Universidad
Politécnica
de Cartagena

industriales
etsii UPCT

GNX Project v.2

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

Titulación: Ingeniero en Organización industrial.
Intensificación: Gestión de la Producción
Alumno: Daniel Carreres Prieto
Director: Dr. Héctor David Puyosa Piña
Codirector: Dr. Juan Suardíaz Muro

Cartagena, 22 de julio de 2014

Autor	Daniel Carreres Prieto
E-mail del Autor	dcarrpri@gmail.com
Director	Dr. Héctor David Puyosa Piña
E-mail del Director	hector.puyosa@upct.es
Codirector	Dr. Juan Suardíaz Muro
E-mail del Codirector	juan.suardiaz@upct.es
Título del PFC	GNX Project v.2.- Módulo de control programable inalámbricamente para aplicaciones de telemetría, telecontrol y posicionamiento vía satélite.

Resumen

El objetivo del presente proyecto es la implementación física de un dispositivo de control integrado reprogramable sin necesidad de PC ni conocimientos de programación, destinado a la realización de multitud de tareas cooperativas definidas por el usuario. La orientación del dispositivo será de general, es decir, no se encontrará focalizado a un único ámbito de aplicación, sino que se persigue un sistema polivalente que pueda ser orientado tanto al control domótico como a su implementación en vehículos u otros dispositivos, así como sistema de adquisición de datos y logger GPS.

El sistema constará de un Módulo Principal basado en la plataforma Arduino, que dispondrá de los siguientes: GPS, Bluetooth, Almacenamiento flash removible, GSM/GPRSS, Conectividad por Radiofrecuencia.

A este módulo se conectarán diversos sensores y actuadores que serán controlados y configurados por medio de una aplicación diseñada para dispositivos Android.

A fin de facilitar su aplicabilidad a áreas de mayor extensión, se plantea como parte del objetivo del presente proyecto el diseño e implementación de otros módulos de reducidas dimensiones ubicados remotamente y en constante comunicación con el Módulo Principal, al que se le podrá conectar los mismo periféricos (sensores y actuadores) que el Módulo Principal, los cuales serán gestionado por éste.

Titulación	Ingeniero en Organización Industrial
Departamento	Departamento de Ingeniería de Sistemas y Automática. Departamento de Tecnología Electrónica.
Fecha de presentación	Julio 2014

Índice General

I. Memoria

Agradecimientos

A mis padres, Pedro y Matilde por todo el cariño y apoyo que me han brindado alentándome a seguir aprendiendo y mejorando.

A mis directores de proyecto, Don Héctor David Puyosa Piña y Don Juan Suardíaz Muro, por el apoyo que me han brindado y confianza depositada.

I. Memoria

1. Introducción

1.1. Enunciado y objetivos 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 con una formación muy técnica, por lo que se reduce en cierta medida el acceso a la mayoría de usuarios.

Así mismo, en aquellas situaciones en la que las características del dispositivo comercial no sean suficientes para poder cubrir las necesidades presentes y futuras del usuario, no es posible acometer modificaciones en dichos dispositivos para poder extender o mejorar sus funciones (o al menos, poder hacerlo de una forma sencilla), obligando al usuario a tener que adquirir otros dispositivos complementarios para poder cubrir las carencias que el dispositivo original no ha cubierto, con riesgo de la infrautilización de la capacidad de los mismo.

Por ello, se plantea como objeto del presente proyecto, el **diseño e implementación de un sistema integrado configurable por el usuario** mediante una aplicación para Smartphones Android, **para la realización de un número casi ilimitado de tareas distintas mediante la combinación de diversos sensores y actuadores** presentes en el sistema, así como otros periféricos de comunicación y adquisición presenten en el mismo, tales como: GPS, Bluetooth, GSM/GPRS, Almacenamiento flash removible y Conectividad por Radiofrecuencia.

El sistema se compondrá de los siguientes elementos:

A) <u>Módulo Principal</u>

Es el corazón del sistema, el cual recibe todas las órdenes del usuario y de las Estaciones remotas. En él es posible conectar un máximo de 35 periféricos, entendiendo como tales sensores y actuadores que serán comentados con más detenimiento en capítulos posteriores. El manejo de dichos periféricos se realizan, o bien, por medio de una orden inmediata del usuario (mediante el Smartphone) pudiendo por ejemplo encender o apagar las luces de una habitación a voluntad; o bien al cumplimiento de alguna condición por parte de los sensores de entrada. Es decir, el sistema pueda ser programada para la realización de diversas acciones (activar salidas, registrar posición, mandar SMS, etc.) cuando se cumpla una condición definida por el usuario en un momento determinado. Esto le confiere el carácter adaptativo que se perseguía en el presente proyecto, pudiéndose realizar cualquier tipo de combinación en un gran número de acciones programar, las cuales se ejecutan de manera prácticamente simultánea, dado que en el presente proyecto

hemos empleado un microcontrolador y no un microprocesador (La justificación de la elección se ha detallado en el Capítulo 4), dado que no se permite programación por hilos o multitareas, por lo las distintas acciones a ejecutar al cumplimiento de las distintas condicione definidas se realizará de forma secuencial, es decir, una detrás de otras, aunque con la alta velocidad de procesamiento (16MHz) da la sensación de que las tareas se realizan de forma simultánea.

El sistema contará con cuatro modos de comunicación distintos: Bluetooth, 3G, SMS y Radiofrecuencia. Siendo los tres primeros a la forma en la que el usuario puede interactuar con el sistema y el último al mecanismo de comunicación bidireccional empleado para el flujo de información entre el Módulo Principal y otros módulos remotos llamadas Estaciones.

Además, el Módulo Principal contiene una serie de dispositivos adicionales, que le posibilitan extender sus funciones, proporcionando una mayor adaptabilidad a entorno. Estos dispositivos son los siguientes:

- **Módulo GPS**
  Encargado de recibir la ubicación del Módulo Principal así como otros parámetros como la fecha, hora y velocidad. Además de permitir al usuario conocer la ubicación del Módulo cuando se cumpla alguna condición o bajo demanda inmediata de este, posibilita el registro del recorrido realizado con el sistema para su posterior visualización en Google Earth, así como posibilitar el "***Control por Área Geográfica***" que será explicado en el Capítulo 4.

- **Bluetooth**
  Destinado a la comunicación a corta distancia (máximo 5 metros) entre el usuario (por medio de su Smartphone) y el Módulo Principal. Este modo de comunicación se emplea principalmente para labores de mantenimiento o cuando la corta distancia entre el usuario y el sistema no compense el empleo de otro tipo de modo de comunicación presente en el sistema.

- **GSM/GPRS**
  Posibilita la comunicación a larga distancia con el usuario, mediante dos forma distintas: Control por internet o por SMS. El control por Internet es posible gracias a una tarjeta SIM con una tarifa de datos. Permite además realizar las funciones típicas de un teléfono, tales como mandar y recibir SMS (permitiendo el control por SMS), realizar y recibir llamadas de voz, etc.

- **Almacenamiento flash removible**
  Preciso para poder almacenar los datos de los sensores y actuadores bajo demanda del usuario, funcionando como tarjeta de adquisición de datos. Además de almacenar los datos de las rutas realizadas para su posterior visualización de un mapa del recorrido. La importancia de ser removible, reside en la posibilidad de poder visualizar dichos datos en el PC, para su posterior tratamiento y análisis.

- **Módulo Radiofrecuencia**
  Debido a que en numerosas ocasiones no es posible cablear los periféricos directamente al Módulo Principal, existe la posibilidad de que dichos periféricos se conecten a otros módulos llamados Estaciones que son gestionados por medio de radiofrecuencia por el Módulo Principal. Esto permite extender el número de pines disponibles para poder conectar periféricos y evita la pérdida de señal que acarrearía el excesivo alargamiento del cable.

B) Estaciones

Son módulos de reducidas dimensiones destinados únicamente a la conexión de exactamente los mismo periféricos que pueden ser conectados en el Módulo Principal, los cuales son gestionados por este, dado que las Estaciones no disponen de capacidad para gestionar los periféricos conectados (dado que la información sobre los mismo es almacenada de forma volátil), sino que se limita a acatar las órdenes del Módulo Principal, con el que establece una comunicación bidireccional a fin de proporcional la retroalimentación necesaria. Las órdenes inmediatas del usuario que afecten a alguno de los periféricos conectados en alguna de las 7 Estaciones que puede gestionar el sistema a la vez, son redireccionadas por el Módulo Principal mediante un identificador en las tramas enviadas.

Estas Estaciones pueden albergar un máximo 10 periféricos cada una de ellas y ubicarse a una distancia entre los 50 y 80 metros.

C) Aplicación para Smartphone

Para la programación, control y monitorización del sistema en su conjunte, se hace necesario disponer de una aplicación diseñada para teléfonos Android. Desde la cual se podrá conocer el estado del sistema en su conjunto, tanto de los periféricos conectados y su valor en un momento dado, como el nivel de batería, que estaciones se encuentran operativas, así como la posibilidad de poder contener un una única aplicación la posibilidad de seleccionar los modos de conectividad a emplear para poder interactuar con el sistema en función de la situación en la que se encuentre el usuario. Por ejemplo, si este se encuentra a cientos de kilómetros del Módulo Principal, debería optar por un control por internet si dispone de conexión a la red en su Smartphone en ese momento, o bien optar por una conectividad por SMS que pese a generar unos costes que dependerán de las tarifas fijadas por la operadora, garantizan la posibilidad de interactuar con el sistema en cualquier momento y sin requerir de disponer de una red wifi o 3G. Todo esto será detallado en capítulos posteriores.

Gracias a los elementos y dispositivos que posee el sistema, hace que pueda abordar una gran cantidad de tareas, más allá de realizar la típica activación/desactivación de las salidas, dado que puede emplearse como por ejemplo, en vehículos, para poder comunicar por ejemplo por SMS la posición del sistema en caso de robo accidente, al detectar una desaceleración brusca o al saltar alguno de los sensores ubicados en el vehículo. Además de poder asociar acciones al hecho de que el sistema se encuentre dentro o fuera de una o

varias áreas geográficas predefinidas, lo cual puede permitir, por ejemplo que conforme nos aproximemos con nuestro vehículo al garaje, la puerta se abra sola, que encienda las luces de la vivienda o que realice una llamada de voz o mande un SMS avisando que hemos llegado a nuestro destino.

Por ello, el sistema dispone de un carácter polivalente que hace que se pueda orientar bien al control domótico como un sistema integrado de seguridad y control, así como la posibilidad de emplearlos en vehículos, invernaderos, etc. Las posibilidades de configuración son muy amplia gracias a la estructura de programación que se detallará en próximos capítulos, pudiendo realizar cualquier tarea de una forma fácil y rápida de programar sin precisar de ningún tipo de conocimiento adicional.

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:

- **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**: Descripción y justificación de las consideraciones de diseño del hardware tanto del Módulo Principal como de las Estaciones.

- **Capítulo 4**: Descripción detallada del firmware diseñado para el Módulo Principal y para las Estaciones, así como la explicación de las principales funciones que es capaz de desarrollar el sistema acompañado de una explicación de segmentos de código fuente que lo posibilitan. También se describirá los modos de comunicación que se han implementado en el proyecto.

- **Capítulo 5**: Descripción de la aplicación para Android que permite controlar el sistema.

- **Capítulo 6**: Descripción del proceso de instalación del firmware de Arduino y de la aplicación Android.

- **Capítulo 7**: Esquema eléctrico, planos del PCB y listado de los componentes.

- **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 sub-bandas. 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.

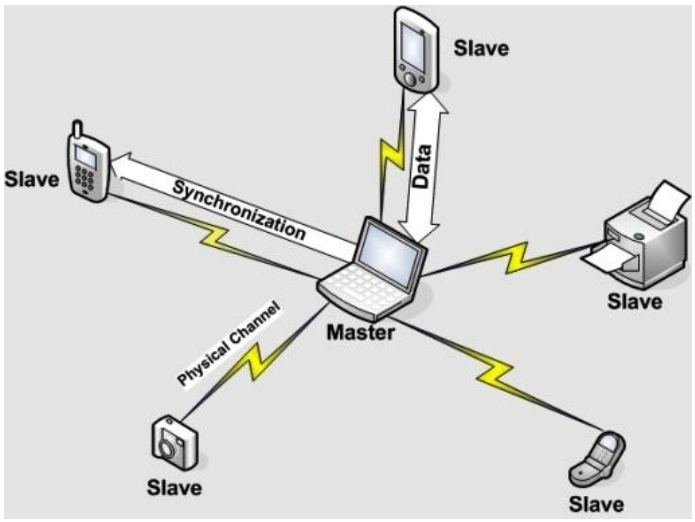


Figura 2.1: Representación gráfica de una red pic (Piconet).

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.

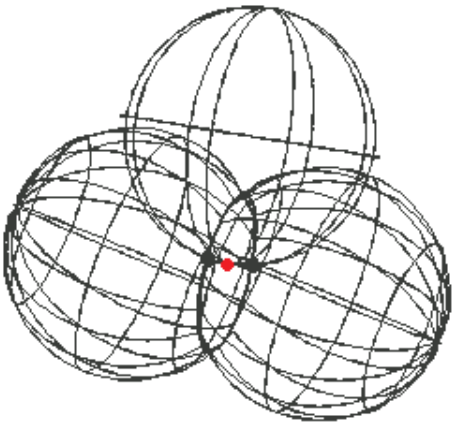


Figura 2.2: Intersección de las esferas imaginarias en un único punto.

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.

Toda la información que recibe del satélite es expresada mediante una estructura de información se realiza de forma ordenada por medio de un sistemas de tramas llamadas NMEA.

La NMEA en realidad es una institución (National Marine Electronic Asociation, pero al protocolo también se le conoce por sus siglas) dedicada a establecer un estándar para la comunicación de dispositivos periféricos marinos, ya que hace algunas décadas atrás, cada marca manejaba su propia forma de comunicarse (protocolo de datos), a fin de hacer más fácil la comunicación entre diversos dispositivos (tanto a nivel del protocolo de datos como de la interfaces eléctrica), se creó esta institución en la cual participan fabricantes, distribuidores, instituciones educacionales y otros interesados en equipos periféricos marinos, esta institución no tiene ánimo de lucro, la última versión del protocolo **NMEA es la 0183**.

En la Figura 2.3, observamos un ejemplo de la adquisición de dichos datos:

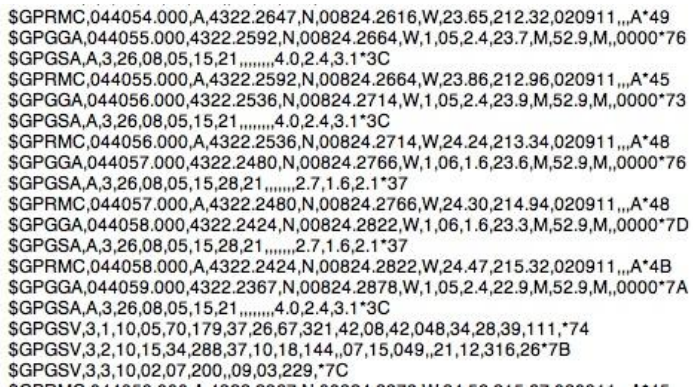


Figura 2.3: Muestra de tramas NMEA.

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.

Vamos a plantear la siguiente trama para poder descomponer los elementos que la forman a fin de poder comprender con mayor claridad, cual debe ser la labor del algoritmo que será necesario implementar dentro del firmware del proyecto para poder trabajar los datos que son recibidos por el puerto serie, a una velocidad máxima de 4800 baudios, dado que los modelos de la gama EM, no funcionan correctamente a velocidades superiores:

$GPRMC,144235.000,A,4322.0289,N,00824.5210,W,0.39,65.46,080714,,,A*44

Donde los datos serán:

- **144235.000** es el es la hora GMT (14:42:35).
- **A:** Indica que el dato de posición está fijado y es correcto, conectado con un mínimo de tres satélites. **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.
- **080714:** Fecha (8 de julio del 2014).

2.4. Sistema de almacenamiento extraíble. SD

Como se ha mencionado en el capítulo anterior, el sistema precisa de disponer de algún tipo de almacenamiento extraíble, a fin de poder revisar los datos que hemos adquirido, ya sean procedente de los datos del GPS para registrar el recorrido realizado, o bien el estado de uno o varios sensores/actuadores ubicados independientemente de donde se encuentren estos ubicados, es decir, tanto en el Módulo Principal como en cada una de las Estaciones remotas.

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 primera opción podría ser el empleo de Pendrives de poca capacidad, dado que solo se van a almacenar archivos de texto. No obstante para poder emplearlos en el presente proyecto, se precisa de una mayor electrónica, encareciendo de esta forma el precio final del prototipo. Lo que se precisa básicamente es un MAX3420E conectado mediante el bus SPI y al conector USB hembra proporciona la posibilidad de funcionar como host.

El MAX3420E opera utilizando un conjunto de registros a los que accede por una interfaz SPI que opera hasta un máximo de 26MHz. Cualquier maestro SPI (microprocesador, ASIC, DSP, etc) puede añadir periféricos USB o la funcionalidad de host utilizando SPI de 4 hilos.

En la Figura 2.4 se puede observar cual sería el esquema general de implementación del USB Host.

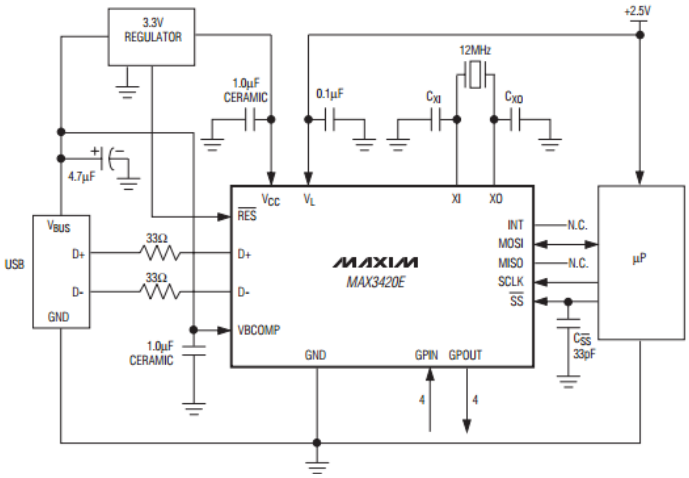


Figura 2.4: Esquema de USB Host con MAX3420E.

Como contrapartida, se puede recurrir al empleo de 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.

Su reducido tamaño y la usencia de circuitería externa que pueda verse afectada por las interferencias hace que se la elección que se empleará para el desarrollo del proyecto. Por ello, a continuación indicaremos el modo de funcionamiento del las tarjetas de memoria 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 2.5.

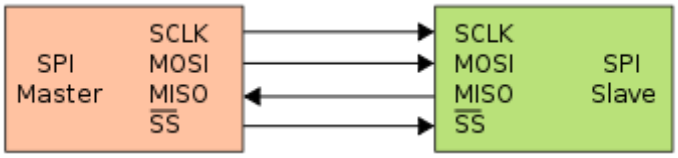


Figura 2.5: Interconexión para comunicación con la SD.

- **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 de comunicación GSM/GPRS

Como se ha mencionado en el Capítulo 1, la comunicación a larga distancia entre el usuario y el Módulo Principal, es posible gracias a la comunicación por internet y por SMS. Esto es permitido por medio de un módulo GSM/GPRS en esencia es un teléfono móvil materializado en forma de integrado como el que aparece en la Figura 2.6.

Figura 2.6: Módulo GSM/GPRS SIM900.

El sistema GSM es el sistema de comunicación de móviles digital de 2ª generación basado en células de radio. Apareció para dar respuestas a los problemas de los sistemas analógicos.

Fue diseñado para la transmisión de voz por lo que se basa en la conmutación de circuitos, aspecto del que se diferencia del sistema GPRS. Al realizar la transmisión mediante conmutación de circuitos los recursos quedan ocupados durante toda la comunicación y la tarificación es por tiempo.

Por otro lado, el GPRS es una nueva tecnología que comparte el rango de frecuencias de la red GSM utilizando una transmisión de datos por medio de 'paquetes'. La conmutación de paquetes es un procedimiento más adecuado para transmitir datos, hasta ahora los datos se habían transmitido mediante conmutación de circuitos, procedimiento más adecuado para la transmisión de voz.

En GSM, cuando se realiza una llamada se asigna un canal de comunicación al usuario, que permanecerá asignado aunque no se envíen datos. En GPRS los canales de comunicación se comparten entre los distintos usuarios dinámicamente, de modo que un usuario sólo tiene asignado un canal cuando se está realmente transmitiendo datos. Para utilizar GPRS se precisa un teléfono que soporte esta tecnología. La mayoría de estos terminales soportarán también GSM, por lo que podrá realizar sus llamadas de voz utilizando la red GSM de modo habitual y sus llamadas de datos (conexión a internet, WAP, etc.) tanto con GSM como con GPRS.

La tecnología GPRS, o generación 2.5, representa un paso más hacia los sistemas inalámbricos de Tercera Generación o UMTS. Su principal baza radica en la posibilidad de disponer de un terminal permanentemente conectado, tarificando únicamente por el volumen de datos transferidos (enviados y recibidos) y no por el tiempo de conexión como hemos podido observar en un punto anterior.

Tradicionalmente la transmisión de datos inalámbrica se ha venido realizando utilizando un canal dedicado GSM a una velocidad máxima de 9.6 Kbps. Con el GPRS no sólo la velocidad de transmisión de datos se ve aumentada hasta un mínimo 40 Kbps y un máximo de 115 Kbps por comunicación, sino que además la tecnología utilizada permite compartir cada canal por varios usuarios, mejorando así la eficiencia en la utilización de los recursos de red.

2.5.1. Protocolo del GPRS

El protocolo GPRS es un protocolo de nivel tres, transparente para todas las entidades de red comprendidas entre el terminal móvil y el nodo SGSN al que el móvil está conectado. Este protocolo soporta tanto el intercambio de informaciones de control como de paquetes PDP-PDU (Packet Data Protocol - Protocol Data Unit) entre el móvil y el nodo al que se encuentre conectado. El formato de una trama GPRS contiene los siguientes campos:

A) Identificador del protocolo GPRS

El identificador del protocolo GPRS es una información numérica cuyo objetivo es el de distinguir las ráfagas (los burst) que contienen paquetes GPRS, de las ráfagas que contienen información GSM.

B) Identificador del protocolo de los PDU

Este identificador, encapsulado en las tramas GPRS, es necesario para direccionar dichas tramas hacia el correcto SAP (Service Access Point) en cuanto son desencapsuladas.

Al igual que el identificador del protocolo GPRS, esta información también es de tipo numérico. Se tendrá, por tanto, un valor que define los paquetes X25, uno que define los paquetes IP (Internet Protocol), uno que define los paquetes CLNP (Connectionless Network Protocol) y así sucesivamente.

Además, dicha información permite la interpretación del mensaje GPRS contenido en la trama GPRS. De hecho, las tramas GPRS son utilizadas tanto para el transporte de mensaje de control como para el transporte de paquetes de datos, por lo que se hace necesario el uso de un indicador que permita distinguir a cuál de las dos categorías posibles pertenece el mensaje GPRS.

C) Mensaje GPRS

Un mensaje GPRS puede contener o bien datos o bien información de control. Los mensajes GPRS de control son definidos por un valor preestablecido del identificador de PDP. Algunos de los posibles mensajes de control se enumeran a continuación:

- Petición de log-on (log-on request).
- Respuesta a una petición de log-on (log-on response).
- Activación del modo de transmisión cifrado (set gprs ciphering mode).

- Petición de actualización de las informaciones de routing (routing update request).
- Respuesta a una petición de actualización de las informaciones de routing
- (routing update response).
- Petición de actualización del indicador de routing area (área de encaminamiento) (gprs ra update request).
- Respuesta a una petición de actualización del indicador de routing area (gprs ra update response).

El nodo SGSN realiza las funciones de encaminamiento. Este nodo encapsula los datos recibidos del terminal móvil, en el protocolo de red usado para el transporte de paquetes en su red de distribución (backbone network).

2.5.2. Comandos AT

Para poder realizar las funciones típicas de un módulo GSM/GPRS, tales como acceder a internet, realizar y recibir llamadas y mensajes, así como poder controlar otros aspectos del mismo como la velocidad o el ajunte del volumen, es necesario proporcionarle al módulo GSM/GPRS la información de forma de tramas ordenadas mediante un lenguaje de programación llamado Comandos AT.

Los comandos AT son instrucciones codificadas que conforman un lenguaje de comunicación entre el usuario y un terminal (Modem). Fueron desarrollados en 1977 por Dennis Hayes como un interfaz de comunicación con un Modem para así poder configurarlo y proporcionarle instrucciones, tales como marcar un número de teléfono. Más adelante, con el avance del baudio, fueron las compañías Microcomm y US Robotics las que siguieron desarrollando y expandiendo el juego de comandos hasta universalizarlo.

Aunque la finalidad principal de los comandos AT es la comunicación con modems, la telefonía móvil GSM también ha adoptado como estándar este lenguaje para poder comunicarse con sus terminales. De esta forma, todos los teléfonos móviles GSM poseen un juego de comandos AT específico que sirve de interfaz para configurar y proporcionar instrucciones a los terminales, permiten acciones tales como realizar llamadas de datos o de voz, leer y escribir en la agenda de contactos y enviar mensajes SMS, además de muchas otras opciones de configuración del terminal.

Estos comandos son mandados a través del puerto serie del microcontrolador o microprocesador, a una velocidad en baudios variable, dada que por defecto el chip SIM900, que es el que se ha empleado en el presente proyecto (Figura 2.6) y del que se hablará más detalladamente en capítulos posteriores, está preconfigurado para adaptarse a la velocidad fijada en el puerto serie (autobauding), que en nuestro caso ha sido establecida a 115200 baudios. Para sincronizar la velocidad del controlador con el SIM900, se debe enviar una “A” y esperar de 3 a 5 segundos, para luego enviar el comando “AT” antes de iniciar la comunicación, donde una vez sincronizado el módulo GSM/GPRS nos devolverá un “OK” (Figura 2.7).

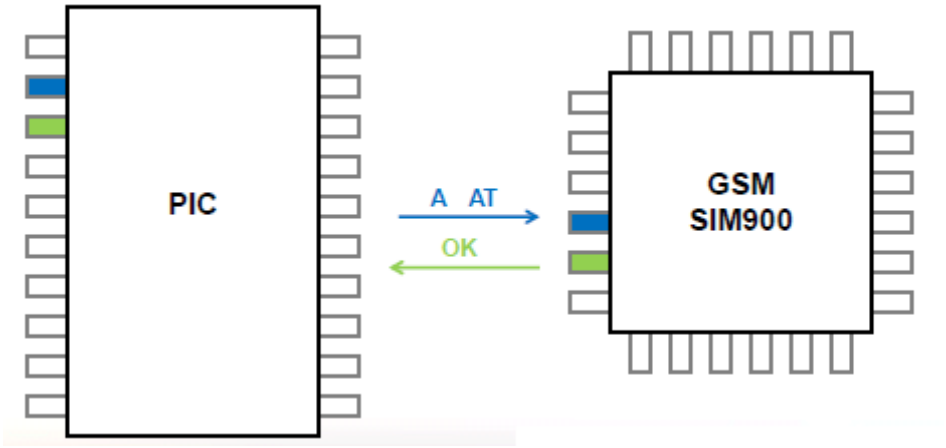


Figura 2.7: Proceso de sincronización SIM900.

Aunque los comandos AT experimentan variaciones en función del fabricante, en su gran mayoría se encuentran estandarizados y son compatibles con prácticamente la totalidad de módulos comerciales de este tipo. En el siguiente ejemplo, vamos a mostrar a modo de esclarecer un poco como se emplea y que forma tiene un programa formado por comandos AT para poder realizar un sencillo en vio de un SMS, a un número de teléfono concreto, por ejemplo: 123456789:

AT
>OK
AT+CMGF=1//Activamos el modo texto
AT+CMGS=”123456789”<0x0d>//El 0x0d representa un salto de línea
Esto es una prueba con comandos AT <0x1a>//Cuerpo del mensaje y envío

2.6. Comunicación por radiofrecuencia

Para concluir el presente capítulo, debemos hablar sobre los fundamentos de la transmisión de datos por radiofrecuencia. Como se ha mencionado con anterioridad, el Módulo Principal, establece una red de comunicación con otros módulos más pequeños llamados Estaciones por medio de un enlace de radiofrecuencia para poder gestionar periféricos ubicados de forma remota. (Figura 2.8).

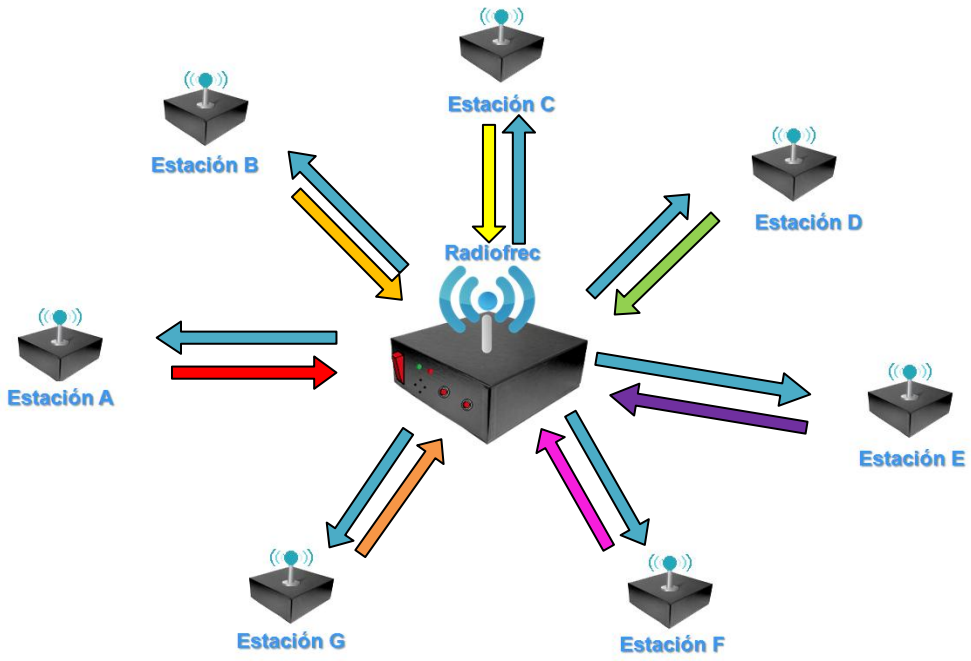


Figura 2.8: Red de comunicación Modulo Principal-Estaciones.

No es posible establecer una comunicación simultánea con las siete Estaciones que componen la red (dado que es el máximo de esclavos permitidos por los módulos empleados, tal y como se mencionará en capítulos posteriores). Por ello, lo que se hace es establecer comunicación con cada una de las Estaciones disponibles intercambiando información de forma bidireccional en cada ciclo de programa, para luego cambiar de frecuencia para poder comunicarse con la siguiente Estación y así sucesivamente. Teniendo en cuenta que esto se realiza a alta velocidad, la sensación de lapsus de tiempo es mínima, dando la impresión que se realiza de forma simultánea.

El término radiofrecuencia, también denominado espectro de radiofrecuencia o RF, se aplica a la porción menos energética del espectro electromagnético, situada entre unos 3 Hz y unos 300 GHz. Las ondas electromagnéticas de esta región del espectro, se pueden transmitir aplicando la corriente alterna originada en un generador a una antena.

Una onda de radio se origina cuando una partícula cargada (por ejemplo, un electrón) se excita a una frecuencia situada en la zona de radiofrecuencia (RF) del espectro electromagnético. Otros tipos de emisiones que caen fuera de la gama de RF son los rayos gamma, los rayos X, los rayos infrarrojos, los rayos ultravioleta y la luz.

Cuando la onda de radio actúa sobre un conductor eléctrico (la antena), induce en él un movimiento de la carga eléctrica (corriente eléctrica) que puede ser transformado en señales de audio u otro tipo de señales portadoras de información.

El emisor tiene como función producir una onda portadora, cuyas características son modificadas en función de las señales (audio o vídeo) a transmitir. Propaga la onda portadora así modulada. El receptor capta la onda y la desmodula para hacer llegar al espectador auditor tan solo la señal transmitida.

3. Diseño del Hardware

3.1. Introducción

Tal y como se ha mencionado con anterioridad, el sistema se componen tanto de un Módulo Principal que alberga todos los dispositivos de comunicación y adquisición (entendiéndose como tales GSM/GPRS, GPS, etc) y actúa de sistema centralizado recibiendo las órdenes del usuarios; así como de diversas Estaciones, que mediante enlace de radiofrecuencia se comunican de forma bidireccional con el Módulo Principal a fin de que éste puede gestionar los dispositivos conectados en las Estaciones remotas como si estas se encontraran físicamente conectadas al Módulo Principal.

Por ello, en el presente capítulo se va a mostrar de forma detallada las características a nivel del hardware de cada uno de estas partes que componen el sistema, a fin de disponer de un enfoque más preciso sobre cada una de las partes antes mencionadas.

3.2. Módulo Principal

El Módulo Principal es el corazón del sistema. Es el que asume la función de gestionar todos los periféricos conectados (tanto en él como en cada una de las Estaciones remotas) además de ser el interlocutor con el usuario y contener otros elementos adicionales tales como GPS o el GSM/GPRS necesario para poder acceder a internet por tarifa de dato, necesarios para lograr la flexibilidad de funcionamiento que hemos marcado como objetivo primordial en el presente proyecto.

3.2.1. Selección de plataforma de desarrollo

El presente módulo está basado en la plataforma Arduino, concretamente Arduino Mega 2560. Los motivos de elección de esta plataforma (y de este modelo en particular) no obedecen a la causalidad, sino que se encuentran fundamentados.

Para poder implementar el presente proyecto se podría emplear otras opciones como los tradicionales microcontroladores PIC de Microchip o microprocesadores 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 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.

