\tSensor/Actuador\t\tValor\t\tFecha\t\tHora");
        myFile.println(" ");
     }
     else{
      myFile = SD.open("SENSORES/Sensores.txt", FILE_WRITE);
     }
  }
  else{//Registro un pin concreto

    char name_R[] = "SENSORES/Pin_00.txt";
    if(AuxPin<8)
      Aux_Auxpin=AuxPin-1;
    else if(AuxPin<35)
      Aux_Auxpin=AuxPin-14;
    else
       Aux_Auxpin=AuxPin-19;
    name_R[13] = Aux_Auxpin/10 + '0';
    name_R[14] =  Aux_Auxpin%10 + '0';
    inicio_reg=AuxPin+1;
    fin_reg=AuxPin+2;
     //Serial.print(name_R);
    if(!SD.exists(name_R)){
        myFile = SD.open(name_R, FILE_WRITE);
        myFile.println("Pin\tSensor/Actuador\t\tValor\t\tFecha\t\tHora");
        myFile.println(" ");
     }
     else{
      myFile = SD.open(name_R, FILE_WRITE);
     }
  }
        if (myFile) {
          //Serial.print("entro");
          for(int go=inicio_reg;go<fin_reg;go++){
```

```
                //Serial.print(Tipo_Pin[go]);
                  if(Tipo_Pin[go]!=0){//El pin esta configurado
                  Serial.println(go);

                    digitalWrite(Led,!digitalRead(Led));

digitalWrite(LedDesmontarSD,!digitalRead(LedDesmontarSD));//Hacemos parpadear los
led frontales 10 veces
                      delay(100);
                      if((go+1)!=10 && (go+1)!=11 && (go+1)!=12 && (go+1)!=13 &&
(go+1)!=TX1 && (go+1)!=RX1 && (go+1)!=TX2 && (go+1)!=RX2 && (go+1)!=TX3
&& (go+1)!=RX3 && (go+1)!=Altavoz && (go+1)!=Led &&
(go+1)!=PinModuleOthers && (go+1)!=49 && (go+1)!=50 && (go+1)!=51 &&
(go+1)!=52 && (go+1)!=53 && (go+1)!=20 && (go+1)!=21){//Pines a excluir de la
monitorizacion
                      Name_Pin_Telemetria++;
                      if(Name_Pin_Telemetria<10)
                       myFile.print("0");
                      myFile.print(Name_Pin_Telemetria);
                      myFile.print("\t");
                      if(strcmp(Tipo_Pin[go], "SR") == 0){
                        myFile.print("Sensor Resistivo");
                          myFile.print("\t");
                          myFile.print(Estado[go]);
                         }
                      else if(strcmp(Tipo_Pin[go], "PL") == 0){
                        myFile.print("Pulsador");
                          myFile.print("\t");
                          myFile.print("\t");
                          myFile.print(Estado[go]);
                          myFile.print(" veces");
                         }
                      else if(strcmp(Tipo_Pin[go], "IN") == 0){
                        myFile.print("Interruptor");
                          myFile.print("\t");
                          myFile.print("\t");
                          if (Estado[go]==1)
                            myFile.print("Cerrado");
                         else
                            myFile.print("Abierto");
                        }
                      else if(strcmp(Tipo_Pin[go], "TP") == 0){
                        myFile.print("Sen.Temperatura(LM35)");
                          myFile.print("\t");
                          myFile.print(Estado[go]);
                          myFile.print(" ºC");
                       }
                      else if(strcmp(Tipo_Pin[go], "US") == 0){
                        myFile.print("Sen.Ultrasonido");
```

```
    myFile.print("\t");
      myFile.print(Estado[go]);
      myFile.print(" Cm");
  }
  else if(strcmp(Tipo_Pin[go], "IR") == 0){
    myFile.print("Sen.Infrarrojo");
    myFile.print("\t");
      myFile.print(Estado[go]);
      myFile.print(" Cm");
  }
  else if(strcmp(Tipo_Pin[go], "RE") == 0){
    myFile.print("Rele");
    myFile.print("\t");
    myFile.print("\t");
    myFile.print("\t");
      if(Estado[go]==1)
        myFile.print("ON");
      else
        myFile.print("OFF");
  }
  else if(strcmp(Tipo_Pin[go], "MP") == 0){
    myFile.print("Modulacion de Pulso");
    myFile.print("\t");
      myFile.print(Estado[go]);
  }
  else if(strcmp(Tipo_Pin[go], "SV") == 0){
    myFile.print("Servo");
    myFile.print("\t");
    myFile.print("\t");
    myFile.print("\t");
      myFile.print(Estado[go]);
      myFile.print(" º");
  }
   else if(strcmp(Tipo_Pin[go], "AL") == 0){
    myFile.print("Altavoz");
    myFile.print("\t");
    myFile.print("\t");
    myFile.print("\t");
      myFile.print(Estado[go]);
      myFile.print(" Hz");
  }
   else if(strcmp(Tipo_Pin[go], "MO") == 0){
    myFile.print("Motor");
    myFile.print("\t");
    myFile.print("\t");
    myFile.print("\t");
      myFile.print(Estado[go]);
      myFile.print(" rpm");
  }
```

```
      else
        myFile.print("Desconocido");



    myFile.print("\t");
    myFile.print("\t");

   if(((Fecha%10000)%100)>0){//Si hay una fecha y hora valida la imprimo
     if(Fecha/10000<10)
      myFile.print("0");
    myFile.print(Fecha/10000);
    myFile.print("-");
    if((Fecha%10000)/100<10)
     myFile.print("0");
    myFile.print((Fecha%10000)/100);
    myFile.print("-");
    myFile.print("20");
    myFile.print((Fecha%10000)%100 );
    myFile.print(",");
   }
  else{
      myFile.print("  -");
      myFile.print(",");
  }


    myFile.print("\t");

 if(((Fecha%10000)%100)>0){//Si hay una fecha y hora valida la imprimo
      if((Meridiano_GPS+Hora/1000000)==24){
       myFile.print("00");
      }
      else if((Meridiano_GPS+Hora/1000000)>24){
       myFile.print("0");
       myFile.print((Hora/1000000)%10);
      }
      else
       myFile.print(Meridiano_GPS+(Hora/1000000));
      myFile.print(":");
      myFile.print((Hora%1000000)/10000);
      myFile.print(":");
      myFile.print(((Hora%1000000)%10000)/100);
      myFile.println(",");

      }
  else{
      myFile.print("\t");
      myFile.print(" -");
```

```
                    myFile.println(",");

                  }
                   }
                   digitalWrite(Led,!digitalRead(Led));

digitalWrite(LedDesmontarSD,!digitalRead(LedDesmontarSD));//Hacemos parpadear los
led frontales 10 veces
                    delay(200);
                   }
                   else{
                     if((go+1)!=10 && (go+1)!=11 && (go+1)!=12 && (go+1)!=13 &&
(go+1)!=TX1 && (go+1)!=RX1 && (go+1)!=TX2 && (go+1)!=RX2 && (go+1)!=TX3
&& (go+1)!=RX3 && (go+1)!=Altavoz && (go+1)!=Led &&
(go+1)!=PinModuleOthers && (go+1)!=49 && (go+1)!=50 && (go+1)!=51 &&
(go+1)!=52 && (go+1)!=53 && (go+1)!=20 && (go+1)!=21){//Pines a excluir de la
monitorizacion
                       Name_Pin_Telemetria++;
                     }
                   }

                }
                if(AuxPin==0)
                  myFile.println("-----------------------------------------------------------------
");

                myFile.println(" ");//Fin de secuencia
                myFile.close();
                Name_Pin_Telemetria=0;
                digitalWrite(Led,HIGH);
                digitalWrite(LedDesmontarSD,RemoveSD);
               }

           //}

          }
        break;

     }
   }

 }//Fin

 void Parpadeo(int z,int velocidad,int led){
  digitalWrite(Led,HIGH);
  digitalWrite(LedDesmontarSD,HIGH);
  for(u=0;u<z;u++){
    if(led==0){
      digitalWrite(Led,!digitalRead(Led));
      delay(velocidad);
```

```
    }
    else if(led==1){
      digitalWrite(LedDesmontarSD,!digitalRead(LedDesmontarSD));
      delay(velocidad);
    }
    else if(led==2){
      digitalWrite(Led,!digitalRead(Led));
      digitalWrite(LedDesmontarSD,!digitalRead(LedDesmontarSD));
      delay(velocidad);
    }

  }
  digitalWrite(Led,HIGH);
  digitalWrite(LedDesmontarSD,LOW);

 }//Fin


 void callphone (long number){//Func
        Serial1.println("AT+CMGF=1");
        delay(1500);
        Serial1.println("ATD"); // Inicia la llamada
        Serial1.println(number); // al numero indicado
        Serial1.println(";"); // SI NO FUNCIONA QUITALO
        delay(5000); // Tiempo que dura la llamada
        Serial1.println("ATH"); //Cuelga

 }//Fin


 int Posicion_EEPROM(){
   for(A=0;A<NAcciones;A++){
     if(EEPROM.read(6*A+0)==0 && EEPROM.read(6*A+1)==0 &&
EEPROM.read(6*A+2)==0 && EEPROM.read(6*A+3)==0 &&
EEPROM.read(6*A+4)==0 && EEPROM.read(6*A+5)==0)
       break;
   }
    Serial.print(A);
 return A;
 }
   void Telemetria(){
   int Name_Pin_Telemetria=0;
   //Serial3.print("-------  DATOS GPS  -------");
   //Serial3.print(";");//Fin de secuencia

  for(int g=1;g<69;g++){
   if(Tipo_Pin[g]!=0){//El pin esta configurado

    if((g+1)!=10 && (g+1)!=11 && (g+1)!=12 && (g+1)!=13 && (g+1)!=TX1 &&
(g+1)!=RX1 && (g+1)!=TX2 && (g+1)!=RX2 && (g+1)!=TX3 && (g+1)!=RX3 &&
```

```
(g+1)!=Altavoz && (g+1)!=Led && (g+1)!=PinModuleOthers && (g+1)!=49 &&
(g+1)!=50 && (g+1)!=51 && (g+1)!=52 && (g+1)!=53 && (g+1)!=20 &&
(g+1)!=21){//Pines a excluir de la monitorizacion
      Name_Pin_Telemetria++;
       if(Name_Pin_Telemetria<10)
         Serial3.print("0");
        Serial3.print(Name_Pin_Telemetria);

       Serial3.print(Tipo_Pin[g]);
      Serial3.print(Estado[g]);
      Serial3.print(";");//Fin de secuencia
     }

     }
     else{//Pin no configurado

     if((g+1)!=10 && (g+1)!=11 && (g+1)!=12 && (g+1)!=13 && (g+1)!=TX1 &&
(g+1)!=RX1 && (g+1)!=TX2 && (g+1)!=RX2 && (g+1)!=TX3 && (g+1)!=RX3 &&
(g+1)!=Altavoz && (g+1)!=Led && (g+1)!=PinModuleOthers && (g+1)!=49 &&
(g+1)!=50 && (g+1)!=51 && (g+1)!=52 && (g+1)!=53 && (g+1)!=20 &&
(g+1)!=21){//Pines a excluir de la monitorizacion
          Name_Pin_Telemetria++;
        if(Name_Pin_Telemetria<10)
          Serial3.print("0");

        Serial3.print(Name_Pin_Telemetria);
       Serial3.print("NN");//No esta configurado
       Serial3.print(";");//Fin de secuencia
     }

     }


   }
    Name_Pin_Telemetria=0;
    Serial3.print("Latitud: ");
    Serial3.print(latitude,5);
    Serial3.print(";");//Fin de secuencia
    Serial3.print("Longitud: ");
    Serial3.print(longitude,5);
    Serial3.print(";");//Fin de secuencia
    Serial3.print("Altitud: ");
    Serial3.print(altitude,5);
    Serial3.print(";");//Fin de secuencia
    Serial3.print("Velocidad: ");
    Serial3.print(speed);
    Serial3.print(" km/h");
    Serial3.print(";");//Fin de secuencia
    Serial3.print("Curso: ");
    Serial3.print(course);
```

```
   Serial3.print(";");//Fin de secuencia
}

 int Read_SRAM(int Save,int bloque){
   if((EEPROM.read(6*bloque+0))/10==1){

     if(Save==1){
       Buffer_SRAM_Sensor[0]=(EEPROM.read(6*bloque+0))/10;
       Buffer_SRAM_Sensor[1]=(EEPROM.read(6*bloque+0))%10;

Buffer_SRAM_Sensor[2]=(EEPROM.read(6*bloque+1))*10+((EEPROM.read(6*bloque+
2))/10);
       Buffer_SRAM_Sensor[3]=(EEPROM.read(6*bloque+2))%10;

Buffer_SRAM_Sensor[4]=(EEPROM.read(6*bloque+3))*10+(EEPROM.read(6*bloque+4
))/10;

Buffer_SRAM_Sensor[5]=((EEPROM.read(6*bloque+4))%10)*100+EEPROM.read(6*bl
oque+5);


     /*
      Serial.print(Buffer_SRAM_Sensor[0]);
      Serial.print(Buffer_SRAM_Sensor[1]);
      Serial.print(Buffer_SRAM_Sensor[2]);
      Serial.print(Buffer_SRAM_Sensor[3]);
      Serial.print(Buffer_SRAM_Sensor[4]);
      Serial.println(Buffer_SRAM_Sensor[5]);
      Serial.print("A=");
      Serial.println(A);*/

     }
       if(Buffer_SRAM_Sensor[1]==0){//Sensor Resistivo
          Tipo_Pin[Buffer_SRAM_Sensor[2]-1]="SR";
       }
       else if(Buffer_SRAM_Sensor[1]==1){//Pulsadores A/D
         Tipo_Pin[Buffer_SRAM_Sensor[2]-1]="PL";
       }
       else if(Buffer_SRAM_Sensor[1]==2){//Interruptor
         Tipo_Pin[Buffer_SRAM_Sensor[2]-1]="IN";
       }
       else if(Buffer_SRAM_Sensor[1]==3){//Sensor de Temp
         Tipo_Pin[Buffer_SRAM_Sensor[2]-1]="TP";
       }
       else if(Buffer_SRAM_Sensor[1]==4){//Ultrasonido
         Tipo_Pin[Buffer_SRAM_Sensor[2]-1]="US";
       }
      else if(Buffer_SRAM_Sensor[1]==5){//Infrarojo
          Tipo_Pin[Buffer_SRAM_Sensor[2]-1]="IR";
       }
```

```
        else
          Tipo_Pin[Buffer_SRAM_Sensor[2]-1]="??";//Desconocido

      return 1;
    }


   else if(EEPROM.read(6*bloque+0)==0 && EEPROM.read(6*bloque+1)==0 &&
EEPROM.read(6*bloque+2)==0 && EEPROM.read(6*bloque+3)==0 &&
EEPROM.read(6*bloque+4)==0 && EEPROM.read(6*bloque+5)==0){
      return 2;

    }
    else{

      if(Save==0){
        Buffer_SRAM_Accion[0]=(EEPROM.read(6*bloque+0))/10;
        Buffer_SRAM_Accion[1]=(EEPROM.read(6*bloque+0))%10;

Buffer_SRAM_Accion[2]=(EEPROM.read(6*bloque+1))*10+((EEPROM.read(6*bloque+
2))/10);
        Buffer_SRAM_Accion[3]=(EEPROM.read(6*bloque+2))%10;

Buffer_SRAM_Accion[4]=(EEPROM.read(6*bloque+3))*10+(EEPROM.read(6*bloque+
4))/10;

Buffer_SRAM_Accion[5]=((EEPROM.read(6*bloque+4))%10)*100+EEPROM.read(6*bl
oque+5);

    /*
       Serial.print(Buffer_SRAM_Accion[0]);
       Serial.print(Buffer_SRAM_Accion[1]);
       Serial.print(Buffer_SRAM_Accion[2]);
       Serial.print(Buffer_SRAM_Accion[3]);
       Serial.print(Buffer_SRAM_Accion[4]);
       Serial.println(Buffer_SRAM_Accion[5]);*/
      }
      if(Buffer_SRAM_Accion[0] !=2)//Si no es una accion de usuari que NO tiene pin
fisico

        if(Buffer_SRAM_Accion[1]==0){//Rele
          Tipo_Pin[Buffer_SRAM_Accion[2]-1]="RE";
        }
        else if(Buffer_SRAM_Accion[1]==1){//Variar corriente
          Tipo_Pin[Buffer_SRAM_Accion[2]-1]="MP";
        }
        else if(Buffer_SRAM_Accion[1]==2){//Servo
          Tipo_Pin[Buffer_SRAM_Accion[2]-1]="SV";
        }
        else if(Buffer_SRAM_Accion[1]==3){//Alarma
```

```
          Tipo_Pin[Buffer_SRAM_Accion[2]-1]="AL";
        }
        else if(Buffer_SRAM_Accion[1]==4){//Motor
          Tipo_Pin[Buffer_SRAM_Accion[2]-1]="MO";
        }
        else
         Tipo_Pin[Buffer_SRAM_Accion[2]-1]="??";//Desconocido


    return 0;
   }

 }

 void setup() {
   Serial.begin(9600); // Iniciamos el puerto serie y lo establecemos a 9600 baudios. Esto
sólo afecta al bluetooth, que es el dispositivo que se conectará al puerto serie "original" de
Arduino
      Serial3.begin(9600);  //Bluetooth
      Serial2.begin(4800);  //GPS
      Serial1.begin(4800);//OTHERS
      pinMode(10, OUTPUT);
      pinMode(PinModuleOthers, OUTPUT);
      pinMode(InterruptorSD, INPUT);
      pinMode(Led, OUTPUT);
      pinMode(LedDesmontarSD, OUTPUT);
      pinMode(PinReset, OUTPUT);
      pinMode(Altavoz, OUTPUT);

    //Temporizaciones escritura GPS
      if (EEPROM.read(7002)==0)//Automático
         Time_GPS=0;
      else if(EEPROM.read(7002)==1)//20 Segundos
         Time_GPS=0.33;
      else if(EEPROM.read(7002)==2)//40 Segundos
         Time_GPS=0.66;
      else if(EEPROM.read(7002)==3)//1 Minuto
         Time_GPS=1;
      else if(EEPROM.read(7002)==4)//5 Minuto
         Time_GPS=5;
      else if(EEPROM.read(7002)==5)//15 Minuto
         Time_GPS=15;
      else if(EEPROM.read(7002)==6)//30 Minuto
         Time_GPS=30;
      else if(EEPROM.read(7002)==7)//1 Hora
         Time_GPS=60;
      else if(EEPROM.read(7002)==8)//1,30 Horas
         Time_GPS=90;
      else if(EEPROM.read(7002)==9)//2 Horas
         Time_GPS=120;
```

