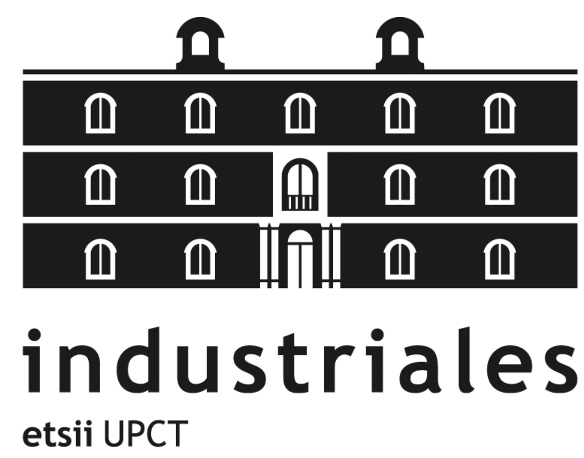


Desarrollo de un dispositivo de telemetría y geolocalización basado en la plataforma Arduino y Shield 3G+GPS

Trabajo de Fin de Grado

Titulación: Grado en Ingeniería en Tecnologías Industriales
Alumno: López Jiménez, Pedro Celestino
Director: Garrigós Guerrero, Francisco Javier

Cartagena, 21 de Julio de 2014

AGRADECIMIENTOS

Dedico este trabajo a mi familia, por su apoyo y preocupación durante el comienzo de mis estudios hasta ahora, y a Javier Garrigós, por depositar su confianza en mí para realizar este trabajo y su gran ayuda durante el desarrollo del mismo.

INFORMACIÓN

Autor	Pedro Celestino López Jiménez
Correo electrónico autor	ninomurcia92@gmail.com
Director	Francisco Javier Garrigós Guerrero
Correo electrónico director	javier.garrigos@upct.es
Titulación	Grado en Ingeniería en Tecnologías Industriales
Departamento	Departamento de Electrónica, Tecnología de Computadoras y Proyectos
Título del trabajo	Desarrollo de un dispositivo de telemetría y geolocalización basado en la plataforma Arduino y Shield 3G+GPS
Title of the work	Development of a telemetry and geopositioning device based on Arduino platform and 3G+GPS Shield

RESUMEN

El objetivo principal del trabajo es el diseño de un dispositivo electrónico basado en la plataforma Arduino capaz de realizar telemetría en tiempo real a través de internet utilizando un Shield con tecnología 3G+GPS. Para ello en primer lugar será necesario diseñar y programar la etapa de adquisición de datos procedentes de diferentes sensores, tanto analógicos como digitales, así como la geolocalización. A continuación se diseñará un procedimiento de empaquetamiento y entramado óptimo de la información, teniendo en cuenta el compromiso entre volumen de datos, ancho del canal, tasa de errores, etc. Finalmente, se diseñará una etapa de comunicaciones, analizando los protocolos más eficientes para esta plataforma y caso de uso, para la transmisión de los datos adquiridos a través de Internet mediante un módem 3G. De esta manera seremos capaces de aplicar el prototipo a una aplicación concreta, en nuestro caso una motocicleta de competición.

ABSTRACT

The main objective of this work is to design an electronic device based on Arduino platform capable of making real-time telemetry through Internet using a Shield with 3G+GPS technology. First of all, it will be necessary to design and program the phase of acquisition of data from different sensors, both analogic and digital, as well as geopositioning data. The next step is to design a method to package and to make an optimal framework of the information, concerning about the compromise between data-volume, and data-width, error tax, etc. Finally, it will necessary to design a communication phase, in which we will study the Internet most-efficient protocols for this platform to transmit the acquired data through Internet by a 3G modem. Then we will be able to apply the prototype to a specific application, in our case a competition motorcycle.

ÍNDICE

1. INTRODUCCIÓN Y OBJETIVOS 9

1.1 INTRODUCCIÓN 9

1.2 DEFINICIÓN DE OBJETIVOS 10

1.3 ELEMENTOS QUE COMPONEN EL SISTEMA 11

1.3.1 Arduino 12

1.3.2 Shield 3G+GPS 14

1.3.3 Sensores 16

1.3.4 Servidor 17

1.4 ENTORNO DE TRABAJO 18

1.5 ESTRUCTURA DE LA MEMORIA 21

2. ESTADO DE LA TÉCNICA 22

2.1 INTRODUCCIÓN A LA TELEMETRÍA 22

2.2 APLICACIONES ACTUALES DE LA TELEMETRÍA 23

2.3 SISTEMAS DE TELEMETRÍA SIMILARES 28

2.3.1 Discusión 30

3. MÉTODOS DE ADQUISICIÓN DE DATOS CON ARDUINO 31

3.1 INTERFACES DE HARDWARE COMPATIBLES CON ARDUINO 31

3.2 COMUNICACIÓN MEDIANTE PINES DIGITALES EN ARDUINO 32

3.2.1 Ejemplo de programa en Arduino para utilización pines digitales 33

3.3 COMUNICACIÓN MEDIANTE PINES ANALÓGICOS EN ARDUINO 34

3.3.1 Ejemplo de programa en Arduino para utilización pines analógicos y PWM ... 37

3.4 COMUNICACIÓN SERIE EN ARDUINO 37

3.4.1 Ejemplo de programa en Arduino para utilización puerto serie 40

3.5 COMUNICACIÓN SPI EN ARDUINO 41

3.6 COMUNICACIÓN I^{2}C EN ARDUINO 43

3.7 COMPARATIVA Y EVALUACIÓN DE LAS PLACAS ARDUINO 46

4. SHIELD 3G+GPS. TRANSMISIÓN DE DATOS A TRAVÉS DE INTERNET Y GEOLOCALIZACIÓN 48

4.1 PRESENTACIÓN SHIELD 3G+GPS. PROTOCOLOS DE INTERNET 48

4.2 DESCRIPCIÓN DE LOS PROTOCOLOS DE INTERNET DISPONIBLES SHIELD 3G+GPS.. 51
4.2.1 Protocolo UDP. Nivel de transporte ... 51
4.2.2 Protocolo TCP. Nivel de transporte ... 52
4.2.3 Protocolo FTP. Nivel de aplicación ... 55
4.3 UTILIZACIÓN DE LOS PROTOCOLOS DE INTERNET EN SHIELD 3G+GPS ... 56
4.3.1 Introducción a los comandos AT ... 56
4.3.2 Inicialización Shield 3G+GPS ... 58
4.3.2.1 Código Arduino para la inicialización Shield 3G+GPS ... 60
4.3.3 Protocolo UDP en Shield 3G+GPS ... 60
4.3.3.1 Código Arduino para la utilización del protocolo UDP con Shield 3G+GPS ... 61
4.3.4 Protocolo FTP en Shield 3G+GPS ... 62
4.3.3.1 Código Arduino para la utilización del protocolo FTP con Shield 3G+GPS ... 63
4.4 GEOLOCALIZACIÓN MEDIANTE SATÉLITES GPS ... 63
4.4.1 Código Arduino para la utilización del módulo GPS ... 66
4.5 EVALUACIÓN DE LOS PROTOCOLOS DE INTERNET PARA UN SISTEMA DE TELEMETRÍA EN TIEMPO REAL ... 67
5. PRUEBAS DE CAMPO ARDUINO MEGA ADK Y SHIELD 3G+GPS ... 69
5.1 DESCRIPCIÓN DE LAS PRUEBAS DE CAMPO ... 69
5.2 TASA DE TRANSMISIÓN DE DATOS ... 69
5.2.1 Tasa de transmisión de datos utilizando protocolo UDP ... 70
5.2.2 Velocidad de transmisión de datos utilizando protocolo FTP ... 71
5.3 TASA DE ERROR DE TRANSMISIÓN DE DATOS ... 72
5.4 TIEMPO DE EJECUCIÓN DEL PROGRAMA ... 73
5.4.1 Tiempo inicialización del programa ... 73
5.4.2 Tiempo de bucle del programa ... 74
5.5 UTILIZACIÓN DE MEMORIA SRAM ... 74
5.6 TAMAÑO DEL PROGRAMA ... 75
5.5.1 Tamaño del programa mínimo ... 76
5.5.2 Número máximo de sensores que pueden incorporarse ... 76
5.7 TASA DE RECOPILACIÓN DE DATOS GPS ... 77
5.8 EVALUACIÓN DE LOS RESULTADOS ... 78
6. APLICACIÓN CASO PARTICULAR: MOTOCICLETA DE COMPETICIÓN ... 80