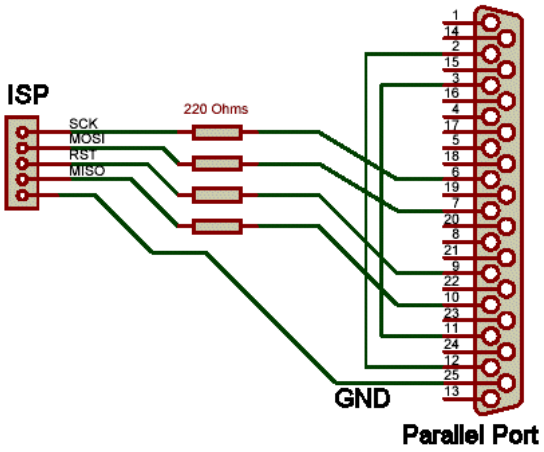


Figura 3.1: Esquema programador paralelo del Bootloader.

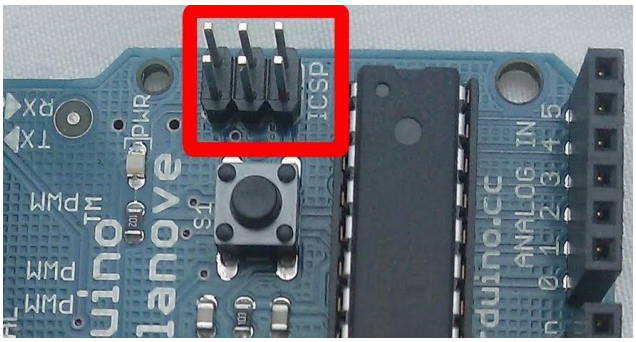


Figura 3.2: Ubicación de los pines ICSP en una placa Arduino.

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.

	Modelo	USB	Micro	Nº Pines	Memoria	Precio
	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€
	Leonardo	Tipo B	ATmega324	20 Digital 7 PWM 12 Analog	Flash:28KByte Ram:2,5KByte EEPROM:1K Byte	12€

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

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).

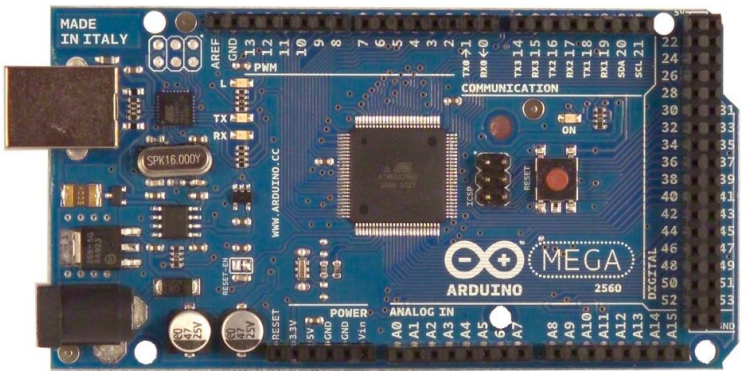


Figura 3.4: Arduino Mega 2560.

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

3.2.2. Periféricos de comunicación y adquisición

Este módulo se compone de diversos dispositivos de control y comunicación, que se emplean para la realización de múltiples operaciones definidas por el usuario. Estos deben aislarse correctamente a fin de evitar interferencias que afecten a su funcionamiento. Con objeto de poder exponer con mayor claridad las interferencias que tienen lugar así como las condiciones particulares de funcionamiento que ha condicionado el diseño, mostraremos a continuación los distintos dispositivos indicando sus características principales.

3.2.2.1. Módulo Transceptor Bluetooth

El módulo transceptor Bluetooth que se ha empleado en el presente proyecto es un RS232, como el que aparece en la Figura 3.5.

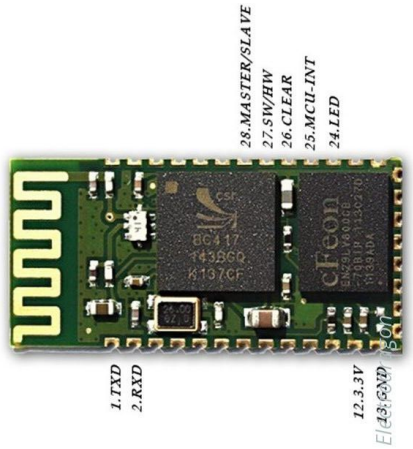


Figura 3.5: Módulo transceptor bluetooth RS232

Está basado en el chip British CSR BlueCore4-Ex, ofreciendo bluetooth 2.0, el cual es suficiente para el proyecto dado que el alcance no es determinante y se trata de reducir los costes en un modo de comunicación que es segundario y opcional.

El módulo es compatible con UART, USB, SPI, PCM, SPDIF. Dispone de una memoria flash de 8Mbit reprogramable mediante puerto serie para poder realizar modificaciones en su configuración por medio de comandos AT, pudiendo alterar el PIN de vinculación (por defecto 1234), el nombre del dispositivo (por defecto es LINVOR), entre otros parámetros.

La velocidad de transmisión es definida por el usuario, entre 1200, 2400, 4800, 9600, 19200, 38400, 57600 y 11520 baudios; siendo la configuración por defecto del puerto serial: 9600, N, 8, 1.

Su antena integrada en el PCB, limita su radio de acción a unos 5 metros en abierto. Opera en la banda de los 2,4GHz-2,48GHz, por lo que puede verse afectado por las interferencias causadas por redes wifis, que recordemos operan en la misma banda, sin embargo en los ensayos realizados, el efecto de dichas interferencias han sido despreciables, dado que el diseño del propio módulo garantiza el adecuado tratamiento de la señal.

Las características técnicas del módulo se encuentran recogidas en la Tabla 3.2.

Características generales	
Voltaje de funcionamiento	3,3V
Rango de tensión	2,2V-4,2V
Rango de temperatura	-40ºC-150ºC
Consumo de escaneo	20mA-30mA
Consumo vinculado	8mA

Tabla 3.2: Características.

El módulo se comercializa sin ningún tipo de soporte. Por lo que se precisa llevar a cabo un PCB para poder conectarlo con el controlador, en este caso Arduino Mega 2560. El esquema básico que se ha seguido para la realización del PCB ha sido el que se muestra en la Figura 3.6, donde se ha contemplado la colocación de un led de estado, que parpadea a intervalos de 150ms cuando está buscando dispositivo al que conectarse y que se mantiene fijo cuando ha establecido dicha conexión.

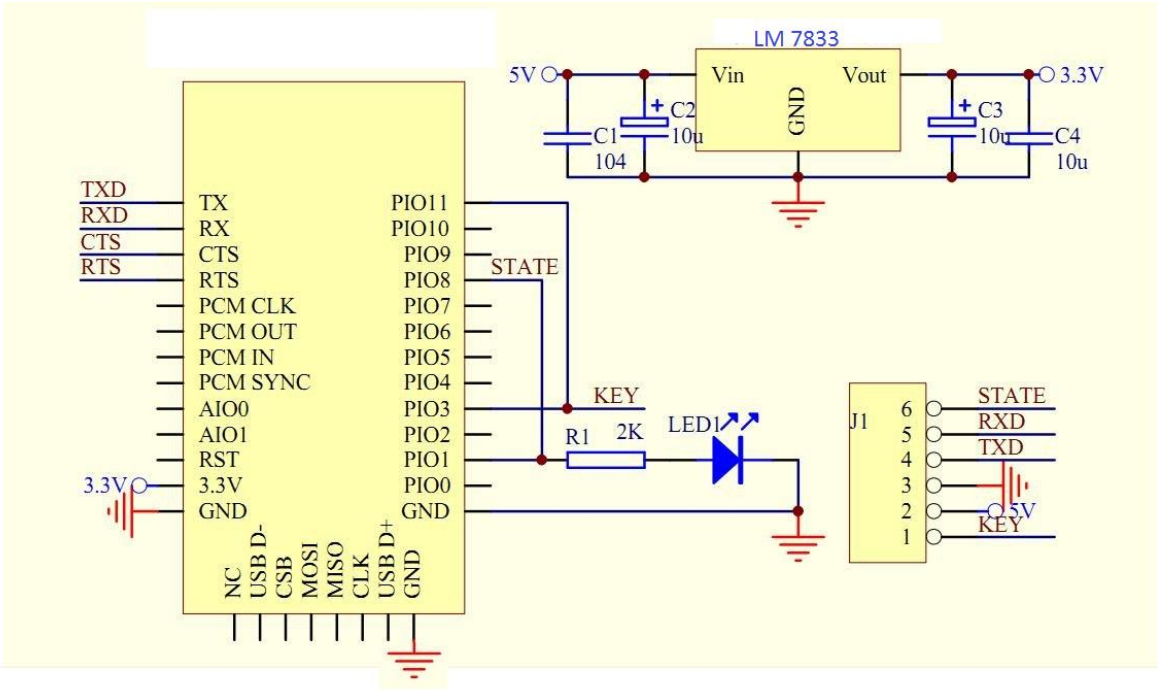


Figura 3.6: Esquema para el uso del transceptor bluetooth RS232.

Dado que el Arduino Mega 2560 dispone de un total de cuatro puertos series, y que deseamos mantener el puerto serie principal (Serial 0) sin conexionar a fin de evitar la aparición de fallos a la hora de cargar el firmware al Arduino, hemos optado por conectar el Bluetooth en el **puerto serie número 3** (Pin 14 y 15).

3.2.2.2. Módulo GPS

Para la adquisición de datos GPS, se ha optado por un módulo de GlobaltSat al ser una gama de productos ya probada en anteriores proyectos y que ha dado una respuesta muy buena al trabajar a altas velocidades. Concretamente, se ha empleado el modelo EM-411 (ver figura 3.7) que pertenece a la gama intermedia de sus productos.

Características

- Chipset: SiRF Star III
- Frecuencia: 1575.42 MHz
- C/A code: 1.023 MHz
- Canales: 20
- Sensibilidad: -159 dBm
- Readquisición: Cada 0.1 seg
- Readquisición en caliente: Cada 1 seg
- Readquisición en frio: Cada 42 seg
- 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

Figura 3.6: GPS EM-411.

Como condiciones de funcionamiento, debemos destacar que la ausencia de antena externa, obliga evitar colocar nada metálico encima del módulo GPS, por lo que obligatoriamente, debe situarse este en la parte superior del PCB. Además la carcasa en la que se alojará el Módulo Principal (y por consiguiente el GPS) deberá de ser de plástico ABS en su cubierta superior, o puede emplearse una cubierta metálica con la necesidad de practicarle orificios en esta de un mínimo de un 0,5 cm de diámetro, en la zona donde se encuentra el GPS a fin de que no actué como pantalla, impidiendo establecer y recibir información del satélite.

El módulo GPS ha sido conectado al **puerto serie número 2** (pin 16,17) a una velocidad de 4800 baudios que como hemos indicado es el máximo soportado por el módulo.

3.2.2.3. Módulo GSM/GPRS

El módulo GSM/GPRS empleado en el presente proyecto, se encuentra basado en el chip SIM900 del fabricante SIMCOM, el cual ya fue descrito en el Capítulo 2 (Ver Figura 2.6).

Para poder emplear dicho módulo, existen en el mercado multitud de placas de desarrollo que proporcionan las conexiones necesarias para poder emplear el SIM900. Atendiendo a unos requisitos de espacio limitado y a mantener los costes del proyecto los más bajos posibles, se ha optado por adquirir el modelo EFComPro del fabricante ElecFreaks (Ver Figura 3.7). Como se puede observar, dispone de su propio regulador de tensión, ranura para la tarjeta SIM, conector para la antena y jack para entrada y salida de audio.

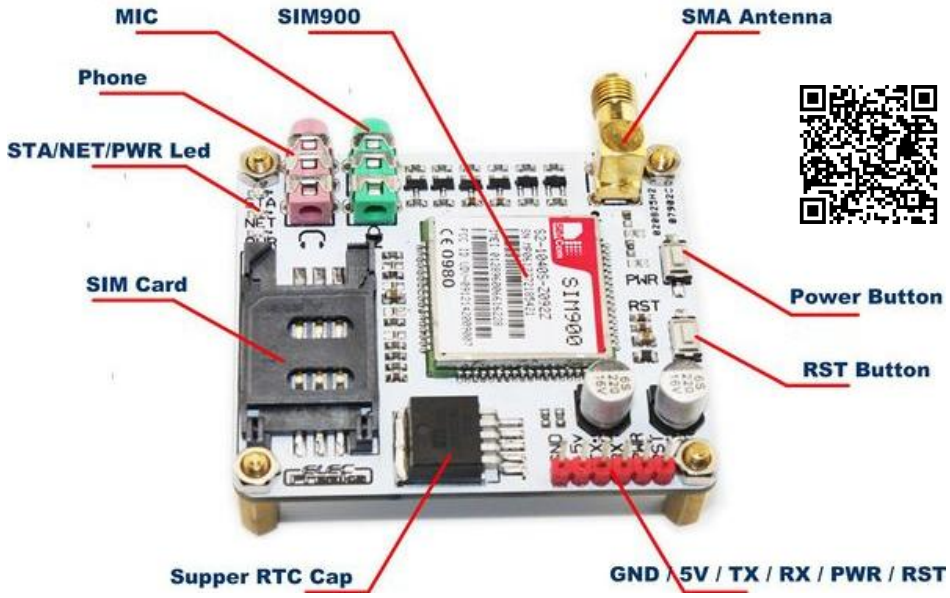


Figura 3.7: Módulo GSM/GPRS EFComPro.

Este módulo completo, consta de unos pocos componentes cuyo esquemático es el que se adjunta a continuación (Figura 3.8).

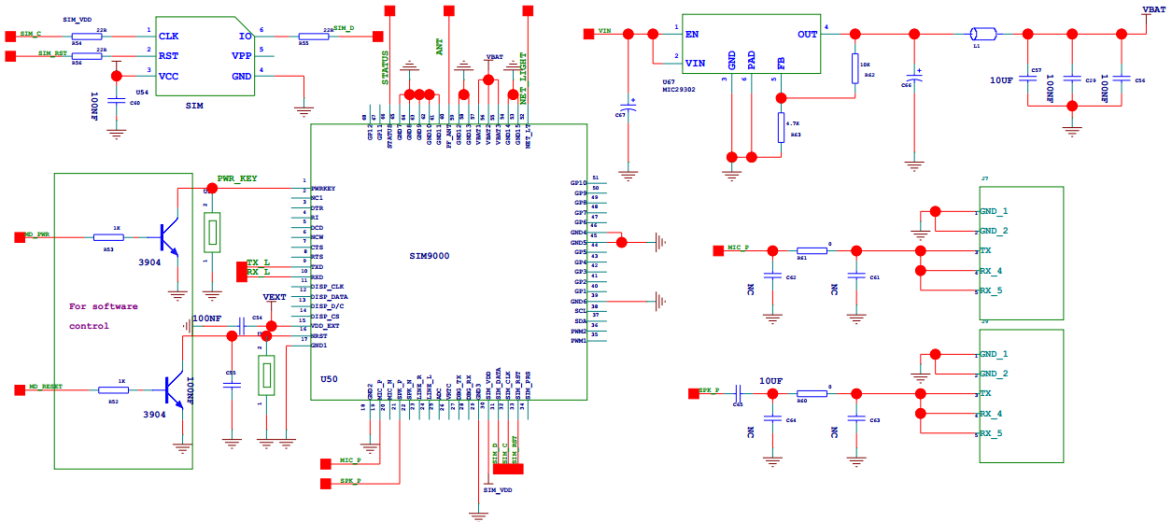


Figura 3.8: Esquema EFComPro.

Este módulo precisa de varios pines para poder funcionar. Por un lado es necesario conectarlo al **puerto serie número 1** para poder establecer la comunicación (pin 18 y 19). Además es preciso disponer de un terminal más para poder encender y apagar el módulo, lo cual se logra manteniendo un pulso alto en tensión durante un periodo superior o igual a 3 segundos. En la Figura 3.9, se puede apreciar el esquema básico de conexionado así como su pinout.

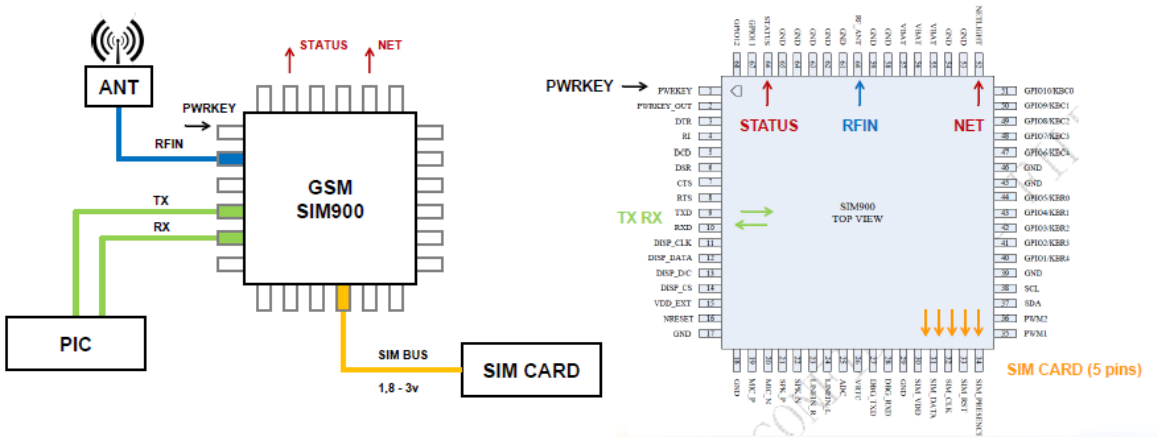


Figura 3.9: Esquema básico de conexionado y pinout.

Los pines "**Status**" (que indica si el módulo está activo) y "**Net**" (que indica si hay conexión a la red GSM/GPRS, es decir, si hay una tarjeta SIM insertada con código PIN conocido), se encuentran conectados a unos led smc que posibilitan conocer de forma visual su estado. Esto es fundamental, dado que el módulo es muy sensible a cualquier alteración en la tensión de alimentación.

Durante su funcionamiento normal, el rango de tensiones debe encontrarse entre los 3,3V y los 4,6V para que el módulo pueda ser empleado para operaciones "bajo consumo", es decir, todas aquellas que no conlleven acceder a internet, como son mandar o recibir mensajes, llamadas o borrar la bandeja de entrada entre otra. Si la tensión de alimentación cae de los 3,3V o sobrepasa los 4,6V (que en las pruebas realizadas, esta tensión correspondería con unos 5,1V), el módulo se apagará automáticamente (Status=LOW). Igual ocurrirá si la temperatura del chip es mayor a los 85ºC. Sin embargo, a 0,1V antes de los límites anteriores (es decir, a 3,4V y 4,5V), el sistema envía warnings por la USART avisando de dicha incidencia (Figura 3.10).

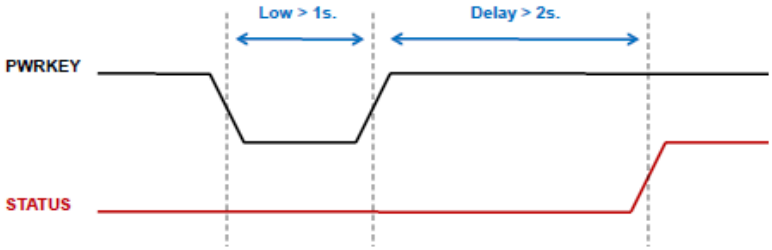


Figura 3.10: Efecto de los niveles de tensión sobre SIM900.

Además, como se ha mencionado, cuando el módulo debe realizar operaciones de acceso a la red, como es el caso de acceder a un servidor FTP, el cual es necesario para el

control del sistema a través de internet y que se comentará en el próximo capítulo; precisamos de un mayor consumo para poder establecer la comunicación, teniendo que suministrar por lo menos 40mA durante los periodos de transmisión. En caso contrario la transmisión se detendrá y el módulo se apagará de igual forma que en los casos expuestos en el párrafo anterior. Igualmente las caídas de tensión durante la transmisión no deben superar los 400mV, sino se reiniciaría el módulo (ver Figura 3.11).

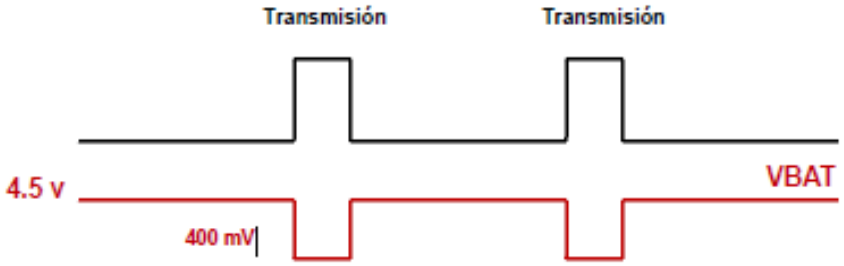


Figura 3.11: Efecto de los niveles de tensión sobre SIM900.

Por otro lado, otro aspecto que condiciona su funcionamiento son las interferencias que se produzcan en el pin POWER del módulo, dado que cualquier alteración provocada ya sea por entrar en contacto por cualquier objeto, ser tocado con los dedos o retorcerse el cable de forma excesiva puede hacer que este se apague.

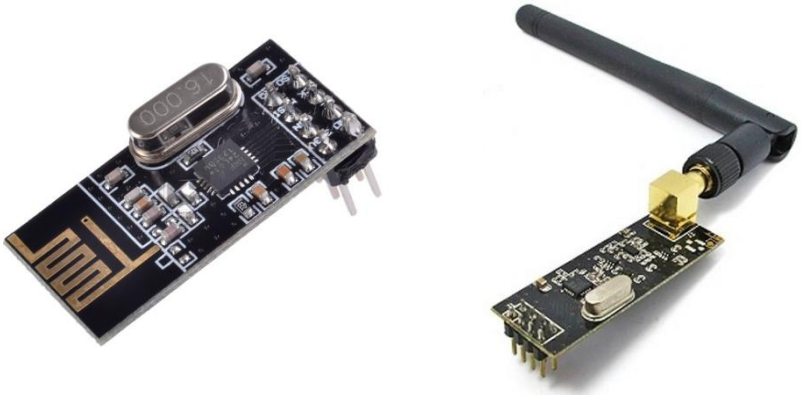
Por ello, para poder solucionar los problemas entes indicados, ha sido necesario:

- Introducir un sistema de regulación de la tensión de alimentación del Arduino Mega 2560 para garantizar la estabilidad en su suministro con la posibilidad de variar la misma. Esto aspecto será tratado al final del presente capítulo dado que se tratará de un circuito externo común tanto al Módulo Principal como a las Estaciones y que además constará con otras características que comentaremos más adelante. Con esta medida se resuelve los problemas relativos al reinicio o apagado del módulo SIM900 durante su acceso a la red.

- Para evitar interferencias en el terminal POWER antes mencionadas, se obstará por reducir al máximo la longitud de la pista que conecta el pin POWER con el pin 49 del Arduino Mega 2560, reduciendo su diámetro a un máximo de 0,5mm y además se recubrirá toda la placa con una laca y barniz proyector acrílico en aerosol de 40kV/mm (http://xurl.es/i22sq).

3.2.2.4. Módulo de Radiofrecuencia

Como se ha mencionado la radiofrecuencia es empleada para la comunicación del Módulo Principal con cada una de las Estaciones presentes en el sistema. Para ello se ha empleado el módulo transceptor NRF24L01 (Figura 3.12 A) con un alcance entre 50 y 80 metros en abierto, aunque también se podría emplear otros de gama más alta como el NRF24L01-PA-LNA (Figura 3.12 B), con antena externa y un alcance de aproximadamente 1Km.



(A) **(B)**

Figura 3.12: Módulos NRF24L01.

En el presente proyecto hemos optado por emplear el NRF24L01 debido a que no era necesario abarcar un área mayor con ellos.

Los transceptores NRF24L01 son una serie de módulos de radio de 2,4 GHz que se basan en el chip de Nordic Semiconductor NRF24L01-PA-LNA (Detalles). El Nordic NRF24L01-PA-LNA integra un completo transceptor RF de 2,4 GHz, un sintetizador de RF y toda la lógica de banda base incluyendo un acelerador de protocolo por hardware Enhanced ShockBurst con una interfaz SPI de alta velocidad para el controlador de la aplicación.

Los datos del alcance son generales, dado el rango es muy dependiente de la situación de los transceptores y tienen mucho más alcance cuando están en la línea de visión, al aire libre que en interior, con obstáculos como paredes y otros materiales. La distancia normal que indican los distintos proveedores para el módulo de baja potencia es de unos 50-80 metros. Pero este valor es para espacio abierto entre unidades funcionando a 250KHz, en interiores, el alcance es mucho menor debido a las paredes, etc.

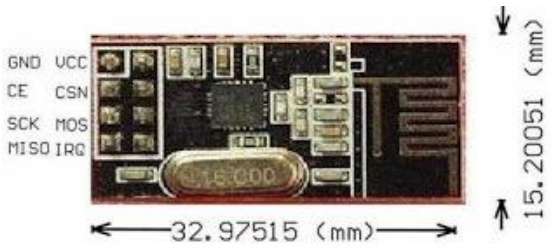


Figura 3.13: Pinout NRF24L01.

El conexionado depende del tipo de placa controladora al que se vaya a conectar. Dado que estos módulos se encuentran presente tanto en el Módulo Principal como en cada una de las Estaciones, en este apartado describiremos el conexionado para ambos, teniendo en cuanta que el Módulo Principal emplea un Arduino Mega 2560 y que las Estaciones usan un Arduino Nano V3, tal y como se indicará más adelante. En la Tabla 3.3, se indica los pines de conexión (Figura 3.13) para cada tipo de Arduino.

Pinout NRF24L01	**Arduino Mega [Módulo Principal]**	**Arduino Nano v3 [Estación]**
VCC	3,3V	3,3V
GND	GND	GND
MISO	50	12
MOSI	51	11
SCK	52	13
CE	8	8
CSN	7	7

Tabla 3.3: Conexionado NRF24L01 para Módulo Principal y Estación.

Otro aspecto clave para el diseño de los módulos finales ha sido el aislamiento de las interferencias causadas por otros dispositivos, como el Wifi, el microondas, etc.

Hay que recordar que estos módulos de radiofrecuencia operan a los 2.4GHz, banda que se encuarta compartida con otros tipos de transmisión como el Wifi o el Bluetooth. Esto es debido a que este tipo de dispositivos se basan en una serie de estándares comunes, como el IEEE 802.11b, IEEE 802.11g e IEEE 802.11n que disfrutan de una aceptación internacional debido a que la banda de 2.4 GHz está disponible casi universalmente, con una velocidad de hasta 11 Mbit/s, 54 Mbit/s y 300 Mbit/s, respectivamente.

En un principio se podría pensar que el módulo transceptor bluetooth que incorpora el Módulo Principal puede causar una fuerte distorsión de la señal limitando su alcance dado que estos se encuentran a escasos centímetros uno de otro en el PCB que más adelante será mostrado. Sin embargo esto no es así, dado que el módulo bluetooth que hemos empleados versión 2.0, y desde la versión 1.2, se realizaron las modificaciones pertinentes en el protocolo para evitar interferencias con otras tecnologías, por lo que no es necesario tomar ninguna medida para aislar sus señales.

Aunque en la actualidad, debido a evitar interferencias con otros dispositivos, se ha empezado a emplear el estándar IEEE 802.11a, conocido como WIFI 5, que opera en la banda de 5 GHz y que disfruta de una operatividad con canales relativamente limpios. La banda de 5 GHz ha sido recientemente habilitada y, además, no existen otras tecnologías (Bluetooth, microondas, ZigBee, WUSB) que la estén utilizando, por lo tanto existen muy pocas interferencias. Su alcance es algo menor que el de los estándares que trabajan a 2.4 GHz (aproximadamente un 10 %), debido a que la frecuencia es mayor (a mayor frecuencia, menor alcance).

Para evitar estas interferencias, la forma más sencilla es adquiriendo el NRF24L01-PA-LNA (Figura 3.12 B) el cual está indicado para poder compensar dichas interferencias

producidas, además de disponer de un alcance de aproximadamente 1Km. Sin embargo dado que durante la realización del presente proyecto no fue posible adquirir dicho modelo (teniendo que emplear Figura 3.12 A), se optó por limitar el efecto de las interferencia desde el punto de vista del software, mediante la implementación de algoritmo de redundancia cíclica que verifican los datos recibidos y en caso de fallo o pérdida de conexión, provocan el reinicio de los módulos en ambos extremos del canal (es decir, tanto en el Módulo Principal como en la Estación/es correspondientes) dado que se ha observado que el reinicio permite volver a restablecer la comunicación entre los módulos. Este algoritmo será explicado en el Capítulo 5.

3.2.2.5. Almacenamiento Removible Flash

Como se han mencionado en el Capítulo 2, el sistema de almacenamiento elegido el tipo SD, concretamente hemos optado por emplear una tarjeta MicroSD de 2Gb que una capacidad más que suficiente para el pequeño tamaño de los archivos .txt que contendrá, además se encuentra por debajo de los 4Gb que es lo máximo que la librería SD.H es capaz de soportar. El hecho que sea MicroSD no condiciona el diseño general del circuito.

En el mercado existen multitud de adaptadores comerciales a un precio que ronda entre los 3 y 6 €. Sin embargo, a comienzo del presente proyecto no fue posible adquirir ninguno comercial, por lo que fue necesario construir uno. A modo de ejemplo práctico, se describe a continuación los pasos necesarios para poder construir uno a partir de un adaptador de tarjetas MicroSD a SD.

En primer lugar debemos observar el esquema dispuesto en la figura 2.5 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 3.15 los pines que conforman el adaptador SD indicando los pines del Arduino.

Figura 3.15: 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**. 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 3.16, 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 3.16: 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 3.15 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 3.17):

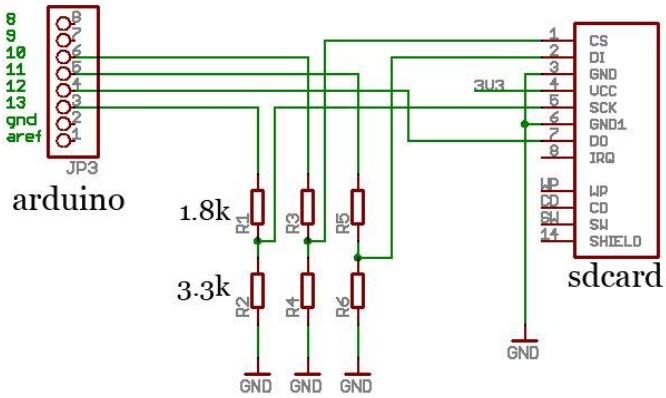


Figura 3.17: Esquema de montaje del lector de tarjetas SD.

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 3.18 e insertamos el adaptador con los alambres en la tarjeta de topos ya soldada (ver figura 3.18 A) tal y como se observa en la figura 3.18 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 3.18 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 3.18: Proceso de construcción del lector de tarjetas MicroSD.

No obstante, pese a que el funcionamiento del adaptador "casero" ha funcionado perfectamente, se ha decidido adquirir uno comercial cuando esto ha sido posible, habiendo elegido el de la Figura 3.19.

Figura 3.19: Distribución de pines SD.

3.2.3. Diseño del PCB del Módulo Principal

Con las indicaciones de diseño expuestas en el apartado anterior, nos encontramos en condiciones de definir el PCB que compondrá el Módulo Principal. El presente proyecto ha sido una continuación del proyecto que fue expuesto para la Ingeniería Técnica de igual nombre, el cual ha sido alterado realizando modificaciones en las pistas a mano para poder disponer de un prototipo para la realización de las pruebas necesarias para poder elaborar el diseño final, cuyos planos viene recogidos en el Apartado II-Planos y especificaciones de la presente memoria.

En la Figura 3.20 se recoge el diseño del prototipo que se diseñado para acometer las pruebas necesarias para poder elaborar el diseño final.

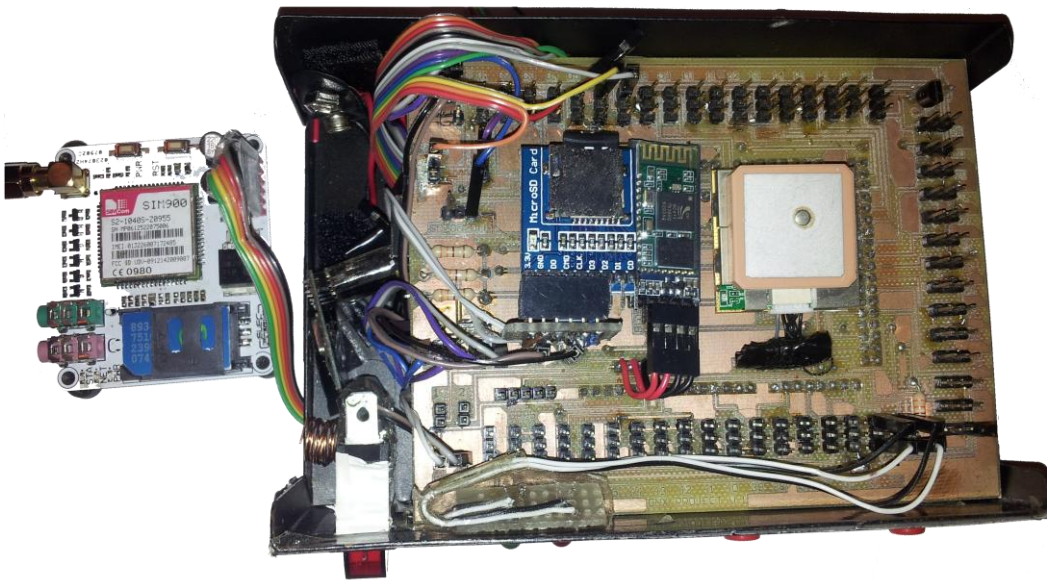
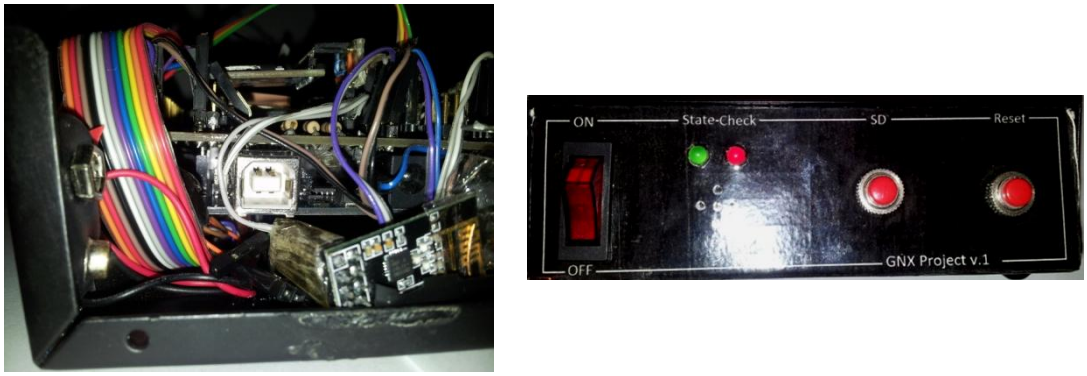


Figura 3.20: Prototipo de prueba del GNX Project v2

En este prototipo los elementos se encuentran en la cara Top del PCB, siendo estos el lector de tarjetas MicroSD, el módulo Bluetooth y el GPS. Observamos que en la periferia de la placa, formando una U, hay un total de 50 conectores, compuesto por tres terminales cada uno (Datos, +5V, GND), siendo estos donde se conectarán los periféricos soportados por el proyecto (sensores y actuadores) y que se describirán más adelante. Si observamos en la Figura 3.21, en el lateral encontramos el módulo de radiofrecuencia NRF24L01 así como el puerto USB del microcontrolador de forma accesible para poder actualizar el firmware. De forma anexa al PCB encontramos el módulo GSM/GPRS SIM900, conectado por cable a un puerto de comunicación establecido en la placa. En el frontal encontramos el interruptor de interruptor general, así como dos pulsadores y dos indicadores led.