```
      else
        Time_GPS=0;
      //Temporizaciones escritura SD
      if (EEPROM.read(7003)==0)//Automático
         Time_SD=0;
      else if(EEPROM.read(7003)==1)//20 Segundos
         Time_SD=0.33;
      else if(EEPROM.read(7003)==2)//40 Segundos
         Time_SD=0.66;
      else if(EEPROM.read(7003)==3)//1 Minuto
         Time_SD=1;
      else if(EEPROM.read(7003)==4)//5 Minuto
         Time_SD=5;
      else if(EEPROM.read(7003)==5)//15 Minuto
         Time_SD=15;
      else if(EEPROM.read(7003)==6)//30 Minuto
         Time_SD=30;
      else if(EEPROM.read(7003)==7)//1 Hora
         Time_SD=60;
      else if(EEPROM.read(7003)==8)//1,30 Horas
         Time_SD=90;
      else if(EEPROM.read(7003)==9)//2 Horas
         Time_SD=120;
      else
        Time_SD=0;
      /*delay(100);
        EEPROM.write(7002, 1);
        delay(100);
        EEPROM.write(7003, 1);
        delay(100);*/
     if (!SD.begin(10)){
         Error_sd=1;
         RemoveSD=1;//Para evitar que se intente grabar nada cuando se produce un error.
       }
      else{
         SD.mkdir("GPS");
         SD.mkdir("SENSORES");
         digitalWrite(Led,HIGH);
       }
     Serial.print("Ram libre=");
     Serial.println(freeMemory());
     Borrar_Buffer_Codigo();
     delay(50);

}

  void loop(){
    if(Error_sd==1 && millis()-Temoirizaciones[NAcciones+4] >=0.01*56850){
       Temoirizaciones[NAcciones+4]=millis();
      digitalWrite(LedDesmontarSD,!digitalRead(LedDesmontarSD));
```

```
  }

   if (digitalRead(InterruptorSD) == HIGH && previous == LOW && millis()+100 -
time > debounce) {//Botos para Montar/Desmontar la SD
      time = millis();
      RemoveSD=!RemoveSD;
      //Serial.println(RemoveSD);
        digitalWrite(LedDesmontarSD,RemoveSD);
      //previous=!digitalRead(InterruptorSD);
    }
    while (Serial2.available()){
          char c = Serial2.read();
          if (gps.encode(c)){
            gps.f_get_position(&latitude,&longitude,&fix_age);
            altitude = gps.f_altitude();
            course  = gps.course(); // Curso en grados
            speed = gps.f_speed_kmph(); // Velocidad en km/hr
            gps.get_datetime(&Fecha, &Hora, &fix_age);
           break;
          }
    }
   if(Detectar_Codigo(ncaracteres)==Password){

      if(ES==0 || ES==2){//Ejecutar una accion por demanda del usuario
       if(Introduciendo_Conf_Sensor==0){//No hay ninguna configuracion de sensores
          Acciones(ES, Modo, Pin, Condicion, Valor1, Valor2);
        }
       else{//Deseo programar las acciones del sensor
          Posicion_EEPROM();

           EEPROM.write(6*A+0,Codigo[4]*10+Codigo[5]);
           delay(50);
           EEPROM.write(6*A+1,Codigo[6]);
           delay(50);
           EEPROM.write(6*A+2,Codigo[7]*10+Codigo[8]);
           delay(50);
           EEPROM.write(6*A+3,Codigo[9]*10+Codigo[10]);
           delay(50);
           EEPROM.write(6*A+4,Codigo[11]*10+Codigo[12]);
           delay(50);
           EEPROM.write(6*A+5,Codigo[13]*10+Codigo[14]);
           delay(50);


       }
       delay(100);
      }
      else{//Deseo configurar un sensor
       if(Inicio_Conf_Sensor==0){//Aun NO le he dicho que acciones asocio al sensor
        Posicion_EEPROM();
```

```
            EEPROM.write(6*A+0,Codigo[4]*10+Codigo[5]);
            delay(50);
            EEPROM.write(6*A+1,Codigo[6]);
            delay(50);
            EEPROM.write(6*A+2,Codigo[7]*10+Codigo[8]);
            delay(50);
            EEPROM.write(6*A+3,Codigo[9]*10+Codigo[10]);
            delay(50);
            EEPROM.write(6*A+4,Codigo[11]*10+Codigo[12]);
            delay(50);
            EEPROM.write(6*A+5,Codigo[13]*10+Codigo[14]);
            delay(50);


          Inicio_Conf_Sensor=1;//Ya he introducido la configuracion del sensor, ahora
puedo indicar las acciones
          Introduciendo_Conf_Sensor=1;//Impido que se ejecute una accion ya sea por la
monitorizacion constante de los sensores o por accion inmediata
        }
        else if(Codigo[4]==3 && Codigo[5]==2){
            Introduciendo_Conf_Sensor=0;
            Inicio_Conf_Sensor=0;

            delay(100);
        }
       }

        Borrar_Buffer_Codigo();
    }//Fin Detectar Codigo

if(Read_SRAM(1,k)==1 && Inicio_Conf_Sensor==0){

      //Hacemos la conversion segun el tipo de sensor y escalamos el valor del mismo
        if(Buffer_SRAM_Sensor[1]==0){//Sensor Resistivo
          Value_Sensor = analogRead(Buffer_SRAM_Sensor[2]);
        }
        else if(Buffer_SRAM_Sensor[1]==1){//Pulsador
          Value_Sensor = digitalRead(Buffer_SRAM_Sensor[2]);
        }
        else if(Buffer_SRAM_Sensor[1]==2){//Interruptor
          Value_Sensor = digitalRead(Buffer_SRAM_Sensor[2]);
        }
        else if(Buffer_SRAM_Sensor[1]==3){//Sensor de temperatura LM35
          Value_Sensor = (5.0 * analogRead(Buffer_SRAM_Sensor[2]) *
100.0)/1024.0;//Fin Sensor LM35
        }
        else if(Buffer_SRAM_Sensor[1]==4){//Sensor Ultrasonico
          float duration;
          pinMode(Buffer_SRAM_Sensor[2], OUTPUT);
```

```
          digitalWrite(Buffer_SRAM_Sensor[2], LOW);
          delayMicroseconds(2);
          digitalWrite(Buffer_SRAM_Sensor[2], HIGH);
          delayMicroseconds(5);
          digitalWrite(Buffer_SRAM_Sensor[2], LOW);
          pinMode(Buffer_SRAM_Sensor[2], INPUT);
          duration = pulseIn(Buffer_SRAM_Sensor[2], HIGH);
          Value_Sensor = duration / 74 / 2;//Nos devuelve el valor en Cm
        }
        else if(Buffer_SRAM_Sensor[1]==5){//Sensor proximidad por infrarrojos sharp
          Value_Sensor = (6787/(analogRead(Buffer_SRAM_Sensor[2])-3))-4;
        }
        else if(Buffer_SRAM_Sensor[1]==6){//INDEPENDIENTE
          Value_Sensor = 0;//Le damos un valor por defecto
        }


      //Analizamos las condiciones y ejecutamos la accion
          if(Buffer_SRAM_Sensor[1]==9 && Buffer_SRAM_Sensor[2]==99 &&
Buffer_SRAM_Sensor[3]==9 && Buffer_SRAM_Sensor[4]==999 &&
Buffer_SRAM_Sensor[5]==999)//Si es una instruccion de un sensor eliminada, salto a la
siguiente
            k++;
          else if(Buffer_SRAM_Sensor[1]==1){// Si se trata de un pulsador hay que hacer
una distincion en la forma de evaluar
            //Recopilamos la informacion sobre cada uno de los pines, dicha informacion
procede del codigo de la EEPROM y del valor del sensor
             //Estado[Buffer_SRAM_Sensor[2]-1]= Estado[k+25];
            if (Value_Sensor == Buffer_SRAM_Sensor[4] && previous ==
!Buffer_SRAM_Sensor[4] && millis()+100 - time > debounce) {
              time = millis();
              Estado[Buffer_SRAM_Sensor[2]-1]++;
              if(Estado[Buffer_SRAM_Sensor[2]-1]>=Buffer_SRAM_Sensor[3]){//Miramos
si se ha pulsado el numero de veces deseado
                float auxMinutosPul;
                if(Buffer_SRAM_Sensor[5]==0)
                  auxMinutosPul =0;//Nunca
                else if(Buffer_SRAM_Sensor[5]==1)
                  auxMinutosPul=0.083;//5 seg
                else if(Buffer_SRAM_Sensor[5]==2)
                  auxMinutosPul=0.166;//10 seg
                else if(Buffer_SRAM_Sensor[5]==3)
                  auxMinutosPul=0.33;//20 seg
                else if(Buffer_SRAM_Sensor[5]==4)
                  auxMinutosPul=0.5;//30 seg
                else if(Buffer_SRAM_Sensor[5]==5)
                  auxMinutosPul=1;//1 Min
                else if(Buffer_SRAM_Sensor[5]==6)
                  auxMinutosPul=5;//5 Min
                else if(Buffer_SRAM_Sensor[5]==7)
```

```
                auxMinutosPul=10;//10 Min
               else if(Buffer_SRAM_Sensor[5]==8)
                auxMinutosPul=15;//15 Min
               else if(Buffer_SRAM_Sensor[5]==9)
                auxMinutosPul=30;//30 Min
              if (millis()-Temoirizaciones[k] >=auxMinutosPul*56850){
               Temoirizaciones[k]=millis();
               Estado[Buffer_SRAM_Sensor[2]-1]=0;//Reiniciamos conteo pasado un
tiempo

              }
              else
               Ejecucion=1;
            }
            k++;
           }

           else{

             k++;
           }

           previous = Value_Sensor;
         }
         else if(Buffer_SRAM_Sensor[1]==2){//Si es un interruptor
          if(Value_Sensor == Buffer_SRAM_Sensor[3]){//Si esta abierto o cerrado
            if(Value_Sensor==HIGH)
             Estado[Buffer_SRAM_Sensor[2]-1]=1;
            else
              Estado[Buffer_SRAM_Sensor[2]-1]=0;
            Ejecucion=1;
            k++;

          }
          else{
           if(Value_Sensor==HIGH)
             Estado[Buffer_SRAM_Sensor[2]-1]=1;
           else
              Estado[Buffer_SRAM_Sensor[2]-1]=0;
           Ejecucion=2;
           k++;
          }
         }
          else if(Buffer_SRAM_Sensor[1]==6){//Si es una accion Independiente de los
sensores
           Ejecucion=1;
           k++;
         }
          else if(Buffer_SRAM_Sensor[3]==4){//Entrada->Salida
           Ejecucion=1;// Hay que reactivarlo
```

```
            k++;
          }
         else{
            //Recopilamos la informacion sobre cada uno de los pines, dicha informacion
procede del codigo de la EEPROM y del valor del sensor
             Estado[Buffer_SRAM_Sensor[2]-1]=Value_Sensor;

            if(Buffer_SRAM_Sensor[3]==0){//Entrada = Valor1
              if(Value_Sensor==Buffer_SRAM_Sensor[4]){
                Ejecucion=1;
                k++;
              }
              else{
                Ejecucion=2;
                k++;
              }
            }
            //-------------------------------
            else if(Buffer_SRAM_Sensor[3]==1){//Entrada > Valor1
             if(Value_Sensor>Buffer_SRAM_Sensor[4]){
                Ejecucion=1;
                k++;
              }
              else{
                Ejecucion=2;
                k++;
              }
            }
            //-------------------------------

           else if(Buffer_SRAM_Sensor[3]==2){//Entrada < Valor1

              if(Value_Sensor<Buffer_SRAM_Sensor[4]){
                Ejecucion=1;
                k++;
              }
              else{
                Ejecucion=2;
                k++;
              }
            }
            //-------------------------------
             else if(Buffer_SRAM_Sensor[3]==3){//Valor1< Entrada < Valor2
              if(Buffer_SRAM_Sensor[4]<Value_Sensor &&
Value_Sensor<Buffer_SRAM_Sensor[5]){
                Ejecucion=1;
                k++;
              }
              else{
                Ejecucion=2;
```

```
              k++;
             }
            }
            //------------------------------
        }
    }
    else{

     k++;
    }

    if(Ejecucion==1){
      while(Read_SRAM(0,k)==0){
       if(Buffer_SRAM_Accion[1]==9 && Buffer_SRAM_Accion[2]==99 &&
Buffer_SRAM_Accion[3]==9 && Buffer_SRAM_Accion[4]==999 &&
Buffer_SRAM_Accion[5]==999){//Si es una instruccion de una accion eliminada, salto a
la siguiente
          k++;
       }
       else{
       if(Buffer_SRAM_Accion[0]==2){//Si es una accion avanzada quiero que se ejecute
mientras se siga cumpliendo la accion pero espacio x minutos
          if(Buffer_SRAM_Accion[1]==1){//Si es un SMS
           if(Estado_SMS==0){//Enviamos cabecera
             if(Temoirizaciones[NAcciones+2]==0 || millis()-
Temoirizaciones[NAcciones+2] >=Tiempo_SMS_Llamadas*56850){
              Temoirizaciones[NAcciones+2]=millis();//Guardo cuando he enviado el sms o
la llamada
              Acciones(Buffer_SRAM_Accion[0], Buffer_SRAM_Accion[1],
Buffer_SRAM_Accion[2], Buffer_SRAM_Accion[3], Buffer_SRAM_Accion[4],
Buffer_SRAM_Accion[5]);
              Estado_SMS=1;
             }

          }
           if(Estado_SMS==1 && Once[k]==0){//Ya he enviado la cabecera, ahora
enviamos el contenido del sms
            if(Buffer_SRAM_Sensor[1]==0){//Sensor Resistivo
              Serial1.print("-Sen.Res pin ");
            }
            else if(Buffer_SRAM_Sensor[1]==1){//Pulsador
              Serial1.print("-Pulsador pin ");
            }
             else if(Buffer_SRAM_Sensor[1]==2){//Interruptor
              Serial1.print("-Interruptor pin ");
            }
            else if(Buffer_SRAM_Sensor[1]==3){//LM35
              Serial1.print("-Sen.Temp pin ");
            }
            else if(Buffer_SRAM_Sensor[1]==4){//Ultrasonido
```

```
                Serial1.print("-Sen.Sonico pin ");
              }
              else if(Buffer_SRAM_Sensor[1]==5){//Infrarrojo
                Serial1.print("-Sen.Ir pin ");
              }
              Serial1.print(Buffer_SRAM_Sensor[2]);
              Serial1.print(": ");
              Serial1.println(Value_Sensor);
              AuxSMS[AuxSMS[0]+1]=k;//Almacenamos el valor de K de esa accion
bloqueada a fin de luego desbloquearla
                AuxSMS[0]++;


            Once[k]=1;
          }


              if(millis()-Temoirizaciones[NAcciones+1] >=0.1*56850 &&
Estado_SMS==1){//Esperamos que pasen 5seg
               Temoirizaciones[NAcciones+1]=millis();
               //Timer[25]=0;
               Serial.println("FIN");
               for(int r=1;r<AuxSMS[0];r++){
                Once[AuxSMS[r]]=0;
                Once[3]=0;

               }
               AuxSMS[0]=0;
               Estado_SMS=0;
              }

         }//fin sms
        else {
          Acciones(Buffer_SRAM_Accion[0], Buffer_SRAM_Accion[1],
Buffer_SRAM_Accion[2], Buffer_SRAM_Accion[3], Buffer_SRAM_Accion[4],
Buffer_SRAM_Accion[5]);
        }
      }
        else if(Buffer_SRAM_Accion[0]==0 && Buffer_SRAM_Accion[1]==0 ){
           if(Once[k]==0){//Quiero que se ejecuta 1 sola vez
           Once[k]=1;
           if(digitalRead(Buffer_SRAM_Accion[2])==LOW)
             Acciones(Buffer_SRAM_Accion[0], Buffer_SRAM_Accion[1],
Buffer_SRAM_Accion[2], Buffer_SRAM_Accion[3], Buffer_SRAM_Accion[4],
Buffer_SRAM_Accion[5]);
          }
        }
       else if(Buffer_SRAM_Accion[0]==0 && (Buffer_SRAM_Accion[1]!=0)){//Si no es
un rele
         int vMin,vMax=0;
```