6.1 DESCRIPCIÓN CASO PARTICULAR 80
6.2 ADQUISICIÓN DE DATOS. SENSORES 81
6.2.1 Sensor digital de efecto Hall AH183 82
6.2.1.1 Especificaciones técnicas AH183 82
6.2.1.2 Código Arduino para utilización del sensor AH183 83
6.2.2 Sensor digital SPI acelerómetro ADXL345 84
6.2.2.1 Especificaciones técnicas ADXL345 84
6.2.2.2 Código Arduino para utilización del sensor ADXL345 86
6.2.3 Sensor digital I^2C barométrico y de temperatura MPL3115A2 87
6.2.3.2 Código Arduino para utilización del sensor MPL3115A2 89
6.2.4 Sensores analógicos para desplazamientos 92
6.2.4.1 Especificaciones técnicas Potenciómetro deslizante Bourns PTE A Series y evaluación 93
6.2.4.2 Especificaciones técnicas Potenciómetro radial Bourns PTV A series y evaluación 94
6.3 TRANSMISIÓN DE DATOS Y GEOLOCALIZACIÓN 95
6.4 DISEÑO DEL ALGORITMO 95
6.4.1 Algoritmo para la monitorización de datos 95
6.4.2 Codificación de los datos 97
6.4.3 Empaquetamiento y envío de los datos 99
6.5 PRUEBAS DE CAMPO DEL PROTOTIPO COMPLETO PARA CASO PARTICULAR 100
6.5.1 Tasa de transmisión de datos 100
6.5.2 Tasa de error de transmisión de datos 101
6.5.3 Tiempo de ejecución del programa 101
6.5.4 Utilización de memoria SRAM 102
6.5.5 Tamaño del programa 102
6.6 PRESUPUESTO 103
6.7 IMÁGNES DEL PROTOTIPO FINAL 104
7. CONCLUSIONES Y LÍNEAS FUTURAS 106
ANEXO 1. CÓDIGO FUENTE SISTEMA DE TELEMETRÍA EN TIEMPO REAL PARA MOTOCICLETA DE COMPETICIÓN 108
BIBLIOGRAFÍA 117

ÍNDICE DE TABLAS Y FIGURAS

1.1 Placas Arduino Uno y Mega ADK ... 12
1.2 Comparativa modelos más comunes ... 14
1.3 Shield 3G+GPS ... 14
1.4 Diagrama de conexiones Shield 3G+GPS ... 15
1.5 Entorno de programación Arduino ... 18
1.6 Cable USB ... 19
1.7 Banco de trabajo Arduino y Shield 3G+GPS ... 20
2.1 Antena de telemetría de un monoplaza ... 23
2.2 Centralita de datos ... 24
2.3 Esquema monoplaza ... 24
2.4 Ejemplo telemetría MotoGP ... 25
2.5 Electrocardiograma obtenido por telemetría ... 27
2.6 Composición Sistema telemetría Álvaro Hernán ... 29
3.1 Mapeado pines Atmega168 ... 32
3.2 Pines digitales Arduino ... 33
3.3 Esquema convertidor analógico/digital ... 34
3.4 Conversión analógico/digital ... 34
3.5 Modulación por ancho de pulso ... 36
3.6 Conexión puerto serie ... 38
3.7 Ejemplo comunicación puerto serie ... 38
3.8 Puerto serie Arduino Uno y Mega ADK ... 38
3.9 Esquema conexión I2C Maestro-Esclavo ... 41
3.10 Pines Arduino comunicación I2C ... 42
3.11 Bus SPI ... 43
3.12 Pines Arduino comunicación SPI ... 44
4.1 Modelo OSI ... 49
4.2 Composición trama protocolo UDP ... 52
4.3 Composición trama protocolo TCP ... 53
4.4 Comandos AT inicialización Shield 3G+GPS ... 58
4.5 Flujograma inicialización Shield 3G+GPS ... 59
4.6 Comandos AT protocol UDP ... 60
4.7 Comandos AT protocolo FTP ... 62
4.8 Comandos AT GPS ... 64
4.9 Trama GPS modo Stand-Alone y Mobile-Based ... 65
4.10 Trama GPS modo Mobile-Assisted ... 65
4.11 Comparativa TCP y UDP ... 67
5.1 Resultados tasa de transmisión protocolo UDP ... 70
5.2 Resultados tasa de transmisión protocolo FTP ... 71
5.3 Resultados tasa de error protocolo UDP y FTP ... 72

5.4 Resultados tiempo de inicialización 73
5.5 Resultados tasa de transmisión protocolo UDP 74
5.6 Resultados utilización de memoria SRAM 75
5.7 Resultados tamaño del programa mínimo 76
5.8 Número máximo de sensores 77
5.9 Resultados tasa de recopilación datos GPS 78
6.1 Sensor AH183 82
6.2 Especificaciones técnicas AH183 83
6.3 Sensor ADXL345 84
6.4 Esquema conexiones ADXL345 85
6.5 Especificaciones técnicas ADXL345 85
6.6 Sensor MPL3115A2 87
6.7 Esquema conexiones MPL3115A2 88
6.8 Esquema pines MPL3115A2 88
6.9 Especificaciones técnicasMPL3115A2 89
6.10 Ejemplo potenciómetro lineal y potenciómetro radial 92
6.11 Especificaciones técnicas Bourns PTV A Series 93
6.12 Curva tensión/desplazamiento Bourns PTV A Series 93
6.13 Especificaciones técnicas Bourns PTE A Series 94
6.14 Curva tensión/rotación Bourns PTE A Series 94
6.15 Magnitudes motocicleta competición 96
6.16 Frecuencias de muestreo sensores motocicleta competición 96
6.17 Flujograma monitorización de datos 97
6.18 Tipos de datos sensores motocicleta de competición 98
6.19 Resultados tasa de transmisión prototipo completo 100
6.20 Comparativa codificación de datos 101
6.21 Resultados tasa de error prototipo completo 101
6.22 Resultados tiempo de ejecución prototipo completo 102
6.23 Resultados utilización de memoria SRAM prototipo completo 102
6.24 Resultados tamaño del programa prototipo completo 102
6.25 Presupuesto sistema telemetría moto competición 103
6.26 Imagen banco de pruebas 104
6.27 Imagen banco de pruebas conectado a PC 104
6.28 Imagen del servidor que procesa y muestra los datos 105

1. INTRODUCCIÓN Y OBJETIVOS

1.1 INTRODUCCIÓN

El tema de este Trabajo de Fin de Grado es el desarrollo de un dispositivo de telemetría y geolocalización en tiempo real basado en la plataforma Arduino, para el cual se ha seleccionado la tarjeta de desarrollo Arduino Mega ADK y el Shield 3G+GPS comercializado por la empresa Cooking Hacks. La telemetría se define como la tecnología que permite la medición remota de magnitudes físicas y químicas, así como el envío de la información generada en la medición hacia el operador del sistema de telemetría, para reportes, controles y toma de decisiones. Por su gran utilidad se ha convertido en una tecnología muy necesaria que se utiliza en infinidad de campos, tales como la exploración científica con naves no tripuladas (satélites, aviones y submarinos), industria (monitorización de procesos y consumo de energía), diversos tipos de competición (Fórmula 1 y MotoGP) e incluso en el campo de la medicina para monitorización de pacientes.

La motivación de este proyecto ha surgido del equipo Moto UPCT, formado por estudiantes de esta universidad y que participa en la competición anual Moto Student, en la que compiten universidades de toda Europa. Uno de sus objetivos para la próxima edición de la competición es la creación de un sistema de telemetría para su moto, y por ello acudieron al Departamento de Electrónica, Tecnología de Computadoras y Proyectos para su desarrollo. Ante la propuesta se decidió realizar un sistema más novedoso que los que se ofrecen en el mercado actualmente, capaz de realizar la medición y envío de los datos en tiempo real, de manera que fuera posible visualizar el comportamiento de la máquina en cada instante de forma remota.

Un sistema de telemetría en tiempo real se puede dividir en varias partes: En primer lugar debe haber un dispositivo que recopile la información de las variables físicas y químicas mediante el uso de sensores, como por ejemplo las revoluciones en una rueda de una moto de competición; debe poseer también un dispositivo de transmisión de datos, que envíe la información recogida de los sensores de forma remota a un servidor mediante un protocolo de comunicación; un servidor con un dispositivo receptor que utilice el mismo protocolo para recibir la información y almacenarla en forma de bases de datos; y por último un software que procese los datos y muestre por pantalla estos datos en forma de gráficas y tablas para su visualización. Ante esta división del sistema en varias partes, el presente Trabajo de Fin de Grado se

centra en la recopilación de datos y posterior envío de los datos a un servidor remoto, dejando las partes restantes como tema para otros proyectos.

Para el desarrollo de este dispositivo de recopilación y envío de datos se ha decidido utilizar Arduino, que es una plataforma de electrónica abierta para la creación de prototipos basada en software y hardware flexibles y fáciles de utilizar. Arduino tiene la ventaja frente a otras plataformas de tener un gran número de *shields* o complementos a su disposición, los cuales permiten implementar un gran abanico de funcionalidades para nuestro dispositivo de manera sencilla. En concreto, para el desarrollo del dispositivo del presente trabajo, se consideró que Arduino es la mejor opción ya que la empresa Cooking Hacks había empezado a comercializar el Shield3G+GPS para Arduino, que le habilita para utilizar redes móviles de alta velocidad como las conocidas 3G y GSM, y cuenta además con GPS interno. Las grandes prestaciones que ofrece esta tecnología para el envío de información (hasta 7.2Mbps de descarga de datos y 5.5Mbps de envío de datos), junto con la posibilidad de geoposicionamiento mediante satélites GPS, hacen que Arduino y el Shield 3G+GPS de Cooking Hacks sea una opción idónea para un prototipo de dispositivo de telemetría en tiempo real por su sencillez y altas prestaciones. Además, el uso de las redes móviles para este tipo de sistemas es una gran novedad con respecto a los dispositivos que se encuentran en estos momentos en el mercado. En el apartado “Estado de la técnica” de este trabajo se profundizará en este tema.