(A) **(B)**

Figura 3.21: Lateral y frontal prototipo de prueba del GNX Project v2.

En la figura 3.21 B observamos que disponemos dos indicadores led, uno de color verde llamada State y otro de color rojo llamada Check, los cuales se encuentran conectados a los pines 33 y 34 del Arduino Mega respectivamente. Estos indicadores se emplean para avisar de múltiples incidencias en el sistema:

- **Led State fijo y Check apagado**: Indica que el sistema ha logrado montar la tarjeta de memoria MicroSD así como que ha logrado establecer comunicación con el módulo GSM/GPRS.

- **Led Check parpadeando a intervalo 1 segundo y State apagado**: Indica al iniciar el sistema, la existencia de algún fallo en alguno de los elementos anteriores, es decir, en el GSM/GPRS y/o la MicroSD. Cuando se da esta situación, el sistema no puede funcionar con normalidad ejecutando y recibiendo órdenes.

- **Led Check parpadeando y State encendido**: Indica que el sistema está realizando una comprobación de la bandeja de entrada de SMS, haciendo que el led Check conmute de estado cada vez que se comprueba el siguiente SMS, tras ser eliminado el anterior (al no ser válido o al ser válido tras ser ejecutado). Esta situación se da cada 45 segundos que es el intervalo predefinido para poder comprobar la bandeja de entrada de SMS para ejecutar las órdenes o configuraciones que el usuario ha mandado empleando el modo de conectividad SMS, del cual se hablará de forma detallada en los próximos capítulos.

- **Led State y Check parpadeando de forma alternativa:** Indica que se está realizando una operación de escritura en la MicroSD o bien que se está eliminado la configuración cargada en el sistema, es decir, las tareas que el usuario ha programado, que se ejecutarán en el momento en el que se cumplan las distintas condiciones antes definidas.

- **Led State parpadeando 3 veces**: Indica al iniciar el sistema ha transmitido información al usuario por cualquiera de los tres modos de comunicación entre el sistema y el Smartphone implementados: Bluetooth, 3G, SMS.

- **Led Check fijo**: Indica que es seguro desmontar la tarjeta MicroSD. Cuando está apagado durante el funcionamiento normal del sistema, indica que la tarjeta está correctamente montada y que se puede escribir y leer de ella de nuevo.

Aparte de los indicadores led, el prototipo de la Figura 3.20, cuenta con dos pulsadores. Durante las pruebas realizadas, se observó 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 47 (State) un segundo indicador luminoso conectado al pin 48 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 conectado al pin 46 cuya función era la de poner un flag a 1 impidiendo la grabación en ella cuando se deseara desmontar la tarjeta. 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 Smartphone durante ese tiempo, de lo contrario, al estar constantemente grabando las órdenes no llegarían a ser nunca ejecutadas.

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 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 45; por último conectar el colector a masa y el emisor al pin de reset de Arduino como vemos en la figura 3.22.

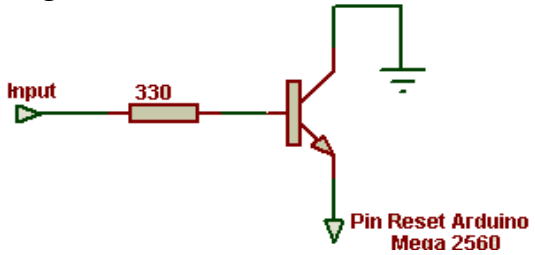


Figura 3.22: Esquema para reiniciar la placa por software.

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**.

Con los datos recogidos en el presente prototipo se ha elaborado una versión definitiva del PCB conde se han mantenido los indicadores, botones y altavoz presenten en el prototipo, pero se han agregado otras opciones y una nueva disposición. En la Figura 3.23, observamos el diseño del PCB del Módulo Principal, donde podemos observar la distribución de los elementos.

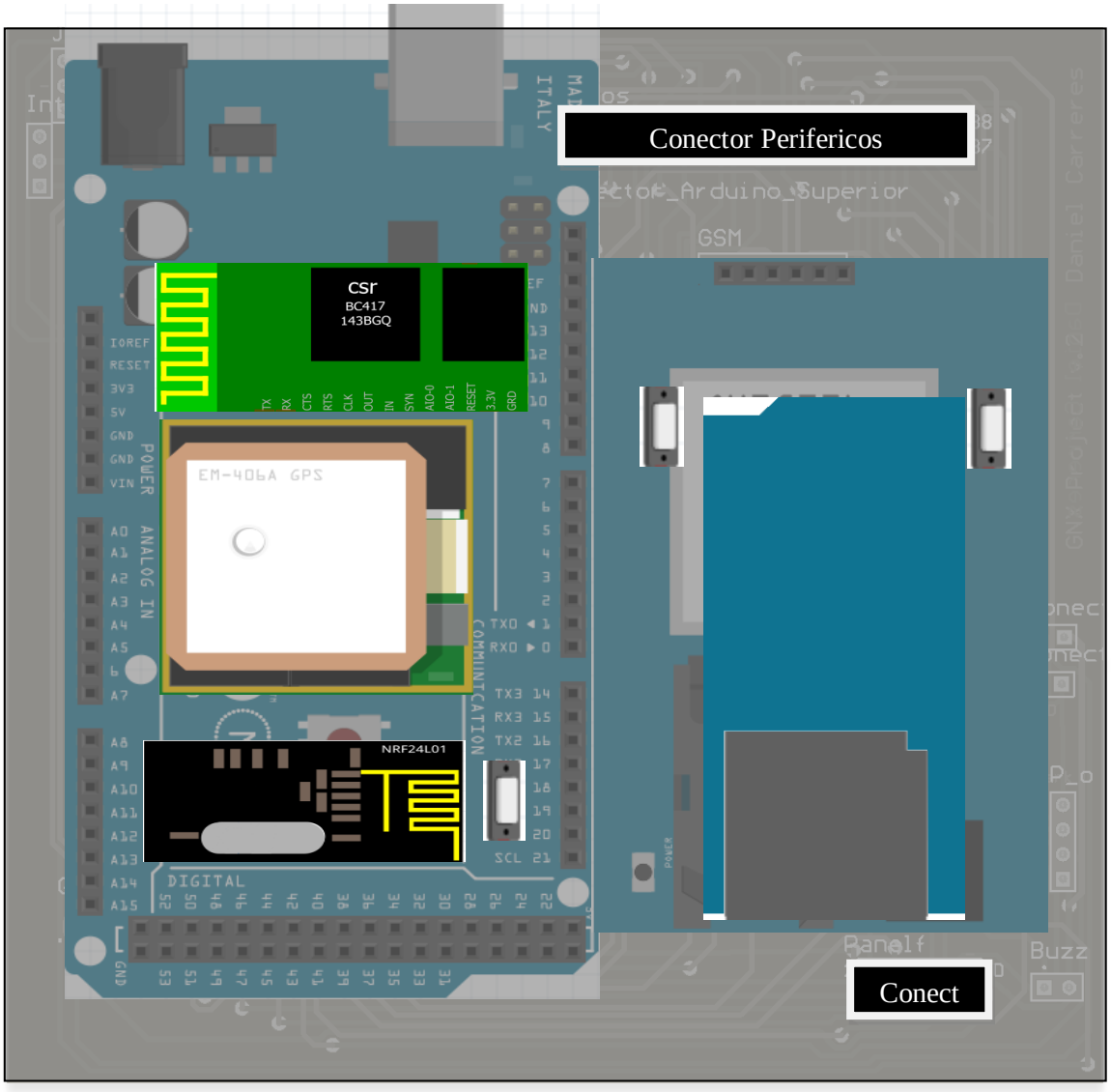


Figura 3.23: Distribución elementos en PCB Módulo Principal.

Los cambios respecto a la placa desarrollada en el prototipo nos notables. En primer lugar el tamaño del PCB se ha visto reducido notablemente pasando a medir 11x10,5 cm. Esto ha sido posible gracias a la nueva distribución de los elementos. El microcontrolador (Arduino Mega), se encuentra emplazado en la izquierda del PCB conectado directamente con la capa Button, de forma que su puerto USB se encuentra accesible desde la parte posterior de la carcasa que donde se emplazará (Ver Figura 2.24).

Figura 3.24: Carcasa del Módulo Principal.

De esta forma se podrá actualizar el firmware del microcontrolador así como posibilitar conectarlo al PC para que en futuros desarrollos del proyecto sea posible el control del sistema por USB a través de una comunicación por cable con el puerto serie por medio del desarrollo de alguna aplicación para PC.

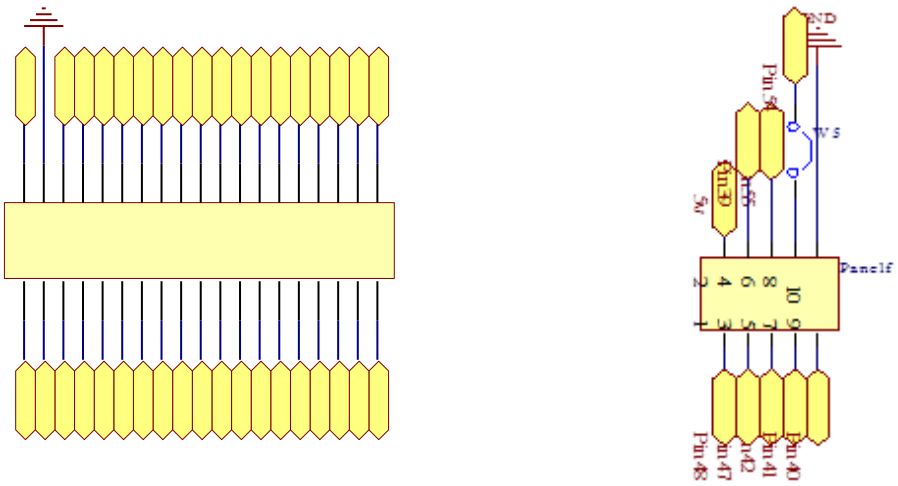
Como se aprecia en la Figura 3.23, en la capa Top y justo encima de la zona donde se encuentra el Arduino Mega 2560 (a fin de reducir el recorrido de las pistas) se emplazan el módulo Bluetooth, el GPS y el módulo de radiofrecuencia NRF24L01. Además justo debajo del NRF24L01 se ha colocado un pulsador de borrado, el cual al ser pulsado durante un periodo superior a los 5 segundos provoca el borrado de todas las tareas programadas en la memoria EEPROM, acompañando este proceso de una señalización por medio de los indicadores Check y State de forma alternativa. A la derecha y a escasos centímetros de los elementos antes citados, encontramos el módulo GSM/GPRS EFComPro, el cual se conectar directamente a la capa Button, de forma que la antena del módulo queda plegada debajo del PCB orientada hacia la parte inferior de la placa.

Justo encima del GSM/GPRS se encuentra el lector de tarjetas MicroSD, justo en el centro de dos pulsadores, uno para montar/desmontar la tarjeta y otro para poder reiniciar el Módulo Principal. Además se cuenta con dos tomas adicionales que permiten poner de forma externa dichos pulsadores en el lugar que al usuario le sea más útil. Por ejemplo, el pulsador de la MicroSD, se encuentra en el diseño final además de en el PCB en el lateral izquierdo de la carcasa.

En la parte superior izquierda observamos los conectores relativos a la alimentación, así como la presencia de un conector adicional pensado para poder acoplar otra placa consistente en un circuito regulador de tensión con sistema de batería de emergencia que se describirá en el último apartado del presente capítulo. También se

dispone de un sistema para medir el voltaje de la batería consistente en un sencillo divisor resistivo conectado a una entrada analógica del controlador.

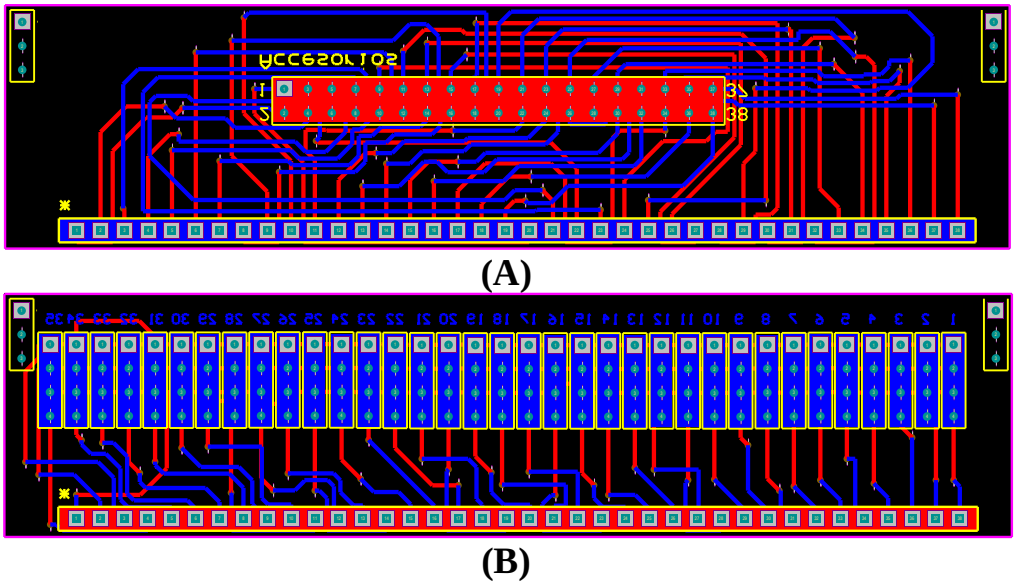
La reducción de espacio del PCB ha sido posible entre otros factores a la eliminación de los pines de conexión de los periféricos que se encontraban en el prototipo de la Figura 3.20 en la periferia formando una U; además estos eran difíciles de conectar dado que requerían que no se colocara la parte superior de la carcasa para poder dejar salir los cables. Su número se ha reducido de 50 a 35 pines configurables por el usuario. Por ello en esta ocasión se ha optado por unificar en un único conector todos los pines, es decir, mediante un conector (Figura 3.25 A) se dispone de todos los pines, tanto de datos como de alimentación y un pin adicional común a todos que será empleado para poder conectar dispositivos que requieran de cuatro terminales de conexión, como es el caso de los ultrasonidos, siendo en este caso este terminal (Pin 38) empleado para dar el pulso. Mediante una faja de conexión, se unirá con un conector externo que se ubicará en el lateral trasero de la carcasa, donde se descompondrá en los terminales necesarios para cada uno de los 35 conectores que dispone el proyecto, descomponiéndose en Datos, Vcc, Gnd y Pulso.



(A) **(B)**

Figura 3.25: Conectores del Módulo Principal.

Este conector a su vez se compone de dos partes dado que por las reducidas dimensiones no era posible realizar el routeado de las pistas en un PCB de las dimensiones apropiadas. Por un lado está el adaptador para pasar de disponer los pines en forma de zócalo de dos filas a uno en forma de una única fila para facilitar el routeado de la segunda parte del conector (Figura 3.26 A) y por otro lado encontramos el PCB que se acopla al anterior y donde se realiza la multiplexión de los pines (Figura 3.26 B).



(A)

(B)

Figura 3.26: Partes del conector de periféricos Módulo Principal.

Otro de los conectores presentes en el PCB son los relativos al panel del control (Figura 3.25 B) donde se concentra todas las conexiones necesarias para los indicadores led e interruptores necesarios.

Los esquemáticos y los PCB's de todos los elementos del Módulo Principal, serán recogidos la parte de la memoria destinada para dicho fin: **Memoria parte II-Planos y especificaciones.**

3.3. Estación

Las Estaciones son módulos de reducidas dimensiones y de escasa entidad constructiva destinadas a prolongar el ratio de acción del Módulo Principal. Puede contener exactamente el mismo tipo de periféricos (sensores y actuadores) que el Módulo Principal, los cuales son gestionados por él, permitiendo de esta forma poder ampliar el número de pines disponibles, más allá de los 35 del Módulo Principal (cada Estación tiene un máximo de 10 pines configurables, siendo siete el número máximo de Estaciones que puede soportar el sistema debido a limitaciones impuestas por el software).

Las Estaciones evitan tener que cablear todos los periféricos al Módulo Principal, de forma que se evitan pérdidas de señal y errores de medida, además permite que las Estaciones puedan emplazarse a gran distancia del Módulo Principal (50-80 metros si se emplea el NRF24L01 o 1Km si se emplea el NRF24L01-PA-LNA). Por lo que permite emplear el sistema para poder realizar por ejemplo tareas de domótica, donde se puede disponer del Módulo Principal en una habitación con algún periférico conectado, y disponer de una Estación en cada una de las restantes habitaciones.

Para el desarrollo de las Estaciones, se ha optado por un modelo de la plataforma Arduino de reducidas dimensiones, con un número de pines no muy elevado y con una rápida velocidad de procesamiento. Por ello, el corazón de cada Estación está formada por un Arduino Nano V3 (ver Figura 3.27).

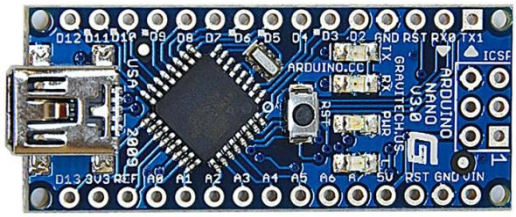


Figura 3.27: Arduino Nano v3.

El otro elemento básico que componen las Estaciones es un módulo de radiofrecuencia NRF24L01, como el del Módulo Principal (Figura 3.12 A) aunque también se podría emplear el de la Figura 3.12 B. En la Figura 3.28 se puede observar el prototipo elaborado mediante tarjeta de topos a fin de poder realizar las pruebas pertinentes.

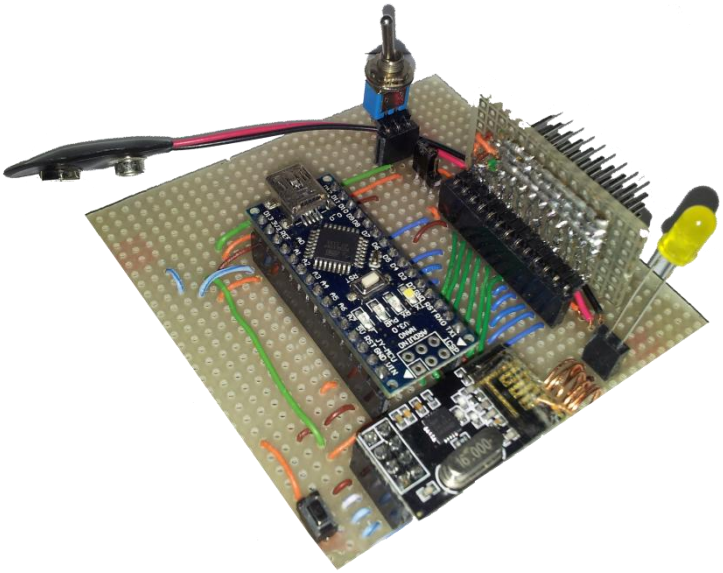


Figura 3.28: Prototipo de Estación.

Además, se incorpora un indicador led, que parpadea a alta velocidad cuando hay una comunicación establecida entre el Módulo Principal y dicha Estación. Para ello, cada Estación dispone de su propio de canal, el cual debe especificarse cuando se instala el firmware, aspecto que se comentará en el próximo capítulo. Los periféricos se conectan por medio de un sistema muy similar al del Módulo Principal.

De las pruebas realizadas con dicho prototipo, se vio la necesidad de introducir otros elementos adicionales y buscar un diseño aún más reducido. En la Figura 3.29 se puede apreciar el PCB final de una Estación.

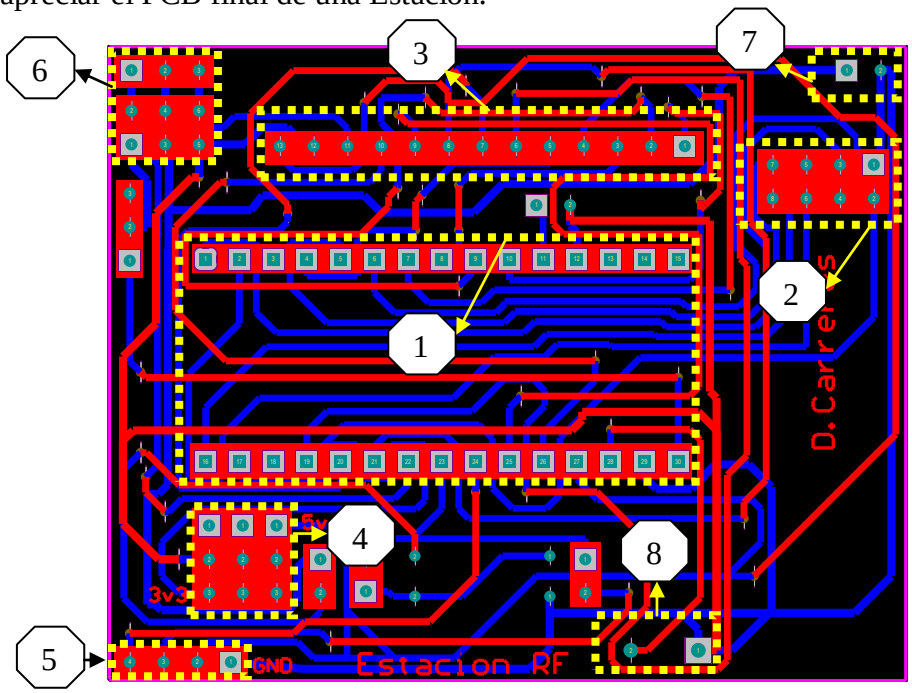


Figura 3.29: PCB Estación.

En la Figura 3.29, se observa que en el centro encontramos el Arduino Nano V3 (Ref.1), dispuesto de forma que el puerto USB se encuentra accesible desde el lado izquierdo. En el lado derecho encontramos el módulo de radiofrecuencia (Ref.2), justo detrás del led indicador de estado (Ref.7). En este diseño se ha añadido la posibilidad de conectar un display LCD por I2C (Ref.5) para dar la posibilidad en el futuro de poder transmitir mensajes de texto entre las Estaciones así como conocer por escrito cualquier eventualidad. También dispone de un zócalo de conectores de alimentación (Ref.4), que proporciona tres tomas de alimentación distinta, donde en cada una es posible elegir entre los 3,3V o los 5V. Se cuenta con un pulsador de reinicio (Ref.8), destinado a poder restablecer la comunicación en caso de fallo. Por último, en disponemos de una serie de conectores para la alimentación del módulo (Ref.6), donde al igual que en el Módulo Principal, está pensado para poder conectar a él un circuito adicional de regulación y alimentación auxiliar, que se describirá en el siguiente punto de forma detallada. Por último disponemos igualmente de la posibilidad de conocer el nivel de batería disponible por medio de un divisor resistivo conectado a un pin analógico.

3.4. Circuito de regulación de tensión y alimentación auxiliar

Como se ha mencionado con anterioridad, el sistema en su conjunto debe de disponer de una alimentación estable para poder realizar todas sus operaciones. Aunque sus necesidades varían según el número de periféricos conectados (y su tipo) así como si se desea acceder a internet o se ha vinculador con el satélite o no.

Módulo debe poder ser alimentado directamente de la red (mediante un transformador externo) o desde el encendedor de un coche (dado que como se comentará en los siguientes capítulos una de las aplicaciones del sistema es orientado a los vehículos), por lo que es necesario un sistema que regule la tensión dentro de los límites operativos del sistema, es decir entre los 5-12V. Pero como se ha mencionado, debe existir la posibilidad de poder variar dicho voltaje para poder ajustarlo a las necesidades concretas del sistema. Si el sistema se va a emplear para poder registrar el recorrido GPS realizado en la MicroSD, no es necesario que se le suministre un voltaje mayor a los 8-9 V, sin embargo si va a tener conectado un número considerable de periféricos de gran caída de tensión (pequeños motores por ejemplo), el voltaje debería ser aumentado.

Además, no siempre es posible disponer de una conexión a la red eléctrica, por lo que el sistema debería ser capaz de funcionar a batería. Se plantea el caso de que en caso de que el suministro de energía principal (de la red) falle, ya sea debido a un apagón o a una desconexión accidental de la red, debería activarse un sistema de alimentación de emergencia que entrara en funcionamiento de forma inmediata para evitar que el módulo se reinicie.

Por ello se ha implementado el siguiente circuito de la Figura 3.30.

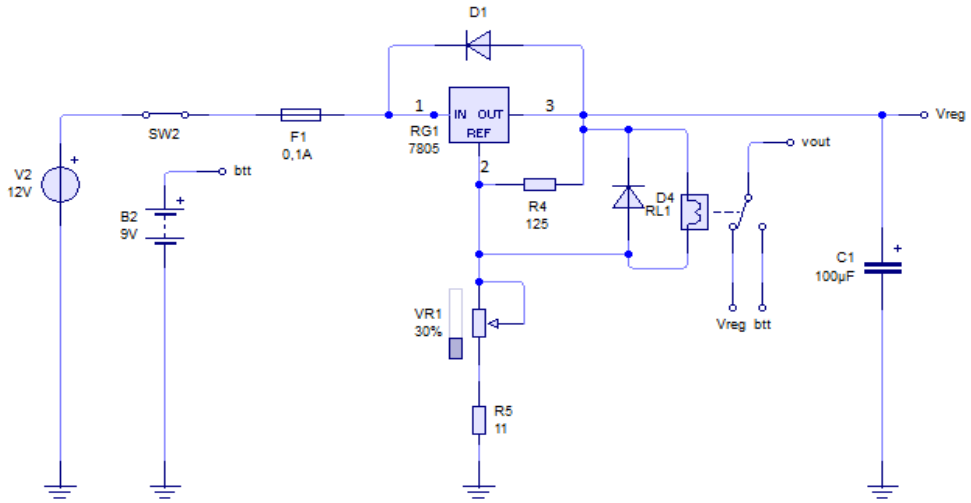


Figura 3.30: Circuito de regulación de tensión y alimentación auxiliar.

El funcionamiento del circuito es el siguiente:

- La tensión de entrada pasar por un regulador de tensión LM7805 que baja la tensión hasta los 5 voltios en la salida.
- Como deseo tener una alimentación variable, coloco un sistema de resistencias cuyos cálculos se describirán a continuación para poder varia la tensión desde los 5,5V hasta los 12V (en realidad un poco meno dado que no alcanzaremos la saturación).
- Cuando circula corriente por el regulador de tensión, entre los terminales 2-3 hay una tensión teórica de 5V, la cual se emplea para alimentar un relé que conmutará para permitir que la tensión entre 3 y masa llegue a la carga.
- Mediante el potenciómetro de 135Ω se puede variar la tensión a la salida entre los 5,5V hasta los 12V.
- Cuando se produce un fallo en la alimentación, el relé deja de estar alimentado volviendo a su estado inicial y permitiendo que la carga se alimente a través de la batería de 9V incorporada.
- El condensador de 100µF permite retener la carga lo suficiente como para que durante la conmutación no se produzca el reinicio de la carga, es decir, el Módulo Principal o de las Estaciones.

Los cálculos de las resistencias son los siguientes:

$$R_1 = \frac{V_{xx}}{5*I_q} = \frac{5}{5*8*10^{-3}} = 125\Omega$$

Para V_{R2}=0V, deseo obtener 0,5V más

$$R_3 = \frac{0,5}{6*I_q} = \frac{0,5}{6*8*10^{-3}} = 10,4 \simeq 11\Omega$$

Para R_2 máxima, V_{out}=12V

$$12 = 5 + V_{R2} + 0,5 \rightarrow V_{R2} = 6,5V$$

$$R_2 = \frac{V_{R2}}{6*I_q} = 135,42 \simeq 135\Omega$$

4. Diseño del firmware y modos de comunicación

4.1. Introducción

En el presente capítulo, vamos a describir de forma detallada el funcionamiento interno de los elementos más importantes del firmware tanto el Módulo Principal como de las Estaciones.

4.2. Firmware Módulo Principal

El sistema GNX Project, 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 6. Debido a la gran extensión del código, a fin de comprender de una forma más profunda el funcionamiento interno del proyecto, se descompondrá en los aspectos más importantes del mismo a fin de poder realizar una mayor profundización en el funcionamiento del mismo.

4.2.1. Detección y procesado de las instrucciones recibidas

Recordemos que el sistema dispone de cuatro modos de comunicación: Bluetooth, 3G, SMS y Radiofrecuencia, donde los tres primeros son empleados para la comunicación Módulo Principal-Usuario.

Independientemente del modo de comunicación que el usuario decida emplear, el cual es seleccionado desde el menú "Conectar" de la aplicación para Smartphone que describiremos en el próximo capítulo de forma detallada; la información que se transmite es en esencia la misma, así como la forma en la que esta es procesada en última instancia, cambiando únicamente el medio de soporte del mensaje y la fase de extracción de la información contenida en el mensaje para posteriormente pasar dicha información por un algoritmo llamado Detectar_Codigo().

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). En la Figura 4.1, se observa de forma resumida los distintos valores que puede tomar en función de la variable E/S.

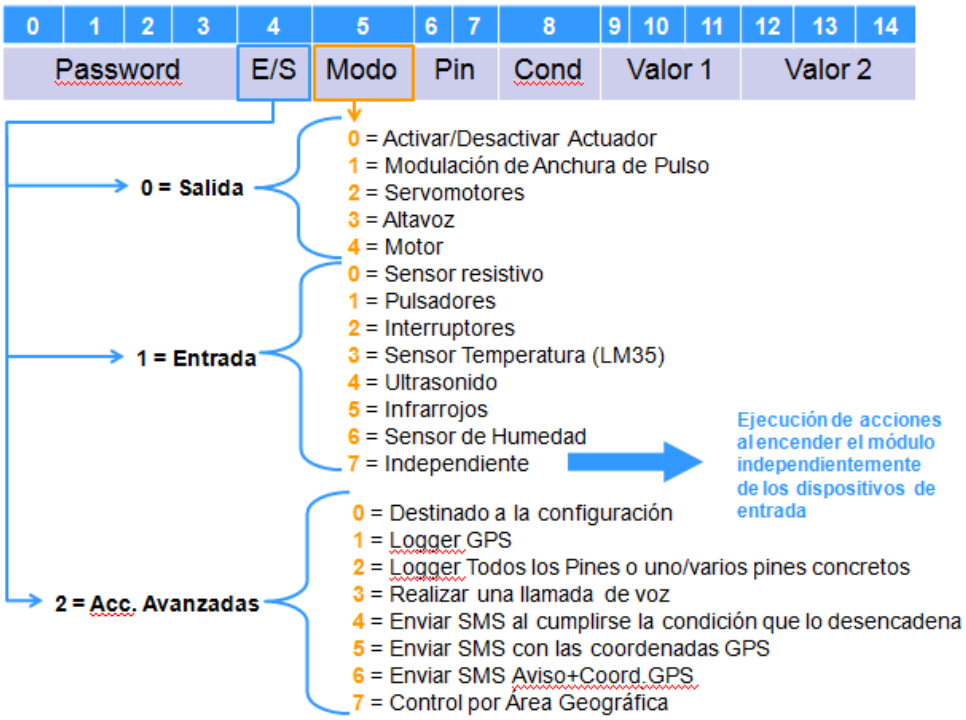


Figura 4.1: Resumen de valores que puede tomar "Modo".

- **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….

- ❖ **<u>Condición =2</u>**→ Si Valor1< Entrada < Valor2….