```
        if(Buffer_SRAM_Sensor[3]==4){//Deseo que la salida=entrada
          if(Buffer_SRAM_Sensor[1]==0){//El sensor es un sensor resistivo
            vMin=0;
            vMax=1023;
          }
          else if(Buffer_SRAM_Sensor[1]==3){//El sensor es de Temperatura
            vMin=-55;
            vMax=105;
          }
          else if(Buffer_SRAM_Sensor[1]==4){//El sensor es un sensor Ultrasonido
            vMin=15;
            vMax=645;
          }
          else if(Buffer_SRAM_Sensor[1]==5){//El sensor es un sensor Infrarrojo
            vMin=5;
            vMax=150;
          }
          //Ahora ajustamos la salida segun el lo que tenemos conectado
            if(Buffer_SRAM_Accion[1]==1)//Modular pulso
              Buffer_SRAM_Accion[4]=map(Value_Sensor, vMin, vMax, 0, 255);
            else if(Buffer_SRAM_Accion[1]==2)//Es un servo
              Buffer_SRAM_Accion[4]=map(Value_Sensor, vMin, vMax, 0, 180);
            else if(Buffer_SRAM_Accion[1]==3)//Alarma
              Buffer_SRAM_Accion[4]=map(Value_Sensor, vMin, vMax, 20, 900);
            else if(Buffer_SRAM_Accion[1]==4)//Motor
              Buffer_SRAM_Accion[4]=map(Value_Sensor, vMin, vMax, 0, 255);


           if(millis()-Temoirizaciones[k] >=0.009*56850){
            Temoirizaciones[k]=millis();
            Acciones(Buffer_SRAM_Accion[0], Buffer_SRAM_Accion[1],
Buffer_SRAM_Accion[2], Buffer_SRAM_Accion[3], Buffer_SRAM_Accion[4],
Buffer_SRAM_Accion[5]);
           }
        }


        if(Once[k]==0 && Buffer_SRAM_Sensor[3]!=4){//Quiero que se ejecuta 1 sola
vez
          Once[k]=1;
          Acciones(Buffer_SRAM_Accion[0], Buffer_SRAM_Accion[1],
Buffer_SRAM_Accion[2], Buffer_SRAM_Accion[3], Buffer_SRAM_Accion[4],
Buffer_SRAM_Accion[5]);
         }
       }

      k++;
    }
    }
    Ejecucion=0;
```

```
    }
    if(Ejecucion==2){
      while(Read_SRAM(0,k)==0){
        if(Buffer_SRAM_Accion[1]==9 && Buffer_SRAM_Accion[2]==99 &&
Buffer_SRAM_Accion[3]==9 && Buffer_SRAM_Accion[4]==999 &&
Buffer_SRAM_Accion[5]==999){//Si es una instruccion de una accion eliminada, salto a
la siguiente
            k++;
        }
        else{

         float auxMinutosAcc;
            if(Buffer_SRAM_Accion[5]==0)
            auxMinutosAcc =0;//Apagado automatico
            else if(Buffer_SRAM_Accion[5]==1)
              auxMinutosAcc=1;//1 Min
            else if(Buffer_SRAM_Accion[5]==2)
              auxMinutosAcc=5;//5 Min
            else if(Buffer_SRAM_Accion[5]==3)
              auxMinutosAcc=10;//10 Min
            else if(Buffer_SRAM_Accion[5]==4)
              auxMinutosAcc=15;//15 Min
            else if(Buffer_SRAM_Accion[5]==5)
              auxMinutosAcc=20;//20 Min
            else if(Buffer_SRAM_Accion[5]==6)
              auxMinutosAcc=30;//30 Min
            else if(Buffer_SRAM_Accion[5]==7)
              auxMinutosAcc=60;//60 Min
            else if(Buffer_SRAM_Accion[5]==8)
              auxMinutosAcc=90;//90 Min
            else if(Buffer_SRAM_Accion[5]==9)
              auxMinutosAcc=120;//120 Min
          if (millis()-Temoirizaciones[k] >=auxMinutosAcc*56850){
            Temoirizaciones[k]=millis();

            if(Buffer_SRAM_Accion[0]==0 && Buffer_SRAM_Accion[1]==0  &&
Once[k]==1){//Si es un rele y esta Encendia
               Once[k]=0;
                Acciones(Buffer_SRAM_Accion[0], Buffer_SRAM_Accion[1],
Buffer_SRAM_Accion[2], Buffer_SRAM_Accion[3], Buffer_SRAM_Accion[4],
Buffer_SRAM_Accion[5]);
            }
            else if(Buffer_SRAM_Accion[0]==0 && Buffer_SRAM_Accion[1]==3 &&
Once[k]==1){//Si es un alarma
              delay(50);
              Once[k]=0;
              Acciones(Buffer_SRAM_Accion[0], Buffer_SRAM_Accion[1],
Buffer_SRAM_Accion[2], Buffer_SRAM_Accion[3], Buffer_SRAM_Accion[4],
Buffer_SRAM_Accion[5]);
            }
```

```
          else if(Buffer_SRAM_Accion[0]==0 && Buffer_SRAM_Accion[1]==1 &&
Once[k]==1){
            Once[k]=0;
            digitalWrite(Buffer_SRAM_Accion[2],LOW);
           }

         }
          k++;

     }
   }
     Ejecucion=0;
   }


    if(k>NAcciones)
    k=0;


  }
```

Apéndice B

Software de la aplicación para Smartphone

```
#Librerias

import os, appuifw, e32, globalui, audio, time
import btsocket as socket
import messaging
import os.path
from graphics import *

#Variables
password ="1234"                #Pass del sistema
Sensores = [""]         #Array donde almacenar los datos de la telemetria
Informacion_Sensor=["Sin configurar","","","","",""]
global Bt_Start
Bt_Start = 0                    #Estado del Bluetooth
Once=0
Once_2=0
i=0                                     #Para la funcion de telemetria
u=0
j=1
l=0
s=2
z=0
Longitud_Elmento=0
AuxTelemetria=""
Aux_pin_telemetria=0
error_flag=0#Indica que se ha producido un error
Aux_p=0
Aux_m=0
Aux_v=0
Aux_a=0
Aux_t=0
Aux_apagado=0
Aux_J=0
aux1 =""
aux2 =""
auxCode=0
grados=""
Pin_Code=0
Pin_Name_Aux=0
Mode_Code =0
Condicion = 0
Value1 = 0
Value2 = 0
ValueAux = 0
```

```
Code =0
#Funciones

#-------------------------------------------------------------------------------------------------------
#                                                  Funciones de comunicacion
#-------------------------------------------------------------------------------------------------------
def choose_service(services):          #Funcion para seleccionar el puerto de comunicacion
        names = []
        channels = []
        for name, channel in services.items():
                names.append(name)
                channels.append(channel)
        index = appuifw.popup_menu(names, u"Seleccione pueto:")
        return channels[index]

#                                                #-----------------#-----------------#

def connect():#Funcion para conectar el telefono con la placa
        #try:
                if Bt_Start == 0:
                        global sock
                        address, services = socket.bt_discover()
                        channel = choose_service(services)
                        sock = socket.socket(socket.AF_BT, socket.SOCK_STREAM)
                        sock.connect((address, channel))
                        global Bt_Start
                        Bt_Start = 1
                        #Sent_PSerie(password+"20130001000")
                        appuifw.note(u"Conexion realizada con exito!" , "conf")
                        #Sent_PSerie("12340020300000")    #Codigo a enviar para indicar a
la placa que  existe comunicacion bluetooth y que empiece a trasmitir datos.
                        e32.ao_sleep(1)
                else:
                        appuifw.note(u"Ya se ha establecido comunicacion", "error")
        #except:
        #       if Bt_Start==0:
        #               appuifw.note(u"Ha ocurrido un error durante la conexion", "error")
        #       else:
        #               appuifw.note(u"Ya se ha establecido comunicacion", "error")

#                                                #-----------------#-----------------#

def Leer_PSerie():     #Lee 1 caracter recibido por el puerto serie
        if Bt_Start ==1:
                global sock
                data = sock.recv(1)
                global AuxTelemetria
                AuxTelemetria = str(data)
                return str(data)
```

```
        else:
                appuifw.note(u"Es necesario establecer comunicacion primero", "error")
                return ""

#                                       #----------------#----------------#

def Sent_PSerie(txt): #Enviar por puerto serie un codigo
        if Bt_Start ==1:
                global sock
                sock.send(txt)
        else:
                appuifw.note(u"Es necesario establecer comunicacion primero", "error")

#                                       #----------------#----------------#

#-------------------------------------------------------------------------------------------------
#                                               Funciones del menu "Opciones"
#-------------------------------------------------------------------------------------------------
def Acercade():
        globalui.global_msg_query(u"GNX Project\nEste proyecto ha sido desarrollado por
Daniel Carreres Prieto para su proyecto final de carrera.", u"Acerca de", 0)

#                                       #----------------#----------------#


def exit_key_handler():
        if globalui.global_query(u"Seguro que desea salir?") == True:
                #if Bt_Start == 1:
                #       Sent_PSerie(password+"20130000000")
                appuifw.app.set_tabs([], None)
                app_lock.signal()
#                                       #----------------#----------------#

#-------------------------------------------------------------------------------------------------
#                                               Funciones
#-------------------------------------------------------------------------------------------------

Opc_favoritos = [(u"Agregar nuevo favorito"),(u"Borrar Todas los favoritos"),(u"

                                        Volver")]
Acc_favoritos = []
BackUp_Name_Favorito=""
saved_Programar_Favoritos = False
def Programar_Favoritos(p,m,v,name):
                                global l

                                Pin_F = [u'Sin seleccionar',u'1 (Digital PWM)',u'2 (Digital
PWM)', u'3 (Digital PWM)', u'4 (Digital PWM)', u'5 (Digital PWM)', u'6 (Digital PWM)',
u'7 (Digital PWM)', u'8 (Digital)', u'9 (Digital)', u'10 (Digital)', u'11 (Digital)', u'12
(Digital)', u'13 (Digital)', u'14 (Digital)', u'15 (Digital)', u'16 (Digital)', u'17 (Digital)', u'18
```

```
(Digital)', u'19 (Digital)', u'20 (Digital)', u'21 (Digital)', u'22 (Digital)', u'23 (Digital)', u'24
(Digital)', u'25 (Digital)', u'26 (Digital)', u'27 (Digital)', u'28 (Digital)', u'29 (Digital)', u'30
(Digital)', u'31 (Digital)', u'32 (Digital)', u'33 (Digital)', u'34 (Digital)', u'35
(Digital/Analogico)', u'36 (Digital/Analogico)', u'37 (Digital/Analogico)', u'38
(Digital/Analogico)', u'39 (Digital/Analogico)', u'40 (Digital/Analogico)', u'41
(Digital/Analogico)', u'42 (Digital/Analogico)', u'43 (Digital/Analogico)', u'44
(Digital/Analogico)', u'45 (Digital/Analogico)', u'46 (Digital/Analogico)', u'47
(Digital/Analogico)', u'48 (Digital/Analogico)', u'49 (Digital/Analogico)', u'50
(Digital/Analogico)']
                        Modo_F =[u'Rele', u'Modular Pulso',
u'Servo',u'Alarma',u'Motor',u'Altavoz placa']
                        Data_F = [(u"Pin", 'combo', (Pin_F,p)),
                                  (u"Salida", 'combo', (Modo_F,m)),
                                  (u"Valor",'number',v)]


                        def salvado_Programar_Favoritos(arg):
                            global saved_Programar_Favoritos
                            saved_Programar_Favoritos = True
                            return True

                        flags_Conf_Favorito = appuifw.FFormEditModeOnly
                        PC_Favoritp = appuifw.Form( Data_F, flags_Conf_Favorito)
                        PC_Favoritp.save_hook = salvado_Programar_Favoritos
                        PC_Favoritp.execute()

                        if saved_Programar_Favoritos == True:
                            if not PC_Favoritp[0][2][1] == 0:#Sin seleccionar
                                    if PC_Favoritp[0][2][1] <8:
                                        Pin_Code =
"0"+str(PC_Favoritp[0][2][1]+1)
                                        Pin_Name_Aux
=str(PC_Favoritp[0][2][1])
                                    elif PC_Favoritp[0][2][1] <35:
                                        Pin_Code =
str(PC_Favoritp[0][2][1]+14)
                                        Pin_Name_Aux
=str(PC_Favoritp[0][2][1])
                                    else:
                                        Pin_Code =
str(PC_Favoritp[0][2][1]+19)
                                        Pin_Name_Aux
=str(PC_Favoritp[0][2][1])

                                    if PC_Favoritp[1][2][1]==0:#Rele
                                        Mode_Code = "0"

                                    elif
PC_Favoritp[1][2][1]==1:#Modular Ancho de Pulso
                                        Mode_Code = "1"
```

```
                                    elif PC_Favoritp[1][2][1]==2:#Servo
                                            Mode_Code = "2"

                                    elif PC_Favoritp[1][2][1]==3:#Alarma
                                            Mode_Code = "3"

                                    elif PC_Favoritp[1][2][1]==4:#Motor
                                            Mode_Code = "4"


                                    Value1 = str(PC_Favoritp[2][2])
                                    if PC_Favoritp[2][2] < 10:
                                            aux1="00"
                                    elif PC_Favoritp[2][2] < 100:
                                            aux1="0"
                                    elif PC_Favoritp[2][2] > 99:
                                            aux1=""
                                    if PC_Favoritp[1][2][1]==1 or
PC_Favoritp[1][2][1]==4:#Si es Variacion de corriente o Motor
                                            if PC_Favoritp[2][2] > 255:
                                                    aux1=""
                                                    Value1 = "255"
                                    elif PC_Favoritp[1][2][1]==2:#Servo
                                            if PC_Favoritp[2][2] > 180:
                                                    aux1=""
                                                    Value1 = "180"
                                    elif PC_Favoritp[1][2][1]==3:#Alarma
                                            if PC_Favoritp[2][2] > 999:
                                                    aux1=""
                                                    Value1 = "999"
                                    if not
PC_Favoritp[1][2][1]==5:#Altavoz placa
                                                    Code =
"0"+Mode_Code+Pin_Code + "0" +aux1+Value1+"000"
                                    else:
                                            Code="03090700000"
                                    Opc_favoritos.insert(l, (u""+name))
                                    Acc_favoritos.insert(l,(u""+Code))
                                    aux1=0
                                    l=l+1

                            else:
                                    if PC_Favoritp[1][2][1]==5:#Altavoz placa
                                            Code="03090700000"
                                            Opc_favoritos.insert(l, (u""+name))
                                            Acc_favoritos.insert(l,(u""+Code))
                                            aux1=0
                                            l=l+1
                                    else:
```

```
                                        appuifw.note(u"Debe especificar el
pin!", "error")

        #Programar_Favoritos(0,0,0,Name_Faforito_Op)


config={}
def Favoritos():
        global l
        global Once
        CONFIG_DIR="e:/"
        CONFIG_FILE="Favoritos.txt"
        if os.path.isfile(CONFIG_DIR+CONFIG_FILE) == True and Once==0:
                while l > 0:
                        del Opc_favoritos[0]
                        del Acc_favoritos[0]
                        l=l-1
                TXT_Favoritos=open(CONFIG_DIR+CONFIG_FILE,'rt')
                content = TXT_Favoritos.read()
                config=eval(content)
                TXT_Favoritos.close()
                for x in range(int(config.get('N_Acciones',''))):
                        Opc_favoritos.insert(x, (u"> "+config.get('Name_'+str(x))))
                        Acc_favoritos.insert(x, (u""+config.get('Acc_'+str(x))))
                l=int(config.get('N_Acciones',''))
                Once=1


        elif l>0:#Si hay alguna accion programada, guardamos todo en el telefono

                CONFIG_FILE=os.path.join(CONFIG_DIR,'Favoritos.txt')
                config={}
                for x in range(l):
                        config["Name_"+str(x)]= Opc_favoritos[x]
                        config["Acc_"+str(x)]= Acc_favoritos[x]
                        #time.sleep(1)
                config["N_Acciones"]= str(l)
                TXT_Favoritos=open(CONFIG_FILE,'wt')
                TXT_Favoritos.write(repr(config))
                TXT_Favoritos.close()
                #config={}
        def Acciones_Favoritos():
                if Menu_Favorito.current() == l:#Agregar nuevo favorito
                        Name_Faforito_Op = appuifw.query(u"Seleccione un nombre",
'text')
                        if Name_Faforito_Op:
                                Programar_Favoritos(0,0,0,Name_Faforito_Op)
                                Favoritos()

                elif Menu_Favorito.current()==l+1:#Borrar Todas los favoritos
```

```
                        global l
                        global Once
                        if not l==0:
                                if globalui.global_query(u"Desea eliminar todos los
favoritos") == True:
                                        while l > 0:
                                                del Opc_favoritos[0]
                                                del Acc_favoritos[0]
                                                l=l-1
                                        os.remove(CONFIG_DIR+CONFIG_FILE)
                                        Once=0
                                        Favoritos()
                                        appuifw.note(u"Todas las acciones ha sido borradas
con exito", "conf")
                        else:
                                appuifw.note(u"No existen acciones programadas", "error")
                elif Menu_Favorito.current()==l+2:#Volver
                        quit_favorito()
                else:#Deso enviar algo
                        if int(Acc_favoritos[Menu_Favorito.current()][1])==1:#Si
Modul.Pulso
                                Etiquetas_Op_Av = [u"Cambiar Ciclo de Trabajo",
u"Cambiar nombre", u"Editar Orden", u"Eliminar"]
                        elif int(Acc_favoritos[Menu_Favorito.current()][1])==2:#Si es un
servo
                                Etiquetas_Op_Av = [u"Cambiar Angulo", u"Cambiar
nombre", u"Editar Orden", u"Eliminar"]
                        elif int(Acc_favoritos[Menu_Favorito.current()][1])==3:#Si es una
Alarma
                                Etiquetas_Op_Av = [u"Cambiar frecuencia", u"Cambiar
nombre", u"Editar Orden", u"Eliminar"]
                        elif int(Acc_favoritos[Menu_Favorito.current()][1])==4:#Si es un
Motor
                                Etiquetas_Op_Av = [u"Cambiar velocidad", u"Cambiar
nombre", u"Editar Orden", u"Eliminar"]
                        else:
                                Etiquetas_Op_Av = [u"Enviar orden", u"Cambiar nombre",
u"Editar Orden", u"Eliminar"]
                        Op_Av = appuifw.popup_menu(Etiquetas_Op_Av, u"Seleccione una
accion:")
                        #if not int(Acc_favoritos[Menu_Favorito.current()][1])==1 or not
int(Acc_favoritos[Menu_Favorito.current()][1])==2 or not
int(Acc_favoritos[Menu_Favorito.current()][1])==3 or not
int(Acc_favoritos[Menu_Favorito.current()][1])==4:

                        if Op_Av == 0:#Enviar orden
                                if Bt_Start == 0:#Bluetooth no activa
                                        appuifw.note(u"Es necesario establecer comunicacion
primero", "error")
                                else:
```

```
                                        if
int(Acc_favoritos[Menu_Favorito.current()][1])==1:#Si Modul.Pulso o motor
                                                Nuevo_Valor_periferico =
appuifw.query(u"Seleccione ciclo de trabajo", 'number')
                                                if Nuevo_Valor_periferico:
                                                        if int(Nuevo_Valor_periferico)<10:
                                                                aux_con="00"
                                                        elif int(Nuevo_Valor_periferico)<100:
                                                                aux_con="0"
                                                        elif int(Nuevo_Valor_periferico)>255:
                                                                Nuevo_Valor_periferico=255
                                                                aux_con=""


        Sent_PSerie(password+"0"+(Acc_favoritos[Menu_Favorito.current()][1:3])+"0"+a
ux_con+str(Nuevo_Valor_periferico)+"000")#Enviamos orden a la placa
                                        elif
int(Acc_favoritos[Menu_Favorito.current()][1])==2:#Si es un servo
                                                Nuevo_Valor_periferico =
appuifw.query(u"Seleccione angulo", 'number')
                                                if Nuevo_Valor_periferico:
                                                        if int(Nuevo_Valor_periferico)<10:
                                                                aux_con="00"
                                                        elif int(Nuevo_Valor_periferico)<100:
                                                                aux_con="0"
                                                        elif int(Nuevo_Valor_periferico)>180:
                                                                Nuevo_Valor_periferico=180
                                                                aux_con=""

        Sent_PSerie(password+"0"+(Acc_favoritos[Menu_Favorito.current()][1:3])+"0"+a
ux_con+str(Nuevo_Valor_periferico)+"000")#Enviamos orden a la placa
                                        elif
int(Acc_favoritos[Menu_Favorito.current()][1])==3:#Si es una alarma
                                                Nuevo_Valor_periferico =
appuifw.query(u"Seleccione frecuencia", 'number')
                                                if Nuevo_Valor_periferico:
                                                        if int(Nuevo_Valor_periferico)<10:
                                                                aux_con="00"
                                                        elif int(Nuevo_Valor_periferico)<100:
                                                                aux_con="0"
                                                        elif int(Nuevo_Valor_periferico)>255:
                                                                Nuevo_Valor_periferico=255
                                                                aux_con=""