Aunque el trabajo propuesto es independiente y la plataforma a desarrollar se espera tenga validez general, para la evaluación de las prestaciones del sistema de telemetría a desarrollar, se hace necesario diseñar un prototipo adaptado y particularizado para un problema concreto. En nuestro caso, se trabajará en estrecha colaboración con el equipo MotoUPCT, para la implementación del Sistema de Telemetría de la moto con el box de carrera.

1.2 DEFINICIÓN DE OBJETIVOS

Para la realización de este trabajo se ha establecido como propósito principal el desarrollo de un prototipo de dispositivo de recopilación y envío de datos a través de internet, utilizando las plataformas ya descritas, sobre el que realizar un *benchmarking*, es decir, una evaluación de la máxima capacidad de lectura y transmisión de datos del dispositivo. De esta manera queremos comprobar si las especificaciones que ofrece la plataforma Arduino son suficientes para este tipo de sistemas en las que la velocidad de adquisición de datos y velocidad de transmisión son primordiales.

Para la realización de este prototipo es necesario definir una serie de objetivos específicos, estos son:

1- Evaluación y selección de la placa Arduino y de los sensores representativos del problema ejemplo. Se seleccionará entre todas las placas que ofrece Arduino la que mejor se adecúe a las necesidades de nuestro dispositivo y los sensores necesarios para medir las variables físicas del sistema, en nuestro caso una moto de competición.

2- Diseño y programación de la etapa de adquisición de datos. A partir de los sensores seleccionados, se programará una etapa de adquisición de datos utilizando el entorno de programación de Arduino. Será necesario establecer un orden de adquisición de los datos en función de las características de los sensores, como las frecuencias de muestreo, y de la prioridad o importancia que tenga la variable dentro del sistema.

3- Evaluación del protocolo de comunicaciones. Se testearán todos los protocolos de transmisión que permite utilizar el Shield 3G+GPS, estableciendo las ventajas e inconvenientes de cada uno y mostrando los resultados de sus máximas capacidades de transmisión. A partir de esta información se elegirá uno de los protocolos para la transmisión de los datos.

4- Diseño del empaquetamiento y entramado de datos óptimo y programación de la etapa de comunicaciones. Una vez diseñada la etapa de adquisición, y seleccionado el mejor protocolo de transmisión, se diseñará la generación de tramas de información óptimas para el protocolo seleccionado y se programará la etapa de transmisión de datos a través de redes móviles.

5- Pruebas de campo y evaluación de los resultados del prototipo completo. Por último se realizará un *benchmarking* para medir la máxima capacidad del prototipo, estableciendo como criterios principales la velocidad de envío de datos y la capacidad de lectura máxima para la mayor velocidad de transmisión.

1.3 ELEMENTOS QUE COMPONEN EL SISTEMA

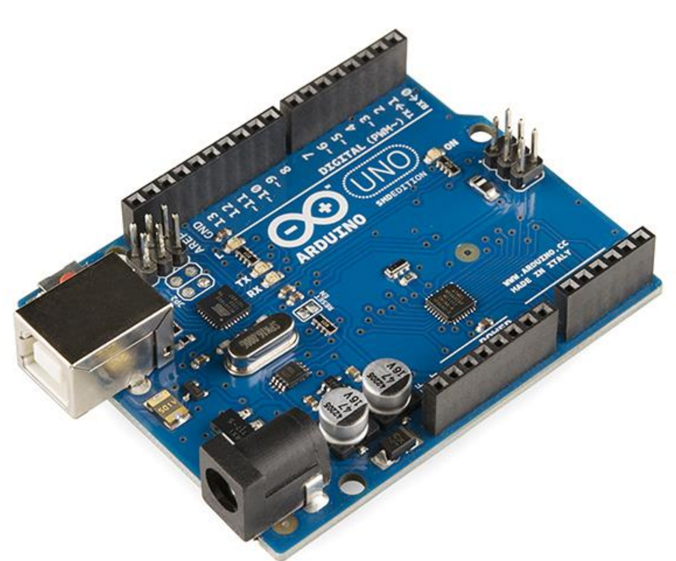
Como se desprende de la descripción hecha de las fases anteriores, una amplia parte del proyecto se ha centrado en buscar aquellos elementos que por sus características parezcan los más convenientes para realizar, lo que en un futuro

constituirá, la arquitectura completa del sistema. Hay que tener en cuenta que estos elementos están en continua evolución.

Los elementos que constituyen el sistema del proyecto son:

1.3.1 Arduino

Consiste en una placa que integra un microcontrolador de la familia ATmega (fabricados por la empresa Atmel Norway), por ejemplo el ATmega 328P-PU que integra la placa de Arduino básica denominada *Uno*. La placa de Arduino además cuenta con una serie de puertos que le permiten comunicarse con otros dispositivos, como un PC, shields o sensores.



1.1 Placas Arduino Uno y Mega ADK

Cada placa de Arduino tiene unas características determinadas, las principales diferencias entre las placas se encuentran en el modelo de microcontrolador y el número de puertos. No obstante, todas tienen una serie de elementos comunes que las diferencian del resto de plataformas de diseño de prototipos electrónicos, estas son [1]:

- Un microcontrolador. Es un circuito integrado programable, capaz de ejecutar las órdenes gradas en su memoria. Un microcontrolador incluye las tres principales unidades funcionales de una computadora: unidad central de procesamiento, memoria y periféricos de entrada/salida.

- Memoria. Todos los modelos poseen un módulo de memoria Flash para guardar en ella los programas. Cuentan también con un módulo de memoria RAM, más rápida pero volátil, que se utiliza durante la ejecución del programa, y una memoria tipo EEPROM que no se borra al apagar el dispositivo. Esto permite guardar datos y no perderlos en caso de fallo o de ser apagado el sistema.

- Alimentación. La placa se puede alimentar a través de varios puertos: El puerto USB, del que recibe 5 Voltios y un conector Jack, del que puede recibir desde 5 hasta 15 Voltios para su funcionamiento. Además, también puede suministrar energía a otros dispositivos a través de sus puertos de alimentación de 5 Voltios y 3.3 Voltios.

- Entradas/Salidas. Todos los modelos de Arduino cuentan con pines de entrada analógica y pines de entrada/salida digital. Algunos de sus pines digitales se pueden utilizar como salidas PWM (modulación por ancho de pulsos) para simular una salida analógica. Estos pines también se pueden utilizar como puertos serie (RS-232), para conexiones i2C/TWI y SPI.

- Comunicaciones. Las placas Arduino poseen al menos un puerto USB a través del cual puede comunicarse con una computadora y un puerto serie para comunicarse con otros dispositivos, como un Shield. También poseen un puerto ICSP para cargar el gestor de arranque (*bootloader*) o programas/firmware.

- Reloj, que puede generar señales de frecuencias desde 16 hasta 20 MHz en el caso de la mayoría de placas, aumentado a 84 MHz en el caso del *Arduino Due* y disminuyendo hasta 8 MHz para los modelos de tamaño reducido, como *Arduino Mini.*

- LEDs de encendido, RX, TX y botón de RESET. Que permiten obtener información visual del estado del dispositivo y de su comunicación, además de poder reiniciar el sistema de manera manual en cualquier momento con el botón RESET.

- Compatibilidad con el entorno de programación de Arduino. Una de las grandes ventajas de estas placas es el software que se utiliza para programarlas, el cual es gratuito y compatible con todas las placas utilizando un mismo lenguaje, basado en *Processing*, siendo portables los códigos entre placas con diferentes microcontroladores. Además, este software se encuentra disponible para todas las plataformas (Windows, Linux , Mac) y destaca por su sencillez.

A continuación se muestra una tabla comparativa de las características comentadas con los modelos más comunes:

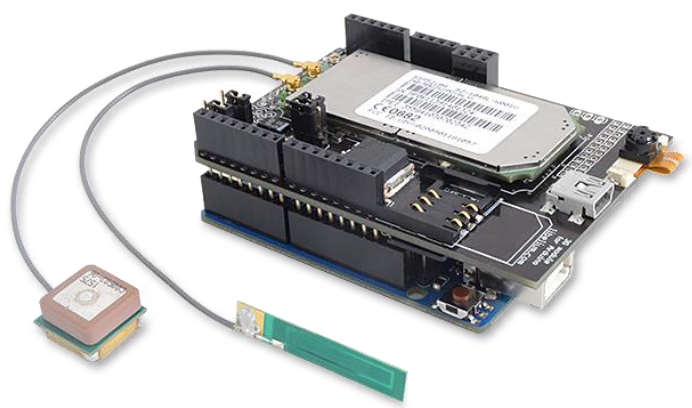
Característica de Arduino	UNO	Mega 2560	Leonardo	DUE
Tipo de microcontrolador	Atmega 328	Atmega 2560	Atmega 32U4	AT91SAM3X8E
Velocidad de reloj	16 MHz	16 MHz	16 MHz	84 MHz
Pines digitales de E/S	14	54	20	54
Entradas analógicas	6	16	12	12
Salidas analógicas	0	0	0	2 (DAC)
Memoria de programa (Flash)	32 Kb	256 Kb	32 Kb	512 Kb
Memoria de datos (SRAM)	2 Kb	8 Kb	2.5 Kb	96 Kb
Memoria auxiliar (EEPROM)	1 Kb	4 Kb	1 Kb	0 Kb

1.2 Comparativa modelos más comunes

1.3.2 Shield 3G+GPS

Los shields de Arduino son placas que a modo de accesorio se pueden conectar a una placa Arduino. Para ello, los pines de sus puertos guardan una disposición de compatibilidad. Existe una gran variedad de shields, con diversa funcionalidad: control de motores, comunicaciones, etc.