- ❖ **<u>Condición =3</u>**→ 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 1=Variar Corriente 2=Servo 3=Altavoz 4=Motor				X	X	X	Temporización			Accionar Relés
									Valor (0-255)			X	X	X	Variar Corriente
									Ángulo(0-180º)			X	X	X	Servo
									Frecuencia(0-255)			Temporización			Altavoz incorporado o uno externo
									Velocidad (0-255)			Temporización			Motor de corriente continua
				1 [In]	0=Sensor Resistivo 1=Pulsadores A/D 2=Interruptor 3=Sensor de Temp. 4=Ultrasonido 5 = Infrarrojo 6=Independiente 7=GPS/Reloj			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 (>) Entrada > Valor1				X	X	X	
								2 (<) Entrada < Valor1				X	X	X	
								3 (< x<) Valor1<Entrada < Valor2							
								4 (->) Valor1→Salida				X	X	X	
				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	1	1	X	X	X	X	X	X	"Eliminar acc programada EEprom" y saltar posicion
						1	2	X	X	X	X	X	X	X	Modificar una instrucción
						2	0	0 Telemetria normal 1 Trasmitir acc progra	X	X	X	X	X	X	Telemetría
					1	X	X	X	X	X	X	X	X	X	Logger GPS
					2	PIN:00 PIN:XX		X	X	X	X	X	X	X	Logger Sensor (**E**l pin 00, indica que se registren todos los pines, o un pin concreto)
					3	Telefono									Llamadas
					4	Telefono									SMS aviso
					5	Telefono									SMS GPS
					6	Telefono									SMS Aviso+GPS
				3	**2**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	Fin de acciones programadas

Pin:00→Velocidad
Pin:01→Altitud

Tabla 4.2: Cuadro resume 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 algoritmo Detectar_codigo():

Algoritmo Detectar_codigo()

```
int Detectar_Codigo(int n) { //Detecta el código enviado por bluetooth o por GSM y transforma de "caracteres individuales" a un numero entero
              if(Datos_GSM_Recibidos==1){

                while(Datos_GSM_Recibidos==1){
                  if((Buffer_GSM[Cursor_GSM]=='X') ||
(Buffer_GSM[Cursor_GSM]=='\0')|| (Buffer_GSM[Cursor_GSM]=='\n') ||
(Buffer_GSM[Cursor_GSM]=='O' || Buffer_GSM[Cursor_GSM]=='K') ||
(Cursor_GSM==sizeof(Buffer_GSM)-1)){
                      Cursor_GSM=0;
                      Datos_GSM_Recibidos=0;
                      Cursor_Aux_GSM=0;
                      memset(Buffer_GSM, 'X', 200);
                      break;
                    }
                    else if(Buffer_GSM[Cursor_GSM]!=';'){

                      Codigo[Cursor_Aux_GSM]=Buffer_GSM[Cursor_GSM];
                      Codigo[Cursor_Aux_GSM]=Codigo[Cursor_Aux_GSM]-48;
                      Cursor_GSM++;
                      Cursor_Aux_GSM++;

                    }
                    else if(Buffer_GSM[Cursor_GSM]==';'){
                      Cursor_GSM++;
                      Cursor_Aux_GSM=0;
                      break;
                    }
                  }
              }
                else{

                  if(Serial3.available()){//Detecto por Bluetooth
                    Modo_Modulo=0;//Modo Bluetooth
                   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 código
}//FIN
```

Este algoritmo se encarga de descomponer las tramas que llegan por cualquiera de los modos de comunicación antes descritos para almacenarlas en una serie de variables (ES; Modo; Pin;Condicion; Valor1; Valor2), además también devuelve la contraseña del sistema, lo cual se empleará para ver si la trama recibida es válida o no, desde el punto de vista de un intento de vulnerabilidad. Por defecto, la contraseña del sistema es 1234, la cual se encuentra almacenada en la memoria EEPROM.

La adquisición de información se hace desde dos puertos serie:

- **Serial 1:** Correspondiente al módulo GSM/GPRS. Cuando se detectan tramas por SMS o por 3G (descargadas del servidor FTP) son procesadas por los algoritmos correspondientes (que se comentarán más adelante) y posteriormente se almacenan en un array llamada Buffer_GSM y poner el flag Datos_GSM_Recibidos=1.

- **Serial 3:** Corresponde al módulo transceptor bluetooth. Es el modo de comunicación que menos esfuerzo computacional requiere. 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 4.2, 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	?

Figura 4.2: 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.

4.2.2. 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" o "Trama de Clausura" (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.

Algoritmo de ejecución y almacenamiento de tareas

```
  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);
                  if(ES==2 && Modo==0 && Pin==20){
                  }
                  else{
                     //ENVIAMOS INFORMACION AL MODULO
                        if(Modo_Modulo==0){//Bluetooth
                          Serial3.print("#OK");
                        }
                        else if(Modo_Modulo==1){//Internet
                         Serial.println("Internet");
                          if (SD.exists("internet.txt")) {
                                 SD.remove("internet.txt");
                          }
                            myFile = SD.open("internet.txt",
FILE_WRITE);
                                if (myFile) {
                                   myFile.print("#OK");
                                   myFile.close();
                                    Serial.println("done.");
                                }
                                else {
                                 Serial.println("Error abriendo
archivo");
                                }

                                Reset_GSM();
                                power_on_GPRS();
                                sendATcommand("AT+CREG?", "+CREG: 0,1",
500);
                                 configure_FTP();
                                 uploadFTP();

                                 if (SD.exists("internet.txt")) {
                                    SD.remove("internet.txt");
                                 }
                                 Reset_GSM();

}
                           else{//SMS-->Hacemos una llamada perdida para no gastar
                              while( (sendATcommand("AT+CREG?", "+CREG:
0,1", 500) || sendATcommand("AT+CREG?", "+CREG: 0,5", 500)) == 0
);
                               Serial1.print("ATD");
                               Obtener_Telefono(Pin, Condicion, Valor1,
Valor2);
```

```
                            Serial1.println(";");
                           Serial1.println("#OK");
                             delay(10 * 1000);
                            Serial1.println("ATH");
                        }
                    }

                    digitalWrite(Led,HIGH);
                     for (int q=0; q<2; q++){
                        digitalWrite(Led,!digitalRead(Led));
                        delay(100);
                      }
                  }
                  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;
                          //ENVIAMOS INFORMACION AL MODULO
                          if(Modo_Modulo==0){//Bluetooth
                            Serial3.print("#OK");
                          }
```

```
                        else if(Modo_Modulo==1){//Internet

                        }
                        else{//SMS-->Hacemos una llamada perdida para no gastar
                         while( (sendATcommand("AT+CREG?", "+CREG:
0,1", 500) || sendATcommand("AT+CREG?", "+CREG: 0,5", 500)) == 0
);
                              Serial1.print("ATD");
                              Obtener_Telefono(Pin, Condicion, Valor1,
Valor2);
                              Serial1.println(";");
                              delay(10 * 1000);
                              Serial1.println("ATH");
                        }
                    //Serial.println("Trama de clausura");
                    digitalWrite(Led,HIGH);
                   for (int q=0; q<2; q++){
                      digitalWrite(Led,!digitalRead(Led));
                      delay(100);
                    }
                    delay(100);
               }
            }

              Borrar_Buffer_Codigo();
       //}
         }//Fin Detectar Codigo
```

La fracción de código mostrada (designada como "Código de ejecución y almacenamiento de tareas") se comprueba en primer lugar si la trama a procesas dispone de la contraseña correcta, de ahí que se pase como parámetro al función Detectar_codigo(), antes descrita. Si la contraseña es incorrecta la trama es desechada, en caso contrario se contemplan varias condiciones distintas:

- **La trama tiene E/S=0 ó E/S=2:** Si no existe ninguna trama que le preceda con E/S=1, indica que se trata de una acción inmediata del usuario que deseamos que se ejecute en ese instante, por lo que no se trata de una tarea programada y por lo tanto no es necesario almacenarla en la EEPROM. Las acciones inmediatas se almacenan también den su posición correspondiente (dependiente del pin que se haya configurado) de un array llamado **Buffer_SRAM_Accion[]**, el cual se empleará posteriormente para las labores de telemetría.

- **La trama comienza E/S=1:** Estas tramas son las llamadas "***Cabecera del Sensor***", que es una trama que incluye el tipo de sensor, el pin en el que se encuentra conectado (ya sea del Módulo Principal o de alguna Estación) y la condición que se tiene que cumplir para que se ejecuten las acciones que siguen a esta. Esta trama inicia el proceso de grabación en la EEPROM en la posición inmediatamente siguiente a la última tarea almacenada. Cuando se ha almacenado la "Cabecera del Sensor" se inicializan una variable (Inicio_Conf_Sensor=1) que indica que las siguientes tramas que tendrán la misma estructura que las órdenes directas (E/S=0 ó E/S=1) son las acciones a asociar al cumplimiento de la condición antes definida, y por lo tanto deben almacenarse en la EEPROM en la posición justo siguiente a la de la cabecera. Este

proceso se repite hasta que se detecta la "Trama de Clausura" (Contraseña+"32000000000"), la cual indica que ya no hay más acciones a asociar y que por lo tanto la tarea ya está definida y que todas las tramas que se mandarán posteriormente serían o bien otra tarea, o bien una acción inmediata.

Cuando se **recibe** una orden o una programación de una tarea, siempre se manda una confirmación al usuario por el mismo medio por el que llegó la información recibida. El sistema detecta el modo de comunicación y en consecuencia manda una respuesta de confirmación de la forma "#**OK**" sin comillas. La forma de envío difiere por tanto en función del modo empleado:

- **Bluetooth**: Es la más sencilla. Se envía una cadena de texto por el puerto serie número 3, mediante el comando Serial.print("#OK").
- **3G**: Se crea un archivo txt con el nombre "internet.txt" en la raíz de la tarjeta MicroSD. En él se escribe el código de confirmación "#OK". Tras esto se realiza el proceso de carga de dicho archivo al servidor FTP donde se almacenará con el nombre: "Buffer_Ordenes.txt". El proceso de subida y bajada de datos del servidor FTP se comentará más adelante.
- **SMS**: Se manda un SMS con la confirmación "#OK" por medio de una serie de comandos AT. Lo que se hace es poner el módulo en modo SMS, mediante AT+CMGF=1. Tras ello definimos el número de teléfono de destino, que está almacenado en el Módulo Principal y tras ello se introduce el cuerpo del mensaje y se envía.

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**.

Si una condición no se cumple en el momento de comprobarla, por ejemplo la condición definida en el Bloque 1 de la Taba 4.4, se saltaría automáticamente al Bloque 2 y se repetiría el proceso de comprobación.

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.

Algoritmo ejecución de las instrucciones asociadas a un sensor.

```
//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
                            //Ejecucion=2;
                         }
                         else
                            Ejecucion=1;
                      }
                      k++;
                    }

                    else{
                         //if(millis()==5000)
                         //Ejecucion=2;
                         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
                         //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;
                        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]==4 || Buffer_SRAM_Accion[1]==5
|| Buffer_SRAM_Accion[1]==6){//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
                         Serial.print("-Sen.Res pin ");
                       }
                       else if(Buffer_SRAM_Sensor[1]==1){//Pulsador
                         Serial.print("-Pulsador pin ");
                       }
                        else if(Buffer_SRAM_Sensor[1]==2){//Interruptor
                         Serial.print("-Interruptor pin ");
                       }
                       else if(Buffer_SRAM_Sensor[1]==3){//LM35
                         Serial.print("-Sen.Temp pin ");
                       }
                       else if(Buffer_SRAM_Sensor[1]==4){//Ultrasonido
                         Serial.print("-Sen.Sonico pin ");
                       }
                       else if(Buffer_SRAM_Sensor[1]==5){//Infrarrojo
                         Serial.print("-Sen.Ir pin ");
                       }
                       Serial.print(Buffer_SRAM_Sensor[2]);
                       Serial.print(": ");
                       Serial.println(Value_Sensor);
                       AuxSMS[AuxSMS[0]+1]=k;//Almacenamos el valor de K de esa accion
bloqueada a fin de luego desbloquearla
                       //Serial.println( AuxSMS[AuxSMS[0]+1]);
                       AuxSMS[0]++;
                       // Serial.println(AuxSMS[0]);
                        //Serial.println(k);

                    Once[k]=1;
                 }
```

```
                        // if(Timer[26]==0)
                            //Timer[26]=millis();
                          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;
                             //Serial.println(AuxSMS[r]);
                            }
                            AuxSMS[0]=0;
                            Estado_SMS=0;
                          }

                    }//fin sms
                  else if(Buffer_SRAM_Accion[1]==3){//Es una llamada, impido que se
realizen llamadas en menos de x tiempo.
                  //A diferencia de los SMS, como las llamadas perdidas no tienes un coste, se permiten que
se realizen llamadas segidas
                  //siempre que sean de acciones distintas.
                    if(millis()-Temoirizaciones[k]
>=Tiempo_SMS_Llamadas*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]);
                     }
                  }
                 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[2]>69)
            Envio_Estacion[Buffer_SRAM_Accion[2]-70]=2; //Indicamos a la estación el tipo de sensor y el pin
donde esta.
        }*/
               }
              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 (Servos, Motores, etc...)
                    Once[k]=1;
                    Name_Pin_Aux=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]);
                  }
                }

              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);
                    }
                    else{//Si es un servo, un motor, etc...
                       Once[k]=0;

                       }
              }
                   k++;

            }
          }
            Ejecucion=0;
          }


           if(k>NAcciones)//Para reiniciar el ciclo
           k=0;
```

Este segmento de código está constantemente en ejecución, salvo en el momento en el que se produce una programación de una tarea a fin de evitar fallos

durante la grabación. Es la fracción de código es el que regula la activación o desactivación de las distintas acciones que componen las tareas. Para ello, debe interactuar con otros dos segmentos de código: Uno encargado de "traducir" la lectura del sensor en función del tipo que sea y otro que contienen todas las acciones que el sistema puede ejecutar, tanto como acción inmediata como parte de una tarea programada.

A continuación se muestra la fracción de código que se encarga de procesas la sensorización y de determinar si la condición definida en la "Cabecera del Sensor" se cumple o no (ha dejado de cumplirse).

Procesado de sensores

```
  if(Read_SRAM(1,k)==1 && Inicio_Conf_Sensor==0){
       if(Buffer_SRAM_Sensor[2]>69){//Se trata de una estación, tomamos los datos
de la estación que se almacenan en el array

            Envio_Estacion[Buffer_SRAM_Sensor[2]-
70]=(Buffer_SRAM_Sensor[1])*10000+(Buffer_SRAM_Sensor[2]-
70)*1000+2;//Indicamos a la estación el tipo de sensor y el pin donde esta.

            Value_Sensor = Estado[Buffer_SRAM_Sensor[2]-
1];//Recuperamos el valor que ya viene procesado de la estación

       }
         else{
         //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)-9;//Fin Sensor LM35
            }
            else if(Buffer_SRAM_Sensor[1]==4){//Sensor Ultrasonico

             if(millis()-Temoirizaciones[k] >=0.005*56850){
               Temoirizaciones[k]=millis();
               //Ultrasonic ultrasonic(PinTriger,Buffer_SRAM_Sensor[2]); // (Trig PIN,Echo
PIN)
                     //Value_Sensor=ultrasonic.Ranging(CM);
                     unsigned long pulso;
                     pinMode(PinTriger, OUTPUT);       // ponemos el pin como
salida
                     digitalWrite(PinTriger, HIGH);    // lo activamos
                     delayMicroseconds(10);      // esperamos 10 microsegundos
                     digitalWrite(PinTriger, LOW);     // lo desactivamos
                     //pinMode(pin, INPUT);     // cambiamos el pin como entrada
                     pulso = pulseIn(Buffer_SRAM_Sensor[2], HIGH);
// medimos el pulso de salida del sensor


                     Value_Sensor =
```

```
((float(pulso/1000.0))*34.32)/2;



                    }
}
              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
                 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
                        //Ejecucion=2;
                      }
                      else
```

```
                        Ejecucion=1;
                    }
                    k++;
                }

                else{
                    //if(millis()==5000)
                    //Ejecucion=2;
                    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
                  //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;
                  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++;
        }
```

El sistema está pensado para soportar una serie de sensores y actuadores que son conectados a los pines de entrada y salida tanto del Módulo Principal como de las Estaciones. Estos elementos soportados son recogidos en la Tabla 4.5.

Sensores	Actuadores
Sensores resistivos	Relés
Pulsadores	Servos
Interruptores	Modular anchura de pulso
Sensor de temperatura LM35	Altavoces
Sensor de infrarrojo	Motores de CC.
Sensor de ultrasonido	

Tabla 4.5: Sensores y Actuadores soportados.

En este código, se analiza los datos del sensor recuperado de la EEPROM y en función de que case de sensor se trate, se realizan las modificaciones pertinentes para poder acondicionar su medida. Por ejemplo, la lectura del sensor de temperatura LM35 se obtiene de realizar la siguiente operación:

$$Valor_{Sensor} = \left(\frac{5 * Lectura * 100}{1024}\right) - 9$$

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 picos 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.

Cuando el valor definitivo del sensor es obtenido, se evalúa la condición que va asociada en cada trama, lo cual bien determinado por las casillas CONDICIÓN, VALOR1, VALOR2. En la Figura 4.2 se observa el ejemplo de una Cabecera del Sensor.

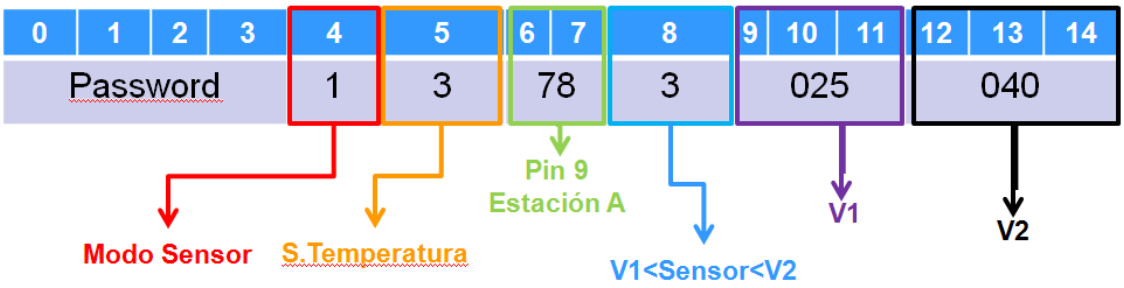


Figura 4.2: Ejemplo de Cabecera del Sensor.

En este ejemplo se trata de un sensor de temperatura (LM35) que se encuentra conectado en el pin 78, que representa el pin número 9 de la Estación remota A. Y la condición que se ha especificado es que si la temperatura está entre los 25ºC (V1) y 40ºC (V2), queremos que se ejecuten una serie de acciones que vendrán representadas por tramas posteriores.

Por otro lado, hemos mencionado que el sistema de ejecución de las tareas depende de otra función, que es la que se encarga de ejecutar las acciones propiamente dichas. Esta función llamada **Acciones()**, la cual recibe como parámetros las distintas partes de las tramas de acciones. Esta función es llamada cada vez que se desea activar una salida, (en el caso de salidas físicas) en el código que hemos designado como **Algoritmo ejecución de las instrucciones asociadas a un sensor**, que hemos descrito con anterioridad.

La función que se va a mostrar a continuación es la que se encarga también de realizar las llamadas de voz, mandar SMS, registrar la posición GPS del sistema en la tarjeta de memoria, registrar el estado de uno o varios periféricos en la MicroSD, etc.

Dado que algunas acciones como por ejemplo las que hemos designado como "Relé" (es decir, de activación/desactivación), cada vez que se ejecuta la función Acciones() cambia su estado, es decir, pasa de encendido a apagado y viceversa, de ahí que se llame a la misma función tanto cuando se cumple la condición, como cuando deja de cumplirse.

Algoritmo Acciones()

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

      if(AuxES==0){//Salida

        if(AuxPin>69){//Usamos una estación
            if(AuxModo==0){//Rele
              Envio_Estacion[AuxPin-70]=(AuxModo)*10000+((AuxPin-
70)*1000)+(!Estado[AuxPin-1]); //Indicamos a la estación el tipo de sensor y el pin donde esta.
            }
            else
              Envio_Estacion[AuxPin-70]=(AuxModo)*10000+((AuxPin-
70)*1000)+AuxValor1; //Indicamos a la estación el tipo de sensor y el pin donde esta.

        }
        else{
        if(Once_3[k]==0){//Quiero que se ejecuta 1 sola vez
                 Once_3[k]=1;
        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;
                    }
                   else if (AuxPin==20){// Telemetría
                       if(AuxCondicion==0){//Telemetria
                      Telemetria();
                    }
                    else if(AuxCondicion==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);
                  }


                }
//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: // 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);
                     }
                   }
                  }
                }
                Borrar_Buffer_Codigo();
                break;
              case 2: //Looger Sensor
                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=89;
                    if(!SD.exists("SENSORES/Sensores.txt")){
                       myFile = SD.open("SENSORES/Sensores.txt",
FILE_WRITE);

myFile.println("Lugar\tPin\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("Lugar\tPin\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)!=PinModuleGSM && (go+1)!=49 && (go+1)!=50 &&
(go+1)!=51 && (go+1)!=52 && (go+1)!=53 && (go+1)!=20 && (go+1)!=21 &&
(go+1)!=7 && (go+1)!=8 && (go+1)!=38 && (go+1)!=39 && (go+1)!=40 &&
(go+1)!=41 && (go+1)!=42 && (go+1)!=43 && (go+1)!=44 && (go+1)!=45 &&
(go+1)!=46 && (go+1)!=47 && (go+1)!=48 && (go+1)!=49 && (go+1)!=50 &&
(go+1)!=51 && (go+1)!=52 && (go+1)!=53 && (go+1)!=54 &&
(go+1)!=55){//Pines a excluir de la monitorizacion
                                  Name_Pin_Telemetria++;
                                  //Indicamos el pin y si pertenece a una estación o no
                                  if(Name_Pin_Telemetria>35 &&
Name_Pin_Telemetria<46){//Estacion A
                                      myFile.print("Est-A");
                                      myFile.print("\t");
                                      if((Name_Pin_Telemetria-35)<10)
                                          myFile.print("0");

myFile.print((Name_Pin_Telemetria-35));
                                  }
                                  else if(Name_Pin_Telemetria>45 &&
Name_Pin_Telemetria<56){//Estacion B
                                      myFile.print("Est-B");
                                      myFile.print("\t");
                                      if((Name_Pin_Telemetria-45)<10)
                                          myFile.print("0");

myFile.print((Name_Pin_Telemetria-45));
```

```
                                    }
                                    else{
                                        myFile.print("Módulo");
                                        myFile.print("\t");
                                        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(" °C");
                                     }
                                    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(" °");
                                  }
                                  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)!=PinModuleGSM && (go+1)!=49 && (go+1)!=50 &&
(go+1)!=51 && (go+1)!=52 && (go+1)!=53 && (go+1)!=20 && (go+1)!=21 &&
(go+1)!=7 && (go+1)!=8 && (go+1)!=38 && (go+1)!=39 && (go+1)!=40 &&
(go+1)!=41 && (go+1)!=42 && (go+1)!=43 && (go+1)!=44 && (go+1)!=45 &&
(go+1)!=46 && (go+1)!=47 && (go+1)!=48 && (go+1)!=49 && (go+1)!=50 &&
(go+1)!=51 && (go+1)!=52 && (go+1)!=53 && (go+1)!=54 &&
(go+1)!=55){//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);
                          }

                     //}

                    }
                    Borrar_Buffer_Codigo();
                 break;


           case 3: //Realizar una llamada perdida

                    while( (sendATcommand("AT+CREG?", "+CREG: 0,1", 500)
|| sendATcommand("AT+CREG?", "+CREG: 0,5", 500)) == 0 );
                    Serial1.print("ATD");
                    Obtener_Telefono(AuxPin,AuxCondicion, AuxValor1,
AuxValor2);
                    Serial1.println(";");
                     delay(10 * 1000);
                    Serial1.println("ATH");
                    Borrar_Buffer_Codigo();
                 break;
            case 4: //Enviar SMS aviso
                    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
                         delay(1500);
                    Serial1.println("Alertas: "); //Cabecera del SMS para saber quien
lo envia
                    Borrar_Buffer_Codigo();
                 break;

              case 5: //SMS GPS
                   if(latitude!=0 && longitude!=0){//Si hay una posicion valida.

                      Serial1.println("AT+CMGF=1"); // set SMS mode to text
                      Serial1.print("AT+CMGS="); // now send message...
                      Serial1.print(34); // ASCII equivalent of "
                      Obtener_Telefono(AuxPin,AuxCondicion, AuxValor1,
AuxValor2);
                      Serial1.println(34); // ASCII equivalent of "
                      delay(500);
                      //Generamos el link con la posicion
                      Serial1.print("https://maps.google.com/maps?q=");
                      Serial1.print(latitude, 6);
                      Serial1.print(",");
                      Serial1.print(longitude, 6);
                      Serial1.print("");
```

```
                    Serial1.print("Velocidad: ");
                    Serial1.print(speed, 6);
                    Serial1.println("Km/h");
                    Serial1.println(char(26)); // ASCII equivalent of Ctrl-Z


                  }
                  Borrar_Buffer_Codigo();
                break;

                case 6: //SMS Aviso+GPS
                  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
                  delay(1500);
                  Serial1.println("Alertas: "); //Cabecera del SMS para saber quien
lo envia
                  Borrar_Buffer_Codigo();
                break;




            }
        }

    }//Fin
```

El tratamiento de los cuatro primeros elementos de la Tabla 4.5 antes mostrada 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.

El resto de funciones de que aparecen en el código antes mostrado serán comentadas en los siguientes aparatados.

4.2.3. Funciones del GPS

A) Envío de la posición por SMS

El usuario por medio puede programar el sistema para que este le envíe la posición del mismo por medio de un SMS, en caso de que se produzca cualquier evento en el sistema o bien puede solicitarla de forma inmediata por medio de una aplicación para Smartphone Android que ha sido desarrollada para este proyecto y que será descrita en el próximo capítulo, con la cual además será posible también conocer la posición del sistema (entre otros datos) desde el menú Monitorizar de la misma como se describirá más adelante.

El envío por SMS de la posición se lleva a cabo mediante el siguiente código:

Algoritmo de envío posición GPS por SMS

```
                                    . . . . . . . . . .
                                    . . . . . . . . . .

case 5: //SMS GPS
                if(latitude!=0 && longitude!=0){//Si hay una posicion valida.

                   Serial1.println("AT+CMGF=1"); // set SMS mode to text
                   Serial1.print("AT+CMGS="); // now send message...
                   Serial1.print(34); // ASCII equivalent of "
                   Obtener_Telefono(AuxPin,AuxCondicion, AuxValor1,
AuxValor2);
                   Serial1.println(34); // ASCII equivalent of "
                   delay(500);
                   //Generamos el link con la posicion

Serial1.print("https://maps.google.com/maps?q=");
                  // https://maps.google.com/maps?q=37.601040,%20-0.978825
                   Serial1.print(latitude, 6);
                   Serial1.print(",");
                   Serial1.print(longitude, 6);
                   Serial1.print("");

                   Serial1.print("Velocidad: ");
                   Serial1.print(speed, 6);
                   Serial1.println("Km/h");
                   Serial1.println(char(26)); // ASCII equivalent of Ctrl-Z


                 }
                 Borrar_Buffer_Codigo();
              break;
                                    . . . . . . . . . .
                                    . . . . . . . . . .
```

Este código se encuentra dentro de la función Acciones() antes descrita. Lo que hace es proporcionar en un SMS un link que al pulsarse permitirá ver en un mapa la posición del sistema. Este mapa pude ser o bien procedente del navegador web instalado

en el Smartphone o bien puede verse en la aplación Google Earth si esta se encuentra instalada en el terminal, dado que al pulsar sobre el link aparecerá un menú para selección con qué aplicación se desea abrir. Además también se indica el valor de velocidad.

El destinatario de dicho mensaje viene especificado en la propia trama, con los caracteres: PIN+CONDICION+VALOR+VALOR. Es la variable MODO la que determina el tipo de mensaje que es, dado que puede ser:

- SMS de posición GPS.
- SMS de incidencia.
- SMS de posición+incidencia.

B) Proceso adquisición de los datos del GPS

Una de las posibilidades de actuación del sistema, es la de usarlo como sistema de registro del recorrido, es decir, como traker GPS. Se puede programar el sistema para que en caso de cumplirse alguna condición (como que se abandone una determinada área, que salte la alarma del vehículo entre otras muchas posibilidades) o simplemente cuando se inicia el dispositivo, se comience a registrar a intervalos regulares configurables por el usuario, los datos adquiridos por el GPS de una forma ordenada en un fichero almacenado en la carpeta "GPS" de la MicroSD.

Los datos recibidos deben ser "descompilados" dando lugar a la siguiente estructura: **Fecha, Hora, Latitud, Longitud, Altitud, Velocidad, Curso**. Luego estos datos son registrados en un archivo de texto en una línea independiente, para así poder añadir los siguientes datos en la posición justamente siguiente.

Como se ha mencionado en el Capítulo 2, el módulo GPS EM-411 transmite constantemente unas tramas por el puerto **Serial 2** de Arduino, las cuales 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.

En el siguiente segmento de código se observa cómo se realiza el proceso de grabación de los datos que nos suministra el GPS tras ser procesados por la librería antes comentada.

Algoritmo de registro de datos GPS en MicroSD

```
  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;
                }
        }
```

. . . . . .

. . . . . .

Dentro de la Estructura CASE contenida en la Función Acciones()