        Sent_PSerie(password+"0"+(Acc_favoritos[Menu_Favorito.current()][1:3])+"0"+a
ux_con+str(Nuevo_Valor_periferico)+"000")#Enviamos orden a la placa
                                        elif
int(Acc_favoritos[Menu_Favorito.current()][1])==4:# motor
```

```
                                        Nuevo_Valor_periferico =
appuifw.query(u"Seleccione velocidad", 'number')
                                        if Nuevo_Valor_periferico:
                                                if int(Nuevo_Valor_periferico)<10:
                                                        aux_con="00"
                                                elif int(Nuevo_Valor_periferico)<100:
                                                        aux_con="0"
                                                elif int(Nuevo_Valor_periferico)>255:
                                                        Nuevo_Valor_periferico=255
                                                        aux_con=""


        Sent_PSerie(password+"0"+(Acc_favoritos[Menu_Favorito.current()][1:3])+"0"+a
ux_con+str(Nuevo_Valor_periferico)+"000")#Enviamos orden a la placa

                                        else:

        Sent_PSerie(password+Acc_favoritos[Menu_Favorito.current()])#Enviamos orden
a la placa
                                        appuifw.note(u"Orden enviada", "conf")
                                elif Op_Av == 1:#Cambiar nombre
                                        Name_Faforito_Op = appuifw.query(u"Seleccione nuevo
nombre", 'text')
                                        if Name_Faforito_Op:
                                                del Opc_favoritos[Menu_Favorito.current()]
                                                Opc_favoritos.insert(Menu_Favorito.current(), (u">
"+Name_Faforito_Op))
                                                Favoritos()
                                elif Op_Av == 2:#Editar orden
                                        global l
                                        backup_l=l
                                        bakup_name = Opc_favoritos[Menu_Favorito.current()]
                                        l=Menu_Favorito.current()
                                        global Aux_p
                                        global Aux_m
                                        global Aux_v

        Aux_p=int(Acc_favoritos[Menu_Favorito.current()][2:4])#pin
                                        if Aux_p <8:
                                                Aux_p = Aux_p-1
                                        elif Aux_p <35:
                                                Aux_p = Aux_p-14
                                        else:
                                                Aux_p = Aux_p-19

        Aux_m=int(Acc_favoritos[Menu_Favorito.current()][1])#Sacamos el MODO

        Aux_v=int(Acc_favoritos[Menu_Favorito.current()][4:8])#Sacamos el Valor
                                        del Opc_favoritos[Menu_Favorito.current()]
                                        Programar_Favoritos(Aux_p,Aux_m,Aux_v,bakup_name)
```

```
                        l=backup_l
                        Favoritos()
                        #appuifw.note(u"Proximamente", "conf")
                elif Op_Av == 3:#Eliminar
                        del Opc_favoritos[Menu_Favorito.current()]
                        del Acc_favoritos[Menu_Favorito.current()]
                        l=l-1
                        if l==0:
                                os.remove(CONFIG_DIR+CONFIG_FILE)
                        Favoritos()
                        appuifw.note(u"Accion eliminada con exito", "conf")



        appuifw.app.exit_key_handler = quit_favorito
        Menu_Favorito = appuifw.Listbox(Opc_favoritos, Acciones_Favoritos)
        appuifw.app.body = Menu_Favorito



#                                       #-----------------#-----------------#
Mostrar_Ocultar_Datos_GPS =0

Buffer_Opc_Monitorizar=[(u"")]
Opc_Monitorizar = [(u"
                                        Actualizar"),(u"
                                    Motrar Datos GPS")]
def Rutina_Monitorizar():
                global Mostrar_Ocultar_Datos_GPS

                global s
                global Longitud_Elmento
                global AuxTelemetria

                Sent_PSerie(password+"24000000000")#Enviamos orden a la placa
para que envie los datos
                while len(Opc_Monitorizar)>2:
                        del Opc_Monitorizar[2]
                        s=s-1
                        Longitud_Elmento=0
                        AuxTelemetria =""
                while len(Buffer_Opc_Monitorizar)>0:
                        del Buffer_Opc_Monitorizar[0]
                for x in range(55):
                        while Leer_PSerie() !=";":
                                Sensores[0]=Sensores[0]+AuxTelemetria
                                Longitud_Elmento=Longitud_Elmento+1
                        if Sensores[0][2:4] !="NN":#Si no se trada de un pin libre
                                #Opc_Monitorizar.insert(s, (u">
"+Sensores[0][0:2]+" ---"))
```

```
                                    if Sensores[0][2:4] =="SR":
                                            Buffer_Opc_Monitorizar.insert(s, (u">
"+Sensores[0][0:2]+" Sen.Resistivo: "+Sensores[0][4:Longitud_Elmento]))
                                    elif Sensores[0][2:4] =="PL":
                                            Buffer_Opc_Monitorizar.insert(s, (u">
"+Sensores[0][0:2]+" Pulsador: "+Sensores[0][4:Longitud_Elmento]+" veces"))
                                    elif Sensores[0][2:4] =="IN":
                                            if Sensores[0][4]=="1":
                                                    Buffer_Opc_Monitorizar.insert(s, (u">
"+Sensores[0][0:2]+" Interruptor: Cerrado"))
                                            else:
                                                    Buffer_Opc_Monitorizar.insert(s, (u">
"+Sensores[0][0:2]+" Interruptor: Abierto"))
                                    elif Sensores[0][2:4] =="TP":
                                            Buffer_Opc_Monitorizar.insert(s, (u">
"+Sensores[0][0:2]+" Sen.Temperat: "+Sensores[0][4:Longitud_Elmento]))
                                    elif Sensores[0][2:4] =="US":
                                            Buffer_Opc_Monitorizar.insert(s, (u">
"+Sensores[0][0:2]+" Sen.Ultrasonido: "+Sensores[0][4:Longitud_Elmento]))
                                    elif Sensores[0][2:4] =="IR":
                                            Buffer_Opc_Monitorizar.insert(s, (u">
"+Sensores[0][0:2]+" Sen.Infrarojo: "+Sensores[0][4:Longitud_Elmento]))

                                    elif Sensores[0][2:4] =="RE":
                                            if Sensores[0][4]=="0":
                                                    Buffer_Opc_Monitorizar.insert(s, (u">
"+Sensores[0][0:2]+" Rele OFF"))
                                            else:
                                                    Buffer_Opc_Monitorizar.insert(s, (u">
"+Sensores[0][0:2]+" Rele ON "))
                                    elif Sensores[0][2:4] =="MP":
                                            Buffer_Opc_Monitorizar.insert(s, (u">
"+Sensores[0][0:2]+" Mod.Pulso:"+Sensores[0][4:Longitud_Elmento]))
                                    elif Sensores[0][2:4] =="SV":
                                            Buffer_Opc_Monitorizar.insert(s, (u">
"+Sensores[0][0:2]+" Servo: "+Sensores[0][4:Longitud_Elmento]+" grados"))
                                    elif Sensores[0][2:4] =="MO":
                                            Buffer_Opc_Monitorizar.insert(s, (u">
"+Sensores[0][0:2]+" Motor: "+Sensores[0][4:Longitud_Elmento]+" rpm"))
                                    elif Sensores[0][2:4] =="AL":
                                            if Sensores[0][4]=="0":
                                                    Buffer_Opc_Monitorizar.insert(s, (u">
"+Sensores[0][0:2]+" Alarma OFF"))
                                            else:
                                                    Buffer_Opc_Monitorizar.insert(s, (u">
"+Sensores[0][0:2]+" Alarma ON "))
                                    elif Sensores[0][2:4] =="??":
                                            Buffer_Opc_Monitorizar.insert(s, (u">
"+Sensores[0][0:2]+" Desconocido"))
                                    else:
```

```
                                        Buffer_Opc_Monitorizar.insert(s, (u">
"+Sensores[0]))
                                Sensores[0]=""
                                Longitud_Elmento=0
                                s=s+1
                                #z=z+1
                                AuxTelemetria =""

                        for y in range(len(Buffer_Opc_Monitorizar)-5):
                                Opc_Monitorizar.insert(y+2, (Buffer_Opc_Monitorizar[y]))
                                #Star_Monitorizar=End_Monitorizar

                        if Mostrar_Ocultar_Datos_GPS==1:#deseo mostrar los datos
                                for fh in range(5):
                                        Opc_Monitorizar.insert(fh+2,
(Buffer_Opc_Monitorizar[fh+(len(Buffer_Opc_Monitorizar))-5]))


def Monitorizar():


        def Acciones_Monitorizar():
                global Mostrar_Ocultar_Datos_GPS

                global Aux_pin_telemetria
                if Menu_Monitorizar.current() == 0:#Actualizar
                        Rutina_Monitorizar()
                        Monitorizar()
                elif Menu_Monitorizar.current() == 1:#Datos GPS
                        if Mostrar_Ocultar_Datos_GPS==0:#deseo mostrar los datos
                                del Opc_Monitorizar[1]
                                Opc_Monitorizar.insert(1, (u"
                                        Ocultar Datos GPS"))
                                for f in range(5):
                                        Opc_Monitorizar.insert(f+2,
(Buffer_Opc_Monitorizar[f+(len(Buffer_Opc_Monitorizar))-5]))
                                Mostrar_Ocultar_Datos_GPS=1
                                Monitorizar()
                        else:
                                Mostrar_Ocultar_Datos_GPS=0
                                del Opc_Monitorizar[1]
                                Opc_Monitorizar.insert(1, (u"
                                        Mostrar Datos GPS"))
                                for lp in range(5):
                                        del Opc_Monitorizar[2]
                                Monitorizar()
                else:
                        if Bt_Start == 0:#Bluetooth no activa
                                        appuifw.note(u"Es necesario establecer comunicacion
primero", "error")
```

```
                    else:
                            if Opc_Monitorizar[Menu_Monitorizar.current()][2]!="L"
and Opc_Monitorizar[Menu_Monitorizar.current()][2]!="A" and
Opc_Monitorizar[Menu_Monitorizar.current()][2]!="V" and
Opc_Monitorizar[Menu_Monitorizar.current()][2]!="C":
                                    if
int(Opc_Monitorizar[Menu_Monitorizar.current()][3:5]) <8:
                                            Aux_pin_telemetria =
"0"+str(int(Opc_Monitorizar[Menu_Monitorizar.current()][3:5])+1)
                                    elif
int(Opc_Monitorizar[Menu_Monitorizar.current()][3:5]) <35:
                                            Aux_pin_telemetria =
str(int(Opc_Monitorizar[Menu_Monitorizar.current()][3:5])+14)
                                    else:
                                            Aux_pin_telemetria =
str(int(Opc_Monitorizar[Menu_Monitorizar.current()][3:5])+19)

                                    if
Opc_Monitorizar[Menu_Monitorizar.current()][6:10]=="Rele":

        Sent_PSerie(password+"00"+Aux_pin_telemetria+"0000000")#Enviamos orden a
la placa para que envie los datos
                                            if
Opc_Monitorizar[Menu_Monitorizar.current()][11:13]=="ON":
                                                    appuifw.note(u"Rele Apagado",
"conf")
                                                    audio.say("Rele Apagado")
                                            else:
                                                    appuifw.note(u"Rele Encendido",
"conf")
                                                    audio.say("Rele Encendido")
                                            Rutina_Monitorizar()
                                            Monitorizar()
                                    elif
Opc_Monitorizar[Menu_Monitorizar.current()][6:11]=="Servo":
                                            A_Giro = appuifw.query(u"Angulo a girar",
'number')
                                            angulo=str(A_Giro)
                                            if A_Giro:
                                                    if int(A_Giro)<10:
                                                            A_Giro="00"+str(A_Giro)
                                                    elif int(A_Giro)<100:
                                                            A_Giro="0"+str(A_Giro)
                                                    if int(A_Giro)>180:
                                                            A_Giro="180"
                                                            angulo="180"
                                                    #appuifw.note(u"Servo
girado"+str(A_Giro)+" grados", "conf")
```

```
        Sent_PSerie(password+"02"+Aux_pin_telemetria+"0"+str(A_Giro)+"000")#Envia
mos orden a la placa para que envie los datos
                                                appuifw.note(u"Servo girado
"+angulo+" grados", "conf")
                                                #audio.say("Servo girado "+angulo+"
grados", "conf")
                                                Rutina_Monitorizar()
                                                Monitorizar()
                                        elif
Opc_Monitorizar[Menu_Monitorizar.current()][6:12]=="Alarma":

        Sent_PSerie(password+"03"+Aux_pin_telemetria+"4000000")#Enviamos orden a
la placa para que envie los datos
                                                if
Opc_Monitorizar[Menu_Monitorizar.current()][13:15]=="ON":
                                                        appuifw.note(u"Alarma Apagada",
"conf")
                                                        audio.say("Alarma Apagada")
                                                else:
                                                        appuifw.note(u"Alarma Encendida",
"conf")
                                                        audio.say("Alarma Encendida")
                                                Rutina_Monitorizar()
                                                Monitorizar()
                                        elif
Opc_Monitorizar[Menu_Monitorizar.current()][6:15]=="Mod.Pulso" or
Opc_Monitorizar[Menu_Monitorizar.current()][6:11]=="Motor":
                                                A_Pulso = appuifw.query(u"Ancho de pulso",
'number')
                                                pulso=str(A_Pulso)
                                                if A_Pulso:
                                                        if int(A_Pulso)<10:
                                                                A_Pulso="00"+str(A_Pulso)
                                                        elif int(A_Pulso)<100:
                                                                A_Pulso="0"+str(A_Pulso)
                                                        if int(A_Pulso)>255:
                                                                A_Pulso="255"
                                                                pulso="255"


        Sent_PSerie(password+"02"+Aux_pin_telemetria+"0"+str(A_Pulso)+"000")#Envia
mos orden a la placa para que envie los datos
                                                appuifw.note(u"Ancho de pulso:
"+pulso, "conf")
                                                #audio.say("Ancho de pulso: "+pulso)
                                                Rutina_Monitorizar()
                                                Monitorizar()
```

```
                    else:
                        Etiquetas_Opc_Monitorizar = [u"Reenviar
coordenadas a...", u"Actualizar"]
                        Op_Av_Opc_Monitorizar =
appuifw.popup_menu(Etiquetas_Opc_Monitorizar, u"Seleccione una accion:")
                        if Op_Av_Opc_Monitorizar == 0 :#Reenviar
coordenadas a...
                            Telefono_Reenvio_Coordenadas =
appuifw.query(u"Especifique numero de destinatario", 'number')
                            if Telefono_Reenvio_Coordenadas:
                                if
Telefono_Reenvio_Coordenadas<999999999:
                                    if
globalui.global_query(u"Desea reenviar las coordenadas GPS al numero:
"+str(Telefono_Reenvio_Coordenadas)) == True:

        messaging.sms_send(str(Telefono_Reenvio_Coordenadas), u"Reenviado desde
GNX
Project\n"+Buffer_Opc_Monitorizar[0]+"\n"+Buffer_Opc_Monitorizar[1]+"\n"+Buffer_O
pc_Monitorizar[2]+"\n"+Buffer_Opc_Monitorizar[3]+"\n"+Buffer_Opc_Monitorizar[4])

        appuifw.note(u"Coordenadas reenviadas con exito", "conf")
                                else:
                                    appuifw.note(u"El telefono
introducido no es valido", "error")
                        elif Op_Av_Opc_Monitorizar == 1 :#Actualizar
                            Rutina_Monitorizar()
                            Monitorizar()
        appuifw.app.exit_key_handler = quit_favorito
        Menu_Monitorizar = appuifw.Listbox(Opc_Monitorizar, Acciones_Monitorizar)
        appuifw.app.body = Menu_Monitorizar



#                           #-----------------#-----------------#
def quit_favorito():
        global Once
        Once=0
        appuifw.app.body = app1 #Pestanna bluetooth
#                           #-----------------#-----------------#
Opc_Panel_Control = [(u"Configurar Sensor"),(u"Programar acciones"),(u"Ver acciones
programadas"),(u"Eliminar acciones programadas"),(u"

        Enviar"),(u"
                                    Volver")]
Opc_Ver_Acciones_Programadas = [(u"Configuracion Sensor",u"Aun no
definido"),(u"Borrar Todas las acciones",u""),(u"

        Volver",u"")]
saved_Programar_acciones = False
```

```
saved_Conf_Sensores = False
def Programar_acciones(p,m,v,a,t,apagado):
                                global j

                                Pin_A = [u'Sin seleccionar',u'1 (Digital PWM)',u'2 (Digital
PWM)', u'3 (Digital PWM)', u'4 (Digital PWM)', u'5 (Digital PWM)', u'6 (Digital PWM)',
u'7 (Digital PWM)', u'8 (Digital)', u'9 (Digital)', u'10 (Digital)', u'11 (Digital)', u'12
(Digital)', u'13 (Digital)', u'14 (Digital)', u'15 (Digital)', u'16 (Digital)', u'17 (Digital)', u'18
(Digital)', u'19 (Digital)', u'20 (Digital)', u'21 (Digital)', u'22 (Digital)', u'23 (Digital)', u'24
(Digital)', u'25 (Digital)', u'26 (Digital)', u'27 (Digital)', u'28 (Digital)', u'29 (Digital)', u'30
(Digital)', u'31 (Digital)', u'32 (Digital)', u'33 (Digital)', u'34 (Digital)', u'35
(Digital/Analogico)', u'36 (Digital/Analogico)', u'37 (Digital/Analogico)', u'38
(Digital/Analogico)', u'39 (Digital/Analogico)', u'40 (Digital/Analogico)', u'41
(Digital/Analogico)', u'42 (Digital/Analogico)', u'43 (Digital/Analogico)', u'44
(Digital/Analogico)', u'45 (Digital/Analogico)', u'46 (Digital/Analogico)', u'47
(Digital/Analogico)', u'48 (Digital/Analogico)', u'49 (Digital/Analogico)', u'50
(Digital/Analogico)']
                                Modo_A = [u'Rele', u'Modular Pulso',
u'Servo',u'Alarma',u'Motor',u'Altavoz placa']
                                Accion_Avanzada = [u'Ninguna',u'Logger GPS', u'Logger All
Pin', u'Logger pin...']
                                Temp_Pulsador_A = [u'Automatico (Rele/Alarma)',u'1 Min
(Rele/Alarma)',u'5 Min (Rele/Alarma)',u'10 Min (Rele/Alarma)',u'15 Min
(Rele/Alarma)',u'20 Min (Rele/Alarma)',u'30 Min (Rele/Alarma)',u'60 Min
(Rele/Alarma)',u'90 Min (Rele/Alarma)',u'120 Min (Rele/Alarma)']
                                Data_A = [(u"Pin", 'combo', (Pin_A,p)),
                                                (u"Salida", 'combo', (Modo_A,m)),
                                                (u"Valor",'number',v),
                                                (u"Acciones", 'combo', (Accion_Avanzada,a)),
                                                (u"Apagado", 'combo',
(Temp_Pulsador_A,apagado))]