Para realizar este proyecto, se va a utilizar el Shield 3G+GPS comercializado por la empresa Cooking Hacks, como ya se ha mencionado anteriormente [10].



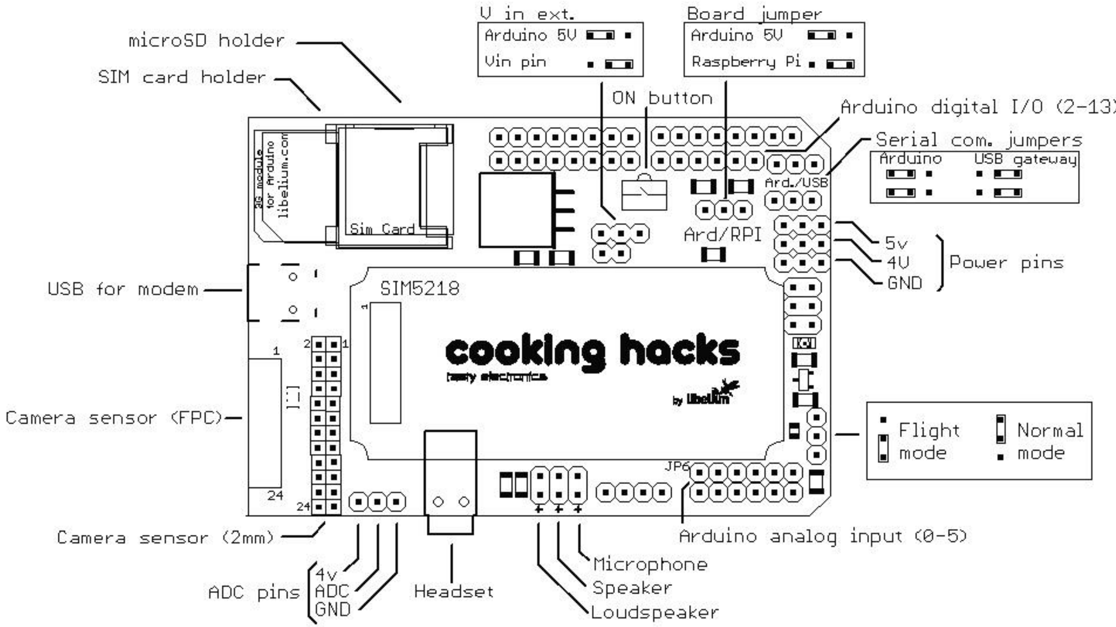
1.3 Shield 3G+GPS

Este Shield está compuesto principalmente por el módem SIM5218, que es un módem de telefonía 3G, que permite conexiones de hasta 7.2Mbps de descarga de datos y 5.76Mbps de subida de datos. El hardware de este módem incluye: UART (*Universal Asynchronous Receiver-Transmiter*, o Transmisor-receptor asíncrono universal), que permite comunicaciones tipo serie, siendo esta la manera en la que se

comunicará con Arduino; Soporte para tarjetas SIM; USB 2.0; GPS interno; conexión para tarjetas SD y soporte para cámara. Este módem permite realizar todas las funciones que se espera de un módem telefónico: Realización de llamadas, mensajes SMS, videollamadas, conexiones de datos a través de internet… Además de la funcionalidad del GPS, que permite utilizar diferentes modos, como el modo Asistido y el modo Stand Alone de los que se hablará más adelante.

Este módem se puede utilizar en Arduino gracias al Shield 3G+GPS, diseñado para que ambos dispositivos se acoplen y puedan comunicarse a través del puerto serie RX-TX y para alimentar el Shield a través de la placa de Arduino directamente. También posee una serie de Jumpers para utilizarlo con distintas configuraciones, como en un Raspberry Pi, con alimentación externa, o directamente con una computadora como un módem 3G USB.

El diagrama de conexiones de este Shield es el siguiente:



1.4 Diagrama de conexiones Shield 3G+GPS

El módulo se puede alimentar como hemos dicho directamente del Arduino, ya que solo requiere entre 3.4V y 4.2 V y una corriente entre 2 A y 3 A. Aunque también está preparado para alimentación externa y a través del puerto USB.

Sus características de comunicación son: SIM5218 es cuatribanda GSM/GPRS/EDGE y UMTS que trabaja a las frecuencias de GSM 850MHz, EGSM 900 MHz, DCS 1800 MHz, PCS1900 MHz, yWCDMA 2100M/1900M/850M.

Las potencias de salida de las señales son:

UMTS 850/900/1900/2100: 0.25W

GSM850/GSM900: 2W

DCS1800/PCS1900: 1W

Otras características de este módem son que implementa los protocolos FTP y FTPS para la gestión de archivos, apertura de puertos TCP y UDP, POP3 y SMTP para el envío de correos electrónicos y el protocolo HTTP, pudiendo obtener y enviar datos de servidores HTTP.

1.3.3 Sensores

Un sensor es un dispositivo capaz de detectar magnitudes físicas o químicas, llamadas variables de instrumentación, y transformarlas en variables eléctricas. Las variables de instrumentación pueden ser por ejemplo: temperatura, intensidad lumínica, distancia, aceleración, inclinación, desplazamiento, presión, fuerza, torsión o humedad. Una magnitud eléctrica puede ser una resistencia eléctrica (como en una RTD), una capacidad eléctrica (como en un sensor de humedad), una Tensión eléctrica (como en un termopar), una corriente eléctrica (como en un fototransistor), etc.

Los sensores son la esencia de la telemetría, gracias a ellos somos capaces de obtener los datos de las variables que queremos medir en nuestro sistema. En el mercado podemos encontrar una gran variedad de sensores, por ello es necesario conocer las características que diferencian a cada uno para poder escoger el sensor más adecuado para medir una variable. Esta decisión suele estar determinada por la naturaleza de la variable, frecuencia de trabajo requerida o precio.

Los sensores se pueden clasificar según la naturaleza de su señal eléctrica de salida en:

-Sensores analógicos. Son aquellos cuya salida es un señal eléctrica analógica, es decir, una tensión que varía con la variable que mide. Para obtener el valor de la variable que miden es necesario aplicar una serie de correlaciones a la tensión de salida, obtenidas experimentalmente y que normalmente aporta el fabricante en forma de curvas.

-Sensores digitales. Son aquellos cuya salida es una señal eléctrica binaria, compuesta por 0 y 1, y que no es necesario transformar. Entre estos sensores se pueden diferenciar el tipo ON y OFF, cuya salida sería booleana (0 o 1), y sensores cuya salida es una variable binaria perfectamente interpretable, como el valor de una temperatura en binario.

Las características que definen a un sensor son:

-Rango de medida: dominio en la magnitud medida en el que puede aplicarse el sensor.

-Precisión: es el error de medida máximo esperado.

-Offset o desviación de cero: valor de la variable de salida cuando la variable de entrada es nula. Si el rango de medida no llega a valores nulos de la variable de entrada, habitualmente se establece otro punto de referencia para definir el offset.

-Linealidad o correlación lineal.

-Resolución: mínima variación de la magnitud de entrada que puede detectarse a la salida.

-Rapidez de respuesta: puede ser un tiempo fijo o depender de cuánto varíe la magnitud a medir. Depende de la capacidad del sistema para seguir las variaciones de la magnitud de entrada.

-Derivas: son otras magnitudes, aparte de la medida como magnitud de entrada, que influyen en la variable de salida. Por ejemplo, pueden ser condiciones ambientales, como la humedad, la temperatura u otras como el envejecimiento (oxidación, desgaste, etc.) del sensor.

-Repetitividad: error esperado al repetir varias veces la misma medida.

1.3.4 Servidor

El servidor es la última de las partes de nuestro sistema. En un sistema de telemetría se define una red formada por dos puntos con una comunicación que puede ser bidireccional entre ellos, un dispositivo que envía información (cliente) a otro dispositivo que recibe la información y la almacena (servidor). Aunque, si en un principio definimos la comunicación de esta manera sería estrictamente unidireccional, decimos que es bidireccional porque la comunicación puede ocurrir en los dos sentidos dependiendo del protocolo de comunicaciones que utilicemos. Por ejemplo, en el caso de utilizar un protocolo basado en UDP (User Datagram Protocol) la comunicación es unidireccional. En cambio, si utilizamos un protocolo basado en TCP (Transmission Control Protocol) la comunicación es bidireccional. Estos conceptos se explicarán con detalle en capítulos posteriores.