```
case 1: // 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);
                      }
                    }
                  }
              }
              Borrar_Buffer_Codigo();
              break;
```

Durante las pruebas realizadas al módulo, se ha observado que en tramas válidas desde el punto de vista del checksum, se producen ciertos desajustes en los datos recibidos tanto en la posición como en la velocidad. Estas alteraciones tienen lugar en determinadas condiciones:

- **Cuando se pasa de estar en movimiento a reposo**. Se ha observado que ha cuando el sistema experimenta una cierta velocidad, como por ejemplo 40Km/h y se detiene a consecuencia de un semáforo por ejemplo, se produce una falsa lectura que indica que la velocidad del sistema es entre 3 y 5 Km/h. Por ello, el algoritmo antes mostrado se encarga de compensar dicho desajustes, dado que no afectan a la posición que lleva el sistema, simplemente estableciendo 0Km/h como velocidad.

- **En el comienzo de la adquisición de datos**. Cuando se inicia el módulo GPS y este empieza a recibir tramas válidas del el punto de vista del checksum, se producen desajustes en la posición de varios metros, los cuales son compensados al cabo de unos segundos. Por ello el algoritmo anterior espera unos segundos después de detectar la primera trama válida antes de iniciar la grabación del resto de tramas.

- **Paso a zona sin cobertura**. Para que el sistema 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 4.3.

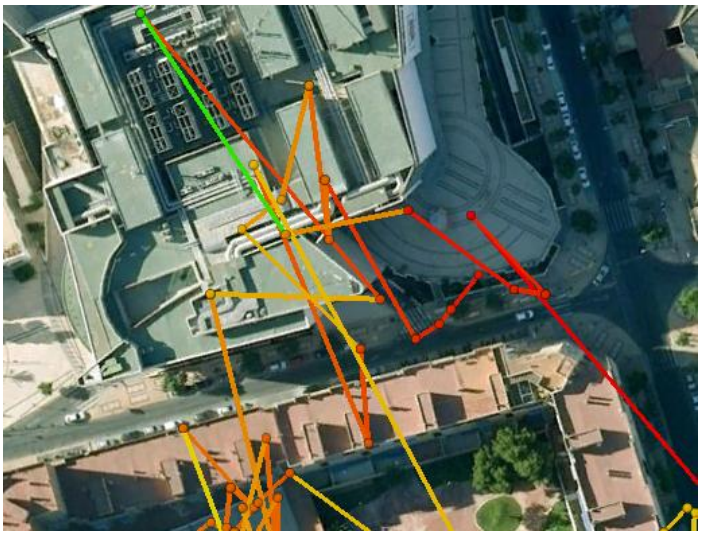


Figura 4.3: Desviación de las coordenadas satélite.

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.

El sistema 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 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,etc.

Como podemos observar en la figura 4.4, 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.

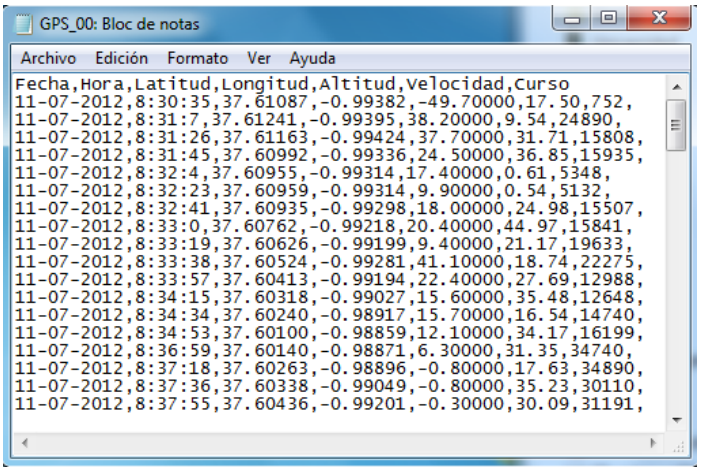


Figura 4.4: Ejemplo de registro de coordenadas GPS en la MicroSD.

En la figura 4.5 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 4.6. La plantilla empleada es:

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

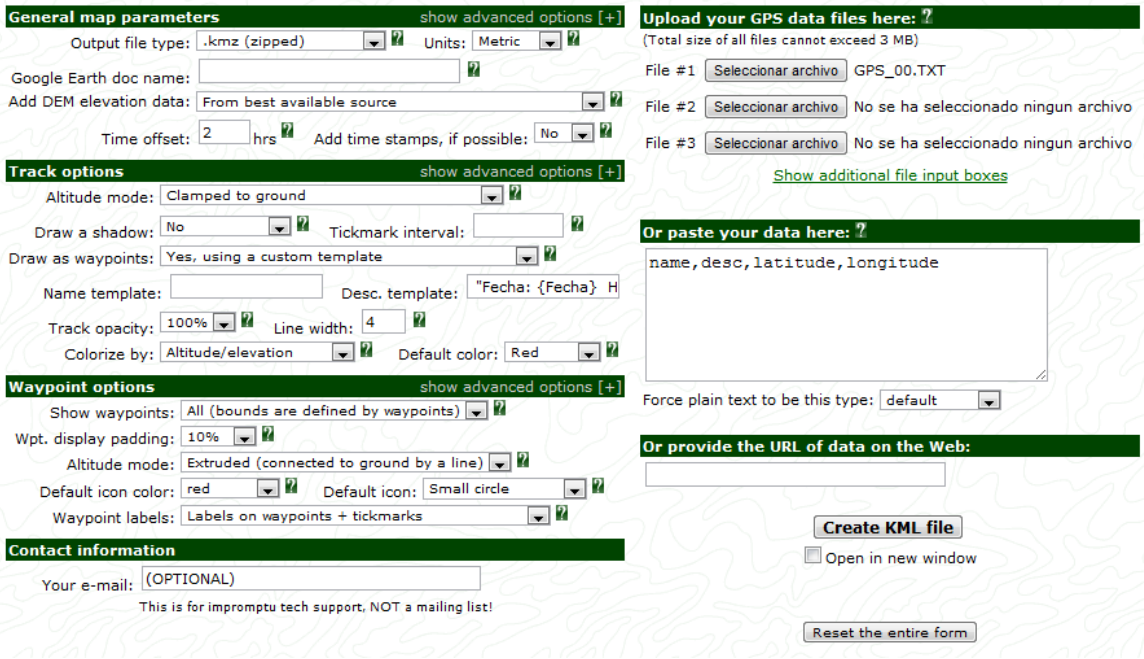


Figura 4.5: Parámetros de configuración de GPSVisualizer.

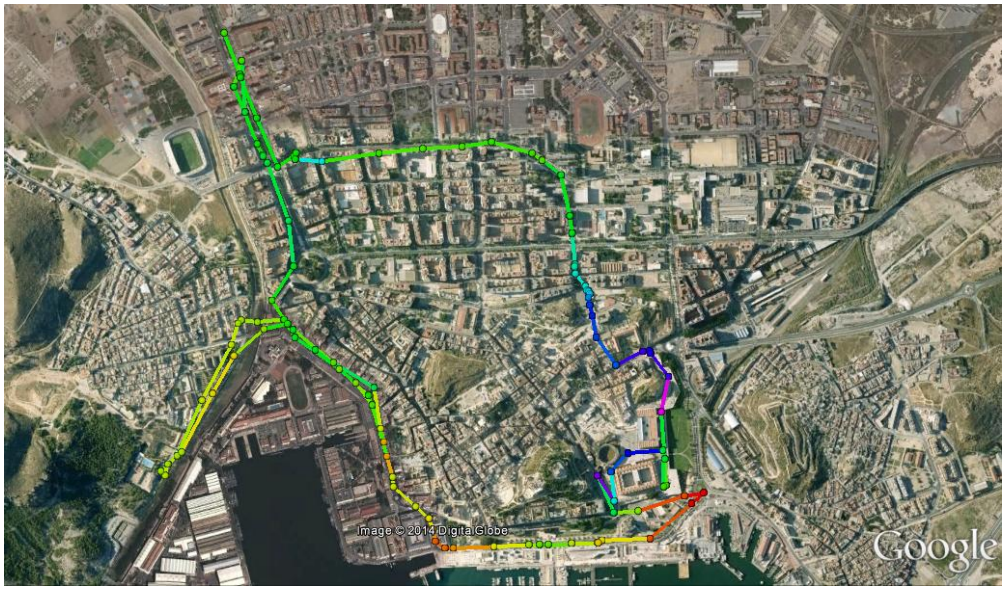


Figura 4.6: Resultado de la ruta en Google Earth.

En la Figura 4.6, se observa el recorrido realizado con el sistema funcionando como Logger GPS (aunque también estaba configurado para realizar otras tareas) por Cartagena, formado por zonas de distinto color en función de la altitud y delimitado por puntos, donde si pulsamos sobre alguno de ellos podremos observar los datos entes almacenados en el txt, tal y como se observa en la Figura 4.7.

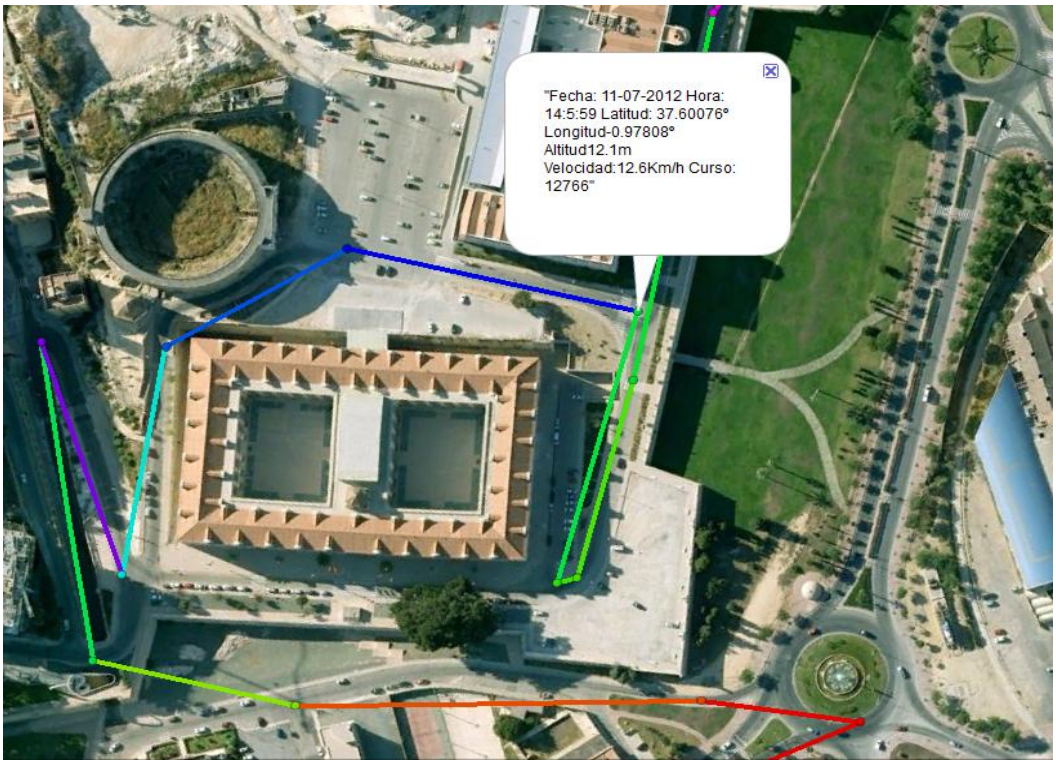


Figura 4.7: Resultado de la ruta en Google Earth.

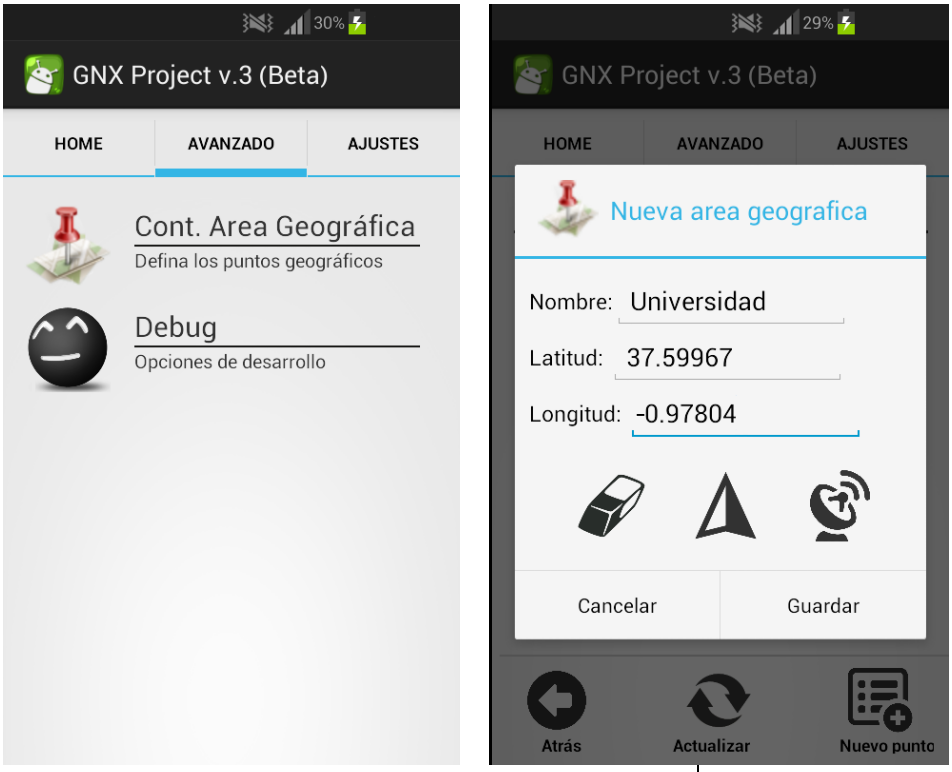
Para más información sobre como poder convertir los datos del GPS en un fichero compatible con Google Earth, escanee el siguiente código.

C) Control por Área Geográfica

Los usos del GPS pueden ir más allá de servir para poder conocer la posición del sistema. Otro uso que se ha planteado en este proyecto es para lo que hemos denominado "**Control por Área Geográfica**", que no es más que asignar condiciones al hecho de encontrarnos dentro o fuera de una determinada área geográfica, es decir, podríamos por ejemplo hacer que se iniciara el proceso de grabación del recorrido antes descritos cuando salgamos del garaje.

Por medio de la aplicación para Smartphone, en una opción contenida en la pestaña **Avanzado** que se describirá en el próximo capítulo, el usuario puede definir únicamente los distintos puntos geográficos, asignándose un nombre fácil de reconocer, como por ejemplo "Casa" o "Universidad". Esta coordenada que define el punto geográfico puede ser introducida a mano o bien puede ser la posición del Módulo Principal que no devuelve por telemetría o bien puede proceder de la posición en el que se encuentre el usuario con su Smartphone, es decir, del propio GPS del teléfono o tablet.

En la Figura 4.8, se puede observar la sección de la aplicación antes comentada.



(A) **(B)**

Figura 4.8: Opción del menú Control por Área Geográfica.

Cuando se define el/los punto/s geográficos, desde el mismo menú desde el que se programa las tareas se selecciona uno de dichos puntos y se especifica un radio de acción en metros y kilómetros dentro de una serie de opciones proporcionadas, especificando además si debemos encontrarnos dentro o fuera de dicha área. Luego se definen las acciones a asociar al hecho de encontrarnos dentro o fuera de dicha área.

Con esta configuración es posible definir varias áreas concéntricas dentro de un mismo punto, tal y como aparece en la Figura 4.9.

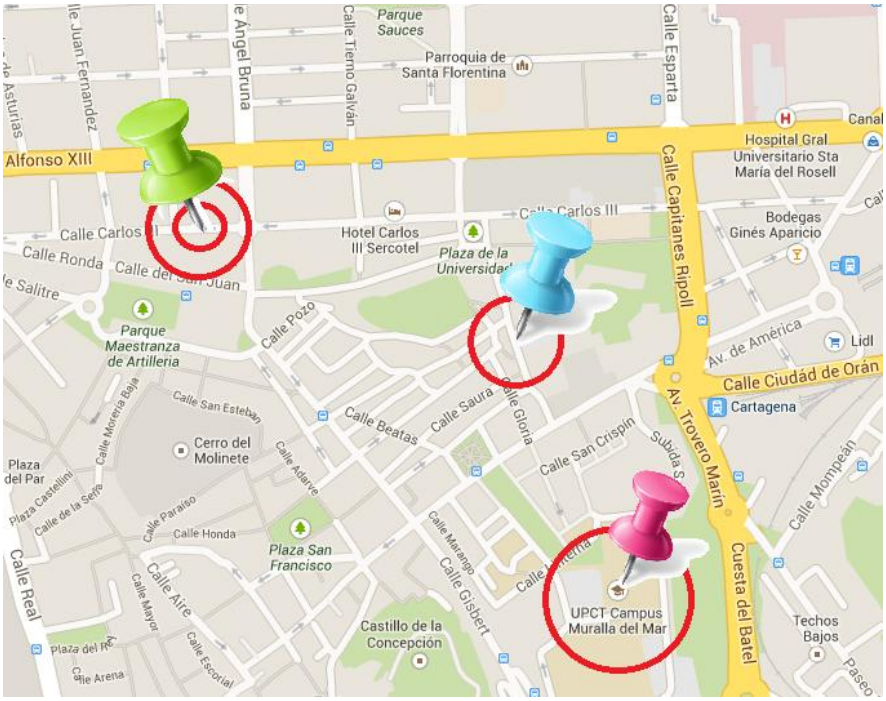


Figura 4.9: Control por Área Geográfica.

4.2.4. 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 y no su correspondencia con los de Arduino.

Todo esto se puede apreciar en la figura 4.10:

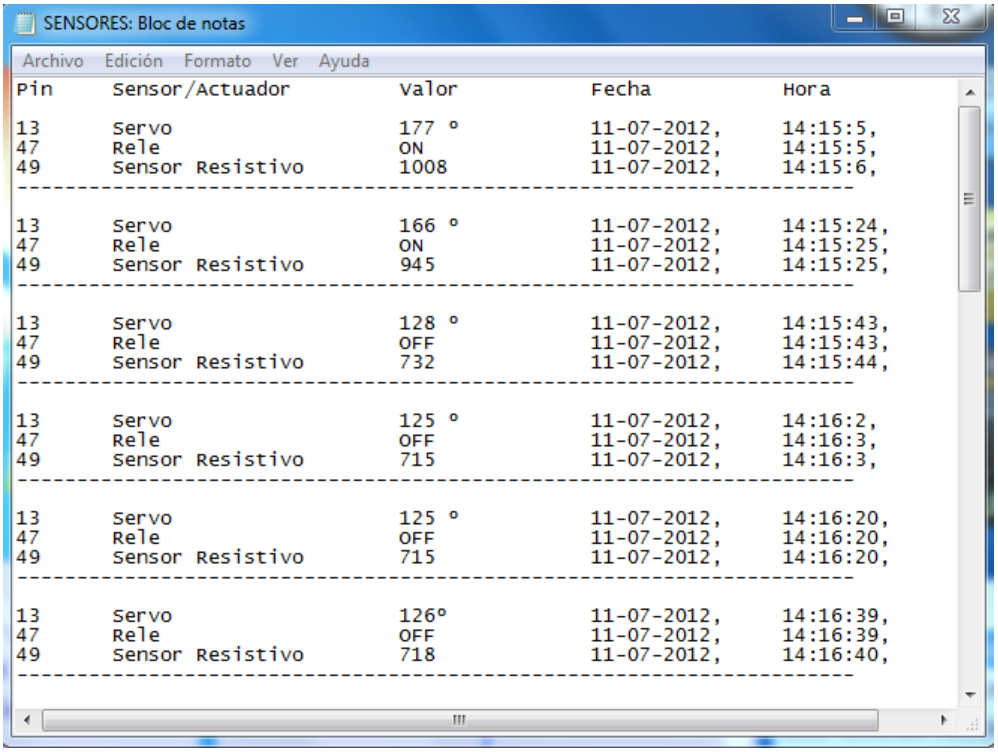


Figura 4.10: Ejemplo de registro de pines en la MicroSD.

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. Firmware Estación

Las Estaciones precisan de un firmware completamente distinto al del Módulo Principal, dado que estas no deben almacenar nada en la memoria EEPROM ni tienen capacidad de decisión, dado que se limita a acatar las órdenes que reciben del Módulo Principal por radiofrecuencia. Las Estaciones no almacenan ningún tipo de información de forma duradera, ya que es el Módulo Principal el que le indica que hay conectada y cuando debe activar o desactivar una salida (o bien que valor debe colocar en la salida). Esto tiene la gran ventaja de que no es necesario que todas las Estaciones estén conectadas en el mismo momento en el que el Módulo Principal recibe la tarea o tareas que debe desarrollar (Figura 4.11).

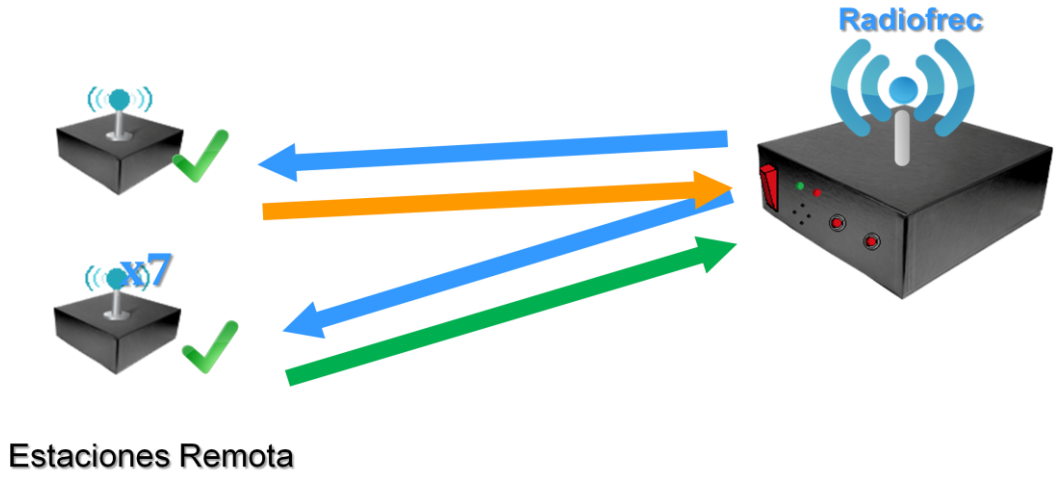


Figura 4.11: Intercambio de información bidireccional con Estación.

Para que esto sea posible, el Módulo Principal debe de mandar información constantemente a cada una de las Estaciones y recibir de esta una retroalimentación necesaria para poder gestionar el sistema correctamente.

El firmware se compone básicamente de tres funciones:

- Captar las órdenes del Módulo Principal.
- Llevar a cabo la orden solicitada.
- Envío de retroalimentación al Módulo Principal.

Algoritmo de captación de los datos

```
void loop(){
    //Si recibo datos por radiofrecuencia (nrf24l01)
    if(transreceiver.available()){
      transreceiver.read();
        transreceiver.rxPL(Buffer_Recepcion_Modulo,10);
    }
    ////////////////////////////////////////
    //Enviamos los datos al módulo principal
     Telemetria[0]=Password;

        transreceiver.txPL(Telemetria,11);
```

```
          transreceiver.send(FAST);
          delay(5);
          if(k==0)
          digitalWrite(2,!digitalRead(2));
       ////////////////////////////////////

      //Procesamiento
      if(Buffer_Recepcion_Modulo[k]!=2){//Pin Configurado

          Procesamiento[0]=(Buffer_Recepcion_Modulo[k]/10000);//Modo

Procesamiento[1]=(Buffer_Recepcion_Modulo[k]%10000)/1000;//Pin

Procesamiento[2]=(Buffer_Recepcion_Modulo[k]%10000)%1000;//Valor

          //Hacemos la conversión de pines
            if(Procesamiento[1]<4)
              Pin=Procesamiento[1]+3;
            else if(Procesamiento[1]<5)
              Pin=9;
            else if(Procesamiento[1]<9)
              Pin=Procesamiento[1]+9;
            else
              Pin=20;
             Acciones(Pin,Procesamiento[0],Procesamiento[2]);

      }

     k++;
    if(k>9)  k=0;

    }
```

Su funcionamiento es muy simple. En el momento que se detectan datos de entrada, se verifica que las órdenes recibidas, que son descritas como un array de 11 posiciones (Tabla 4.6), donde la primera posición del mismo (posición 0) contenía la contraseña del sistema, la cual servía para verificar si los datos recibidos procedían de una fuente conocida. El resto de posiciones del array corresponden a las órdenes de cada uno de los 10 pines que componen las Estaciones, donde se indica: Qué hay conectado y Valor o Estado a poner en la salida (si lo que hay conectado es un actuador). En el caso de sensores, solo se indica el tipo de sensor, dado que la condición y cuando debe cumplirse es comunicado por el Módulo Principal.

0	1	2	3	4	5	6	7	8	9	10
PASS	Orden PIN 1	Orden PIN 2	Orden PIN 3	Orden PIN 4	Orden PIN 5	Orden PIN 6	Orden PIN 7	Orden PIN 8	Orden PIN 9	Orden PIN 10

Tabla 4.6: Datos recibos por Estación del Módulo Principal

Cuando llegan dichas órdenes y tras verificar la contraseña incluida en ellas, se descomponen para cada pin y se almacena de forma temporal para después pasarla como parámetro a una función llamada Acciones(); la cual se encarga de procesar la salida o entrada como corresponda atendiendo al tipo de periférico conectado.

Algoritmo Acciones() de Estación

```
void Acciones(byte pin,byte mode,byte valor){
    if(valor==2){//Es una entrada(Sensor)
      pinMode(pin,INPUT);
            if(mode==0){//Sensor Resistivo
             Telemetria[Procesamiento[1]+1] = analogRead(pin);
             }
            else if(mode==1){//Pulsador
              Telemetria[Procesamiento[1]+1] = digitalRead(pin);
            }
            else if(mode==2){//Interruptor
              Telemetria[Procesamiento[1]+1] = digitalRead(pin);
            }
            else if(mode==3){//Sensor de temperatura LM35
              Telemetria[Procesamiento[1]+1] = ((5.0 *
analogRead(pin) * 100.0)/1024.0)-9;//Fin Sensor LM35
            }
            else if(mode==4){//Sensor Ultrasonico
                    unsigned long pulso;
                    pinMode(PinTriger, OUTPUT);       // ponemos el pin como
salida
                    digitalWrite(PinTriger, HIGH);    // lo activamos
                    delayMicroseconds(10);      // esperamos 10 microsegundos
                    digitalWrite(PinTriger, LOW);     // lo desactivamos
                    //pinMode(pin, INPUT);     // cambiamos el pin como entrada
                    pulso = pulseIn(pin, HIGH);   // medimos el pulso de
salida del sensor
                    Telemetria[Procesamiento[1]+1] =
((float(pulso/1000.0))*34.32)/2;

            }
            else if(mode==5){//Sensor proximidad por infrarrojos sharp
              Telemetria[Procesamiento[1]+1] =
(6787/(analogRead(pin)-3))-4;
            }
    }
    else{//Es una salida (Actuador)
      if(valor!=Telemetria[Procesamiento[1]+1]){
      pinMode(pin,OUTPUT);
      if(mode==0){//Rele
        //if(valor!=digitalRead(pin)){
          digitalWrite(pin,valor);
        //}
        if(digitalRead(pin)==LOW)
          Telemetria[Procesamiento[1]+1]=digitalRead(pin);
      }
      else if(mode==1){//Modular Pulso
        analogWrite(pin,valor);
        Telemetria[Procesamiento[1]+1]=valor;
      }
     else if(mode==2){//Servo
        Servo_Device.attach(pin);
        delay(15);
        Servo_Device.write(valor);
```

```
          delay(15);
          Telemetria[Procesamiento[1]+1]=Servo_Device.read();
        }
      else if(mode==4){//Alarma
          if(Telemetria[Procesamiento[1]+1]==0){
             Telemetria[Procesamiento[1]+1]=1;
             tone(pin, valor);
             delay(100);
           }
           else{
              Telemetria[Procesamiento[1]+1]=0;
              noTone(pin);
              delay(100);
           }
        }
       else if(mode==4){//Motor
          if(valor>0){
             analogWrite(pin, valor);
          }

Telemetria[Procesamiento[1]+1]=map(valor,0,255,0,3000);//Revoluciones
por minuto;
        }
     }
   }
```

Como hemos mencionado, para que el sistema pueda gestionar todos los periféricos indistintamente de donde se encuentre conectado, es decir, en el Módulo Principal o en alguna de las Estaciones remotas, es necesario que el sistema sea en bucle cerrado, es decir, que exista una retroalimentación.

Esto se logra mandando por radiofrecuencia el estado de todo lo que hay conectado en los 10 pines (en caso de que alguno no esté configurado, se manda un "2"), mediante un array que se genera mediante el siguiente código.

Algoritmo retroalimentaión desde Estación