                                def salvado_Programar_acciones(arg):
                                        global saved_Programar_acciones
                                        saved_Programar_acciones = True
                                        return True

                                flags_Conf_Acciones = appuifw.FFormEditModeOnly
                                PC_Acc_Sensores = appuifw.Form( Data_A,
flags_Conf_Acciones)
                                PC_Acc_Sensores.save_hook = salvado_Programar_acciones
                                PC_Acc_Sensores.execute()

                                if saved_Programar_acciones == True:
                                        global error_flag
                                        error_flag=2
                                        if PC_Acc_Sensores[0][2][1] <8:
                                                Pin_Code =
"0"+str(PC_Acc_Sensores[0][2][1]+1)
```

```
                                        Pin_Name_Aux
=str(PC_Acc_Sensores[0][2][1])
                                elif PC_Acc_Sensores[0][2][1] <35:
                                        Pin_Code =
str(PC_Acc_Sensores[0][2][1]+14)
                                        Pin_Name_Aux
=str(PC_Acc_Sensores[0][2][1])
                                else:
                                        Pin_Code =
str(PC_Acc_Sensores[0][2][1]+19)
                                        Pin_Name_Aux
=str(PC_Acc_Sensores[0][2][1])
                                if Accion_Avanzada[PC_Acc_Sensores[3][2][1]] ==
"Ninguna":
                                        if not PC_Acc_Sensores[0][2][1] == 0:#Sin
seleccionar pin
                                                Value1 = str(PC_Acc_Sensores[2][2])
                                                if PC_Acc_Sensores[2][2] < 10:
                                                        aux1="00"
                                                elif PC_Acc_Sensores[2][2] < 100:
                                                        aux1="0"
                                                elif PC_Acc_Sensores[2][2] > 99:
                                                        aux1=""

                                                if
PC_Acc_Sensores[1][2][1]==0:#Rele
                                                        Mode_Code = "0"
                                                        grados=""
                                                elif
PC_Acc_Sensores[1][2][1]==1:#Modular Ancho de Pulso
                                                        Mode_Code = "1"
                                                        grados=""
                                                        if PC_Acc_Sensores[2][2] >
255:
                                                                aux1=""
                                                                Value1 = "255"
                                                elif
PC_Acc_Sensores[1][2][1]==2:#Servo
                                                        Mode_Code = "2"
                                                        grados="Grados"
                                                        if PC_Acc_Sensores[2][2] >
180:
                                                                aux1=""
                                                                Value1 = "180"
                                                elif PC_Acc_Sensores[1][2][1]==3 or
PC_Acc_Sensores[1][2][1]==5:#Alarma
                                                        Mode_Code = "3"
                                                        grados=""
                                                elif
PC_Acc_Sensores[1][2][1]==4:#Motor
```

```
                                        Mode_Code = "4"
                                        grados=""

                                    if PC_Acc_Sensores[1][2][1]==0 or
PC_Acc_Sensores[1][2][1]==3 or PC_Acc_Sensores[1][2][1]==5:#Si es un Rele o una
Alarma
                                        if PC_Acc_Sensores[4][2][1]
==0:#Nunca
                                            Value2="000"
                                            aux2=""
                                        elif PC_Acc_Sensores[4][2][1]
==1:#1 Min
                                            Value2="001"
                                            aux2=""
                                        elif PC_Acc_Sensores[4][2][1]
==2:#5 Min
                                            Value2="002"
                                            aux2=""
                                        elif PC_Acc_Sensores[4][2][1]
==3:#10 Min
                                            Value2="003"
                                            aux2=""
                                        elif PC_Acc_Sensores[4][2][1]
==4:#15 Min
                                            Value2="004"
                                            aux2=""
                                        elif PC_Acc_Sensores[4][2][1]
==5:#20 Min
                                            Value2="005"
                                            aux2=""
                                        elif PC_Acc_Sensores[4][2][1]
==6:#30 Min
                                            Value2="006"
                                            aux2=""
                                        elif PC_Acc_Sensores[4][2][1]
==7:#60 Min
                                            Value2="007"
                                            aux2=""
                                        elif PC_Acc_Sensores[4][2][1]
==8:#90 Min
                                            Value2="008"
                                            aux2=""
                                        elif PC_Acc_Sensores[4][2][1]
==9:#120Min
                                            Value2="009"
                                            aux2=""
                                        else:
                                            Value2="000"
                                            aux2=""
                                    else:
```

```
                                                    Value2="000"
                                                if not
PC_Acc_Sensores[1][2][1]==5:#no es Altavoz placa
                                                    Code =
"0"+Mode_Code+Pin_Code + "0" +aux1+Value1+Value2
                                                else:

        Code="03090"+aux1+Value1+"000"
                                                if not PC_Acc_Sensores[1][2][1]==0
and not PC_Acc_Sensores[1][2][1]==3 and not PC_Acc_Sensores[1][2][1]==5:#Si no es
ni un rele ni una alarma

        Opc_Ver_Acciones_Programadas.insert(j, (u"Pin "+Pin_Name_Aux+"
"+Modo_A[PC_Acc_Sensores[1][2][1]]+" "+Value1+" "+grados,u""+Code))
                                                    aux1=0
                                                    j=j+1
                                                else:
                                                    if not
PC_Acc_Sensores[1][2][1]==5:#Altavoz placa

        Opc_Ver_Acciones_Programadas.insert(j, (u"Pin "+Pin_Name_Aux+"
"+Modo_A[PC_Acc_Sensores[1][2][1]],u""+Code))
                                                    else:#Altavoz placa

        Opc_Ver_Acciones_Programadas.insert(j,
(u""+Modo_A[PC_Acc_Sensores[1][2][1]]+" "+aux1+Value1,u""+Code))
                                                    aux1=0
                                                    j=j+1

                                            else:
                                                if not
PC_Acc_Sensores[1][2][1]==5:#Altavoz placa
                                                    appuifw.note(u"Debe
especificar el pin!", "error")

        #Programar_acciones(0,0,0,0,0,0)
                                                else:
                                                    Value1 =
str(PC_Acc_Sensores[2][2])
                                                    if PC_Acc_Sensores[2][2] < 10:
                                                        aux1="00"
                                                    elif PC_Acc_Sensores[2][2] <
100:
                                                        aux1="0"
                                                    elif PC_Acc_Sensores[2][2] >
99:
                                                        aux1=""

        Code="03090"+aux1+Value1+"000"
```

```
        Opc_Ver_Acciones_Programadas.insert(j, (u"Altavoz placa"+"
"+aux1+Value1,u""+Code))
                                                        aux1=0
                                                        j=j+1

                                        else:
                                                Value1 = str(PC_Acc_Sensores[4][2])
                                                global error_flag
                                                error_flag=2
                                                if PC_Acc_Sensores[3][2][1]==1:#Logger
GPS
                                                        Mode_Code = "5"
                                                        Code =
"2"+Mode_Code+"00"+"0"+"000"+"000"

        Opc_Ver_Acciones_Programadas.insert(j,
(u""+Accion_Avanzada[PC_Acc_Sensores[3][2][1]],u""+Code))
                                                elif PC_Acc_Sensores[3][2][1]==2:#Logger
Sensor
                                                        Mode_Code = "6"
                                                        Code =
"2"+Mode_Code+"00"+"0"+"000"+"000"

        Opc_Ver_Acciones_Programadas.insert(j,
(u""+Accion_Avanzada[PC_Acc_Sensores[3][2][1]],u""+Code))
                                                elif PC_Acc_Sensores[3][2][1]==3:#Logger
Pin...
                                                        if not PC_Acc_Sensores[0][2][1] ==
0:#Sin seleccionar pin
                                                                Mode_Code = "6"
                                                                Code =
"2"+Mode_Code+Pin_Code+"0"+"000"+"000"

        Opc_Ver_Acciones_Programadas.insert(j,
(u""+Accion_Avanzada[PC_Acc_Sensores[3][2][1]],u""+Code))
                                                        else:
                                                                appuifw.note(u"Debe
especificar el pin!", "error")

        #Programar_acciones(0,0,0,0,0,0)
                                                aux1=0
                                                j=j+1



def Conf_Sensores(p,m,c,v1,v2):
        Pin_S = [u'Sin seleccionar',u'1 (Digital PWM)',u'2 (Digital PWM)', u'3 (Digital
PWM)', u'4 (Digital PWM)', u'5 (Digital PWM)', u'6 (Digital PWM)', u'7 (Digital PWM)',
u'8 (Digital)', u'9 (Digital)', u'10 (Digital)', u'11 (Digital)', u'12 (Digital)', u'13 (Digital)',
```

```
u'14 (Digital)', u'15 (Digital)', u'16 (Digital)', u'17 (Digital)', u'18 (Digital)', u'19 (Digital)',
u'20 (Digital)', u'21 (Digital)', u'22 (Digital)', u'23 (Digital)', u'24 (Digital)', u'25 (Digital)',
u'26 (Digital)', u'27 (Digital)', u'28 (Digital)', u'29 (Digital)', u'30 (Digital)', u'31 (Digital)',
u'32 (Digital)', u'33 (Digital)', u'34 (Digital)', u'35 (Digital/Analogico)', u'36
(Digital/Analogico)', u'37 (Digital/Analogico)', u'38 (Digital/Analogico)', u'39
(Digital/Analogico)', u'40 (Digital/Analogico)', u'41 (Digital/Analogico)', u'42
(Digital/Analogico)', u'43 (Digital/Analogico)', u'44 (Digital/Analogico)', u'45
(Digital/Analogico)', u'46 (Digital/Analogico)', u'47 (Digital/Analogico)', u'48
(Digital/Analogico)', u'49 (Digital/Analogico)', u'50 (Digital/Analogico)']
        Modo_S = [u'Resistivo', u'Pulsador',u'Interruptor', u'Temperatura(LM35)',
u'Ultrasonido', u'Infrarrojo',u'Independiente']
        Condicion_S = [u'Entrada=Valor 1',u'Entrada>Valor 1', u'Entrada<Valor 1', u'Valor
1<Entrada<Valor 2', u'Entrada->Salida',u'Cerrado(Interruptor)', u'Abierto(Interruptor)']
        Nivel =         [u'Ascendente (Solo Pulsador)', u'Descendente (Solo Pulsador)']
        N_Pulsaciones = [u'1 (Solo Pulsador)',u'2 (Solo Pulsador)',u'3 (Solo Pulsador)',u'4
(Solo Pulsador)',u'5 (Solo Pulsador)',u'6 (Solo Pulsador)',u'7 (Solo Pulsador)',u'8 (Solo
Pulsador)',u'9 (Solo Pulsador)']
        Temp_Pulsador_S = [u'Nunca (Solo Pulsador)',u'5 Seg (Solo Pulsador)',u'10 Seg
(Solo Pulsador)',u'20 Seg (Solo Pulsador)',u'30 Seg (Solo Pulsador)',u'1 Min (Solo
Pulsador)',u'5 Min (Solo Pulsador)',u'10 Min (Solo Pulsador)',u'15 Min (Solo
Pulsador)',u'30 Min (Solo Pulsador)']
        Data_S = [(u"Pin", 'combo', (Pin_S,p )),
                        (u"Sensor", 'combo', (Modo_S,m )),
                        (u"Condicion", 'combo', (Condicion_S, c )),
                        (u"Valor 1",'number', v1 ),
                        (u"Valor 2",'number', v2 ),
                        (u"Pulsos", 'combo', (N_Pulsaciones, c )),
                        (u"Flanco", 'combo', (Nivel, 0 )),
                        (u"Apagado", 'combo', (Temp_Pulsador_S, 0))]


        def salvado_Conf_Sensores(arg):
                global saved_Conf_Sensores
                saved_Conf_Sensores = True
                return True
        flags_Conf_Sensores = appuifw.FFormEditModeOnly
        PC_Conf_Sensores = appuifw.Form( Data_S, flags_Conf_Sensores )
        PC_Conf_Sensores.save_hook = salvado_Conf_Sensores
        PC_Conf_Sensores.execute( )
        if saved_Conf_Sensores == True:
                if not PC_Conf_Sensores[0][2][1] == 0:
                        if PC_Conf_Sensores[0][2][1] <8:
                                Pin_Code = "0"+str(PC_Conf_Sensores[0][2][1]+1)
                                Pin_Name_Aux =str(PC_Conf_Sensores[0][2][1])
                        elif PC_Conf_Sensores[0][2][1] <35:
                                Pin_Code = str(PC_Conf_Sensores[0][2][1]+14)
                                Pin_Name_Aux =str(PC_Conf_Sensores[0][2][1])
                        else:
                                Pin_Code = str(PC_Conf_Sensores[0][2][1]+19)
                                Pin_Name_Aux =str(PC_Conf_Sensores[0][2][1])
```

```
                    #Tipor de Sensor
                    Mode_Code = str(PC_Conf_Sensores[1][2][1])
                    while len(Informacion_Sensor)>0:
                        del Informacion_Sensor[0]
                    if PC_Conf_Sensores[1][2][1] ==1:#Si es un Pulsador/Magnetico
                        Condicion = str(PC_Conf_Sensores[5][2][1]+1)
                        if PC_Conf_Sensores[6][2][1] == 0: #Nivel Alto
                            ValueAux = "1"
                            Informacion_Sensor.insert(3,"\nActivado cuando pase
a flanco: Ascendente")
                        elif PC_Conf_Sensores[6][2][1] ==1:#Nivel Bajo
                            ValueAux = "0"
                            Informacion_Sensor.insert(3,"\nActivado cuando pase
a flanco: Descendente")
                        Value1 = "00"+ValueAux
                        aux1=""

                        if PC_Conf_Sensores[7][2][1] ==0:#Nunca
                            Informacion_Sensor.insert(4,"\nApagado: Nunca")
                            Value2="000"
                            aux2=""
                        elif PC_Conf_Sensores[7][2][1] ==1:#5 Seg
                            Informacion_Sensor.insert(4,"\nApagado despues de:
5 Seg")
                            Value2="001"
                            aux2=""
                        elif PC_Conf_Sensores[7][2][1] ==2:#10 Seg
                            Informacion_Sensor.insert(4,"\nApagado despues de:
10 Seg")
                            Value2="002"
                            aux2=""
                        elif PC_Conf_Sensores[7][2][1] ==3:#20 Seg
                            Informacion_Sensor.insert(4,"\nApagado despues de:
20 Seg")
                            Value2="003"
                            aux2=""
                        elif PC_Conf_Sensores[7][2][1] ==4:#30 Seg
                            Informacion_Sensor.insert(4,"\nApagado despues de:
30 Seg")
                            Value2="004"
                            aux2=""
                        elif PC_Conf_Sensores[7][2][1] ==5:#1 Min
                            Informacion_Sensor.insert(4,"\nApagado despues de:
1 Min")
                            Value2="005"
                            aux2=""
                        elif PC_Conf_Sensores[7][2][1] ==6:#5 Min
                            Informacion_Sensor.insert(4,"\nApagado despues de:
5 Min")
```

```
                        Value2="006"
                        aux2=""
                    elif PC_Conf_Sensores[7][2][1] ==7:#10 Min
                        Informacion_Sensor.insert(4,"\nApagado despues de:
10 Min")
                        Value2="007"
                        aux2=""
                    elif PC_Conf_Sensores[7][2][1] ==8:#15 Min
                        Informacion_Sensor.insert(4,"\nApagado despues de:
15 Min")
                        Value2="008"
                        aux2=""
                    elif PC_Conf_Sensores[7][2][1] ==9:#30 Min
                        Informacion_Sensor.insert(4,"\nApagado despues de:
30 Min")
                        Value2="009"
                        aux2=""
                    else:
                        Value2="000"
                        aux2=""


                    if PC_Conf_Sensores[5][2][1]+1 == 1:
                        Informacion_Sensor.insert(2,"\nCondicion: Ser
pulsado "+Condicion +" vez")
                    else:
                        Informacion_Sensor.insert(2,"\nCondicion: Ser
pulsado "+Condicion +" veces")


                else:
                    #Condicion
                    Condicion = str(PC_Conf_Sensores[2][2][1])
                    if PC_Conf_Sensores[2][2][1]==5:#Cerrado, interruptor
                        Condicion="1"
                    elif PC_Conf_Sensores[2][2][1]==6:#Abierto, interruptor
                        Condicion="0"

                    #Valor 1
                    Value1 = str(PC_Conf_Sensores[3][2])
                    if PC_Conf_Sensores[3][2] > 1000:
                        aux1=""
                        Value1 ="999"
                    elif PC_Conf_Sensores[3][2] < 10:
                        aux1="00"
                    elif PC_Conf_Sensores[3][2] < 100:
                        aux1="0"
                    elif PC_Conf_Sensores[3][2] > 99:
                        aux1=""
```

```
                                #Valor 2
                                Value2 = str(PC_Conf_Sensores[4][2])
                                if PC_Conf_Sensores[4][2] > 1000:
                                        aux2=""
                                        Value2 ="999"
                                elif PC_Conf_Sensores[4][2] < 10:
                                        aux2="00"
                                elif PC_Conf_Sensores[4][2] < 100:
                                        aux2="0"
                                elif PC_Conf_Sensores[4][2] > 99:
                                        aux2=""

                                if PC_Conf_Sensores[2][2][1]==0:#Entrada=Valor 1
                                        Informacion_Sensor.insert(2,"\nCondicion: V.Sensor
= "+Value1)
                                elif PC_Conf_Sensores[2][2][1]==1:#Entrada>Valor 1
                                        Informacion_Sensor.insert(2,"\nCondicion: V.Sensor
> "+Value1)
                                elif PC_Conf_Sensores[2][2][1]==2:#Entrada<Valor 1
                                        Informacion_Sensor.insert(2,"\nCondicion: V.Sensor
< "+Value1)
                                elif PC_Conf_Sensores[2][2][1]==3:#Valor
1<Entrada<Valor 2
                                        Informacion_Sensor.insert(2,"\nCondicion: V.Sensor
entre "+Value1+" y "+Value2)
                                elif PC_Conf_Sensores[2][2][1]==4:#Cerrado
                                        Informacion_Sensor.insert(2,"\nCondicion: Estar
Cerrado")
                                elif PC_Conf_Sensores[2][2][1]==5:#Abierto
                                        Informacion_Sensor.insert(2,"\nCondicion: Estar
Abierto")
                                Informacion_Sensor.insert(3,"")
                                Informacion_Sensor.insert(4,"")