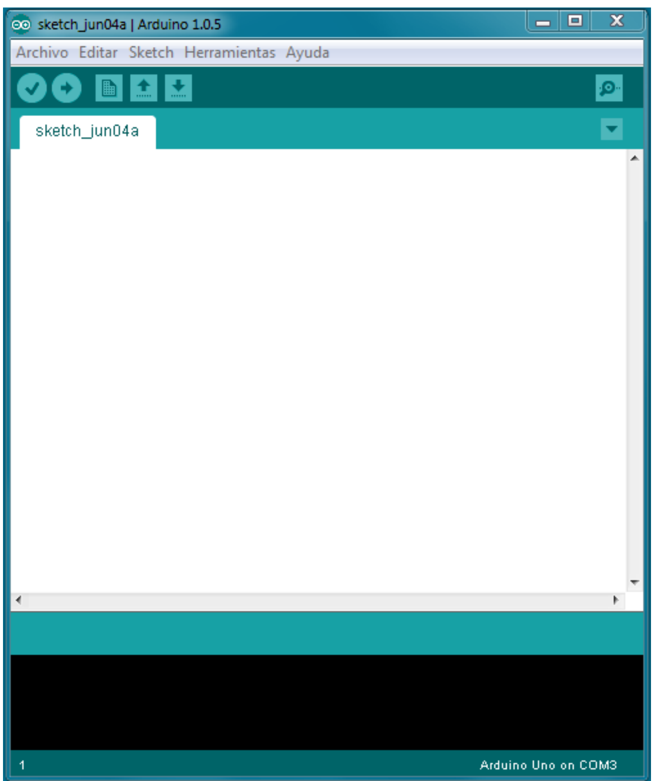
Como servidor para este proyecto se ha utilizado una computadora en el laboratorio del departamento, que está conectada a la red Internet a través del

acceso de la Universidad. La computadora tiene el sistema operativo Windows 7 sobre el que se ejecutan los diversos servidores, que utilizan distintos protocolos con los que se ha trabajado en este proyecto, como un servidor UDP compilado por nosotros, y un conocido servidor FTP, filezilla, que es de uso gratuito y se puede descargar desde su página oficial: (https://filezilla-project.org/).

1.4 ENTORNO DE TRABAJO

En los apartados anteriores se han mencionado distintos elementos particulares que se emplearán a lo largo del proyecto para la consecución de los objetivos planteados. Por ello, este apartado trata de dotar al lector de una visión más clara sobre el entorno global en el que se ha trabajado.

En primer lugar se contará con un PC sobre el que está instalado el sistema Windows 7, que se utilizará para programar el microcontrolador gracias al entorno de programación de Arduino. Este software es gratuito y multiplataforma, y se puede descargar en la página oficial gratuitamente : (http://arduino.cc/en/Main/Software). Aunque Arduino también permite utilizar otro software menos específico para su programación, este ha sido creado especialmente para estas placas.



1.5 Entorno de programación Arduino

Este entorno de programación utiliza su propio lenguaje para las placas Arduino. El lenguaje de programación de Arduino es una implementación de *Wiring*, una plataforma de computación física utilizada para microconroladores, que a su vez se basa en *Processing*, un entorno de programación multimedia. Toda la información sobre el lenguaje de programación Arduino se encuentra disponible en su página web, además cuenta con una gran comunidad a su alrededor de profesionales y aficionados que comparten sus programas y librerías, y que intentan ayudar a resolver problemas y dudas a través del foro oficial: http://forum.arduino.cc/ De esta manera se ha creado una gran comunidad de personas entorno a Arduino que le han impulsado y llevado a ser la plataforma de desarrollo de dispositivos electrónicos más conocida.

Algunas de las características del entorno de desarrollo de Arduino son:

- Editor de textos, en el que se escribe el código fuente del programa. Estos documentos son llamados *sketches* y tienen extensión ".ino". Permite realizar las mismas operaciones que cualquier editor de textos.

- Una consola que nos muestra por pantalla mensajes sobre la compilación y la carga en la placa del código, como por ejemplo mensajes de error o carga completada.

- Un monitor del puerto serie, que es una ventana que muestra los datos que envía la placa de Arduino a través del puerto serie. También posee una función de send para enviar datos por el puerto serie desde la computadora.

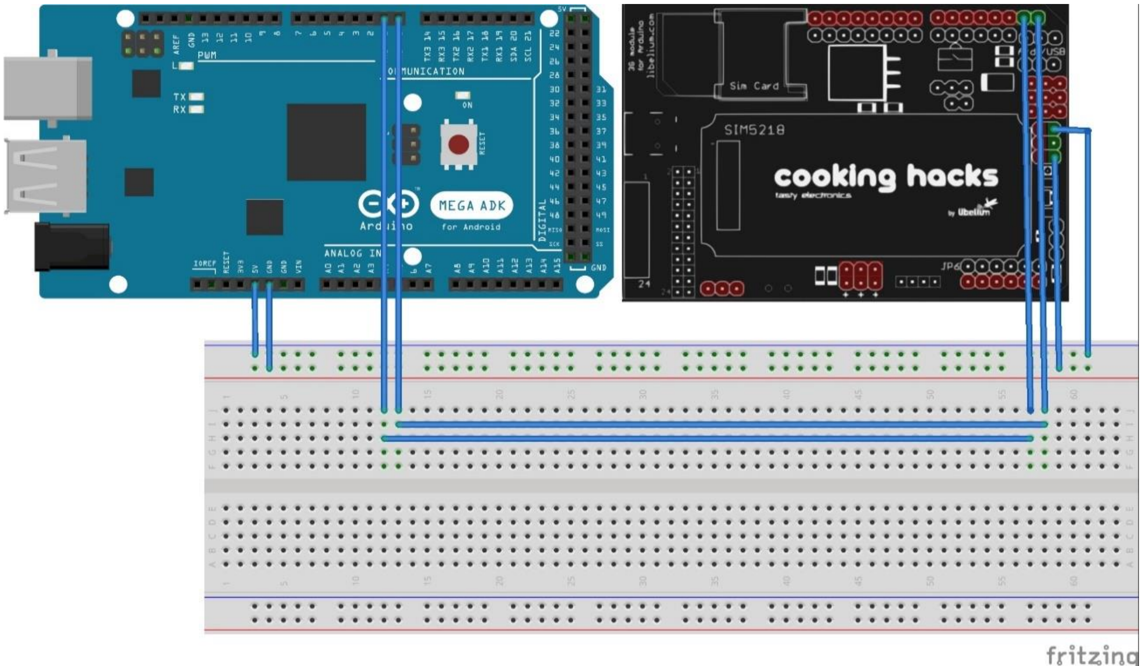
- Ejemplos, programas que utilizan la mayoría de las funciones de este lenguaje y que están a tu disposición para su consulta de manera sencilla y rápida.

- Drivers de todas las placas de Arduino, para una instalación sencilla en nuestra computadora.

A continuación, para cargar los códigos en nuestra placa, solo es necesario conectar Arduino con el PC mediante un cable USB A/B, los drivers para Windows van incluidos con el entorno de programación:

1.6 Cable USB

Para utilizar el último componente, el Shield 3G+GPS, es necesario explicar el siguiente problema: La comunicación entre el Arduino y el shield se realiza a través del puerto serie, y este mismo puerto serie se comparte para comunicar la placa Arduino con el PC. De esta manera si están acoplados Arduino y Shield, y esto se conecta al PC, el Arduino siempre da preferencia al Shield, y no se podrá cargar el código del programa desde el PC. Para solucionar este problema, realizamos el siguiente esquema de conexión, sobre el que montaremos nuestro banco de trabajo:



1.7 Banco de trabajo Arduino y Shield 3G+GPS

En esta figura podemos ver las conexiones que hay que realizar para comunicar ambos dispositivos sin la necesidad de acoplarlos físicamente. En primer lugar, se ha conectado los pines de 5V y GND de Arduino a los correspondientes del Shield para la alimentación. Para la comunicación serie con el Shield se utilizan los pines 0 (RX) y 1 (TX) de la placa Arduino. Para evitar el problema que se ha descrito, utilizamos la configuración de esquema, ya que no es necesario acoplar ambas placas, y con un simple interruptor podemos cerrar la conexión serie entre los pines 0 y 1 y cargar el código fácilmente. De esta manera, para volver activar el Shield solo es necesario volver a activar el interruptor y la comunicación serie entre placa y Shield se restablece. Evidentemente, esta estrategia se ha utilizado únicamente durante la fase de desarrollo, en la que las modificaciones en el código y las pruebas son continuas, para facilitar el proceso de reconexionado. Una vez finalizado el desarrollo, para su funcionamiento en campo, la placa Arduino y el Shield 3G pueden ser acoplados en la forma habitual, lo que reduce el espacio y hace al conjunto más consistente.