```
//Procesamiento
     if(Buffer_Recepcion_Modulo[k]!=2){//Pin Configurado

          Procesamiento[0]=(Buffer_Recepcion_Modulo[k]/10000);//Modo

Procesamiento[1]=(Buffer_Recepcion_Modulo[k]%10000)/1000;//Pin

Procesamiento[2]=(Buffer_Recepcion_Modulo[k]%10000)%1000;//Valor

          //Hacemos la conversión de pines
            if(Procesamiento[1]<4)
              Pin=Procesamiento[1]+3;
            else if(Procesamiento[1]<5)
              Pin=9;
            else if(Procesamiento[1]<9)
              Pin=Procesamiento[1]+9;
            else
              Pin=20;
             Acciones(Pin,Procesamiento[0],Procesamiento[2]);
     }
```

4.4. Modos de conectividad

Como se ha mencionado con anterioridad, el sistema dispone de un total de cuatro modos de conectividad distintos, destinados a cubrir las deficiencias o limitaciones que presentan cada uno de ellos.

Estos modos de conectividad son: **Bluetooth**, **3G**, **SMS** y **Radiofrecuencia**, tal y como aparece reflejado en la Figura 4.12.

Figura 4.12: Modos de comunicación soportados.

Donde los tres primeros (**Bluetooth**, **3G**, **SMS**) son empleados para la comunicación directa entre el usuario y el Módulo principal y el último modo (**Radiofrecuencia**) es exclusivamente para la comunicación entre el Módulo Principal y cada una de las Estaciones. Hay que recordar que se trata de un sistema centralizado, en el que el Módulo Principal es el corazón del sistema y el que recibe todas las órdenes del usuario y se encarga de transmitirlas a la Estación/es correspondientes.

Centrándonos en los tres primeros modos de conectividad, el motivo de su implementación no es otro que el de cubrir las limitaciones inherente a los mismos:

- **Bluetooth**

  Pensado para comunicaciones a corta distancia, es decir, cuando el usuario se encuentre en el mismo emplazamiento que el Módulo Principal, como por ejemplo en la misma habitación, en el coche, etc. Con un alcance máximo de 5 metros, ideal para poder realizar tareas de mantenimiento, como el establecer los parámetros de configuración necesarios para que el Módulo Principal pueda emplear el modo SMS o el 3G (definiendo el servidor FTP, punto de acceso APN, etc.). Otra de las ventajas es que no genera coste alguno por flujo de información, siendo por tanto su único inconveniente su reducido alcance, que impide que el usuario pueda manejarlo desde kilómetros de distancia, aspecto que sería fundamentar para un control domótico/seguridad o si se instala en un vehículo o en un invernadero, por ejemplo.

- **3G**

  Posibilita disponer de una cobertura mundial. Funciona a través del módulo GSM/GPRS por medio de una tarifa de datos. Como inconveniente, es el coste que acarrea disponer de una tarifa de datos para funcionar, aunque como se verá más adelante, los paquetes de información no llegan a los 20 Bytes, por lo que las tarifas de datos que se comercializan actualmente difícilmente serán consumidas por lo que no se generarán costes adiciones por sobrepasar el límite contratado. Precisa disponer de conexión a internet en el Smartphone para poder emplear dicho modo de conectividad.

- **SMS**

  Su función es idéntica al del modo 3G antes descrito. Es posible gracias al módulo GSM/GPRS y proporciona una cobertura mundial, con la diferencia que no es necesario disponer una tarifa de datos ni acceso a internet desde el Smartphone, con el inconveniente de que genera unos costes por mensajes que variarán en función del operador contratado. Este modo de conectividad es especialmente útil en situaciones en las que no se encuentre cerca del Módulo Principal y no se disponga de una conexión a internet desde el Smartphone. Imaginemos que el sistema está instalado en un vehículo, el cual en plena calle es sustraído. Cuando el usuario se percata del hurto trata de conocer la ubicación del vehículo así como la realización de alguna otra orden como activar el claxon, desconectar la batería del motor, etc. (en función de cómo ha sido configurado el sistema). Si el usuario no dispone de acceso a internet desde su Smartphone el modo 3G estaría descartado, por lo que emplearía el modo SMS, con el que lograría controlar el Módulo Principal de forma remota y conocer su ubicación.

A continuación vamos a proceder a explicar cómo funcionan internamente los distintos modos de conectividad antes descritos.

4.4.1. Modo de conectividad Bluetooth

Para poder emplear este modo de conectividad es necesario en primer lugar establecer una comunicación entre el Smartphone y el Módulo Principal. Esto se lleva a cabo por medio de la aplicación para Android que se ha elaborado para este proyecto. En el menú principal de la aplicación (Figura 4.13 A) se debe seleccionar la opción "Conectar". Acto seguido aparecerá un menú se abre un menú con las tres opciones de conexión disponibles (Figura 4.13 B), donde seleccionamos la opción "Bluetooth". Si el bluetooth del Smartphone o Tablet se encuentra desconectado aparecerá un aviso como en el del Figura 4.13 C preguntándole al usuario si desea activarlo.

Figura 4.13: Establecimiento de comunicación por bluetooth

Una vez activado, aparece la lista de dispositivos que el terminal ya tenía registrado con anterioridad (Figura 4.13 D) tras lo cual se selecciona el dispositivo bluetooth que corresponde con el Módulo Principal, llamado "linvor". Tras lo cual se

confirma que se desea establecer la conexión con el dispositivos (Figura 4.13 E) y si se logra establecer la comunicación, el icono del bluetooth del menú de opciones se volverá de color azul tal y como aparece en la Figura 4.13 F. En caso de no establecerse la conexión o perderse, el icono volverá a ser de color gris y saldrá un mensaje de alerta.

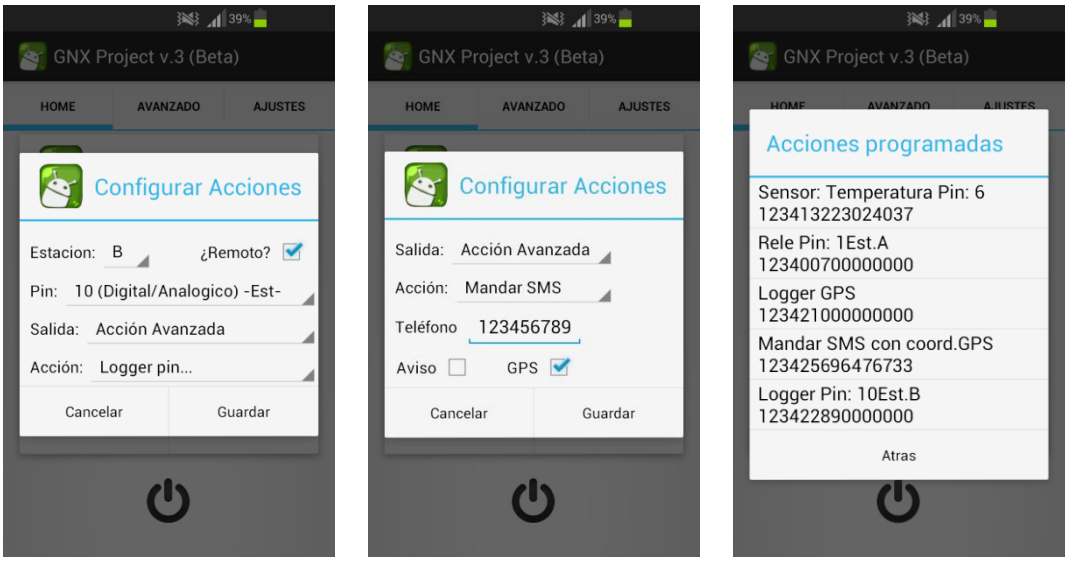
Una vez establecida la comunicación por bluetooth, ya se puede controlar el sistema para poder realizar diversas opciones como:

- Acciones inmediatas, es decir, controlar los periféricos u otras acciones como el registro del recorrido entre otras.
- Programar una o varias tareas.
- Adquirir los datos de telemetría.
- Cambiar la configuración.

Para ilustrar este apartado, vamos a suponer que se desea programar una tarea al sistema, dado que el proceso de transmisión de información es el mismo independientemente de que se desee enviar, por ello dado que programar una tarea precisa el envío de un número mínimo de tres tramas (frente a las otras opciones que solo precisan una) vamos a ilustrar el proceso de programar una tarea nueva en el sistema.

Para programar una tarea simplemente hay que dirigirse a la opción "Panel del Control" y a continuación definir, por un lado, la condición que se tiene que cumplir por medio de la opción "Configurar Sensor" (Figura 4.13 C) y por otro lado definir una a una todas la acciones que deseo que se ejecute al cumplimiento de la condición antes definida (Figura 4.13 D y E).

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



(D) **(E)** **(F)**

Figura 4.14: Programación de una tarea.

La descripción detallada de dichos menús se llevará a cabo en el próximo capítulo donde se hablará de forma detallada de las características de la aplicación para Android.

Cuando se han programado la tarea (Condición + Acciones), se almacenan en un buffer de salida que puede ser consultado mediante la opción "Acciones programadas" donde se pueden realizar modificaciones en la tarea así como eliminar alguna. En el ejemplo de la Figura 4.14 F, la tarea que vamos a programar es la siguiente: ***Si el sensor de temperatura (LM35) que se encuentra conectado al pin 6 del Módulo Principal registra una temperatura entre los 24ºC y 37ºC, queremos se active el pin 1 de la Estación remota A; nos registre el recorrido GPS que se haga con el sistema en la MicroSD; nos mande un SMS con la posición GPS y además que nos registre en un fichero de la MicroSD todas las fluctuaciones de valor que experimente el pin 10 de la Estación remota B.***

Cuando la tarea está definida completamente se procede a su envío mediante la opción "Enviar" que aparece en el submenú "Panel de Control" (Figura 4.14 B). Tras lo cual se produce el envío ordenado de las tramas, comenzando con la "Cabecera del Sensor" (dado que es esta la que inicia en el Módulo Principal el proceso de grabación de la tarea en la EEPROM) y luego cada una de las acciones que hemos asociado al cumplimiento de la condición anterior, mandándose una detrás de la otra con una pausa de 0,5 segundos para evitar solapamientos en el puerto serie del Módulo Principal.

Cuando se han enviado todas las tramas antes definidas, se envía de forma automática la llamada "Trama de clausura", cuya estructura es Contraseña+"32000000000", que como se ha comentado en capítulos anteriores, sirve para indicarle al sistema que ya no hay más acciones que componen la tarea y que por lo tanto no es necesario almacenarlas en la EEPROM, haciendo que cualquier trama adicional sea interpretada como una acción inmediata o como una nueva tarea.

Cuando se recibe las tramas y son todas verificadas antes de ser guardas o ejecutadas, se envía una confirmación al usuario empleando el mismo modo de comunicación que el empleado para adquirir dichos datos. En este caso, simplemente se manda por el puerto serien número 3, la siguiente cadena de texto: "#OK".

Cuando la aplicación para Android recibe dicha respuesta, muestra un mensaje en pantalla llamados en programación "toast", indicando que los datos enviados han sido recibidos por el Módulo Principal. En caso de no recibirse dicha respuesta durante un periodo de 10 segundos, la aplicación mostrará un error en pantalla, indicando que los datos no han sido entregados.

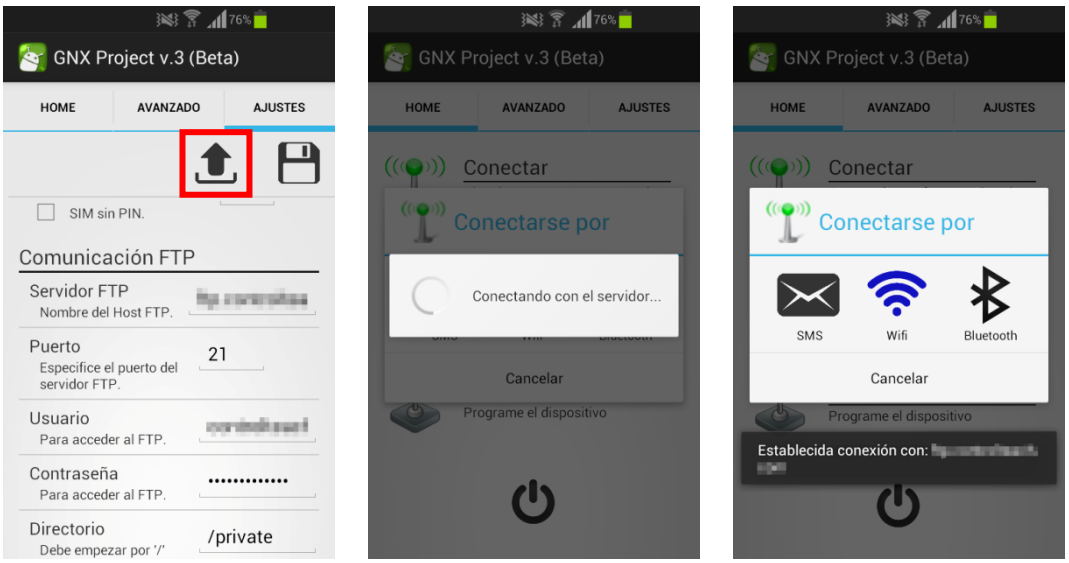
Con esto se termina el ciclo de programación de una tarea por bluetooth y el sistema ya está preparado para ejecutarla cuando llegue el momento mientras puede seguir ejecutando otras tareas y órdenes del usuario.

4.4.2. Modo de conectividad 3G

El modo de conectividad 3G se basa en el tráfico de archivos de txt a un servidor FTP que los aloja como punto intermedio. El proceso de transmisión de datos es idéntico en todos los modos de conectividad desde el punto de vista del usuario, es decir, la interfaz es la misma independientemente del modo empleado, sin embargo internamente el proceso difiere.

Explicaremos este proceso continuando con el caso del envío de una tarea, dado que precisa mandar un mayor número de tramas, pero que al estar contenidas en un archivo de texto es indiferente el número a mandar.

La primera vez que se emplee el sistema, es necesario especificar una serie de parámetros tanto en la aplicación Android como en el Módulo Principal. Para ello se debe pulsar sobre la pestaña "Ajustes" donde centrándonos únicamente en el modo 3G, deberemos rellenar los datos del apartado "Comunicación FTP" para introducir los datos el servidor FTP que alojará temporalmente las órdenes y respuestas del sistema para posibilitar el control a través de internet, indicando el servidor, puerto de comunicación, usuario, contraseña y directorio donde se van a almacenar los datos (Figura 4.15 A). Además es preciso especificar los datos del punto de acceso APN, es decir, los datos de la tarifa de datos de la SIM insertada en el módulo GSM/GPRS EFComPro. Estos datos deben ser enviados por medio del icono remarcado en rojo de la Figura 4.15 A **la primera vez** por el Modo Bluetooth, dado que es el único de los tres modos que no precisa de ningún parámetro de configuración previo.



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

Figura 4.15: Modo de conectividad 3G.

Una vez establecido estos datos antes descritos, desde la pestaña "Home" se selecciona la opción "Conectar" y luego a "Wifi/3G" (En el icono aparecerá Wifi o 3G en función de qué forma de conectarse a internet le hemos indicado a la aplicación

Android desde el menú de ajustes, en este caso disponíamos de conexión wifi en nuestro Smartphone). Cuando se pulsa sobre este modo de conectividad, la aplicación comprueba (Figura 4.15 B) si es posible acceder al servidor FTP antes especificado antes de activar este modo de conexión. Cuando la aplicación logra acceder al servidor (lo que indica que está disponible) se habilitado el modo, quedando como se ve en la Figura 4.15 C el icono en color azul. Si no se logra establecer una respuesta, saltará un aviso en pantalla indicando que no se puede llevar a cabo dicha operación.

Ahora procedemos con se ha descrito en el Modo Bluetooth a definir aquello que deseamos enviar (una tarea es lo que estamos enviando en este ejemplo, pero podría ser también la solicitud de los datos de telemetría) (Figura 4.14).

Una vez establecido lo que se va a enviar, la aplicación crea un archivo de texto con el nombre "Buffer_Ordenes.txt" en la raíz del almacenamiento interno del teléfono. (Figura 4.16).

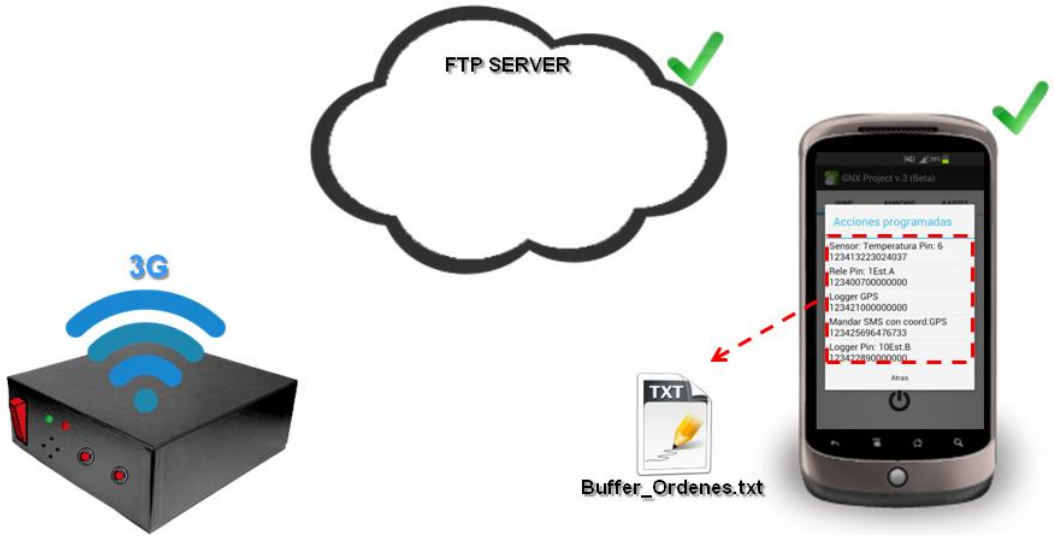


Figura 4.16: Generación del archivo Buffer_Ordenes.txt.

Tras ello, por medio de una librería en Android llamada "Apache.lib", se sube el archivo Buffer_Ordenes.txt al servidor especificado y en la ruta establecida. Una vez está subido, el Smartphone realiza una llamada perdida al Módulo Principal para indicarle que hay nuevos datos disponibles en el servidor y que por lo tanto debe acceder a él y descargarlos. El motivo de realizar una llamada perdida no es otro que el de evitar gastar tarifa de datos en realiza comprobaciones periódicas del servidor. De esta forma solo se consume cuando es necesario. (Figura 4.17)

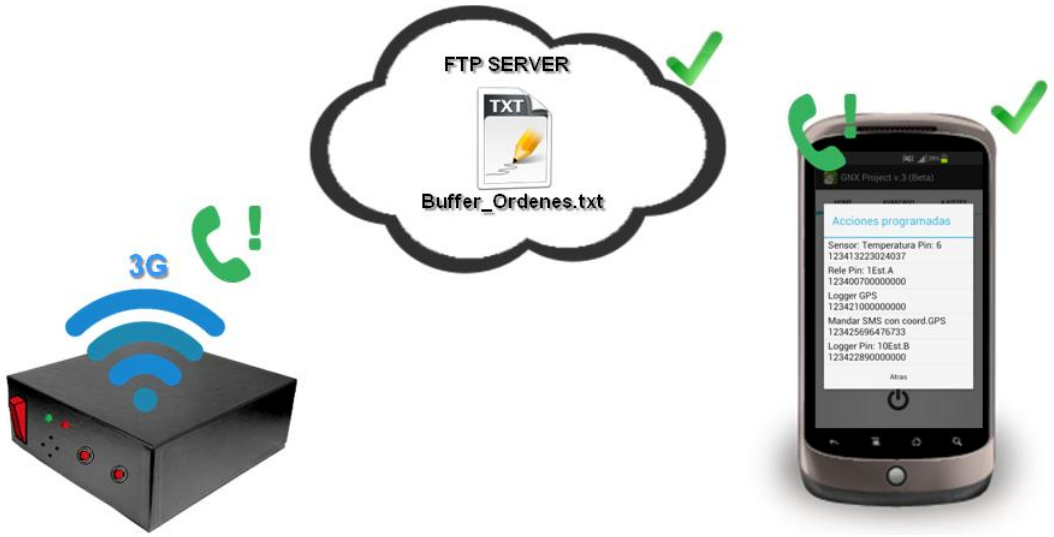


Figura 4.17: Subida del archivo Buffer_Ordenes.txt.

Una vez que el Módulo Principal detecta la llamada perdida de un número de teléfono contenido en la lista de teléfonos permitidos, se produce la descarga de dicho archivo, cuyo código de descarga y subida de datos, será expuesto más adelante. Cuando el archivo de texto es descargado, se elimina del servidor FTP para evitar procesar dos veces el mismo archivo y acto seguido tras verificar que el archivo contenga la contraseña del sistema (por defecto 1234), se manda una confirmación de recepción por el mismo canal. Para ello el Módulo Principal genera un nuevo archivo de texto llamado "Buffer_Datos.txt" que contiene en este caso "#OK" (aunque también puede contener los datos de telemetría como se comentará más adelante en el presente capítulo). Luego procede al envío de dicho archivo a la misma ruta del servidor FTP, tras lo cual elimina el archivo en la MicroSD. Cuando el Smartphone detecta la respuesta en el servidor, elimina dicho archivo y muestra un mensaje (toast) indicando que el Módulo Principal ha recibido correctamente los datos. (Figura 4.18)