                        if not PC_Conf_Sensores[1][2][1] ==6:#Si NO es una accion
independiente de los sensores
                                #if not PC_Conf_Sensores[1][2][1]==4 and not
PC_Conf_Sensores[1][2][1]==4
                                if not PC_Conf_Sensores[1][2][1] ==2 and
(PC_Conf_Sensores[2][2][1]==5 or PC_Conf_Sensores[2][2][1]==6):#Si  NO es un
interruptor y he puesto como condicion abierto/cerrado salta un error
                                        appuifw.note(u"Esta condicion es solo para el
Interruptor", "error")
                                        Conf_Sensores(0,0,0,0,0)
                                elif PC_Conf_Sensores[1][2][1] ==2 and
PC_Conf_Sensores[2][2][1]<4: #Si es un Interruptor y No he puesto una condicion valida
salta error
```

```
                                        appuifw.note(u"Condicion no valida", "error")
                                        Conf_Sensores(0,0,2,0,0)
                                else:
                                        del Opc_Ver_Acciones_Programadas[0]#Borramos lo
que ya huviera antes programado
                                        Code = "1"+Mode_Code+Pin_Code + Condicion
+aux1+Value1+aux2+Value2
                                        Opc_Ver_Acciones_Programadas.insert(0, (u"Sensor
"+Modo_S[PC_Conf_Sensores[1][2][1]]+" pin: "+Pin_Name_Aux,u""+Code))
                                        Informacion_Sensor.insert(0,"Pin: "+Pin_Name_Aux)
                                        Informacion_Sensor.insert(1,"\nSensor:
"+Modo_S[PC_Conf_Sensores[1][2][1]])
                                        appuifw.note(u"Configuracion del sensor guardada
con exito", "conf")
                        else:#Si es una accion independiente de los sensores
                                Code = "1"+"5"+"00" + "0" +"000"+"000"
                                Opc_Ver_Acciones_Programadas.insert(0, (u"Sensor:
Independiente",u""+Code))
                                Informacion_Sensor.insert(0,"Pin: Ninguno")
                                Informacion_Sensor.insert(1,"\nCondicion: No precisa de
condicion")
                                Informacion_Sensor.insert(2,"")
                                Informacion_Sensor.insert(3,"")
                                Informacion_Sensor.insert(4,"")
                                appuifw.note(u"Configuracion del sensor guardada con
exito", "conf")


                elif PC_Conf_Sensores[1][2][1] ==6:#Si es una accion independiente de los
sensores:#Sin seleccionar
                        Code = "1"+"6"+"00" + "0" +"000"+"000"
                        del Opc_Ver_Acciones_Programadas[0]#Borramos lo que ya
huviera antes programado
                        Opc_Ver_Acciones_Programadas.insert(0, (u"Sensor:
Independiente",u""+Code))
                        Informacion_Sensor.insert(0,"Pin: Ninguno")
                        Informacion_Sensor.insert(1,"\nCondicion: No precisa de
condicion")
                        Informacion_Sensor.insert(2,"")
                        Informacion_Sensor.insert(3,"")
                        Informacion_Sensor.insert(4,"")
                        appuifw.note(u"Configuracion del sensor guardada con exito",
"conf")
                else:
                        appuifw.note(u"Debe especificar el pin!", "error")
                        Conf_Sensores(0,0,0,0,0)
def Panel_Control():


        def quit_Panel_Control():
```

```
                appuifw.app.body = Menu_Panel_Control

        def Acciones_Panel_Control ():
                global j
                if Menu_Panel_Control.current() == 0:#Configurar Sensor
                        Conf_Sensores(0,0,0,0,0)
                elif Menu_Panel_Control.current() == 1:
                        Programar_acciones(0,0,0,0,0,0)
                        global error_flag
                        if error_flag==2:
                                appuifw.note(u"Accion guardada con exito", "conf")
                                error_flag=0
                        else:
                                error_flag=0

                elif Menu_Panel_Control.current() == 2:#Ver acciones programadas
                        def Menu_Panel_Control_Acciones_Programadas():
                                def Acciones_Ver_Acciones_Programadas():
                                        #if Menu_Ver_Acciones_Programadas.current() == 0:
                                        if Menu_Ver_Acciones_Programadas.current() ==
0:#Sensor
                                                Etiquetas_Edit_Acciones = [u"Ver detalles del
sensor",u"Editar Sensor",u"Eliminar Sensor"]
                                                Op_Etiquetas_Edit_Acciones =
appuifw.popup_menu(Etiquetas_Edit_Acciones, u"Seleccione una accion:")
                                                if Op_Etiquetas_Edit_Acciones == 0 :#Ver
detalles del sensor

        globalui.global_msg_query(u""+Informacion_Sensor[0]+Informacion_Sensor[1]+I
nformacion_Sensor[2]+Informacion_Sensor[3]+Informacion_Sensor[4], u"Detalles del
sensor", 0)
                                                elif Op_Etiquetas_Edit_Acciones == 1
:#Editar Sensor
                                                        if globalui.global_query(u"Desea editar
la configuracion del sensor") == True:
                                                                Conf_Sensores(0,0,0,0,0)

        Menu_Panel_Control_Acciones_Programadas()
                                                elif Op_Etiquetas_Edit_Acciones == 2
:#Eliminar Sensor
                                                        if globalui.global_query(u"Desea
eliminar la configuracion del sensor") == True:

        Informacion_Sensor.insert(0,("Sin configurar"))

        Informacion_Sensor.insert(1,(""))

        Informacion_Sensor.insert(2,(""))

        Informacion_Sensor.insert(3,(""))
```

```
        Informacion_Sensor.insert(4,(""))
                                                    del
Opc_Ver_Acciones_Programadas[0]

        Opc_Ver_Acciones_Programadas.insert(0, (u"Configuracion Sensor",u"Aun no
definido"))

                                        elif Menu_Ver_Acciones_Programadas.current() ==
j:#Borrar todas las acciones programadas
                                                if not j==1:
                                                    if globalui.global_query(u"Desea
eliminar todas las acciones programadas") == True:
                                                        while j > 1:
                                                            global j
                                                            del
Opc_Ver_Acciones_Programadas[1]
                                                            j=j-1

        Menu_Panel_Control_Acciones_Programadas()
                                                        appuifw.note(u"Todas las
acciones ha sido borradas con exito", "conf")
                                                else:
                                                    appuifw.note(u"No existen acciones
programadas", "error")

                                        elif Menu_Ver_Acciones_Programadas.current() ==
j+1:#Volver
                                                quit_Panel_Control()
                                        else:

                                                Etiquetas_Edit_Acciones = [u"Enviar esta
Accion",u"Editar Accion",u"Eliminar Accion"]
                                                Op_Etiquetas_Edit_Acciones =
appuifw.popup_menu(Etiquetas_Edit_Acciones, u"Seleccione una accion:")

                                                if Op_Etiquetas_Edit_Acciones == 0 :#Enviar
esta Accion

        Sent_PSerie(password+Opc_Ver_Acciones_Programadas[Menu_Ver_Acciones_Pr
ogramadas.current()][1])
                                                    appuifw.note(u"Enviado:
"+Opc_Ver_Acciones_Programadas[Menu_Ver_Acciones_Programadas.current()][1],
"info")
                                                elif Op_Etiquetas_Edit_Acciones == 1
:#Editar una accion
                                                    if globalui.global_query(u"Desea editar
esta accion?") == True:
                                                        global j
                                                        global Aux_p
```

```
                                                    global Aux_m
                                                    global Aux_v
                                                    global Aux_a
                                                    global Aux_t
                                                    global Aux_apagado
                                                    global Aux_J
                                                    Aux_J = j
                                                    j =
Menu_Ver_Acciones_Programadas.current()#Para sobreescribir en la posicion deseada
                                                    if not
int(Opc_Ver_Acciones_Programadas[Menu_Ver_Acciones_Programadas.current()][1][0])
==2:#No es una accion de usuario

        Aux_p=int(Opc_Ver_Acciones_Programadas[Menu_Ver_Acciones_Programadas.c
urrent()][1][2:4])#pin
                                                            if Aux_p <8:
                                                                    Aux_p = Aux_p-
1
                                                            elif Aux_p <35:
                                                                    Aux_p = Aux_p-
14
                                                            else:
                                                                    Aux_p = Aux_p-
19



        Aux_m=int(Opc_Ver_Acciones_Programadas[Menu_Ver_Acciones_Programadas.
current()][1][1])#Sacamos el MODO

        Aux_v=int(Opc_Ver_Acciones_Programadas[Menu_Ver_Acciones_Programadas.c
urrent()][1][4:8])#Sacamos el Valor
                                                            Aux_a=0
                                                            Aux_t=0
                                                            if Aux_m==0 or
Aux_m==3:#Si es un rele o una alarma ver si tiene temporizacion asociada

        Aux_apagado=int(Opc_Ver_Acciones_Programadas[Menu_Ver_Acciones_Progra
madas.current()][1][8:12])#Sacamos el tempor
                                                            else:
                                                                    Aux_apagado=0
                                                    else:#Accion de usuario
                                                            Aux_p=0
                                                            Aux_m=0
                                                            Aux_v=0
                                                            Aux_apagado=0

        Aux_a=int(Opc_Ver_Acciones_Programadas[Menu_Ver_Acciones_Programadas.c
urrent()][1][1])-4#Sacamos Accion
                                                            Aux_t=0
```

```
                                                    del
Opc_Ver_Acciones_Programadas[Menu_Ver_Acciones_Programadas.current()]

        Programar_acciones(Aux_p,Aux_m,Aux_v,Aux_a,Aux_t,Aux_apagado)

                                                    global error_flag
                                                    if error_flag==1:
                                                            appuifw.note(u"Accion
modificada con exito", "conf")
                                                            error_flag=0
                                                            j=Aux_J
                                                    else:
                                                            error_flag=0

        Menu_Panel_Control_Acciones_Programadas()

                                                elif Op_Etiquetas_Edit_Acciones == 2
:#Eliminar una accion
                                                        if globalui.global_query(u"Desea
eliminar esta accion?") == True:
                                                    del
Opc_Ver_Acciones_Programadas[Menu_Ver_Acciones_Programadas.current()]
                                                    #global j
                                                    j=j-1

        Menu_Panel_Control_Acciones_Programadas()
                                                    appuifw.note(u"Accion
eliminada con exito", "conf")

                                appuifw.app.exit_key_handler = quit_Panel_Control
                                Menu_Ver_Acciones_Programadas =
appuifw.Listbox(Opc_Ver_Acciones_Programadas,
Acciones_Ver_Acciones_Programadas)
                                appuifw.app.body = Menu_Ver_Acciones_Programadas

                        Menu_Panel_Control_Acciones_Programadas()

                elif Menu_Panel_Control.current() == 3:#Borrar configuracion
                        if globalui.global_query(u"Desea regresar a la configuracion de
fabrica") == True:
                                Sent_PSerie(password+"20100000000")
                                appuifw.note(u"Se ha regresado a la confirguracion de
fabrica con exito", "conf")

                        #Pin_delete = appuifw.query(u"Selecciona el pin", 'number')

                elif Menu_Panel_Control.current() == 4:#Enviar
                        #Sent_PSerie(password+"10141900000")
                        #appuifw.note(u"Enviado conf sensor", "info")
                        if Bt_Start == 0:#Bluetooth no activa
```

```
                                appuifw.note(u"Es necesario establecer comunicacion
primero", "error")
                        elif len(Opc_Ver_Acciones_Programadas)>3:
                                for x in range(j):

        Sent_PSerie(password+Opc_Ver_Acciones_Programadas[x][1])
                                        appuifw.note(u"Enviado:
"+password+Opc_Ver_Acciones_Programadas[x][1], "info")
                                Sent_PSerie(password+"32000000000")
                                appuifw.note(u"Enviado: "+password+"32000000000",
"info")#Codigo de clausura de acciones
                                if globalui.global_query(u"Desea eliminar todas las acciones
programadas") == True:
                                        while j > 1:
                                                del Opc_Ver_Acciones_Programadas[1]
                                                j=j-1
                                        appuifw.note(u"Todas las acciones ha sido borradas
con exito", "conf")
                        else:
                                appuifw.note(u"Es necesario establecer la configuracion del
sensor", "error")

                elif Menu_Panel_Control.current() == 5:#Volver
                        quit_favorito()
                else:
                        appuifw.note(u"Error", "error")
        appuifw.app.exit_key_handler = quit_favorito
        Menu_Panel_Control = appuifw.Listbox(Opc_Panel_Control,
Acciones_Panel_Control)
        appuifw.app.body = Menu_Panel_Control



#                                       #-----------------#-----------------#


#                                       #-----------------#-----------------#


def Opciones_avanzadas():
        Etiquetas_Op_Av = [u"Modo manual", u"Altavoz", u"Reinicia sistema",
u"Eliminar acciones programadas"]
        Op_Av = appuifw.popup_menu(Etiquetas_Op_Av, u"Seleccione una accion:")
        if Op_Av == 0 :#Modo manual
                if Bt_Start == 1:
                        Orden = appuifw.query(u"Codigo a enviar", 'number')
                        if Orden<99999999999:
                                if Orden!="None":
                                        if globalui.global_query(u"Desea enviar
"+password+str(Orden)) == True:
```

```
                                        Sent_PSerie(password+str(Orden))
                                        globalui.global_note(u"Enviado:
"+password+str(Orden), 'conf')
                        else:
                                globalui.global_note(u"Codigo introducido no valido",
'error')
                else:
                        appuifw.note(u"Es necesario establecer comunicacion primero",
"error")
        elif Op_Av == 1 :#Altavoz
                if Bt_Start == 1:
                        Sent_PSerie(password+"03090700000")
                        appuifw.note(u"Orden enviada", "conf")
                else:
                        appuifw.note(u"Es necesario establecer comunicacion primero",
"error")
        elif Op_Av == 2 :#Reiniciar placa
                if Bt_Start == 1:
                        Sent_PSerie(password+"20000000000")
                        appuifw.note(u"Orden enviada", "conf")
                else:
                        appuifw.note(u"Es necesario establecer comunicacion primero",
"error")
        elif Op_Av == 3:#Eliminar acciones programadas
                if Bt_Start == 1:
                        Sent_PSerie(password+"20100000000")
                        appuifw.note(u"EEPROM borrada. Todas la acciones ha sido
eliminadas con exito", "conf")
                else:
                        appuifw.note(u"Es necesario establecer comunicacion primero",
"error")


#                                       #-----------------#-----------------#


#-------------------------------------------------------------------------------------------------
#                                                       Pestanna Bluetooth
#-------------------------------------------------------------------------------------------------
OpcBt = [u"Conectar", u"Favoritos", u"Monitorizar", u"Panel de control", u"Opciones
avanzadas", u"Salir"]

def handler_OpcBt ():
        if app1.current() == 0:
                connect()
        elif app1.current() == 1:
                Favoritos()
        elif app1.current() == 2:
                if Bt_Start == 0:#Bluetooth no activa
```

```
                        appuifw.note(u"Es necesario establecer comunicacion primero",
"error")
                else:
                        Rutina_Monitorizar()
                        Monitorizar()
        elif app1.current() == 3:
                #appuifw.note(u"Proximamente", "info")
                Panel_Control()
        elif app1.current() == 4:
                Opciones_avanzadas()
        elif app1.current() == 5:
                exit_key_handler()

        else:
                appuifw.note(u"Error", "error")


app1 = appuifw.Listbox(OpcBt, handler_OpcBt)

#-------------------------------------------------------------------------------------------------
#                                                       Pestanna Configuracion
#-------------------------------------------------------------------------------------------------
Opc_Pestanna_Conf = [u"Cambiar password: "+password, u"Guardar datos GPS:",
u"Guardar datos SD: "]
Etiquetas_Opc_Pestanna_Conf = [u"20 seg",u"40 seg",u"1 min",u"5 min",u"15 min", u"30
min",u"60 min",u"90 min",u"120 min",]
Valor_configuracion=[u"0",u"0"]
def Menu_conf():
        global Bt_Start
        global app2
        global password
        global Once_2
        CON_DIR="e:/"
        CON_FILE="Configuracion.txt"
        if os.path.isfile(CON_DIR+CON_FILE) == True and Once_2==0:
                for G in range(3):
                        del Opc_Pestanna_Conf[0]
                for AK in range(2):
                        del Valor_configuracion[0]

                TXT_Conf=open(CON_DIR+CON_FILE,'rt')
                contentL = TXT_Conf.read()
                config=eval(contentL)
                TXT_Conf.close()

                Opc_Pestanna_Conf.insert(0, (u"Cambiar password:
"+config.get("Conf_0")))
                Opc_Pestanna_Conf.insert(1, (u"Guardar datos GPS:
"+Etiquetas_Opc_Pestanna_Conf[int(config.get("Conf_1"))]))
```

```
                Opc_Pestanna_Conf.insert(2, (u"Guardar datos SD:
"+Etiquetas_Opc_Pestanna_Conf[int(config.get("Conf_2"))]))

                password=config.get("Conf_0")
                Valor_configuracion.insert(1, (u""+config.get("Conf_1")))
                Valor_configuracion.insert(2, (u""+config.get("Conf_2")))
                Once_2=1


        def handler_OpcConfi ():
                if app2.current() == 0:
                        if Bt_Start == 1:
                                NEW_Pass = appuifw.query(u"Nueva Password", 'number')
                                if      NEW_Pass:
                                        if NEW_Pass<9999:
                                                if globalui.global_query(u"Cambiar password
por: "+str(NEW_Pass)+"\nEs correcto?") == True:
                                                        if NEW_Pass<10:

        password="000"+str(NEW_Pass)
                                                        elif NEW_Pass<100:
                                                                password="00"+str(NEW_Pass)
                                                        elif NEW_Pass<1000:
                                                                password="0"+str(NEW_Pass)
                                                        else:
                                                                password=str(NEW_Pass)
                                                        del Opc_Pestanna_Conf[0]

        Opc_Pestanna_Conf.insert(0,(u"Cambiar password: "+password))
                                                        CON_DIR="e:/"

        CON_FILE=os.path.join(CON_DIR,'Configuracion.txt')
                                                        con_M={}
                                                        con_M["Conf_0"]= password
                                                        for x in range(2):
                                                                con_M["Conf_"+str(x+1)]=
Valor_configuracion[x]
                                                        #con_M["Conf_1"]= str(Opc_Conf+1)
                                                        TXT_Conf=open(CON_FILE,'wt')
                                                        TXT_Conf.write(repr(con_M))
                                                        TXT_Conf.close()