1.5 ESTRUCTURA DE LA MEMORIA

La memoria de este Trabajo de Fin de Grado se ha estructurado del siguiente modo:

1. Introducción. En el actual capítulo se explica la motivación de este proyecto y se ha determinado el alcance que debe tener el mismo, así como los pasos que se han llevado para su realización.

2. Estado de la técnica. En este capítulo se lleva a cabo una investigación de la telemetría y de su evolución desde sus inicios hasta la actualidad, profundizando en los sistemas remotos de telemetría. Será útil para comparar la tecnología que existe en el mercado con la que se desarrolla en este trabajo.

3. Métodos de adquisición de datos con Arduino. En primer lugar, se estudian las interfaces de hardware compatibles con la plataforma Arduino con algunos ejemplos que explican cómo utilizarlas. Por último se realiza una evaluación de las placas Arduino disponibles en el mercado para un sistema de telemetría.

4. Shield 3G+GPS. Transmisión de datos a través de internet y geolocalización. En este apartado se evaluarán los protocolos de transmisión a través de internet que permite utilizar el Shield 3G + GPS estableciendo las ventajas e inconvenientes de cada uno. Se estudiará el funcionamiento del Shield 3G+GPS y la forma de comunicación con el dispositivo Arduino, como se utilizan los protocolos descritos, además de la geolocalización mediante GPS en Arduino.

5. Pruebas de campo Arduino Mega ADK Shield 3G+GPS. En este capítulo se muestran los resultados de las pruebas de campo que se han realizado sobre el prototipo diseñado en los capítulos anteriores.

6. Aplicación caso particular: Motocicleta de competición. En este punto se describe la solución concreta para el problema particular de una moto de competición, analizando en profundidad los métodos de adquisición y de recopilación de datos óptimos para este caso.

7. Conclusiones y trabajos futuros. Por último, se analizan los resultados obtenidos con el prototipo y las conclusiones finales obtenidas de dichos resultados. También se comentan las posibilidades en la evolución de este dispositivo para futuros trabajos.

2. ESTADO DE LA TÉCNICA

2.1 INTRODUCCIÓN A LA TELEMETRÍA

La telemetría incluye un conjunto de procedimientos para medir magnitudes físicas y químicas desde una posición distante al lugar donde se producen los fenómenos que queremos analizar y además, abarca el posterior envío de la información hacia el operador del sistema. El término telemetría procede de los términos griegos "*tele*" que significa remoto, y "*metron*", que significa medida. Aunque el termino telemetría se suele aplicar para sistemas remotos sin cables, en algunas bibliografías también se puede encontrar para definir sistemas de transmisión cableados.

Un sistema de telemetría normalmente está constituido por un transductor como dispositivo de entrada, un medio de transmisión en forma de líneas de cable u ondas de radio, dispositivos de procesamiento de señales, y dispositivos de grabación o visualización de datos.

- El dispositivo de entrada se puede distinguir como conjunto de dos partes fundamentales: el sensor, que es el elemento sensible primario que responde a las variaciones de estado de las magnitudes físicas de estudio, y el transductor, que es el que se encarga de convertir el valor de temperatura, presión o vibraciones en la señal eléctrica correspondiente, una vez detectada la variable a seguir, como puede ser la velocidad, las revoluciones del motor, la posición de la moto o el estado de las suspensiones.
- El medio de transmisión puede establecerse de forma guiada por medios como redes de telefonía clásica, redes de ordenadores o enlaces de fibra óptica, o de forma no guiada, por ondas de radio, comunicación por *bluetooth* o *wifi* o incluso por redes de telefonía móvil, que será la más adecuada para nuestra aplicación.
- El dispositivo de procesamiento de la señal está compuesto por un servidor remoto encargado de analizar y transformar los datos, según sea conveniente, para almacenar toda la información en una base de datos interna del propio ordenador.
- Para la visualización de los datos utilizaremos una herramienta software capaz de mostrar automáticamente los valores recogidos en las gráficas pertinentes. Y además, será capaz de representar la posición GPS de la moto en tiempo real.

2.2 APLICACIONES ACTUALES DE LA TELEMETRÍA

A continuación se presentan algunos de los numerosos campos en los que se aplica esta tecnología en la actualidad, como podemos ver se utiliza en una gran variedad de campos:

- **COMPETICIÓN. FÓRMULA1 Y MOTOGP**

En el caso de Fórmula1 los sistemas de telemetría son los sistemas auxiliares más importantes de los que se dispone. Los sistemas que utilizan se basan en ondas microondas en la banda UHF (300MHz-300GHz)y en conexiones punto a punto coche-portátil (PC). En las transmisiones inalámbricas la propagación ha de ser por línea de vista, es decir, que no haya ningún obstáculo sólido entre las antenas, porque las ondas utilizadas no son capaces de superarlos. Por ello se trabaja con envío de información a corta distancia mediante el uso de distintas antenas, aunque cuando el coche pasa lejos de los boxes puede haber pérdida de información. Podrían usarse también ondas de radio, que serían más rápidas, pero también menos fiables y con un menor ancho de banda (y por lo tanto, no podría transmitirse tanta información). Para poder enviar información a corta distancia, a lo largo de todos y cada uno de los circuitos del Mundial existen una serie de antenas repetidoras a las que llegan los datos desde los monoplazas.

Cada monoplaza lleva incorporada una pequeña (y aerodinámica) antena situada en el morro y a más de 10cm de altura, para evitar que la curvatura de la tierra sea un obstáculo más. Es omnidireccional, trabaja a una frecuencia de entre 1,45 y 1,65 GHz, tiene una ganancia de aproximadamente +3 dBi y una potencia de 160W. En la parte trasera del coche también se incorpora una segunda antena unidireccional.

2.1 Antena de telemetría de un monoplaza

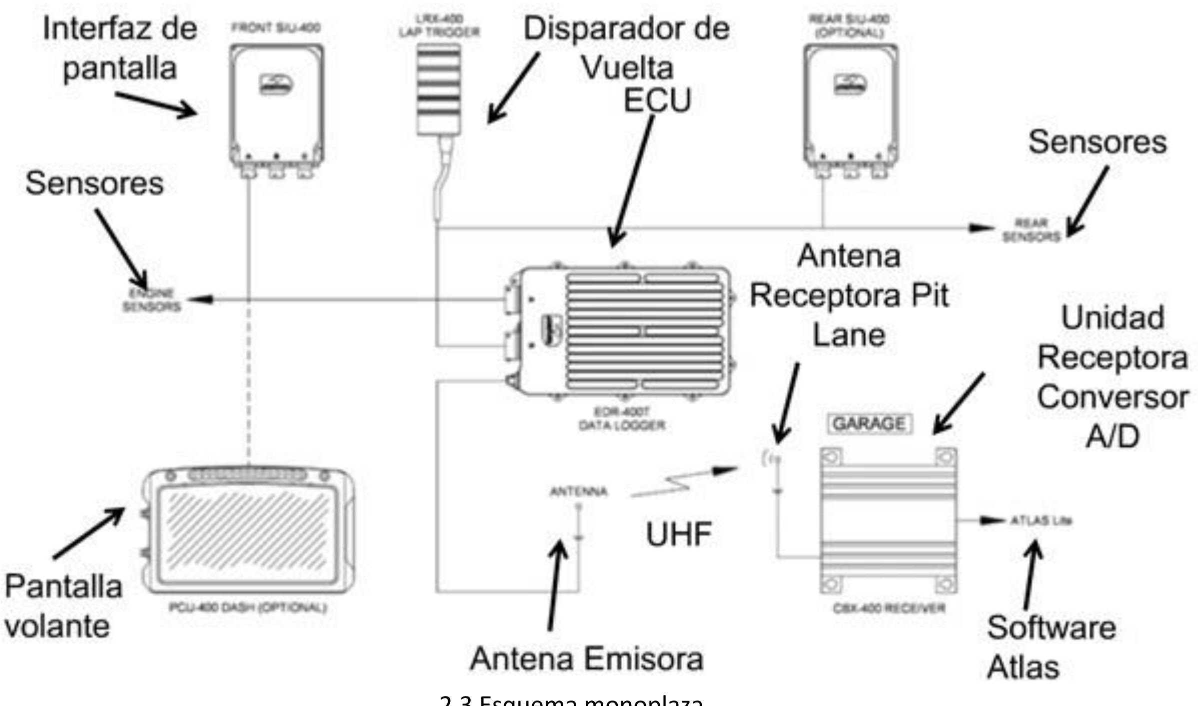
Esta antena base va conectada a una unidad emisora/receptora CBR-610 que actúa como modem y des/encripta la señal con los datos codificados. Cuenta con una tasa de transferencia con picos de hasta 100Mbps. Esta unidad prepara la información

registrada por los sensores de coche de tal forma que pueda gestionarse mediante el potente software 'Atlas', que permite la lectura de los datos mediante complejas gráficas.

2.2 Centralita de datos

Desde la propia "centralita de datos" también se envía la información directamente a la fábrica de la escudería vía satélite, usando antenas parabólicas trabajando en la banda SHF.