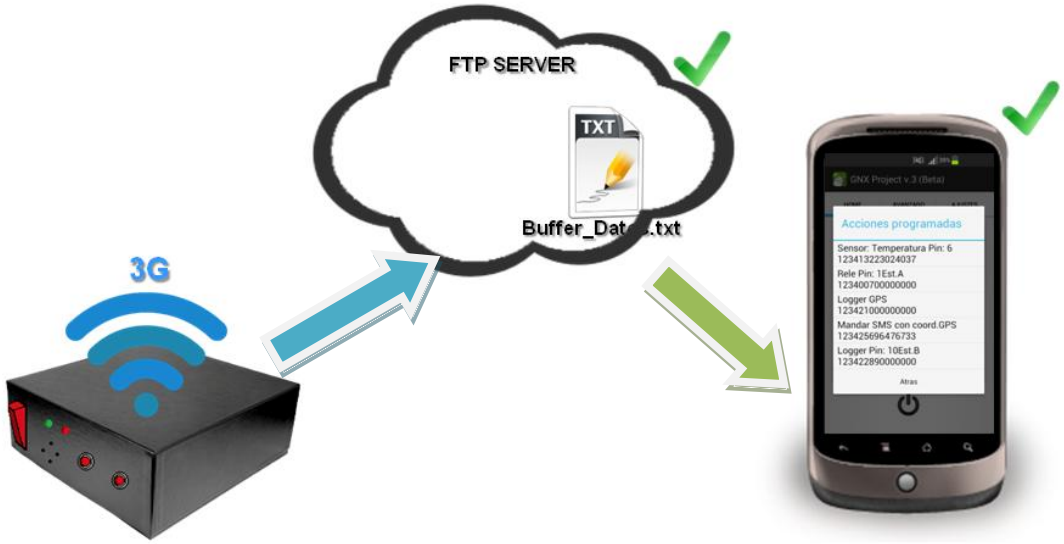


Figura 4.18: Descarga de respuesta del Módulo Principal (Buffer_Datos.txt).

Con esto el Módulo Principal está preparado para ejecutar la tarea que le ha sido enviada cuando se cumpla la condición.

4.4.3. Modo de conectividad SMS

El modo de conectividad SMS, es el último de los modos de comunicación que pueden emplearse entre el usuario y el Módulo Principal. En este modo de conectividad, no es preciso encontrarse conectado a internet para poder manejar el sistema, dado que esto se realiza mediante un simple SMS que contiene las tramas que antes hemos descrito separas por ";".

Para poder emplear este modo, es preciso (al igual que en el Modo 3G) definir una serie de parámetros desde la pestaña "Ajustes", donde hay que rellenar el número de teléfono de la tarjeta SIM que se encuentra presente en el Módulo Principal así como el PIN de desbloqueo que esta posea (si no lleva PIN, se marca la casilla "SIM sin PIN"). Estos datos deben ser enviados, al menos la primera vez por bluetooth, dado que es el único modo de conectividad que no precisa de ningún tipo de configuración. (Figura 4.19 A).

(A) **(B)**

Figura 4.19: Modo de conectividad SMS.

Una vez realizado este paso, simplemente hay que seleccionar el modo de conectividad SMS por medio del mismo menú que hemos descrito en los anteriores modos, tal y como se observa en la Figura 4.19 B.

Ahora se puede mandar las órdenes o tareas que el usuario desee por medio de la interface que hemos descrito en los anteriores modos. Cuando se emplea el Modo SMS, la aplicación limita a un máximo de 10 tramas, las que puede ser enviadas por SMS a fin de no sobrepasar los 160 caracteres, lo que obligaría a mandar un segundo SMS con el coste adicional que esto conlleva.

Las tramas a mandar se materializan en un SMS, donde se encuentran separadas por ";" que es enviado al número de teléfono del Módulo Principal (Figura 4.20).

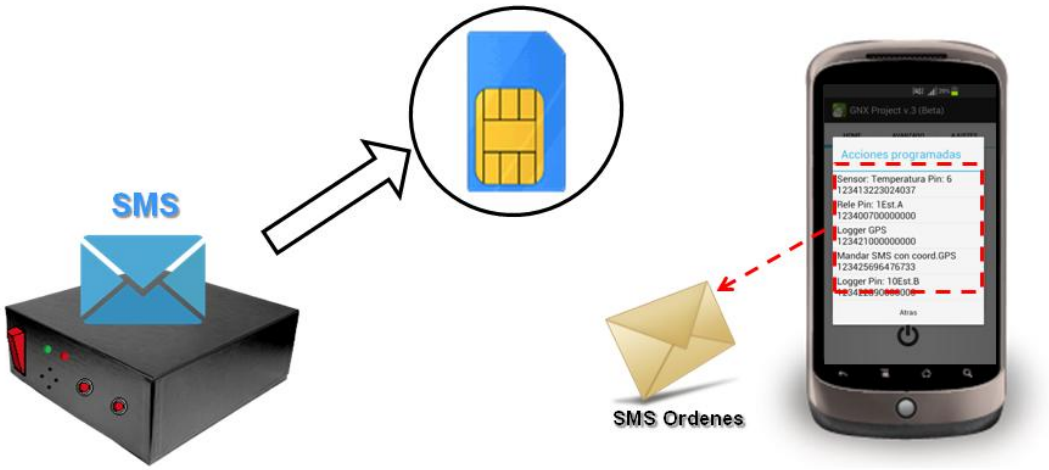


Figura 4.20: Envío de SMS con tramas.

El mensaje se queda almacenado en la bandeja de entrada de la SIM. A intervalos regulares de 30 segundos, el Módulo Principal comprueba la bandeja de entrada comenzando por la primera posición. El sistema ha sido diseñado para contener un máximo de 5 SMS, siendo los restantes SMS eliminados automáticamente.

Los SMS se consideran válidos o no atendiendo a dos criterios:

- Que el remitente del mismo esté registrado dentro de la lista de teléfonos permitidos.
- Que contenga la contraseña del sistema.

Cuando se detecta un mensaje que no cumple alguno de los anteriores requisitos, se elimina y se comprueba la siguiente posición de la bandeja de entrada. Si el mensaje resulta válido, este es ejecutado por medio de los algoritmos mencionados en el presente capítulo y tras ello eliminados, tal y como se observa de forma gráfica en la Figura 4.21, donde se aprecia que existía mensajes no válidos como son mensajes de publicidad o de otra índole, hasta encontrar uno que resulta válido.

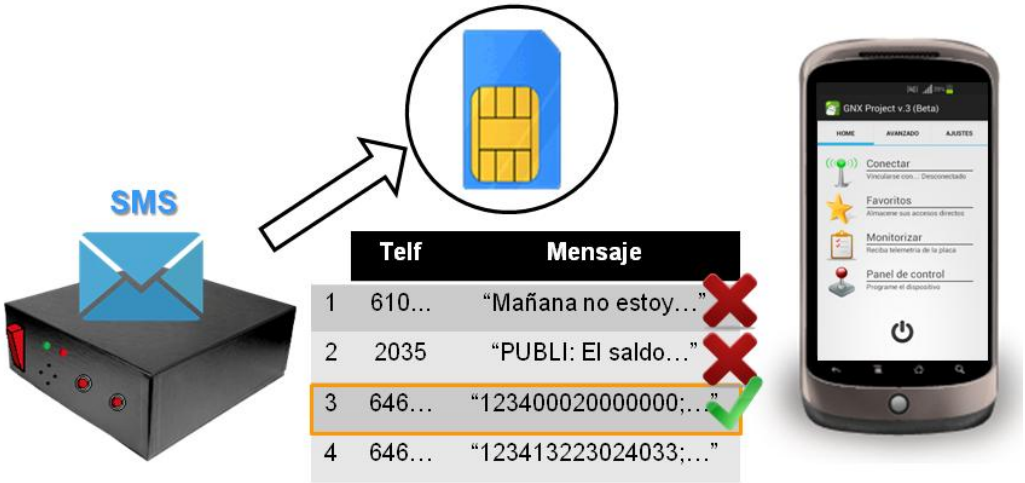


Figura 4.21: Análisis de la bandeja de entrada del GSM/GPRS.

Al igual que en los anteriores modos, el Módulo Principal manda una confirmación al usuario que puede ser en forma de SMS o una llamada perdida, según se haya especificado.

Hay que tener en cuenta que si entre la recepción del SMS y su procesamiento transcurre un periodo mayor a 1 hora, no se manda ningún tipo de respuesta al usuario.

4.4.4. Conectividad Módulo Principal - Estaciones

El modo de conectividad empleado entre el Módulo Principal y cada una de las Estaciones que compone el sistema es la radiofrecuencia. Este modo no es extensible al usuario, por lo que la aplicación para Android no juega ningún papel.

El Módulo Principal es el encargado de dar las órdenes a cada una de las Estaciones para que activen/desactiven las salidas correspondientes así como que tiene conectado en cada uno de sus pines, además es el que se encarga de comprobar si las condiciones definidas por los sensores se cumplen; para que el Módulo Principal pueda gestionar los pines de las Estaciones como si estos estuvieran físicamente presentes en él es preciso que exista una retroalimentación. (Figura 4.22).

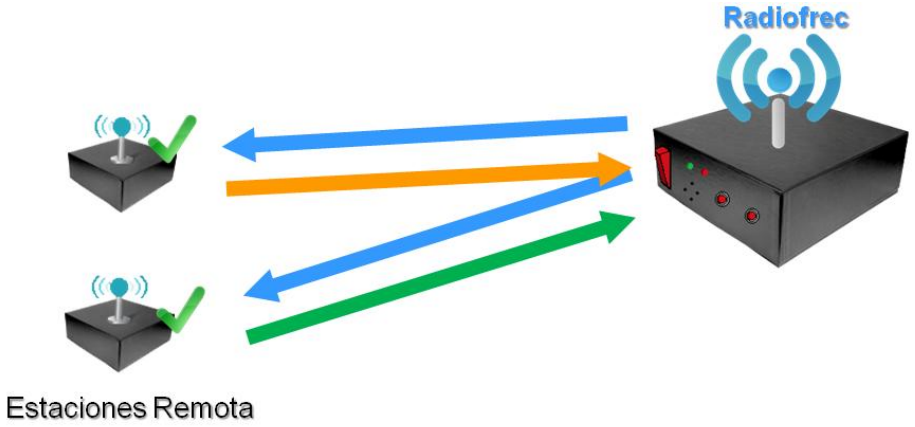


Figura 4.22: Flujo de información Módulo Principal-Estaciones.

Sin embargo e Módulo Principal no puede conectarse a más de una Estación a la vez. Por ello, lo que hace es conectarse a cada una de las Estaciones disponibles en cada ciclo de programa, cambiando la frecuencia (el canal en la librería). En todo momento existe un flujo constante de información aunque por la tarea u orden enviada al sistema no se precise de ninguna Estación.

Esto tiene la ventaja de que no es preciso que todas las Estacione estén conectadas en el momento de en el que el usuario programa una tarea que implica alguna de las Estaciones, sino que simplemente, cuando la Estación correspondiente se encuentre disponible, automáticamente recibirá las órdenes del Módulo Principal. Lo que permite que una Estación se pueda conectar en cualquier momento y en caso de fallo, simplemente reiniciando dicha Estación por medio del pulsado que incorporan cada una de ella, se volverá a restablecer el flujo de información.

El sistema planteado en el presente proyecto se describe como un sistema centralizado, donde el Módulo Principal es el que se encarga de recibir todas las órdenes del usuario y transmitirlas a la Estación correspondiente, tal y como se refleja en la Figura 4.23.

Figura 4.23: Comunicación Smartphone-Estaciones.

El Módulo Principal sabe a qué estación mandar las órdenes inmediatas del usuario, gracias a que la trama mandad incluye el pin y para valores superiores a 69. Donde, los pines correspondientes a cada Estación:

- **Estación A**: 70-79
- **Estación B**: 80-89
- **Estación C**: 90-99
- **Estación D**: 100-109
- **Estación E**: 110-119
- **Estación F**: 120-129
- **Estación G**: 130-139

El algoritmo empleado para el tráfico de información por radiofrecuencia comienza inicializando una serie de variables a cero (Canal=80; ErrorRF=0), tras ello trata de establecer comunicación con la Estación remota cambiando la frecuencia. Dado que para manejar el módulo se ha empleado una librería llamada nRF24L01p.H, donde los distintos saltos de frecuencia están codificado por canales que van del 0 al 125. Tras haber realizado un análisis de las bandas con un programa que no es objeto del proyecto, se observó que los canales menos "contaminado" correspondían a los canales entre el 80 y el 118, de ahí que se inicialice el Canal en 80, que sería el canal asignado para la Estación A (81 para la Estación B y así sucesivamente). Si la comunicación no se establece, se espera 0,5 segundos, se marca ErrorRF=1 y se trata de volver a conectar. Si en este segundo intento no se establece la comunicación, se cambia al canal siguiente (Canal++) y se repite el proceso. Si por el contrario se logra establecer comunicación, se produce el intercambio de las tramas.

4.5. Telemetría

Entendemos por telemetría a toda la información que el sistema devuelve al usuario por medio de la aplicación para Smartphone que se ha desarrollado. Esta información es solicitada por el usuario por medio del envío de la siguiente trama: "Contraseña+2000000000" independientemente del modo de comunicación que se emplee para ello, es decir, los datos pueden recibirse por cualquiera de los tres modos de conectividad antes descritos, siendo la forma de visualización de los mismos idéntica, tal y como se comentará a continuación.

Antes de hablar de cómo se realizar la adquisición de dichos datos, es preciso comentar de que se componen:

A) Datos de los periféricos

Aunque se mandan todos los datos en un solo bloque, al principio del mismo aparecen los datos referentes a los periféricos, entendiendo como tales tanto los sensores como los actuadores, separados cada uno por un ";". El Módulo Principal, lleva internamente un registro de todos los pines que son configurables por el sistema, contando tanto los propios del módulo principal (35 pines) como los pertenecientes a las Estaciones (10 pines cada una). Para reducir al máximo el tamaño del la trama de respuesta de telemetría, únicamente se envía información de aquellos pines que se encuentren configurados, es decir, aquellos que hayan formado parte de una o varias tareas programadas o que sean una acción inmediata para el usuario, es decir, si el usuario programa varias tareas que no emplean un relé en concreto, dicho relé solo aparecerá en los datos de telemetría solo si el usuario ha decidido controlarlo en un momento dado.

El estado de los pines configurados se manda en forma de acrónimos para reducir el tamaño. En la Figura 4.24, se observa que para un pin configurado se compone dos o tres dígitos para el pin, luego un acrónimo de lo que hay conectado y luego su valor o estado seguido de un ";".

Figura 4.24: Estructura del estado de los pines en telemetría.

En este caso, nos indican que en el pin 35 del Módulo Principal (pin 69 del Arduino Mega) hay un sensor de temperatura (de ahí el acrónimo ST) y que el valor que ha registrado en el momento de solicitar dicha información ha sido de 35ºC.

El motivo de reducir las tramas lo máximo posible es principalmente para cuando se opera con el Modo SMS, a fin de reducir al máximo el número de SMS necesarios para transmitirlo.

B) Datos del GPS

Tras mandar los datos de todos los pines configurados, se mandan los datos del GPS, siguiendo la siguiente estructura:

… **Latitud;Longitud;Altitud;Velocidad;Fecha;Hora** …

C) Datos generales sobre el Módulo Principal y las Estaciones

Por último se manda en el mismo paquete otros datos generales como son: Nivel de batería del Módulo Principal y cada una de las Estaciones, si están funcionando conectadas a la red o a batería, así como si una Estación está conectada o no.

Esto viene dado mediante la estructura de la Figura 4.25:

Figura 4.25: Estructura datos generales Módulo Principal y Estaciones.

Donde en el caso del Módulo Principal, si se encuentra funcionando a batería, mostrará un valor del porcentaje de carga. Si el número termina en un uno (por ejemplo 981), indica que se encuentra funcionando a corriente. Así si apareciese "981" indicaría que la batería del Módulo Principal está al 98% de capacidad y que se encuentra conectada a la red, por lo que la batería no se descargaría (tampoco se carga porque el circuito de la Figura 3.30 no contempla dicha posibilidad debido al precio de las batería recargables de 9V).

Luego se describe las distintas Estaciones, donde se indica solo se indica mediante una escala de 0-10 su nivel de batería, siendo 10 que está completamente cargada. El cero indica que la Estación correspondiente no se encuentra disponible.

Una vez descrito que compone los datos de telemetría, vamos a mostrar el proceso que ocurre cuando el usuario solicita dicha información.

En primer lugar, el usuario debe de seleccionar el modo de conectividad que desea emplear: Bluetooth, 3G o SMS. Desde el punto de vista del usuario el procedimiento es indiferente de qué modo usar, sin embargo internamente, cambia la forma en que las órdenes viajan del Smartphone al Módulo Principal y viceversa.

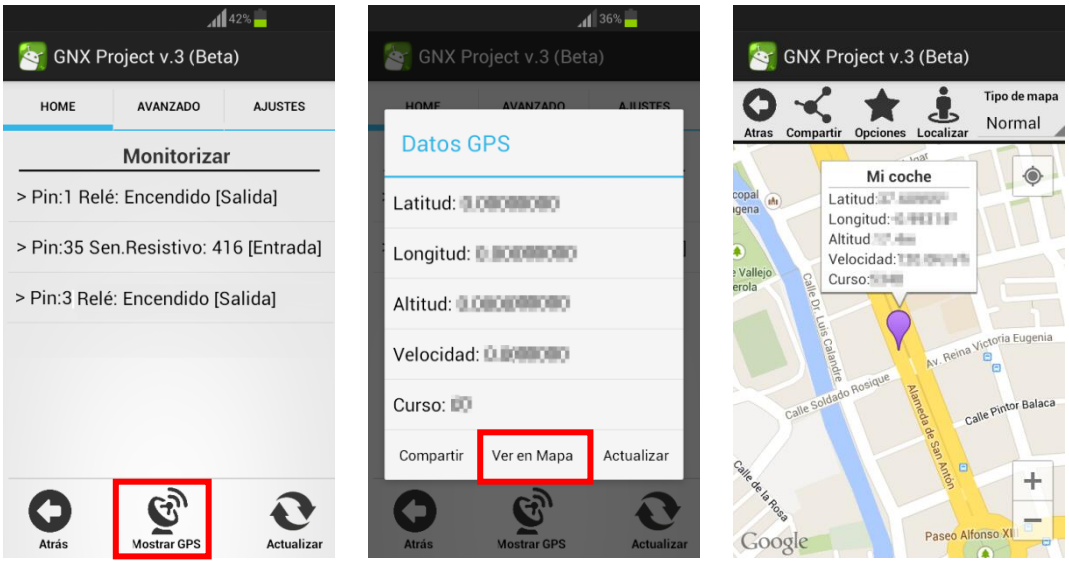
Una vez seleccionado el modo, se debe pulsar sobre la opción del menú principal llamada "**Monitorizar**", lo cual provocará el envío inmediato de la trama "Contraseña+2000000000" donde tal y como se ha recogido en la Tabla 4.2, sirve para solicitar los datos de telemetría. (Figura 4.26).

Figura 4.26: Envió de trama de telemetría.

Cuando la trama llega al Módulo Principal y una vez verificada, se envía un paquete con toda la información antes descrita, como se observa en la Figura 4.27.

Figura 4.27: Recepción de los datos de telemetría.

Cuando los datos llegan a la aplicación Android, descompone dichos datos y traduce los acrónimos en frases de fácil comprensión, tal y como se observa en la Figura 4.28 A, donde se indica el pin, lo que hay conectado y su valor, además si hubiera información de alguna Estación, se especificaría de cual sería dando la posibilidad de poder restablecer la comunicación en caso de fallo con solo pulsar sobre el nombre de la estación.



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

Figura 4.28: Menú Monitorizar.

Además desde este menú podremos actualizar los datos, dado que estos no se actualizan de forma automática, sino bajo demanda del usuario, con el fin de no sobrecargar el sistema en realizar solicitudes periódicas dado que el Módulo Principal no puede realizar tareas en segundo plano, cosa que sí puede la aplicación. Además el tráfico constante de información podría acarrear unos altos costes si se emplease con el Modo 3G o SMS.

Desde el menú Monitorizar podemos observar los datos del GPS mediante la opción "Mostrar GPS", que conduce a una ventana como la de la Figura 4.28 B donde aparecen todos los datos proporcionados en sus unidades correspondientes. Esta información puede ser compartida por medio de la opción "Compartir" por Twitter, Facebook, SMS, Whatsapp, Telegram,etc. Además también podremos ver dichos datos en un mapa por medio de la opción "Ver en Mapa", lo que nos conducirá a una sección de la aplicación basada en Google Maps API2, que es una librería de Google que proporciona bajo una licencia de desarrollo con una caducidad de unos meses que permite dotar a las aplicaciones Android del uso del Google Maps nativo. Como se aprecia en la Figura 4.28 C, nos indica en un mapa con un señalizador la posición del Módulo Principal (Desde el menú de Ajustes se puede asignar un nombre representativo al módulo, en este caso, como se encontraba en un vehículo, se le ha dado el sobre nombre de "Mi Coche"), donde si se pulsa sobre dicho señalador aparecerán los datos antes mostrados. Desde este menú podremos cambiar el tipo de vista a: Normal, Satélite, Híbrida y Tráfico.

5. Diseño de la aplicación para Smartphone

5.1. Introducción

En el presente capítulo, se describirá de forma detallada los distintos menús y funciones que posee la aplicación para Android que se ha desarrollado expresamente para este proyecto.

5.2. Descripción del menú principal

El menú principal de la aplicación para Smartphone cuenta con tres pestañas: **Home**, **Avanzados** y **Ajustes** (Figura 5.1). La primera contiene las opciones para lograr un control de todas las funcionalidades del proyecto. La segunda incluye algunas opciones adicionales que no son precisas para poder realizar un control directo en el sistema pero que sí permite ampliar sus funcionalidades. La tercera incluye los parámetros que pueden ser editados por el usuario y que rigen el funcionamiento del sistema.