        Sent_PSerie(password+"2001"+password+"000")
                                                        appuifw.note(u"Password cambiada
con exito", "conf")
                                        else:
                                                appuifw.note(u"El password no dispone de la
longitud correcta", "error")
                                Menu_conf()
                        else:
```

```
                        appuifw.note(u"Es necesario establecer comunicacion
primero", "error")

            elif app2.current() == 1:
                global password
                if Bt_Start == 1:
                    Opc_Conf =
appuifw.popup_menu(Etiquetas_Opc_Pestanna_Conf, u"Seleccione una opcion:")
                    del Opc_Pestanna_Conf[1]
                    del Valor_configuracion[0]
                    Opc_Pestanna_Conf.insert(1,(u"Guardar datos GPS:
"+Etiquetas_Opc_Pestanna_Conf[Opc_Conf]))
                    Valor_configuracion.insert(0,(u""+str(Opc_Conf)))

                    CON_DIR="e:/"
                    CON_FILE=os.path.join(CON_DIR,'Configuracion.txt')
                    con_M={}
                    con_M["Conf_0"]= password
                    for x in range(2):
                        con_M["Conf_"+str(x+1)]= Valor_configuracion[x]
                    #con_M["Conf_1"]= str(Opc_Conf+1)
                    TXT_Conf=open(CON_FILE,'wt')
                    TXT_Conf.write(repr(con_M))
                    TXT_Conf.close()

                    Sent_PSerie(password+"2002"+str(Opc_Conf+1)+"0000")
                    appuifw.note(u"Tiempo de grabacion del GPS ha sido
cambiado con exito", "conf")
                else:
                    appuifw.note(u"Es necesario establecer comunicacion
primero", "error")

                Menu_conf()
            elif app2.current() == 2:
                global password
                if Bt_Start == 1:
                    Opc_Conf =
appuifw.popup_menu(Etiquetas_Opc_Pestanna_Conf, u"Seleccione una opcion:")
                    del Opc_Pestanna_Conf[2]
                    del Valor_configuracion[1]
                    Opc_Pestanna_Conf.insert(2,(u"Guardar datos SD:
"+Etiquetas_Opc_Pestanna_Conf[Opc_Conf]))
                    Valor_configuracion.insert(0,(u""+str(Opc_Conf)))

                    CON_DIR="e:/"
                    CON_FILE=os.path.join(CON_DIR,'Configuracion.txt')
                    con_M={}
                    con_M["Conf_0"]= password
                    for x in range(2):
                        con_M["Conf_"+str(x+1)]= Valor_configuracion[x]
```

```
                                #con_M["Conf_1"]= str(Opc_Conf+1)
                                TXT_Conf=open(CON_FILE,'wt')
                                TXT_Conf.write(repr(con_M))
                                TXT_Conf.close()
                                Menu_conf()
                                Sent_PSerie(password+"2003"+str(Opc_Conf+1)+"0000")
                                appuifw.note(u"Tiempo de grabacion en la MicroSD ha sido
cambiado con exito", "conf")
                        else:
                                appuifw.note(u"Es necesario establecer comunicacion
primero", "error")
                else:
                        appuifw.note(u"Error", "error")

        app2 = appuifw.Listbox(Opc_Pestanna_Conf, handler_OpcConfi)
        appuifw.app.body = app2
#-------------------------------------------------------------------------------------------------------
#                                                       Menu Principal
#-------------------------------------------------------------------------------------------------------
# Creamos un controlador que cambie las opciones en funcion de la pestanna
def Seleccion_Pestanna(index):
   if index == 0:
      appuifw.app.body = app1 # Pestanna Bluetooth
   if index == 1:
      Menu_conf()

#Opciones menu Opciones de la parte inferior izquierda
appuifw.app.menu = [(u"Acerca de", Acercade), (u"Salir", exit_key_handler)]

app_lock = e32.Ao_lock() # Creamos un objeto activo
appuifw.app.set_tabs([u"Home", u"Configuracion"],Seleccion_Pestanna) # Creamos las
pestannas de la aplicacion
appuifw.app.title = u'GNX Project v.1' #Establecemos el nombre de la aplicacion
appuifw.app.body = app1 # Pestanna que esta activa por defecto