Por otro lado, el elemento clave sin el cual no sería posible la telemetría en la Fórmula1 es la ECU (*Electronic Control Unit*). Podríamos decir que es la CPU del monoplaza, que se encarga de recoger todos los datos de los sensores. Es estándar y obligatoria para los 24 coches de la parrilla y está fabricada por la escudería McLaren en colaboración con Microsoft.



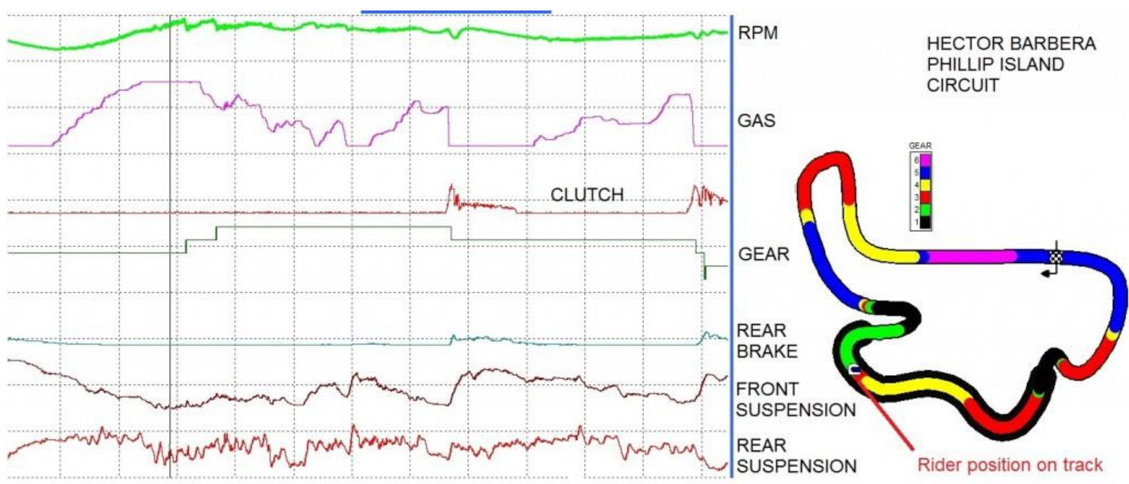
2.3 Esquema monoplaza

La ECU está basada en la arquitectura Power-PC, cuenta con dos procesadores de 40MHz, 1GB de memoria estática, 1MB de memoria flash ROM y 1MB de memoria SRAM. Su tasa máxima de transmisión de datos es de 230Kbps. Los ingenieros usan un cable Ethernet o RS-232 para conectarla con un ordenador portátil y configurarla adecuadamente (aunque está bastante limitada por la normativa de la FIA).

En la competición MotoGP se lleva a cabo un sistema de telemetría totalmente distinto. Por reglamento, en MotoGP la telemetría es offline, es decir, se recaba la información mientras el piloto está en pista, y luego, mediante el software propio de cada centralita, se descarga en el ordenador desde el que se trabajará .A diferencia de la Fórmula1 donde se trabaja en tiempo real, en MotoGP este sistema está prohibido.

En esta competición cada moto descarga toda la información después de cada expedición en pista. La información se transmite a través de canales, por ejemplo la Ducati GP11 puede llegar a tener más de doscientos. Una vez registrada la información, los gráficos resultantes de la misma ayudarán al piloto, junto al analista de datos y al ingeniero de pista, a conocer la situación de su moto y de su propio pilotaje, en cada curva del circuito. Asimismo, podrá averiguar la marcha con la que negocia cada ángulo o la presión que ejerce sobre los frenos en todo momento, para conocer sus límites, y la mejor estrategia para sobrepasarlos sin exponerse a un accidente.

A continuación, podemos ver un gráfico que contempla variables como las revoluciones por minuto, las marchas, las suspensiones. Es un gráfico real de la telemetría del piloto MAPFRE Aspar, Héctor Barberá, en el circuito australiano de Phillip Island.



2.4 Ejemplo telemetría MotoGP

- **DOMÓTICA**

A mediados de los años 90, los automatismos destinados a edificios de oficinas, junto con otros específicos, se comenzaron a aplicar a las viviendas particulares y otro tipo de edificios, dando origen a la vivienda domótica.

La vivienda domótica es un hogar capaz de tomar las decisiones óptimas ante diferentes eventos. Para ello integra una serie de mecanismos en materia de electricidad, electrónica, robótica, informática y telecomunicaciones, con el objetivo de simplificar algunas labores, dotar de mayor confort y seguridad, y aumentar el ahorro energético.

Con la integración de la telemetría en el campo de la domótica, no sólo es posible manipular un equipo de la casa in situ, sino monitorizar y controlar su estado de forma remota en tiempo real. Esta técnica se puede implementar mediante pasarelas residenciales, a través de las cuáles se puede acceder a la red interna de la vivienda, tan sólo disponiendo de un servidor web con conexión a Internet, entre otras herramientas. Esto permite a propietarios controlar y gestionar los componentes de su vivienda desde cualquier lugar.

- **ROBÓTICA**

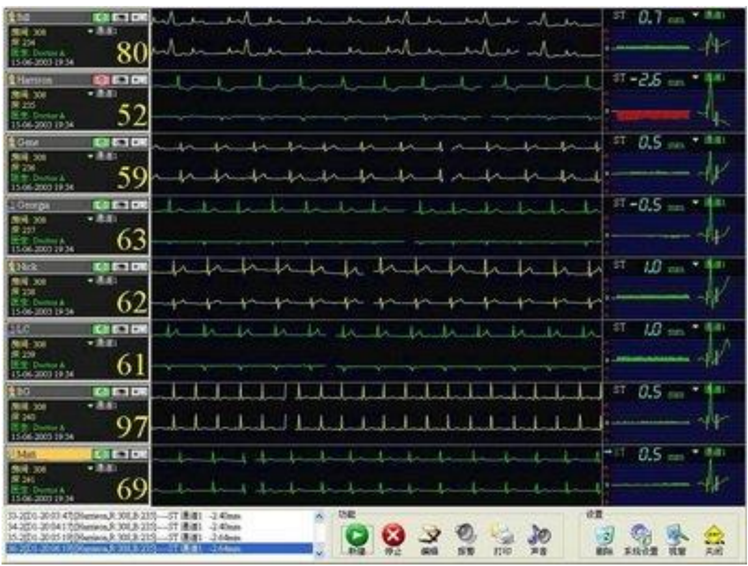
La robótica es una rama de la tecnología que estudia el diseño y construcción de máquinas capaces de desempeñar tareas realizadas por el ser humano, o que requieren del uso de inteligencia. La informática, la electrónica, la mecánica y la ingeniería son sólo algunas de las disciplinas que se combinan para elaborar un sistema robótico.

Estos sistemas pueden realizar tareas como manipular materiales radioactivos, limpiar residuos tóxicos, minería, búsqueda y rescate de personas y localización de minas terrestres. Para ser posible, es necesario recibir información del entorno e interpretarla, para lo que existen técnicas como la telemetría láser. Ésta consiste en lanzar un rayo láser que determina la distancia a la que se encuentran los objetos que lo rodean, lo que le permite desplazarse e interactuar por el mismo.

❖ MEDICINA

En medicina la telemetría es comúnmente usada en numerosas situaciones y utilizando diferentes técnicas, fundamentalmente con el objetivo de tener monitorizadas constantes vitales u otros parámetros de los pacientes de la forma menos invasiva posible, sobre todo cuando el periodo de observación se extiende a días o incluso semanas. En estos casos, los modernos sistemas de telemetría y alarma basada en comunicaciones móviles permiten incluso el seguimiento domiciliario de la evolución de los pacientes crónicos o con largos periodos de convalecencia, que de otra forma deberían permanecer en el entorno hospitalario o realizar continuas visitas, con un control discontinuo de su evolución. La telemetría tiene la ventaja de proporcionar un sistema de monitorización continua, como en el caso de estar hospitalizado, pero con el ahorro de costes y la comodidad para el paciente de estar en su entorno habitual.

Entre otras aplicaciones, una de las más extendidas es el uso de la telemetría para registrar eventos electrocardiográficos a distancia. Los radiotransmisores están conectados al paciente mediante 5 electrodos adheridos a la piel; esto permite a los pacientes libertad para deambular y moverse. El ordenador central refleja los E.C.G. de los pacientes conectados a él y guarda los eventos importantes ocurridos durante las últimas 24 horas. En un estudio realizado por Pérez Titos CB y Oliver Ramos MA, del Hospital Universitario Médico-Quirúrgico «Virgen de las Nieves» de Granada se obtuvo que el 80% de los pacientes estudiados registraron eventos en la Telemetría y el 23%, de éstos, fueron eventos graves.



2.5 Electrocardiograma obtenido por telemetría

2.3 SISTEMAS DE TELEMETRÍA SIMILARES

En este punto se pretende realizar una búsqueda de sistemas telemetría similares que se han diseñado recientemente, con el objetivo de estudiar distintas posibilidades que otros investigadores han tenido en cuenta y establecer una comparación con nuestro sistema, e incorporar si fuera necesario alguna de las características de otros diseños que resulten de especial interés en nuestro caso. Es de señalar que la mayoría de información encontrada se centra en el sector de la automoción..