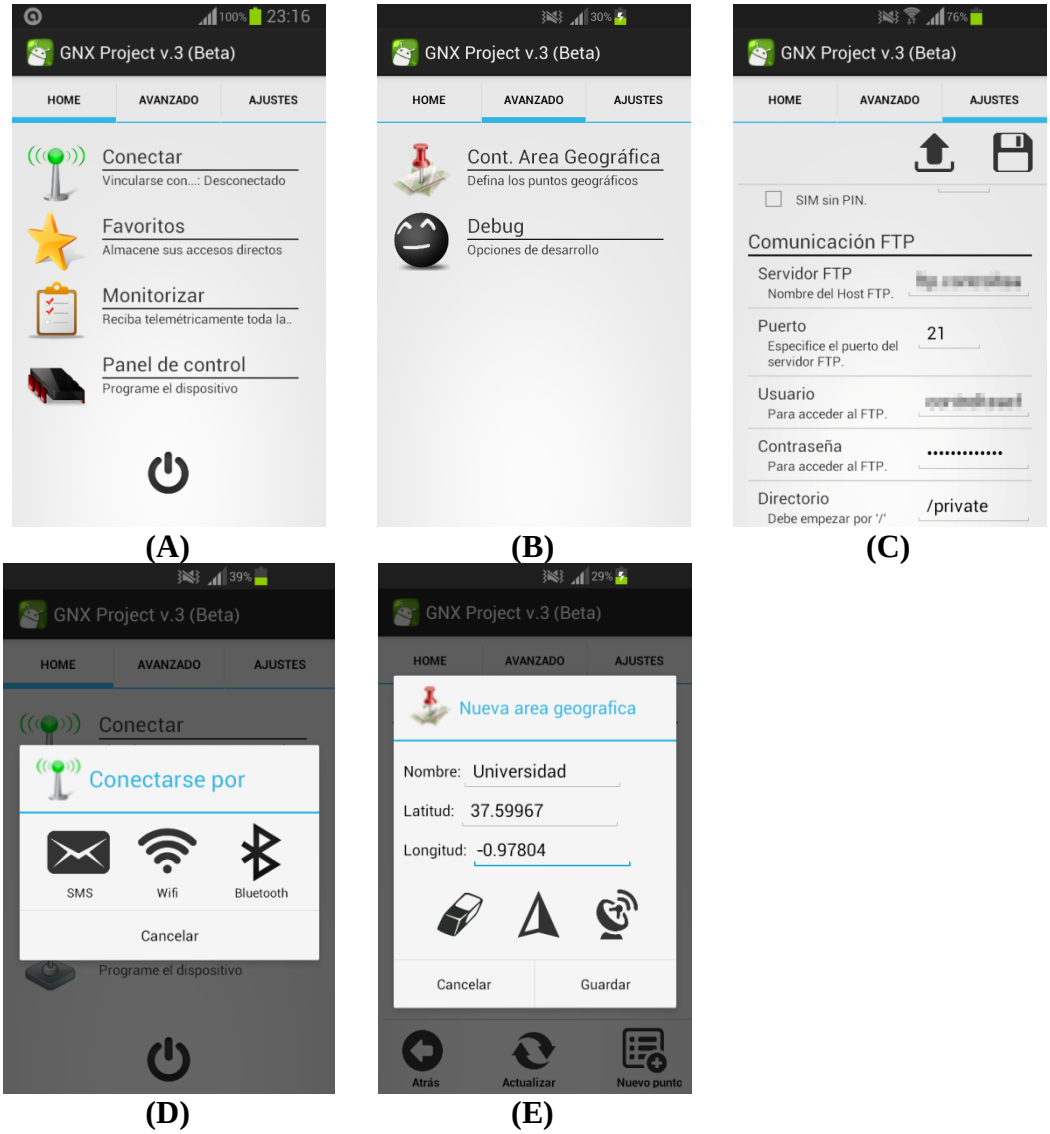


Figura 5.1: Menú Principal aplicación Android.

En la Figura 5.1 A, se observa la existencia de un botón de salir. El motivo de este es que en la aplicación se ha desactivado la tecla "back" que dispone el Smartphone o Tablet, a fin de evitar que el usuario salga de la aplicación sin que se haya desconectado la conexión por bluetooth y guardado la configuración.

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

- **Conectar**: Por medio de esta opción, el usuario puede elegir el modo de conectividad que desee emplear, eligiendo entre Bluetooth, 3G o SMS, tal y como se aprecia en la Figura 5.1 D. El modo seleccionado determinará toda la dinámica interna de la aplicación dado que se precisa de distintos procedimiento para poder transmitir y recibir los datos.

- **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 del Módulo Principal. 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 la posibilidad de conocer los datos del GPS con posibilidad de compartir dicha información o verla en un mapa.

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

En lo referente a la pestaña **Avanzados** (Figura 5.1 B), encontramos las siguientes opciones:

- **Control por área geográfica:** En este menú (Figura 5.1 D), se pueden definir los distintos puntos geográficos que se emplearán con posterioridad en la programación de una tarea. En este menú se crean puntos geográficos a los que se le asigna un nombre representativo, como por ejemplo "Casa" o "Universidad" y acto seguido se establecen sus coordenadas o bien a mano, o automáticamente a partir de la posición del usuario (ubicación del Smartphone o Tablet) o del Módulo Principal.

- **Debug:** Esta opción es para fines únicamente de mantenimiento y experimentación para el programador, donde se alojan determinadas opciones como la posibilidad de introducir tramas de forma manual por medio de un teclado en pantalla.

5.3. Menú Favoritos

Como hemos mencionado, el este menú podremos añadir las acciones que deseamos ejecutar con regularidad, sin necesidad que estas se encuentren asociadas a alguna condición, es decir, pueden ser ejecutadas de forma inmediata por el usuario. En este menú encontramos con tres opciones de partida: **Agregar nuevo favorito** y **Atras**. (Figura 5.2 A).

Para añadir un nuevo acceso directo o favorito, simplemente debemos pulsar la opción "N**uevo favorito**". Tras esto deberemos seleccionar el nombre (Figura 5.2 B) que queremos que aparezca en el menú principal para a continuación definir la acción asociada mediante el menú de la Figura 5.2 C.

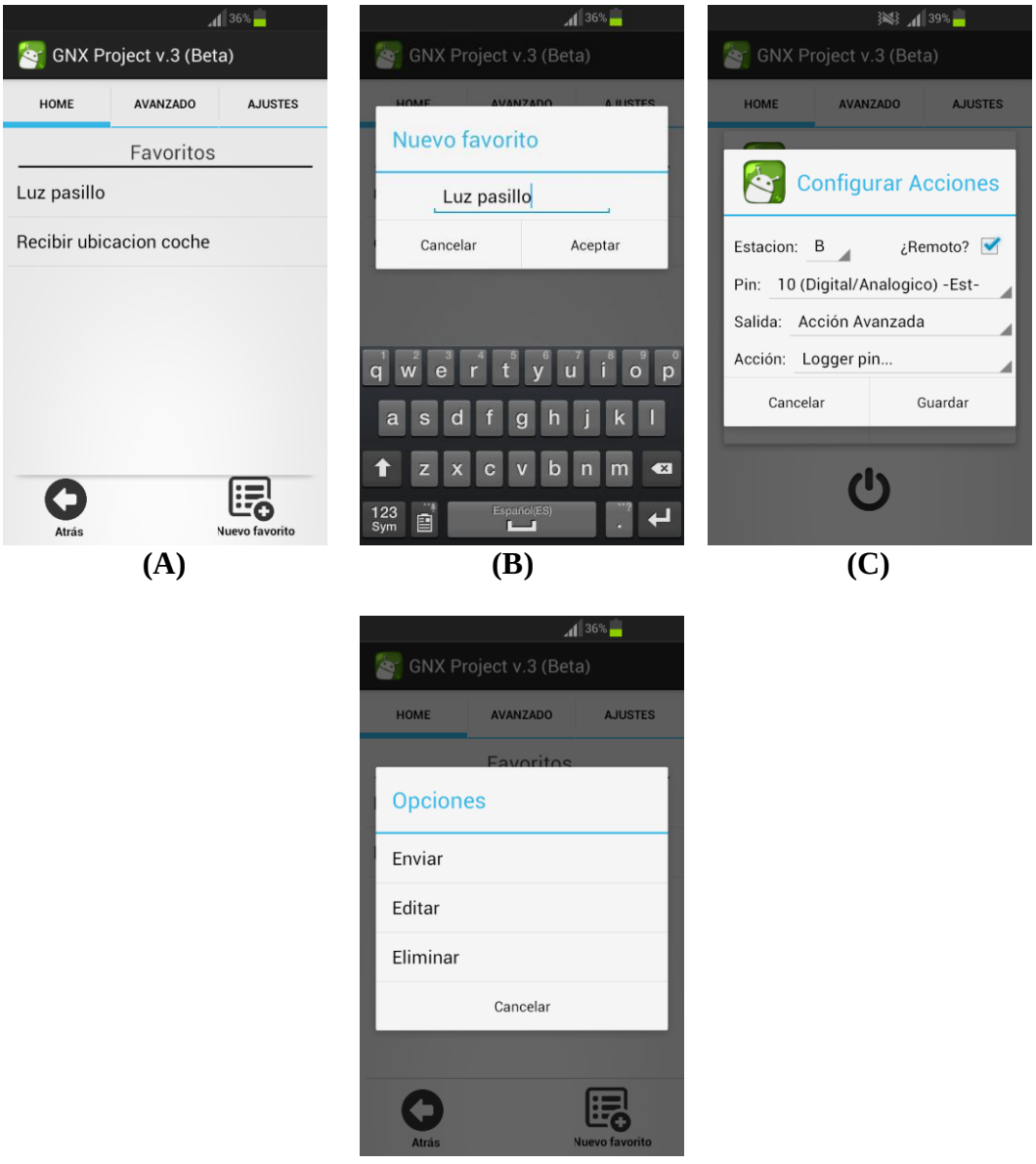


Figura 5.2: Menú de Favoritos aplicación Android.

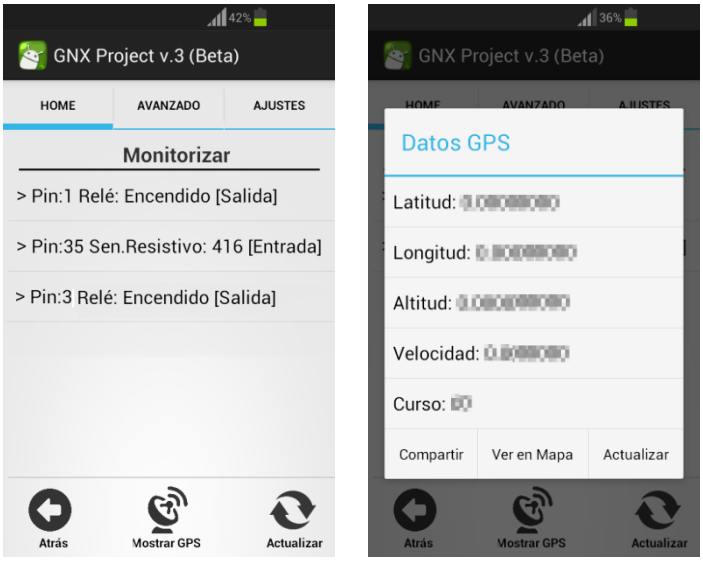
Una vez creado, es posible realizar varias acciones sobre éste simplemente pulsando sobre él. Nos aparecerá un menú como el de la Figura 5.2 D donde podremos realizar las siguientes operaciones:

- **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.
- **Editar orden**: Posibilita redefinir los parámetros de la instrucción almacenada en este menú así como poder asignarle un nuevo nombre.
- **Eliminar favorito**

5.4. 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.



(A) **(B)**

Figura 5.3: Menú Monitorizar aplicación Android.

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 5.3 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 "**Datos GPS**", aparecerá una nueva pantalla indicándose la longitud, altitud, latitud, velocidad y curso como podemos observar en la Figura 5.3 B. Teniendo la posibilidad de compartir dichos datos mediante la opción "Compartir" y visualizarlos en un mapa siempre y cuando existan datos GPS.

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. Después de cada cambio, se actualiza la pantalla de manera automática.

5.5. 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 5.4 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 5.4 B podemos definir esta condición, seleccionando el pin de los 35 disponibles de Módulo Principal más los de las Estaciones (indicándose el tipo de pines digital, analógicos o si permiten PWM como se aprecia en la Figura 5.4 D), el tipo de sensor (Figura 5.4 C) y la condición a cumplir (Figura 5.4 E).

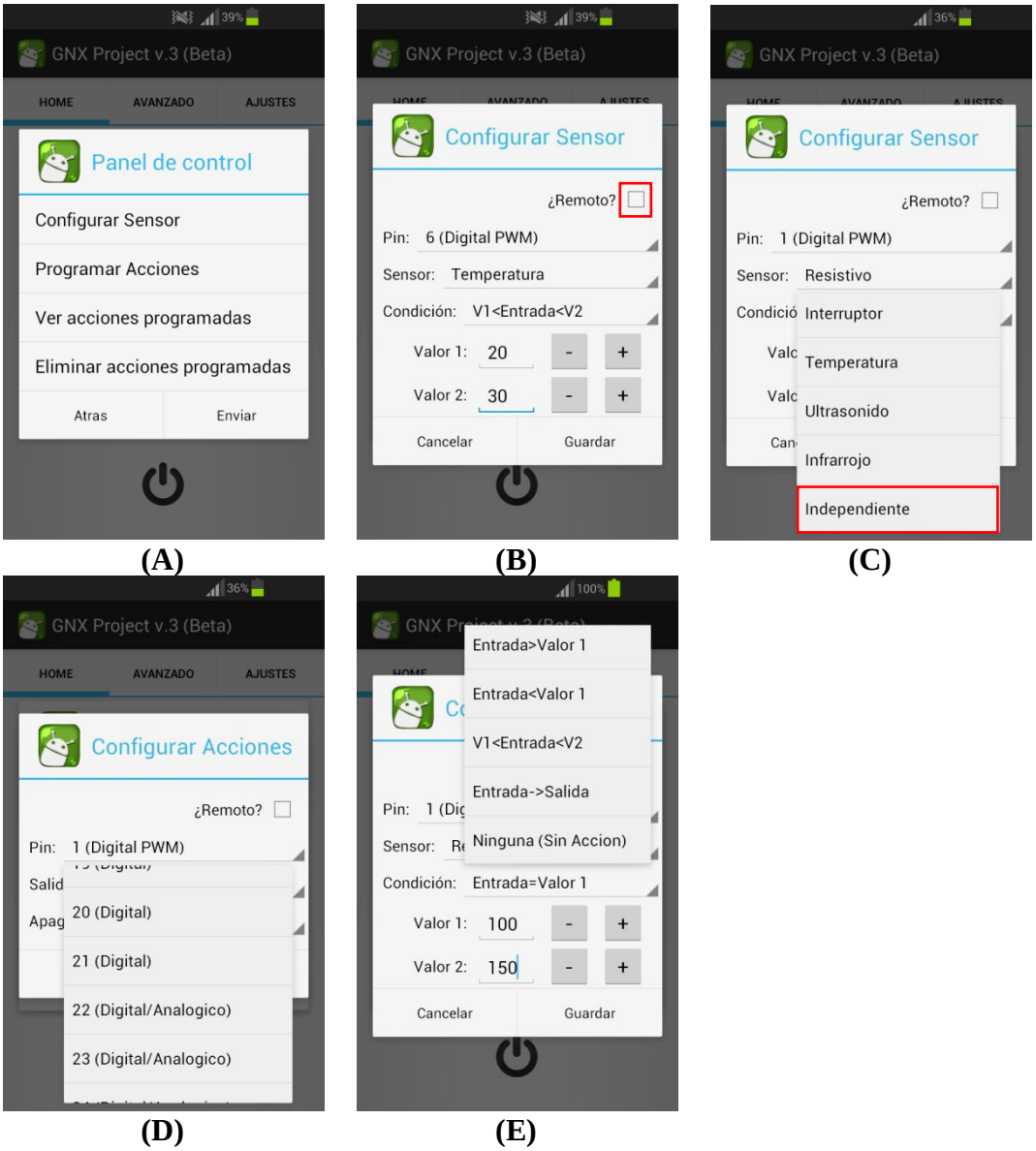


Figura 5.4: Menú Panel de Control. Configurar Sensor.

Como apreciamos en la Figura 5.4 E existen varias opciones a elegir, siendo las tres últimas las que merecen una explicación más en profundidad:

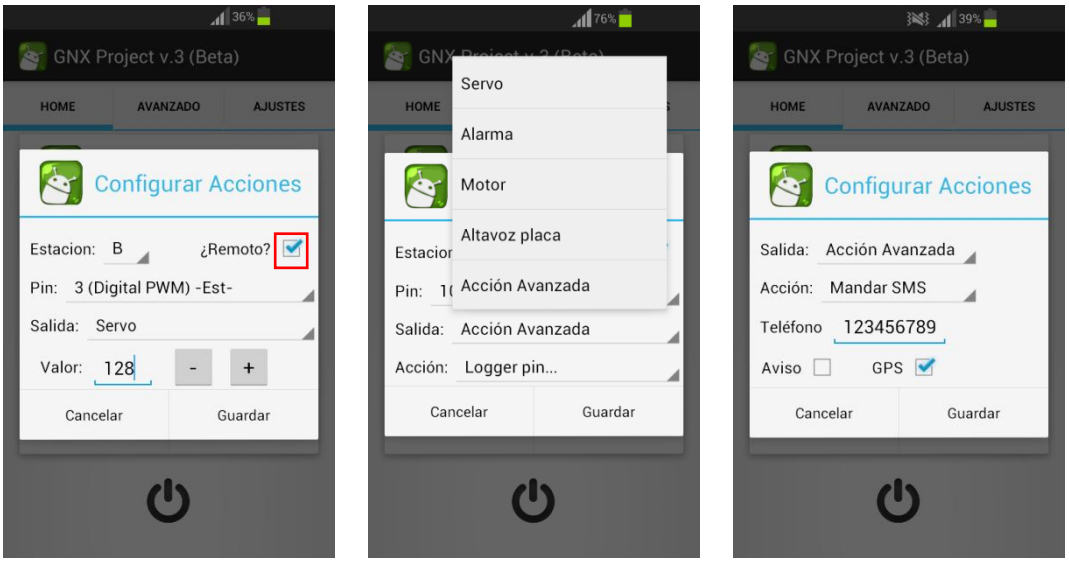
- **Entrada→Salida**: Mediante esta opción podemos hacer que el valor presente a la entrada, es decir, el valor del sensor del tipo y pin indicado se transfiera directamente una o más salidas que definiremos en el submenú "**Programar Acciones**". Esto permite por ejemplo, mover un servo en función de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

Se debe observar que existe una casilla en la parte superior derecha designada como "¿Remoto?" que sirve para poder indicar si la configuración es referida a una Estación y no al Módulo Principal. Cuando se marca esta opción aparece un nuevo desplegable donde elegiremos la Estación deseada.

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**.



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

Figura 5.5: Menú Panel de Control. Programar acciones.

En ésta podemos observar (Figura 5.5 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 pines que incorpora el proyecto. Como se aprecia en la Figura 5.5 A, se encuentra asociado a cada uno de ellos una descripción.

- **Salida**: Permite seleccionar entre los actuadores que pueden ser utilizados para este proyecto (Figura C.5 B). 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 5.5 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.

Como se aprecia, igual que en el menú "Configurar Sensor", existe la posibilidad de definir un pin de una Estación. Como se aprecia en la Figura 5.5 A, se está actuando sobre el pin 3 de la Estación B.

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 5.6 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 15 dígitos justo debajo de cada descripción. Este es el código que se define esa trama. Los motivos de visualizarlo son para comprobar el correcto funcionamiento de la aplicación.

(A) **(B)**

Figura 5.6: Menú Panel de Control. Buffer de salida.

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. 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.

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. (Figura 5.6 B)

- **Enviar**: Mediante esta opción podemos cargar todo el contenido configurado. Durante la descarga de datos, se va visualizando todo aquello que se transmite 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**".

5.6. Pestaña Configuración

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

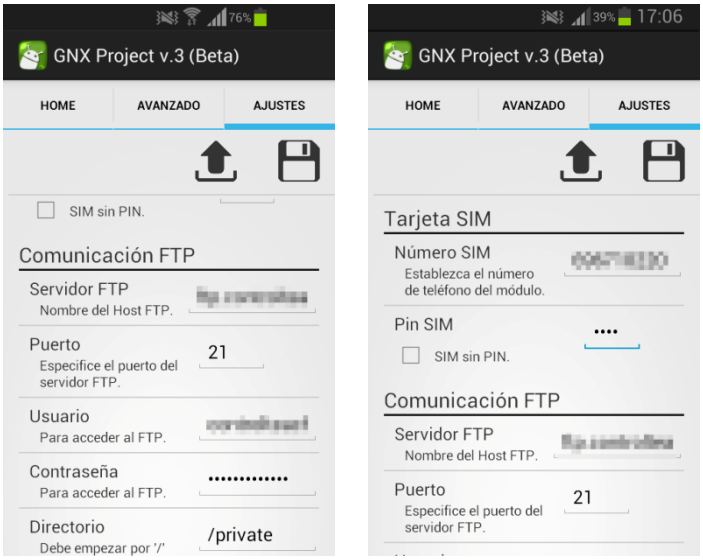
Este consta de tres opciones que pueden observarse en la Figura 5.6 A:

- **Cambiar Contraseña**: 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.

- **Configuración FTP**: Como se ha mencionado en capítulos anteriores, este aparatado es necesario para definir los parámetros que permitirán el uso del Modo 3G. (Figura 5.7 A).

- **Tarjeta SIM**: En él se especifican el número de teléfono de la SIM insertada en el Módulo Principal, así como su PIN si lo tuviera. Es esencial para el Modo SMS. (Figura 5.7 B).

- **Otros ajustes generales**: Entendiendo como tales el intervalo de grabación de los datos GPS o de los Sensores en la MicroSD del Módulo Principal, entre otras opciones.



(A) **(B)**

Figura 5.7: Pestaña Configuración aplicación Android.

Una vez definida la configuración, es preciso guardarla mediante el icono del disquete y luego mandarla mediante el icono de su izquierda siempre y cuando se haya establecido previamente una comunicación con el Módulo Principal por el modo que se desee.

6. Proceso de carga del firmware y la aplicación Android

6.1. Introducción

En capítulos anteriores se han descrito el firmware que debe cargarse al Módulo Principal así como a las Estaciones y la aplicación para Smartphone que permite manejar el sistema. En este capítulo se describirá los pasos para poder instalar dichos elementos.

6.2. 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 6.1 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.

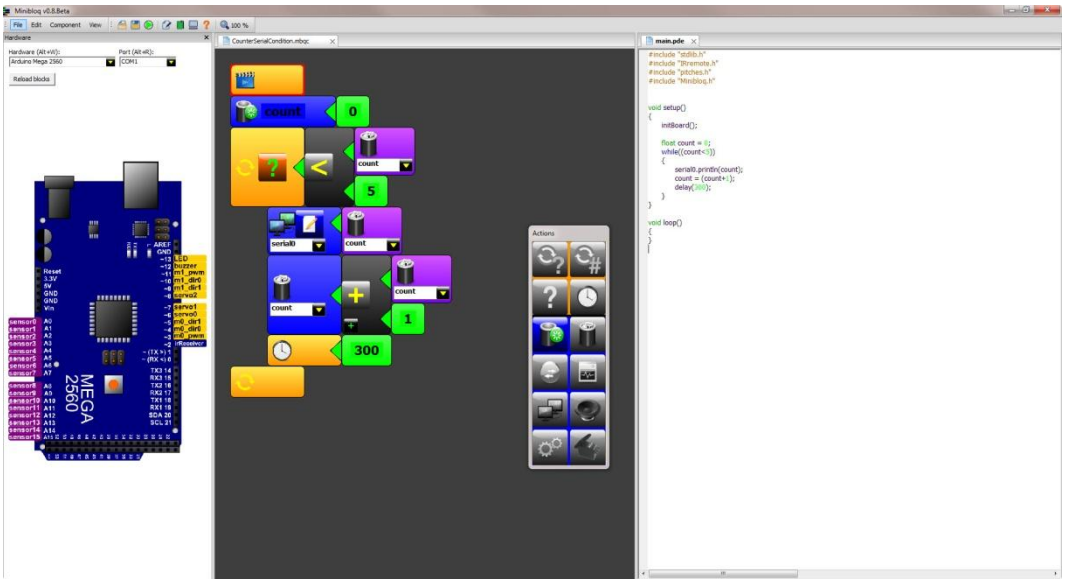


Figura 6.1: Menú principal del software de desarrollo Miniblg.

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 1.0.5 del IDE de Arduino que es la última versión estable hasta la fecha. (Figura 6.2).

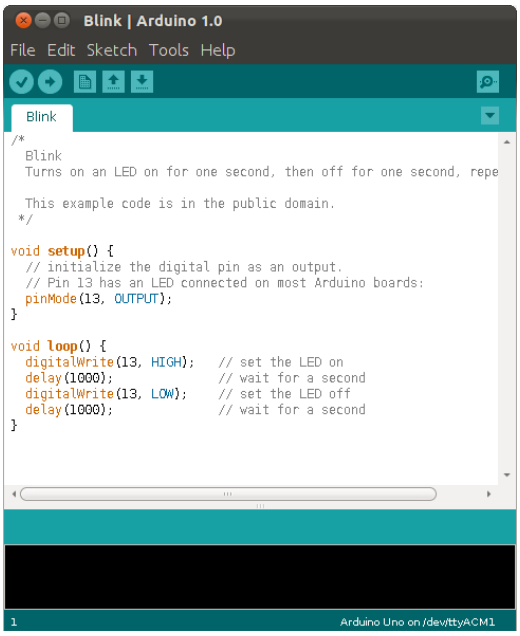


Figura 6.2: IDE Arduino 1.0.5.

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 6.1, vamos a presentar cada unas de ellas 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 6.1: Descripción de las opciones de la IDE de Arduino.

Como hemos mencionado con anterioridad, el proyecto está basado en el Arduino Mega 2560 y Arduino Nano v3. Para poder utilizar esta placa con la IDE de Arduino se precisa indicarle en el menú **Tools/Board** el tipo de placa como vemos en la Figura 6.3.

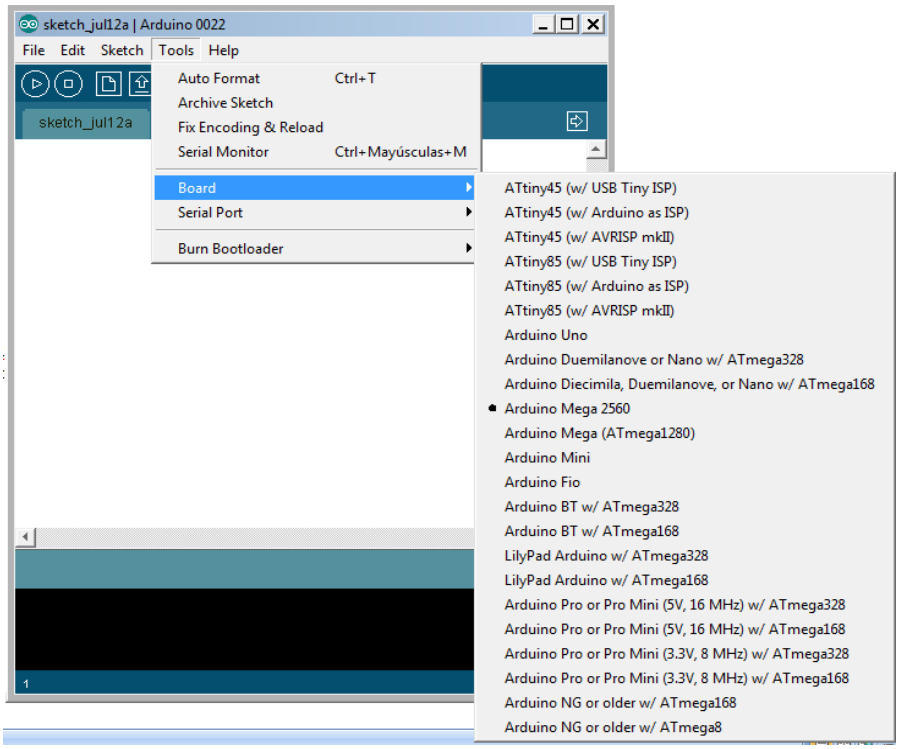


Figura 6.3: Selección de modelo de placa en la IDE.

Esta operación sería suficiente para cualquier otro modelo (a excepción de la versión UNO), sin embargo, al conectar el Arduino 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 6.4).

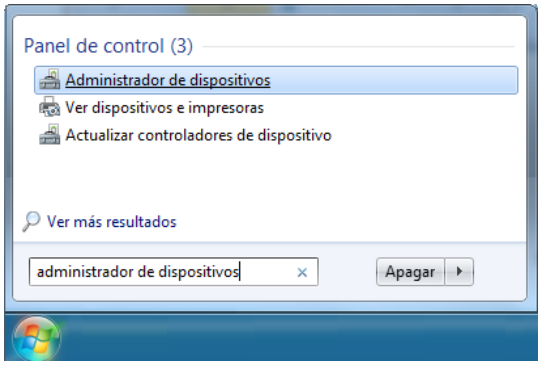


Figura 6.4: Acceso al Administrador de Dispositivos.

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 6.5.

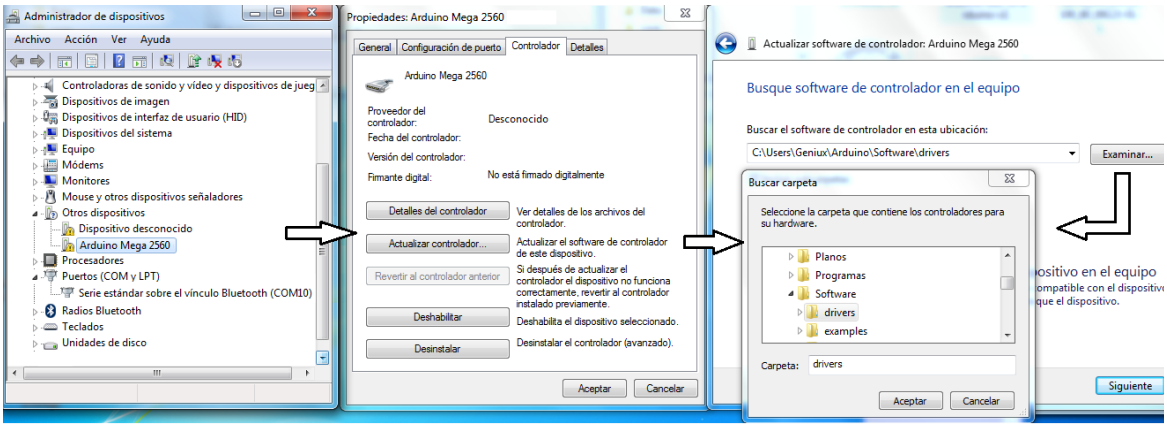


Figura 6.5: Pasos para instalar drivers.

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.

En caso de cualquier fallo durante la instalación, aparecerá un mensaje de error como el de la Figura 6.6.

Figura 6.6: 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 6.7 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.

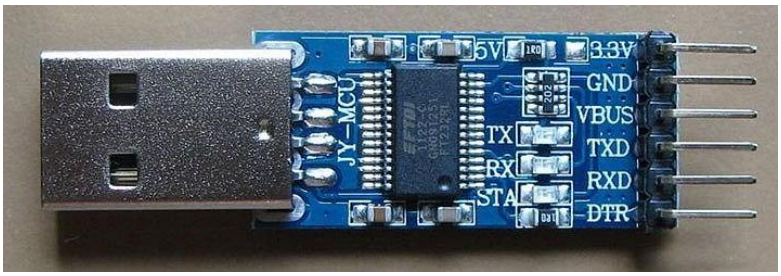


Figura 6.7: Adaptador USB-Serie.

6.3. Implementación de la aplicación para Smartphone

El proceso de carga de la aplicación para Android en el Smartphone o Tablet es muy sencillo, aunque difiere según si deseamos compilar el código fuente del proyecto o si deseamos instalar el archivo .apk ya compilado.

6.3.1. Instalación desde el código fuente

Si se parte del código fuente de la aplicación, es necesario compilarlo para poder obtener el archivo .apk a instalar.

Para ello, se ha empleado la herramienta de desarrollo Eclipse, que es un editor de múltiples lenguajes de programación al que se le ha instalado el SDK el Android desde su web con el fin de poder programar directamente en él.

El procedimiento dentro de la aplicación (una vez se cuente con todo el entorno de desarrollo instalado) es:

1.- Haz click sobre la pestaña de File, en la parte superior del entorno de desarrollo Eclipse.

2.- Seleccionar la opción de import.

3.- Dar click en General --> Existing proyects into workspace.

4.- Se mostrará la ventana Import Proyects, seguido del cuadro Select root directory, dar click en el botón browse

5.- Seleccionar el espacio de trabajo (workspace) en donde se encuentra el proyecto a importar.

6.- Una vez seleccionado el proyecto, se mostrará la ruta específica donde se encuentra el proyecto.

7.- Haz click en finalizar para que se muestre el proyecto importado.

Una vez importado se procede a su compilación. Para ello es necesario pulsar sobre el icono  que se encuentra en el centro de la barra superior. Acto seguido saldrá una ventana como de la Figura 6.8, donde aparecenrán los dispositivos Android que tenemos conectados al PC por cable USB. Solo seleccionamos uno de ellos y pulsamos “OK” y comenzará el proceso de compilación y e instalación de la aplicación en el Smartphone o Tablet.

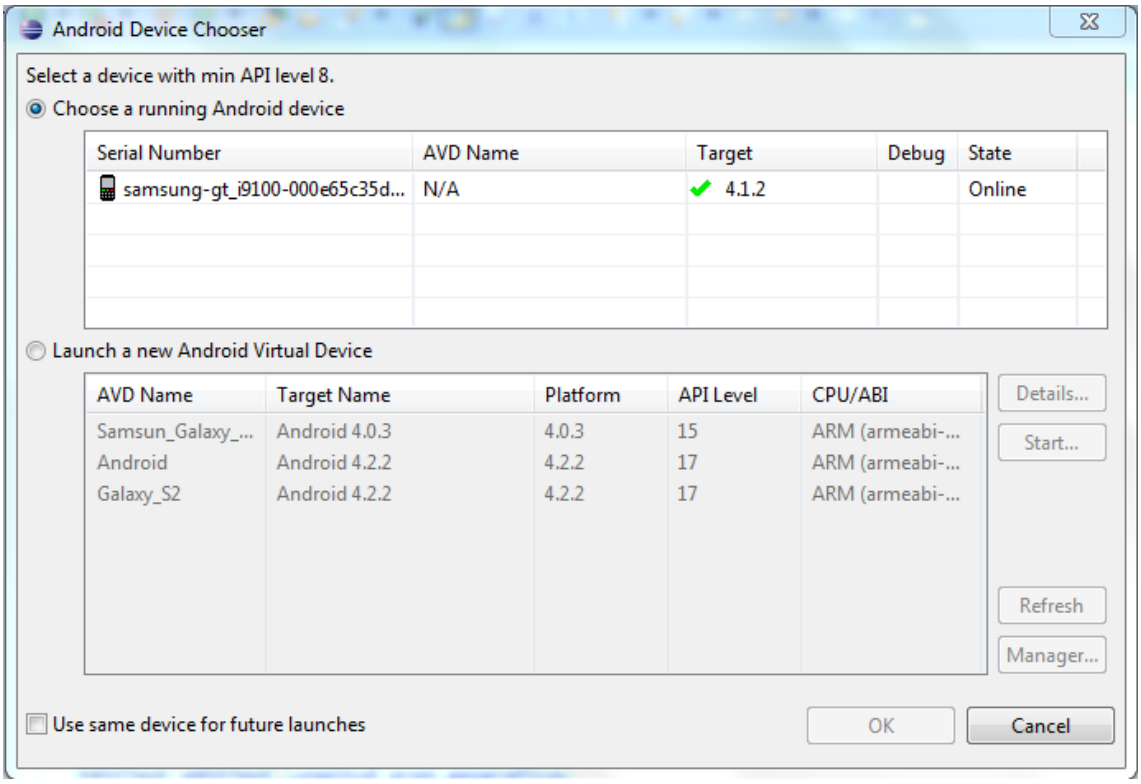


Figura 6.8: Compilación e instalación de .apk.

6.3.2. Instalación directa del .apk

En el caso de instalar el archivo .apk directamente, es tan sencillo como escanear el siguiente código QR para poder descargar e instalar la aplicación.

II. PLANOS Y ESPECIFICACIONES

7. Planos

7.1. Módulo Principal

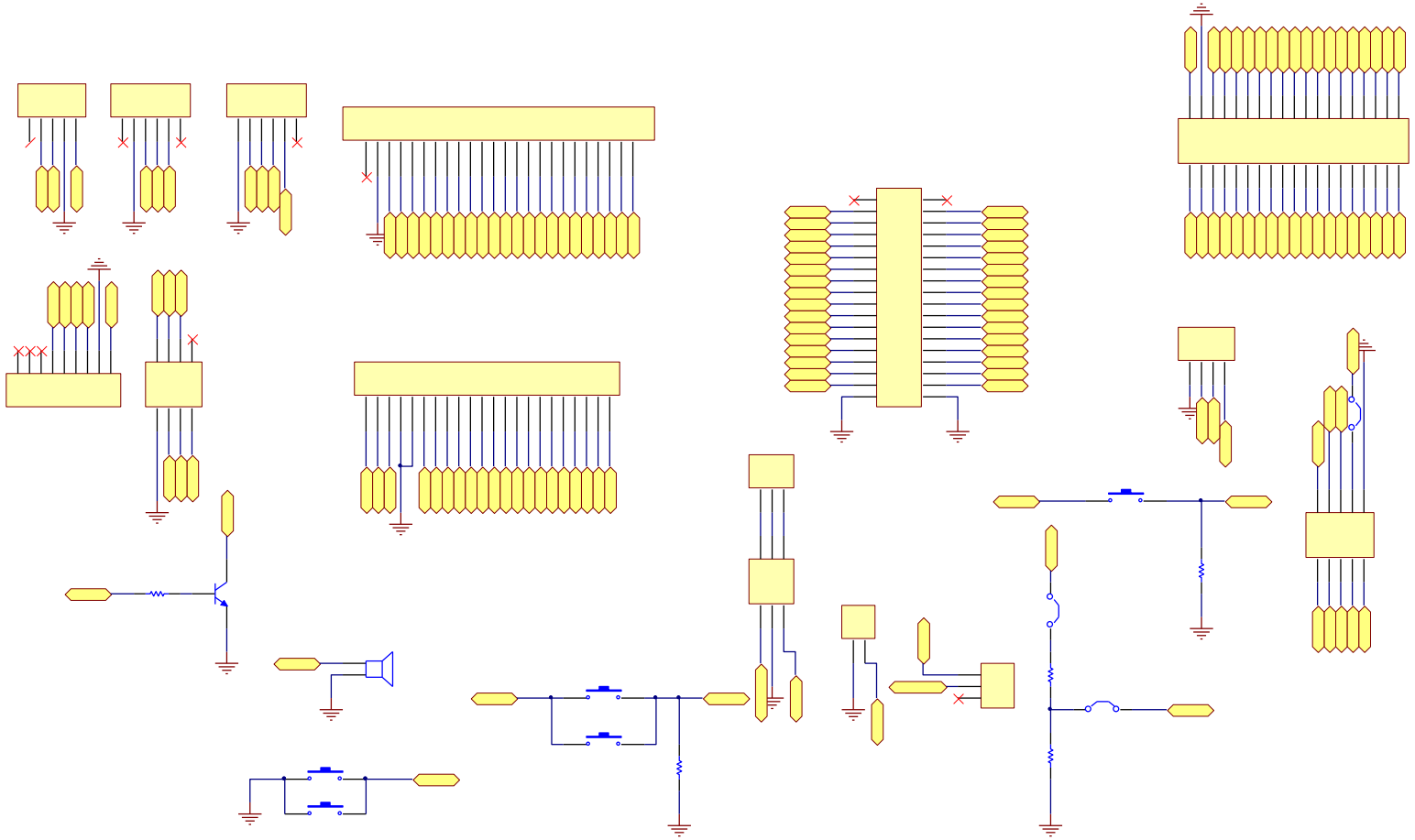


Figura 7.1: Parte 1 del esquemático Módulo Principal.

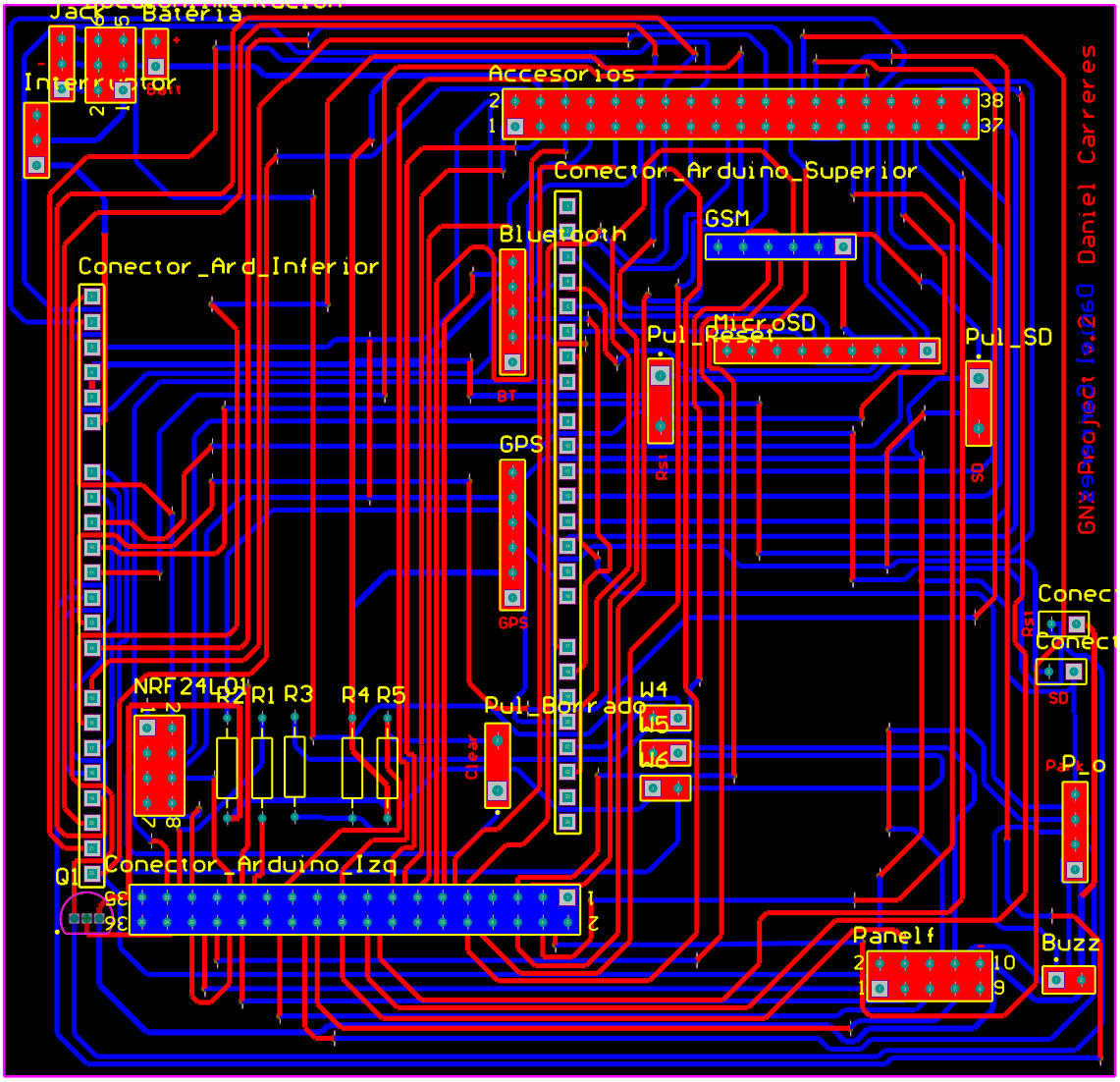


Figura 7.1: PCB del Módulo Principal.

7.2. Conector Módulo Principal

7.2.1. Conector Parte 1 Módulo Principal

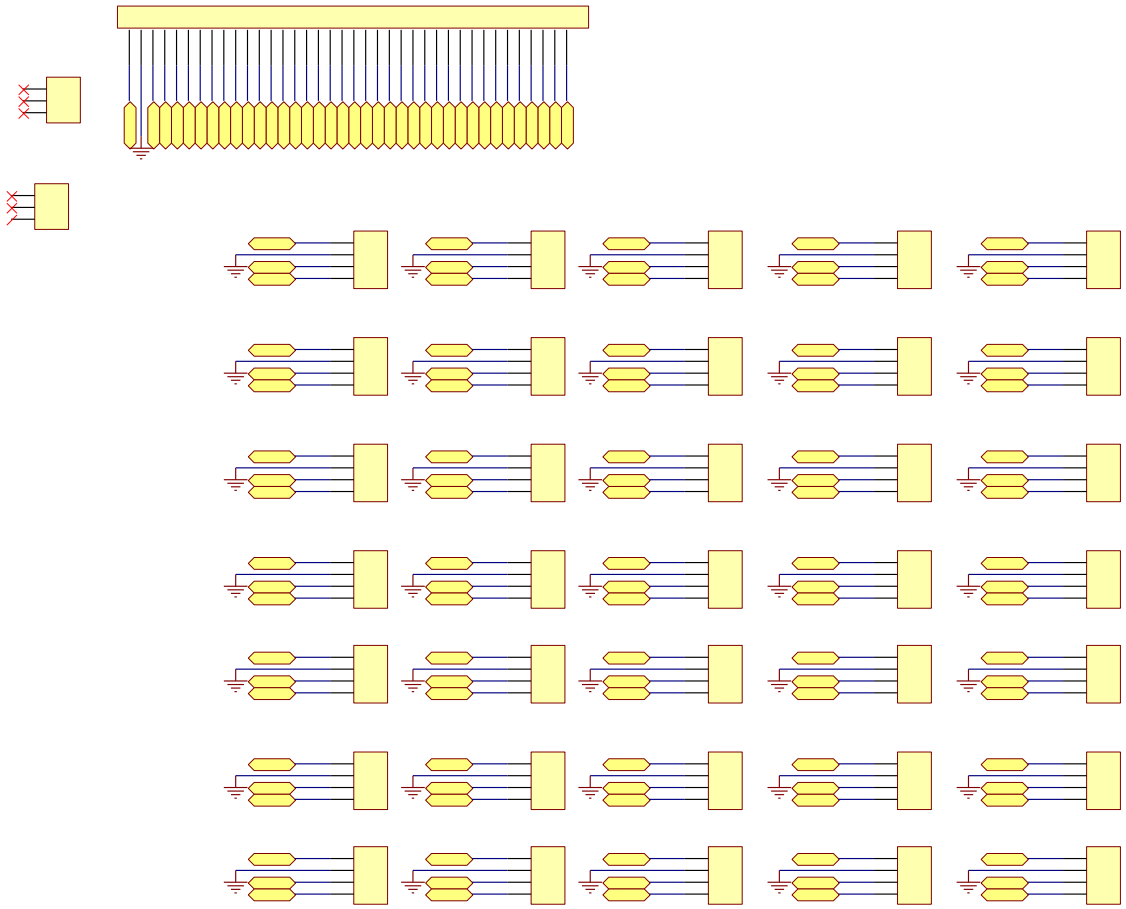


Figura 7.3: Esquemático conector Parte 1 del Módulo Principal.

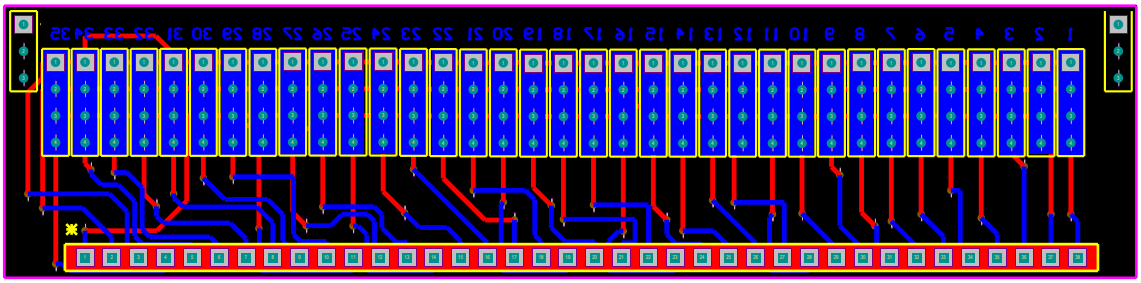


Figura 7.4: PCB conector Parte 1 del Módulo Principal.

7.2.2. Conector Parte 2 Módulo Principal

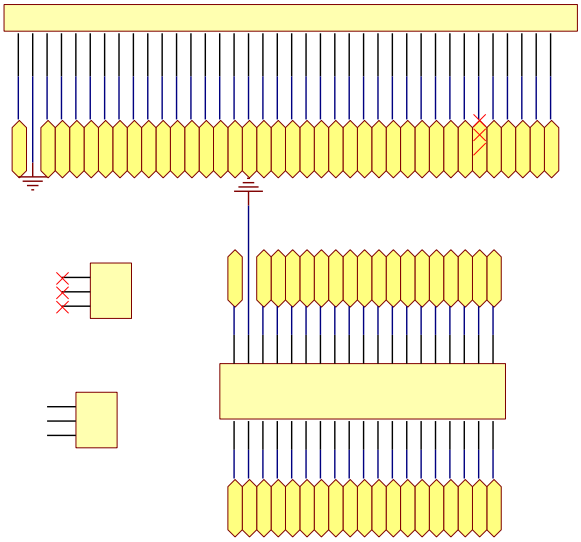


Figura 7.5: Esquemático conector Parte 2 del Módulo Principal.

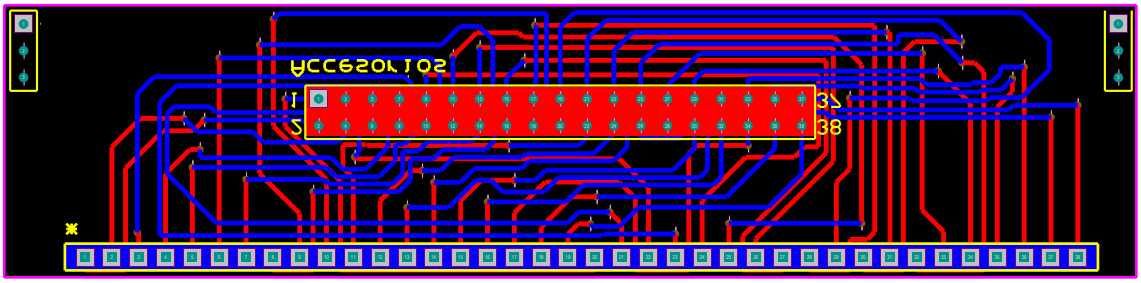


Figura 7.6: PCB conector Parte 2 del Módulo Principal.

7.3. Estación

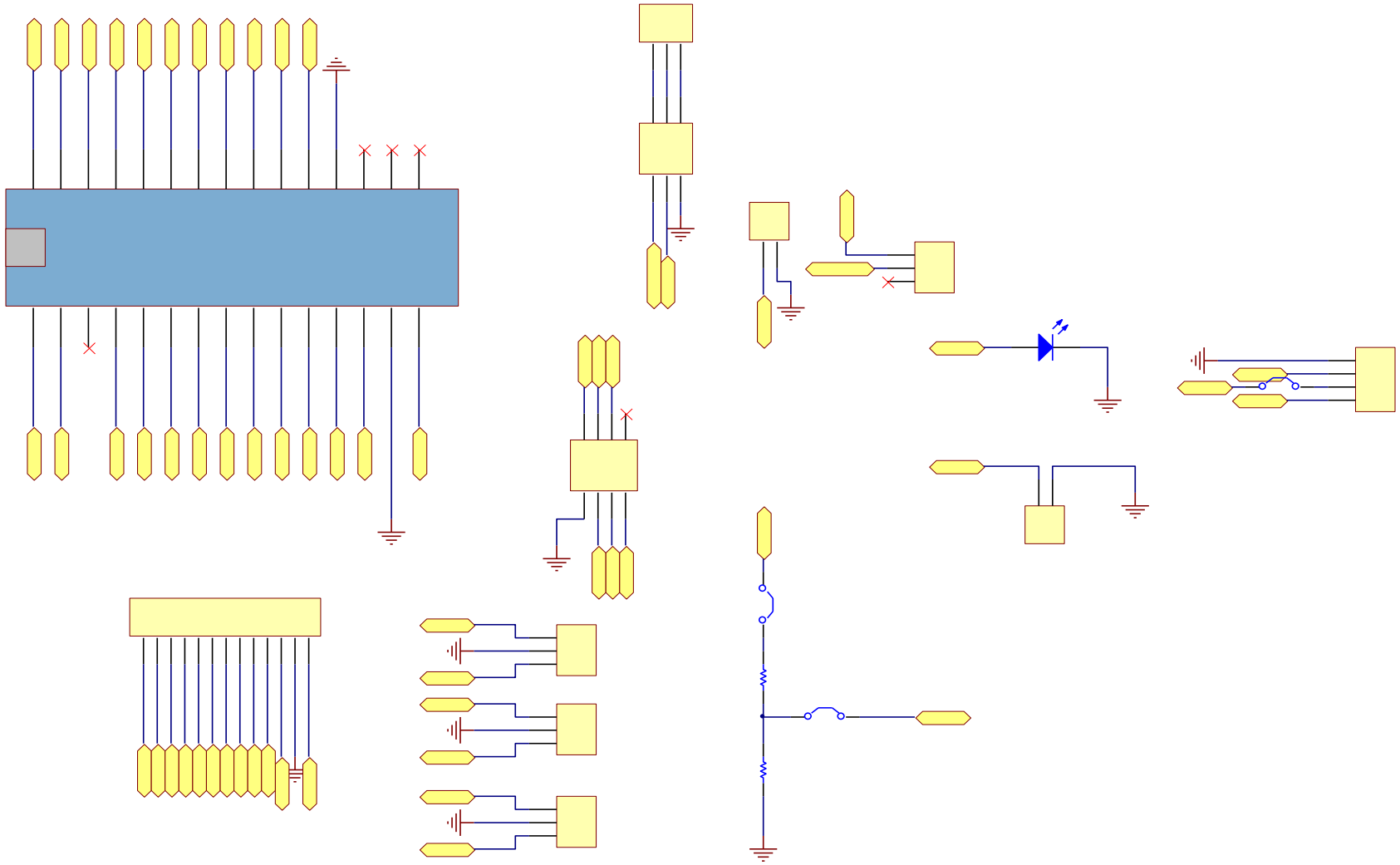


Figura 7.7: Esquemático Estación.

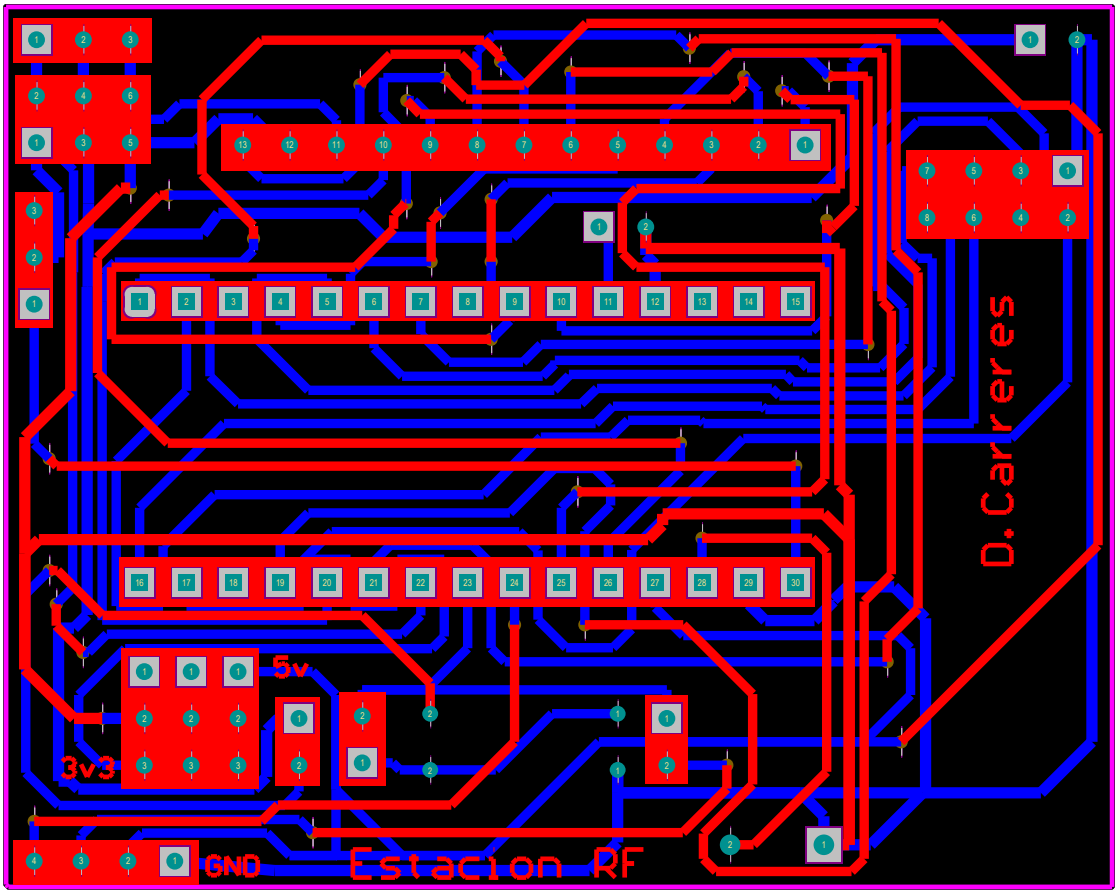


Figura 7.8: PCB Estación.

7.4. Conector Estación

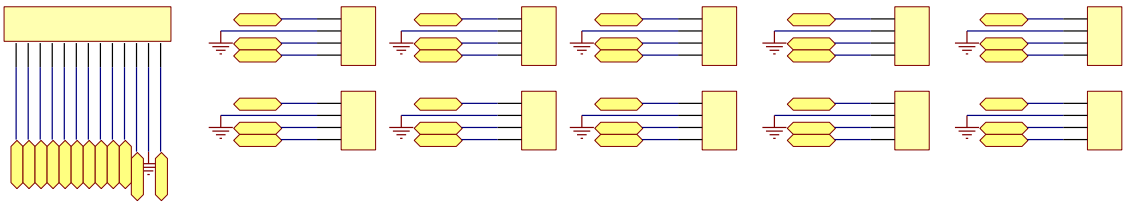


Figura 7.9: Esquemático conector Estación.

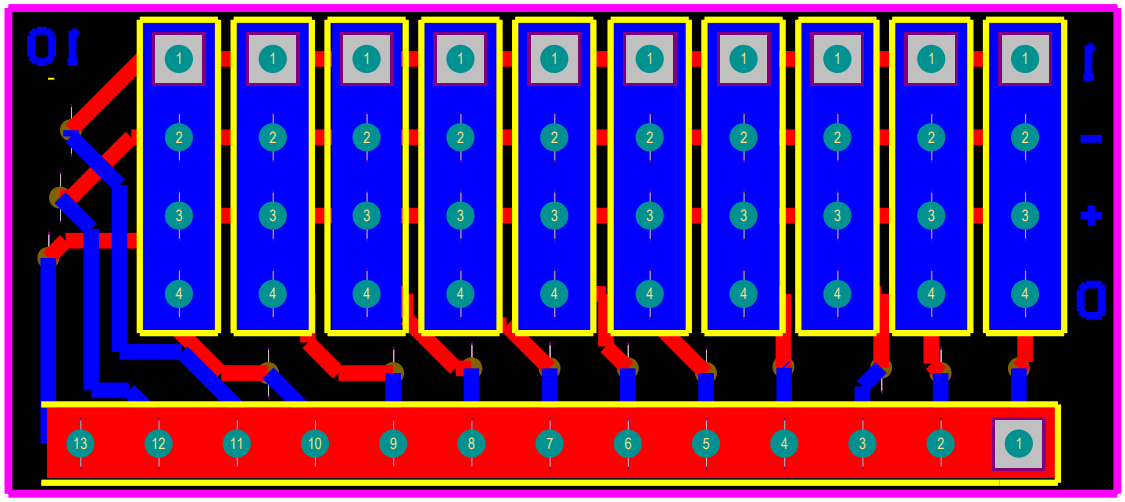


Figura 7.10: PCB conector Estación.

7.5. Pinout

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

Tabla 7.2: Pinout Arduino Mega 2560 y correlación con el sistema GNX Project.

Estación			
Pin	**Correlación**	**Pin**	**Correlación**
AREF	Ninguna	8	Radiofrecuencia
GND	GND	9	Pin 5
RESET	RESET	10	Radiofrecuencia
3.3V	3.3V	11	Radiofrecuencia
5V	5V	12	Radiofrecuencia
VIN	VIN	13	Radiofrecuencia
0-TX0	Ninguna	14-A0	Pin 6
1-RX0	Ninguna	15-A1	Pin 7
2	Led indicador	16-A2	Pin 8
3	Pin 1	17-A3	Indicador batería
4	Pin 2	18-A4	SDA (LCD)
5	Pin 3	19-A5	SCL (LCD)
6	Pin 4	20-A6	Pin 9
7	Radiofrecuencia	21-A7	Pin 10

Tabla 7.3: Pinout Arduino Nano V3 y correlación con el sistema GNX Project.

7.6. Arduino Mega 2560

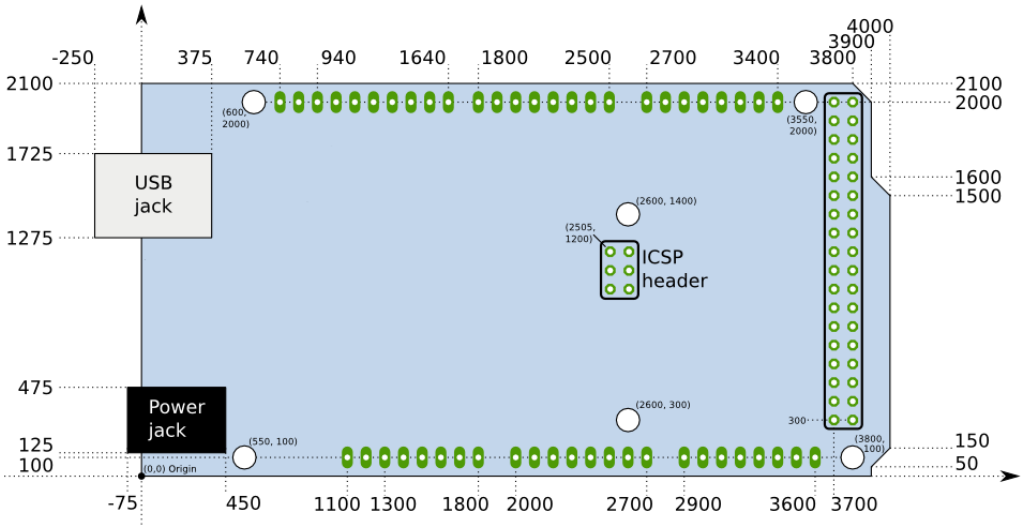


Figura 7.11: Dimensiones generales del Arduino Mega 2560.

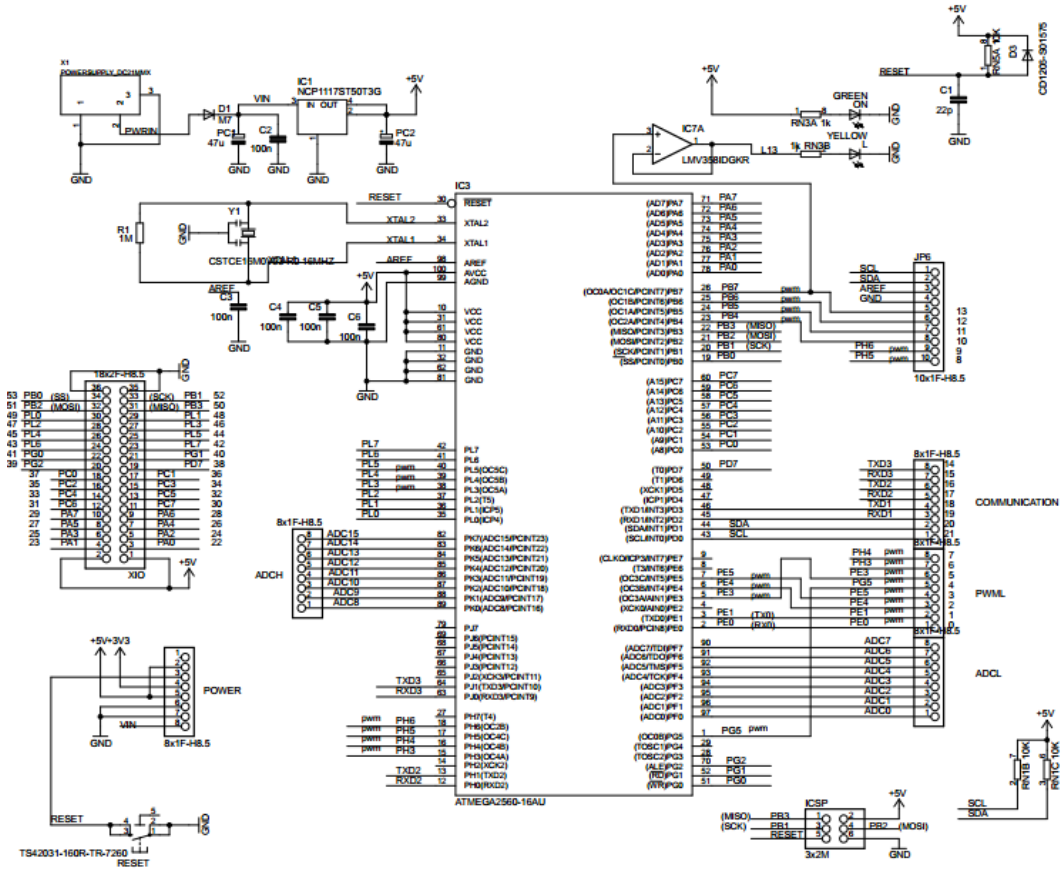


Figura 7.12: Esquemático Arduino Mega 2560.

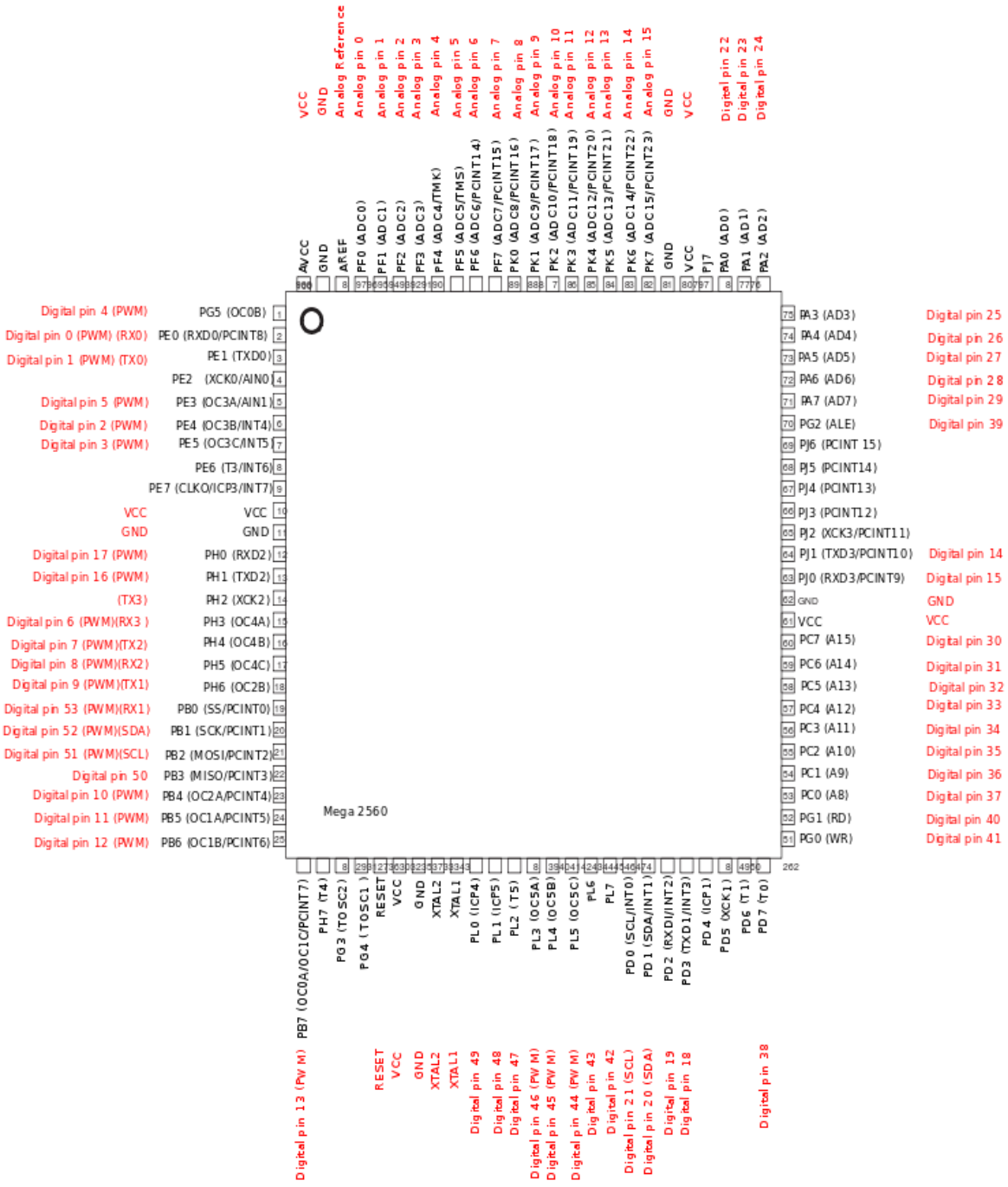


Figura 7.13: Pinout Atmel Mega 2560 y correspondencia con los pines de Arduino.

Bibliografía

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

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

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

Curso de Android
http://www.sgoliver.net/blog/?page_id=3011

EFComPro
http://www.elecfreaks.com/wiki/index.php?title=EFCom_Pro_GPRS/GSM_Module

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.