appuifw.app.screen = 'normal'
appuifw.app.exit_key_handler = exit_key_handler
app_lock.wait()
```

Apéndice C

Manual de usuario del sistema GNX Project v.1

C.1. Descripción del menú principal

El menú principal de la aplicación para Smartphones cuenta con dos pestañas: **Home** y **Configuración** (ver figuras C.1 A y B). La primera contiene las opciones para lograr un control de todas las funcionalidades del proyecto. La segunda incluye los parámetros que pueden ser editados por el usuario y que rigen el funcionamiento del sistema.

Figura C.1: Menú principal

Dispone de un menú adicional al que se puede acceder por medio del botón "**Option**" situado en la esquina inferior izquierda (ver figura C.1 C). Este menú contine dos opciones: **Acerca de…** donde se indica el nombre de autor del proyecto (ver figura C.1 D), y **Salir**.

La pestaña **Home**, observamos las siguientes opciones que se detallaran los siguientes aparatados:

- **Conectar**: Mediante esta opción el posible vincular la aplicación con el módulo hardware por medio del Bluetooth, si este no se encuentra activo, saltará una ventana de diálogo preguntándonos si deseamos activarlo. Para ello, nos saldrá una lista con los últimos dispositivos conectados (ver figura C.1 E). Para encontrar el módulo la primera vez, deberemos pulsar la opción: "**Más dispositivos**". Una vez finalizada la búsqueda debemos seleccionar aquella que se designe por "linvor". Nos pedirá una contraseña para vincularse la cual viende dada por el fabricante, esta es "1234". Una vez realizado esto, sólo debemos establecer el puerto de comunicación de la lista emergente (ver figura C.1 F) y si la conexión se ha realizado con éxito deberá aparecer una notificación al respecto.

- **Favoritos**: En esta sección, el usuario podrá definir aquellas acciones que emplee con mayor frecuencia.

- **Monitorizar**: Permite visualizar los datos transmitidos desde el módulo hardware por Bluetooth referentes al estado de los pines configurados, así como los datos referentes al GPS.

- **Panel de Control**: Menú que proporciona todo lo necesario para configurar el módulo hardware.

- **Opciones avanzadas**: Contiene una serie de funcionalidades consideradas de interés para el usuario.

- **Salir**: Finaliza la comunicación Bluetooth y cierra la aplicación.

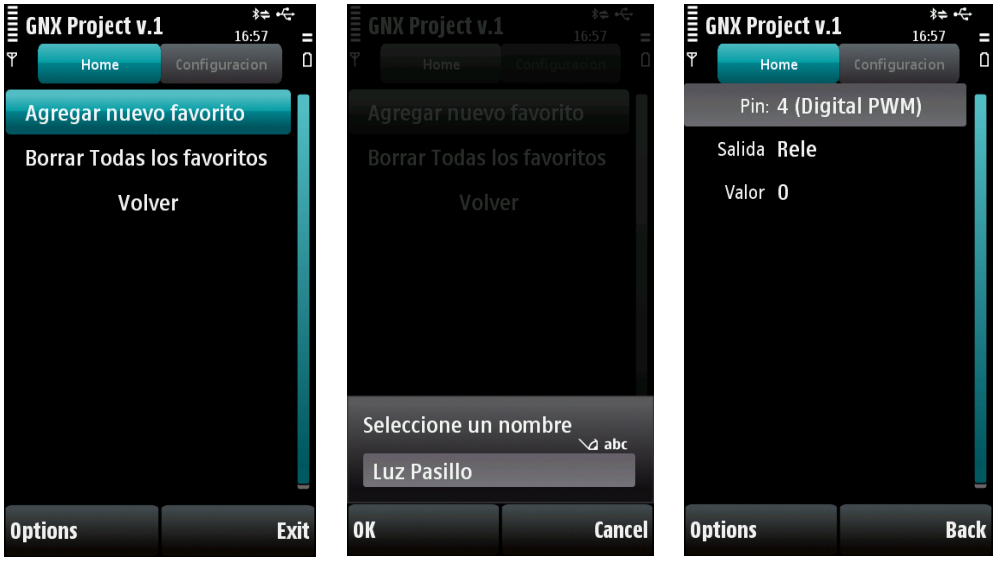
En lo referente a la pestaña Configuración, encontramos las siguientes opciones:

- **Cambiar password**.
- **Guardar datos GPS**: Donde especificamos cada cuanto tiempo queremos que se graben los datos del satélite en la tarjeta MicroSD del módulo. Este periodo de tiempo va desde los 20 segundos hasta los 120 minutos.
- **Guardar datos SD**: Permite seleccionar el espacio de tiempo entre dos grabaciones de datos en la tarjeta de memoria del estado de los pines del módulo.

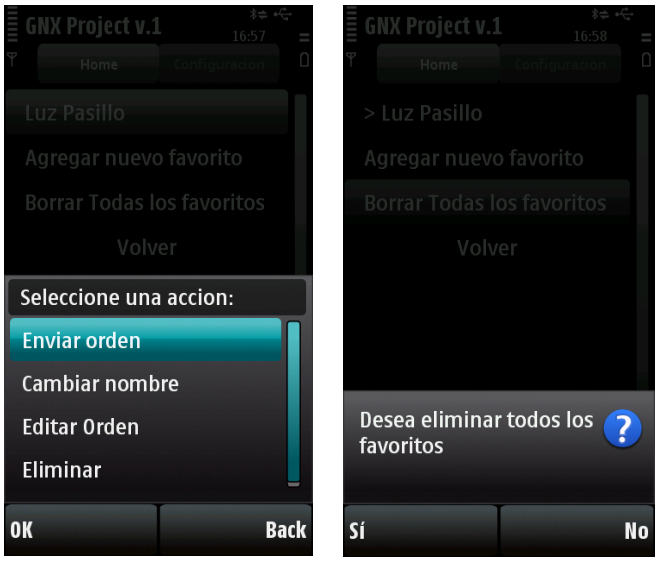
C.2. Menú Favoritos

Como hemos mencionado, el este menú podremos añadir las acciones que deseamos ejecutar con regularidad. En este menú encontramos con tres opciones de partida: **Agregar nuevo favorito, Borrar todos los favoritos** y **Volver**. (Ver figura C.2 A).

Para añadir uno, simplemente debemos pulsar la opción "**Agregar nuevo favorito**". Tras esto deberemos seleccionar el nombre (ver figura C.2 B) que queremos que aparezca en el menú principal para a continuación definir la acción asociada mediante el menú de la figura C.2 C.



(A) (B) (C)



(D) (E)

Figura C.2: Menú de Favoritos

Una vez creado, es posible realizar varias acciones sobre éste simplemente pulsando sobre él. Nos aparecerá un menú como el de la figura C.2 D donde podremos realizar las siguientes operaciones:

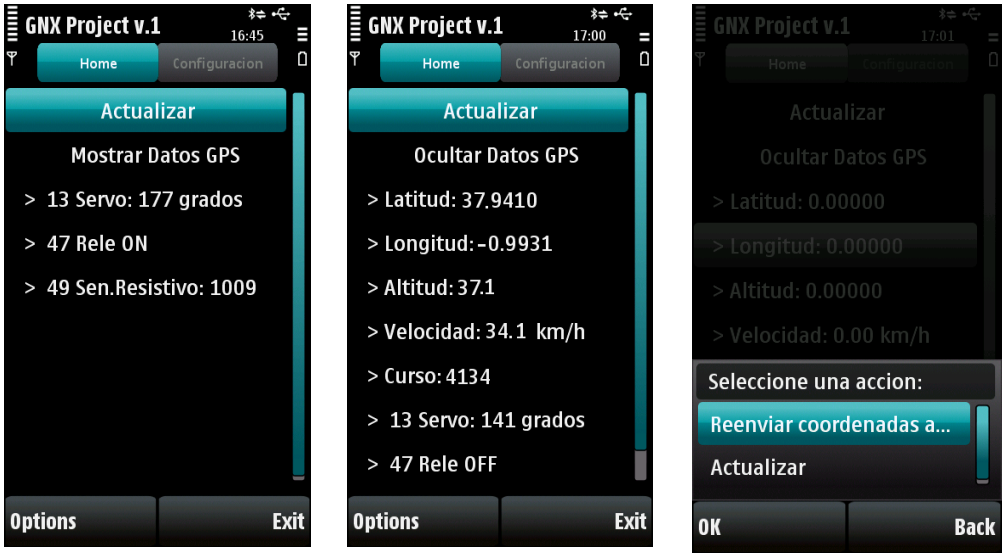
- **Enviar orden**: Mediante esta opción tenemos la posibilidad de enviar la instrucción que hemos guardado como favoritos. En el caso de servos, motores, altavoces y operaciones de modulación de anchura de pulso, permitirá seleccionar el valor que deseamos poner a la salida de dichos elementos.
- **Cambiar nombre**
- **Editar orden**: Posibilita redefinir los parámetros de la instrucción almacenada en este menú.
- **Eliminar favorito**

Frente a esta última opción, tenemos la posibilidad de eliminar todos los favoritos creados por medio de la opción que dispone este menú. En este caso se solicitará confirmación antes de realizar ninguna acción como observamos en la figura C.2 D.

C.3. Menú Monitorizar

En esta sección, tenemos la posibilidad de visualizar toda la información que nos proporciona el módulo que hemos creado para este proyecto.

Una vez establecida la comunicación con el dispositivo (de lo contrario saldría un error avisándonos de este hecho y que no nos permitiría acceder a este menú) el Smartphone realiza una petición de información al módulo el cual responde vía Bluetooth con el estado de todos los pines configurados, entendiéndose como tales aquellos de los que el sistema tenga constancia de qué hay conectado a ellos. También se reciben las coordenadas del GPS, siendo su valor de 0 mientras no se haya logrado establecer conexión con el satélite.



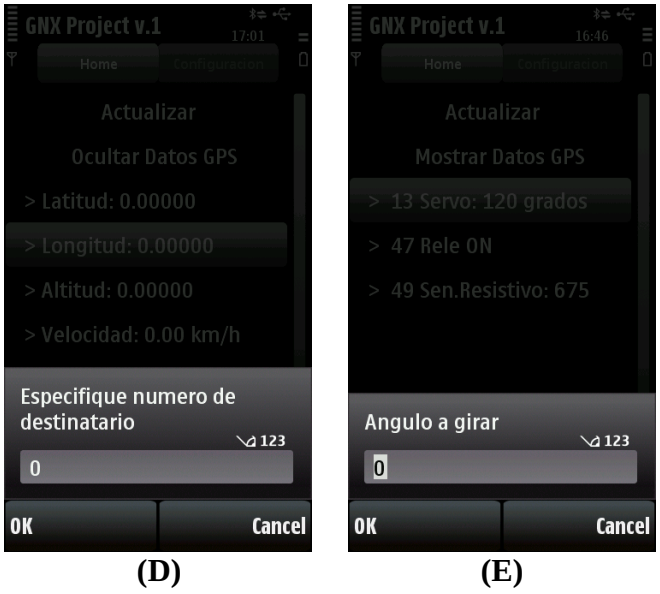


Figura C.3: Menú Monitorizar

A fin de exponer de una forma clara la información, se han excluido de ser mostrados aquellos pines que no se encuentran configurados, pese a que estos son también transmitidos. Como observamos en la figura C.3 A, los datos se exponen de forma que se indica el pin, lo que hay conectado y el estado o valor del mismo. La actualización de dichos datos se realiza previa petición por parte del usuario por medio de la opción "**Actualizar**" a fin de no sobrecargar al sistema y el tráfico de datos cuando se sean realizar otras tareas.

Los datos del GPS permanecen inicialmente ocultos al usuario para no mostrar más información que la que necesita el usuario. Si se pulsa la opción "**Mostrar Datos GPS**", aparecerá en la pantalla junto con los datos de los pines (si los hay) la longitud, altitud, latitud, velocidad y curso como podemos observar en la figura C.3 B. Estando estos datos mostrados en pantalla podremos actualizarlos así como reenviarlos por SMS al destinatario que designemos con simplemente seleccionar la opción "**Reenviar coordenadas a…**" del menú emergente de la figura C.3 C que resulta de pulsar alguna de los datos del GPS. Simplemente debemos especificar el destinatario (ver ifigura C.3 D) y estos datos serán enviado automáticamente siempre y cuando el teléfono sea válido, en caso contrario se notificará este hecho impidiéndose el envío.

Regresando a la información de los pines, tenemos la posibilidad de manipular el valor de las salidas de los actuadores. Con sólo pulsar sobre una de estas salidas aparecerá una interface que se adecuará al elemento a manipular pudiendo por ejemplo variar el ángulo de giro del servo. En el caso de relés estos no disponen de interface al pulsarlo, sino que cambia de estado, es decir, pasa de On a Off o viceversa, **notificándose por voz** este hecho. Después de cada cambio, se actualiza la pantalla de manera automática.

C.4. Menú Panel de Control

Este es el apartado más importante de los que componen la aplicación. Es en éste menú donde se define la configuración de los sensores y actuadores así como la asociación entre estos mediante una serie de opciones como apreciamos en la figura C.4 A. Para simplificar el proceso, se diferencia entre configuración de sensores y configuración de acciones, proporcionando en cada una de ellas opciones distintas.

Cuando programamos una tarea, esta debe ejecutarse en base al cumplimiento de una determinada condición que normalmente va referida a un determinado sensor (o no como es el caso del **Sensor Independiente**). Mediante el menú de la figura C.4 B podemos definir esta condición, seleccionando el pin de los 50 disponibles (a los cuales se le asocian si se tratan de pines digital, analógicos o si permiten PWM), el tipo de sensor (ver figura C.4 C) y la condición a cumplir.

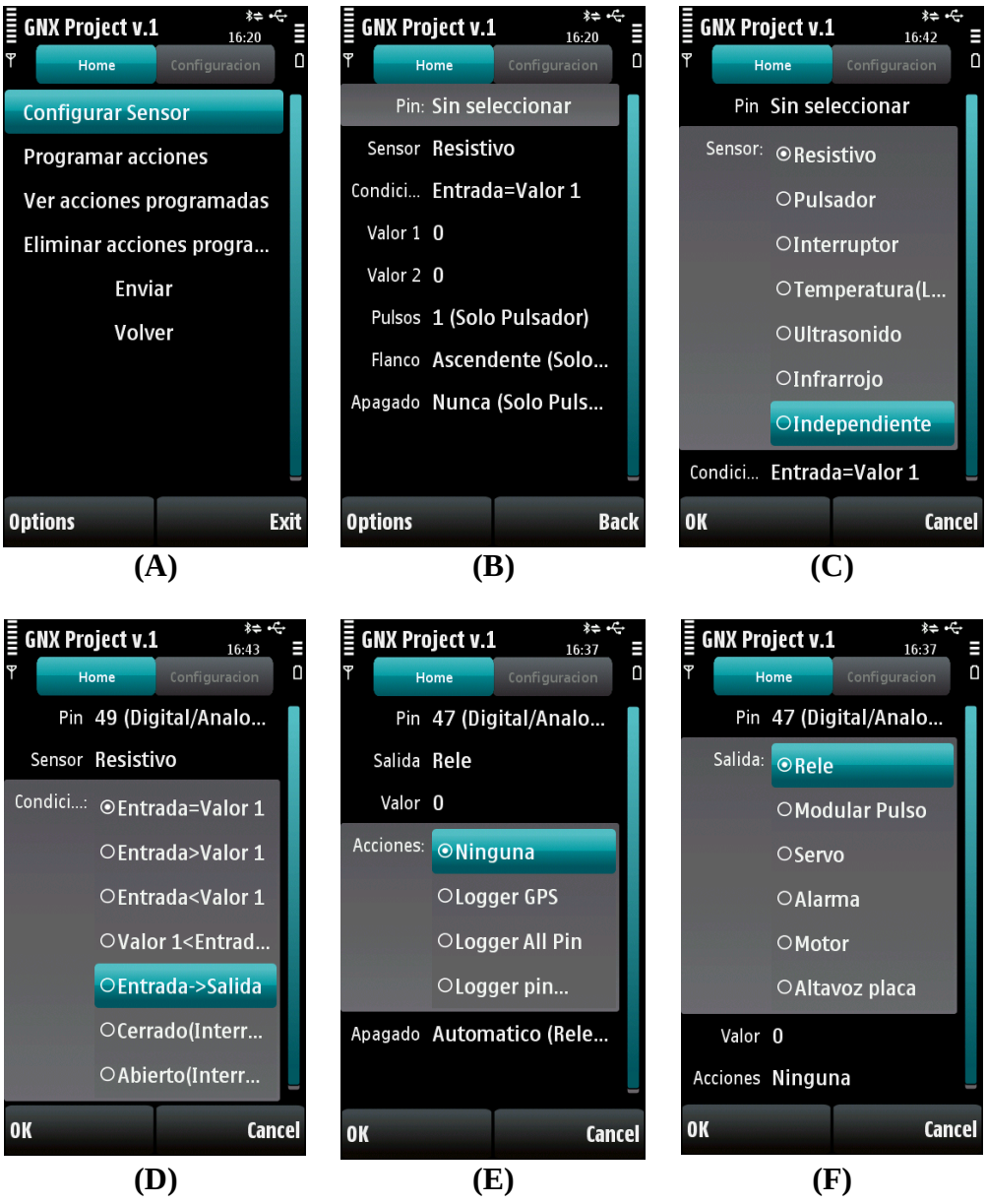


Figura C.4: Menú Panel de Control. Configurar Sensor.

Como apreciamos en la figura C.4 D existen varias opciones a elegir, siendo las tres últimas las que merecen una explicación más en profundidad:

- **Entrada→Salida**: Mediante esta opción podemos hacer que el valor presente a la entrada, es decir, el valor del sensor del tipo y pin indicado se transfiera directamente una o más salidas que definiremos en el submenú "**Programar Acciones**". Esto permite por ejemplo, mover un servo en función de la cantidad de luz detectada por una LDR, variar la velocidad de un motor de corriente continua en función de la distancia que detecte un sensor de ultrasonido.

- **Cerrado (Sólo interruptor)**: Condición de uso exclusivo con interruptores donde se establece como condición que se encuentre cerrado. Esto puede ser utilizado para detectar si una puerta se encuentra abierta o no y en consecuencia hacer sonar una alarma por ejemplo.

- **Abierto (Sólo interruptor)**: Igual que el punto anterior pero la condición a evaluar es "estar abierto".

Otras opciones que se pueden observar en la figura C.4 B son las siguientes:

- **Pulsos**: De uso exclusivo con los pulsadores, permite definir el número de veces que deberán ser pulsados para que se ejecuten las tareas asociadas. Por motivos de respetar la estructura de la trama descrita en la tabla 4.1, el valor de las pulsaciones va de 1 a 9.

- **Flanco**: De uso exclusivo con los pulsadores, define si se desea considerar el flanco de subida o bajada.

- **Apagado**: De uso exclusivo con los pulsadores, indica cuando se debe reiniciar el conteo interno a fin de poder volver a ejecutar las acciones asociadas a este.

Ahora, una vez definida la parte del sensor, debemos indicar que acciones queremos asociar a éste. Para ello debemos dirigirnos a la segunda opción: **Programar acciones**.

En ésta podemos observar (ver figura C.5 A) que dispone opciones muy parecidas al menú anterior. Estas son:

- **Pin**: Al igual que la opción anterior, tenemos la posibilidad de escoger entre los 50 pines que incorpora el proyecto. Como se aprecia en la figura C.5 B, se encuentra asociado a cada uno de ellos una descripción.

- **Salida**: Permite seleccionar entre los actuadores que pueden ser utilizados para este proyecto (ver figura C.5 A). Estos son: **Relé, Modular Pulso, Servo, Alarma, Motor y Altavoz placa**. Cabe mencionar que en caso de Alarma y Altavoz placa, el tratamiento es el mismo a diferencia de que

para este último no se precisa de seleccionar el pin ya que este está predefinido.

- **Valor**: Representa el valor que se podrá en la salida. En el caso del servo indicará su ángulo de giro o si se trata de un altavoz, servirá para especificar la frecuencia a la que se desea que opere. Este dato debe ser como máximo de tres dígitos, donde en caso de excederse, el programa asignará automáticamente el valor más alto admisible.

- **Acciones**: Aumentan las opciones del proyecto proporcionando funcionalidades extras como podemos observar en la figura C.5 C:

    - **Logger GPS**: Permite la captura de los dados de satélite en la tarjeta MicroSD.
    - **Logger All pin**: Mediante esta opción podemos registrar el estado de todos los pines de la placa que se encuentren configurados. Se almacenarán en la siguiente ruta de la tarjeta: **e://SENSORES/SENSORES.txt.**
    - **Logger pin…**: Es igual que la opción anterior pero permite registrar un único pin, creándose en la carpeta **SENSORES** un archivo .txt con el nombre del pin a registrar.

(A) **(B)** **(C)**

Figura C.5: Menú Panel de Control. Programar acciones.

Esta operación se repetirá para cada una de las acciones a asociar. Todas ellas se almacenan en un buffer de salida incluida la que define el sensor y su condición. Este buffer es accesible por medio de la opción: **Ver acciones programadas**. Como podemos observar en la figura C.6 A, únicamente se muestra la información más elemental de lo que hemos programado, sobre todo en lo referente a los sensores, donde sólo se indica el tipo de sensor conectado y el pin donde se ubica. También podemos apreciar que aparece un código de 11 dígitos justo debajo de cada descripción. Este es el código (a falta de la contraseña, de ahí que sean 11 caracteres en vez de 15) que se define esa trama. Los motivos de visualizarlo son para comprobar el correcto funcionamiento de la aplicación.

Sin embargo, si pulsamos sobre esta opción, nos aparecerá un menú emergente como el de la figura C.6 B que nos brindará entre otras opciones la posibilidad de ampliar esta información. Para ello debemos seleccionar: **Ver detalles del sensor**.

Esto nos remitirá a una pantalla como la de figura C.6 C donde vemos a parte de lo ya indicado en el menú anterior, la condición que se debe de cumplir.

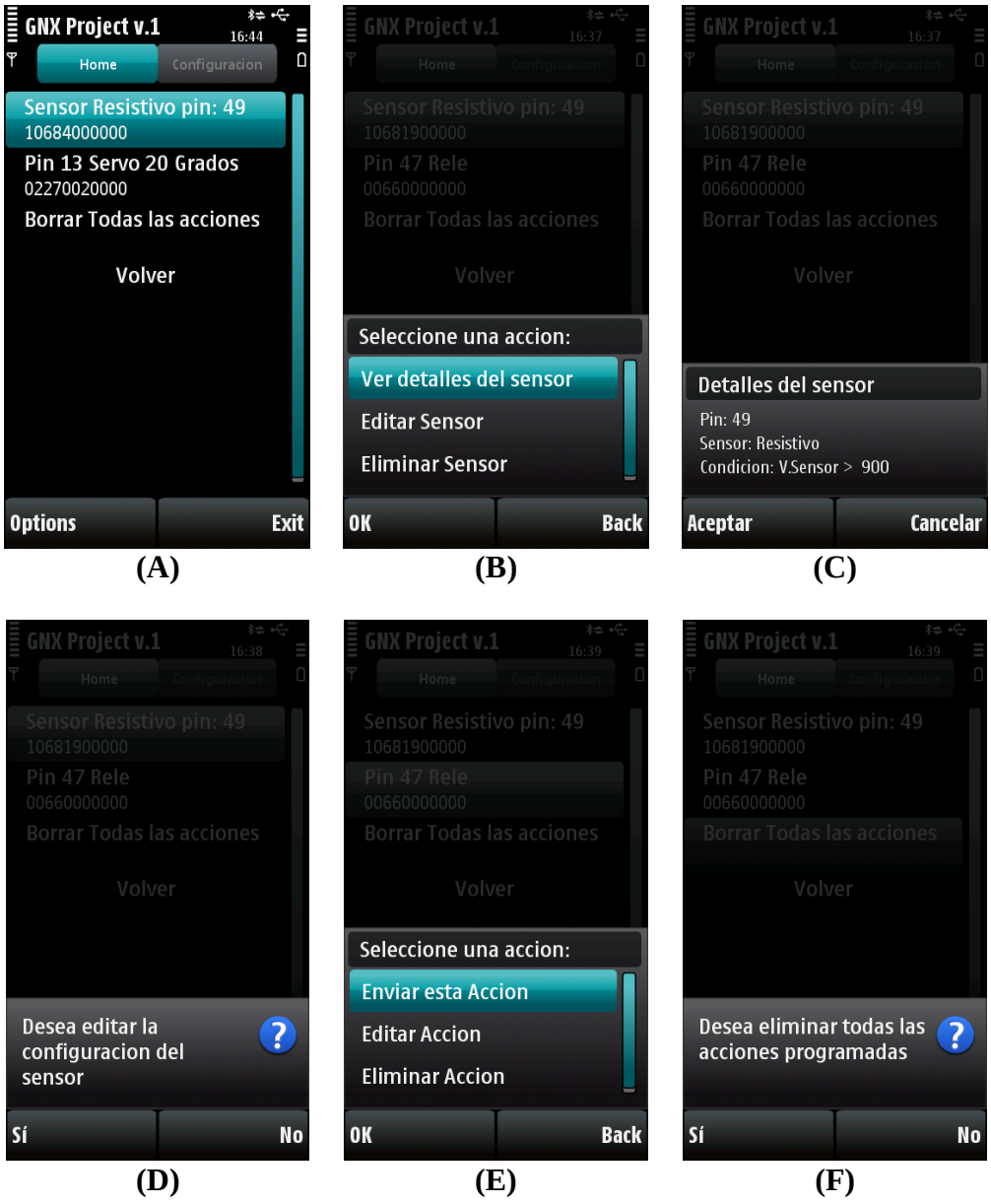


Figura C.6: Menú Panel de Control. Ver acciones programadas.

Las otras opciones de este menú permiten: Editarlo (solicitándonos confirmación, nos permitirá redefinir las trama del sensor) y eliminarlo. Si no se dispone de la trama del sensor, la aplicación impedirá el envío de todas las instrucciones al módulo hardware.

Para las acciones, también disponemos de un menú emergente que resulta de pulsar sobre alguna de ellas (ver figura C.6 E). Este dispone de las siguientes opciones:

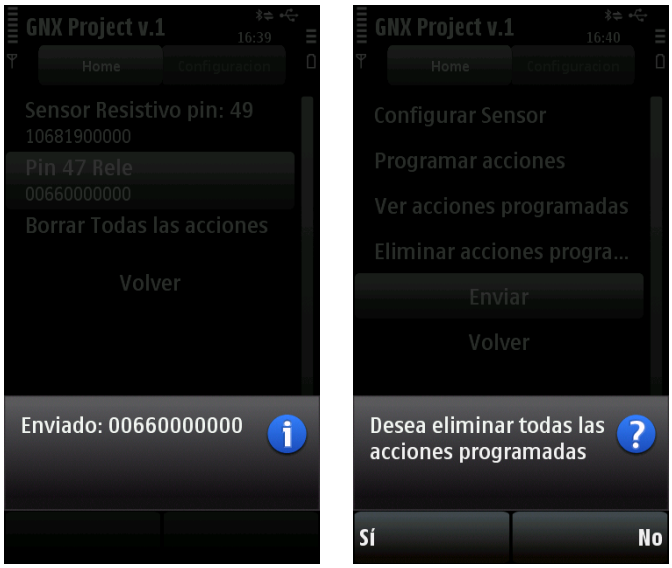
- **Enviar esta Acción**: Permite enviar una trama como acción inmediata sin necesidad de enviar el resto.
- **Editar Acción**: Solicitará confirmación antes de editar la acción.
- **Eliminar Acción**: Solicitará confirmación antes de eliminar.

Para concluir con la opción Ver acciones programadas, solo mencionar que disponemos al igual que ocurría en el menú Favoritos, de la posibilidad de eliminar todas las acciones pulsando un solo botón: **Borrar todas las acciones**. Solicitando confirmación en ese caso como se aprecia en la figura C.6 F.

Las dos opciones que quedan por definir del menú Panel de Control son:

- **Eliminar acciones programadas**: Mediante esta opción podemos borrar la EEPROM de la placa Arduino y en consecuencia eliminar todas las acciones programadas. Cuando ocurre esta acción, los led indicadores del panel principal empiezan a parpadear de manera alternativa.

- **Enviar**: Mediante esta opción podemos cargar todo el contenido configurado. Durante la descarga de datos, se va visualizando todo aquello que se transmite (ver figura C.7 A) a fin de garantizar su correcto funcionamiento. Cuando trasmite por último lugar la instrucción "**Fin de acciones programadas**" (Contraseña+ "32000000000"). Tras esto nos preguntará si deseamos o no eliminar todo lo tenemos en "**Ver acciones programadas**".



(A) **(B)**

Figura C.7: Menú Panel de Control. Enviar.

C.5 Menú Opciones avanzadas

Para concluir con la pestaña **Home**, debemos hablar acerca del menú **Opciones avanzadas** (ver figura C.8 A). Al pulsar sobre la opción del menú principal aparece un menú emergente con las siguientes opciones destinadas a proporcionar algunas herramientas al usuario:

- **Modo manual**: Permite introducir un código de instrucción "a mano", es decir, por medio un cuadro de texto como el de la figura C.8 B.

- **Altavoz**: Mediante esta opción podemos encender/apagar el altavoz que incorpora el módulo hardware preseleccionándose una frecuencia de 700 Hz.

- **Reiniciar sistema**: Con la ayuda del transistor descrito en la figura 4.9 podemos provocar el reset de la placa diseñada de forma remota. Cuando esto ocurra, el sistema avisará mediante el parpadeo de los dos led de forma conjunta.

- **Eliminar acciones programadas**: Misma opción que la del menú **Panel de Control** que permite borrar la EEPROM y consigo todas las acciones programadas.

(A) **(B)**

Figura C.8: Menú Opciones avanzadas.

C.6 Pestaña Configuración

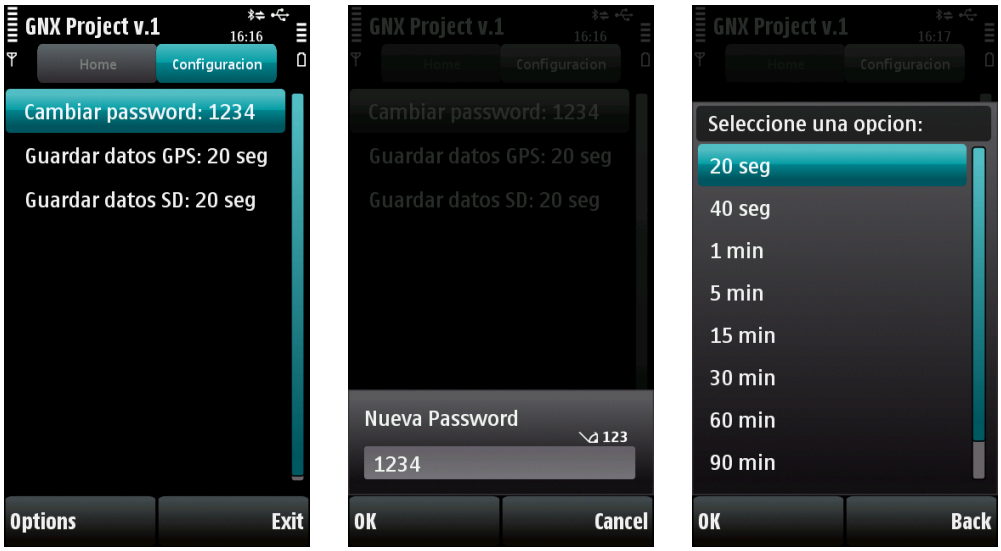
Una vez descrita la pestaña **Home** y todo su contenido, nos queda por comentar el menú de configuración.

Este consta de tres opciones que pueden observarse en la figura C.9 A:

- **Cambiar password**: Permite seleccionar una contraseña nueva para el sistema. Para ello sólo deberemos pulsar sobre él y nos aparecerá un cuadro de texto donde la introduciremos. (Ver figura C.9 B).

- **Guardar datos GPS**: Sirve para especificar cada cuánto tiempo se desea hacer el registro de los datos del GPS en la MicroSD. Esta temporización se encuentra tabulada, donde las opciones a elegir son las que se muestran en la figura C.9 C.

- **Guardar datos SD**: Permite definir el espacio de tiempo que separa a dos grabaciones del estado de los pines. Al igual que el punto anterior, la temporización se selecciona de los tabulados en la figura C.9 C.



(A) **(B)** **(C)**

Figura C.9: Pestaña Configuración.

Bibliografía

Tutoriales de Arduino:
http://arduino.cc/en/Tutorial/HomePage

Tutoriales avanzados de Arduino:
http://tronixstuff.wordpress.com/tutorials/

Tutoriales de programación de Nokia:
http://www.developer.nokia.com/

Web sobre funcionamiento del GPS:
http://www.asifunciona.com/electronica/af_gps/af_gps_10.htm

Información sobre Python
http://es.wikipedia.org/wiki/Python

Beginning Arduino por Michael McRoberts. Editorial: Technology in Action. 2010.

Diseño práctico con Microcontroladores por José Mª Angulo, Susana Romero e Ignacio Angulo. Editorial: Thomson. 2004.