- **SISTEMA DE ADQUISICIÓN DE DATOS DE UNA MOTOCICLETA DE COMPETICIÓN, Proyecto de Fin de Carrera realizado por GUILLEM BATLER SIQUIER [2]**

En este Proyecto se diseña un dispositivo de adquisición de datos para una motocicleta de competición. Para su diseño se utiliza el microcontrolador PIC 16F887 que almacena los datos en una tarjeta SD. También se centra en la búsqueda de los sensores que mejor se adapten a este sistema midiendo temperatura, revoluciones, frenada… mediante la utilización de una serie de sensores analógicos. Es un proyecto interesante para definir lo parámetros que son importantes para medir y los sensores concretos necesarios para lograrlo.

- **PROTOTIPO DE UN SISTEMA DE TELEMETRÍA Y CONTROL PARA SEGURIDAD EN VEHÍCULOS, SOPORTADO EN REDES MÓVILES, Proyecto de Fin de Grado por ÁLVARO HERNÁN CÁRDENAS VALENCIA [3]**

Para este dispositivo se seleccionó el microcontrolador PIC, junto con el modem Sony Ericsson GM28/29. La parte de telemetría es instalada en un vehículo, que por medio de un modem de telefonía celular GSM, es el encargado de establecer la comunicación y la trasmisión de la información en un mensaje de texto simple (SMS), este va dirigido al número del propietario suministrándole información actual de los estados del mismo. En cuanto a la parte de control: el usuario utiliza un teléfono celular; llama al número del vehículo, por consiguiente el sistema modem contesta automáticamente y establece la comunicación en la cual el usuario puede controlar la alarma por medio del teclado numérico del teléfono, el bloqueo, encender o apagar el vehículo o el aire acondicionado, solicitar información de la temperatura, nivel de la gasolina o el aceite.

El sistema final diseñado se puede ver en la siguiente figura:

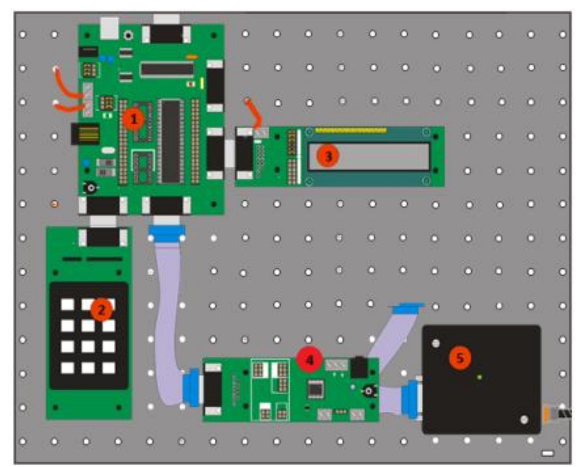
1. Multiprogramador de MCU PICmicro EB006.

2. Board Keypad EB014.

3. Board LCD EB005.

4. Board RS232 EB015.

5. Modem Sony Ericsson GM28



2.6 Composición Sistema telemetría Álvaro Hernán

- **SISTEMA DE TELEMETRÍA EN TIEMPO REAL MEDIANTE REDES MÓVILES GSM PARA EL MONITOREO DE LOS PARÁMETROS BÁSICOS DE UN VEHÍCULO, Proyecto de Fin de Grado por ANGEL DANILO CORNEJO ORTEGA [4].**

El objetivo de este proyecto consiste en el diseño y construcción de un sistema de telemetría, para realizar el monitoreo de los parámetros de presión de aceite, temperatura del líquido refrigerante, velocidad de giro del motor y velocidad de desplazamiento del vehículo, y su respectiva implementación en el vehículo Chevrolet Optra 2008. Los parámetros son adquiridos de las señales de los sensores que posee el auto, de acuerdo al respectivo parámetro; para luego ser procesados y almacenados en el microcontrolador PIC 16F877A y posteriormente enviados por el puerto serie a un modem GSM Motorola C261, el mismo que va a ser el encargado de enviarlo hacia el modem GSM ZTE en formato de mensaje de texto cuando reciba la debida orden.

- **TELEMETRÍA INALÁMBRICA POR RED CELULAR GSM, Proyecto de Fin de Carrera realizado por SONIA CASILLAS [5]**

Este trabajo expone el desarrollo de un sistema de telemetría que usa como medios de transmisión las redes GPRS (*General Packet Radio Service*) e Internet. El sistema tiene dos versiones en la parte correspondiente a la transmisión de las señales: una con la tarjeta Starlert ST-1 y la otra con un teléfono celular. Ambos casos consideran el uso de una computadora para el procesado y/o generación de los datos. Lo que también difiere es la comunicación entre dicha unidad y el transmisor: la primera por medio del puerto serie y la segunda por la tecnología Bluetooth. Se expone también una aplicación en un sistema de ambulancia para la medición de señales ECG, de presión

sanguínea y de temperatura corporal y su posterior transmisión por GSM y recepción en el hospital de destino. A la par de telemetría, el sistema también permite la localización por medio de la red GPS (*Global Position System*), cuya aplicación se muestra con el rastreo de la ambulancia vía Internet.

2.3.1 Discusión

En este punto estamos en disposición de realizar una comparativa de los proyectos similares presentados y el sistema propuesto en este trabajo:

- La tecnología de comunicaciones utilizada en la mayoría de los casos es GSM, en este trabajo se propone utilizar la tecnología 3G, que proporciona un aumento de velocidad de conexión notable.

- Los microcontroladores utilizados en cada sistema son modelos específicos, esto no permite que puedan ser sustituidos por otros similares. En cambio, en este trabajo se propone utilizar Arduino, una plataforma que facilita el desarrollo creando un lenguaje de programación común para todos sus microcontroladores. Esto hace que cualquier programa sea válido e intercambiable entre cada modelo, lo que dota al sistema de la capacidad de ajustarse a las necesidades y el presupuesto.

- Los sistemas descritos son sistemas cerrados, es decir, no permiten la modificación de sus componentes o de sus parámetros de funcionamiento. En este proyecto se presenta un sistema abierto, que utiliza todas las posibilidades que ofrece Arduino (como las interfaces serie, I^2C, SPI…) capaz de adaptarse a las distintas necesidades que se pueden buscar en sistema de telemetría.

3. MÉTODOS DE ADQUISICIÓN DE DATOS CON ARDUINO

3.1 INTERFACES DE HARDWARE COMPATIBLES CON ARDUINO

En este capítulo se trata de estudiar los métodos disponibles para adquirir datos en Arduino. Para ello es necesario un estudio de las interfaces de hardware compatibles con las placas Arduino, además del hardware y el software necesario para conectar una placa Arduino con diferentes dispositivos como sensores o actuadores. Interfaz es un término que procede del vocablo inglés *interface* ("superficie de contacto"). En electrónica se utiliza este término para nombrar la conexión física y funcional entre dos sistemas o dispositivos. La interfaz, es por tanto, una conexión entre dos máquinas, a las cuales se les brinda un soporte para la comunicación entre distintos niveles. Un ejemplo conocido es la interfaz USB (Universal Serial Bus), que permite conectar todo tipo de periféricos a una computadora [6][7].

La idea fundamental en el concepto de interfaz es el de mediación. La interfaz es lo que "media", lo que facilita la comunicación, la interacción, entre dos sistemas de diferente naturaleza. Esto implica, además, que se trata de un sistema de traducción, ya que los dos sistemas "hablan" lenguajes diferentes. Se pueden distinguir dos tipos de interfaces:

- Una interfaz de hardware, a nivel de los dispositivos utilizados para ingresar, procesar y entregar los datos: sensores, actuadores...
- Una interfaz de software, destinada a entregar información acerca de los procesos y herramientas de control, a través de lo que el usuario observa habitualmente en la pantalla.

En nuestro caso, el análisis de las interfaces solo se centrará en las interfaces de hardware, ya que nuestro caso de estudio es un dispositivo de telemetría para el cual no se establece comunicación con una persona de forma directa, como por ejemplo a través de una pantalla. En esta sección no se tratarán los *shields* de Arduino, ya que utilizan las mismas interfaces para comunicarse con Arduino que el resto de dispositivos electrónicos, aunque estén diseñados específicamente para acoplarse con la placa. Por otro lado, cada una de las interfaces que se van a estudiar son independientes de la placa Arduino que utilicemos, ya que todas implementan de forma general estas interfaces y en todas se utiliza el mismo código de programación independientemente

del modelo utilizado. En los siguientes apartados de este capítulo se va a tratar de estudiar las diferentes interfaces de hardware compatibles con Arduino, las cuales son: interfaz analógica, digital, puerto serie, SPI (*Serial Peripheral Interface*) e I^{2}C (*Inter-Integrated Circuit*) y, por último, se hará una evaluación de las placas Arduino para el prototipo.

Para lograr estas comunicaciones, se emplean pines digitales y analógicos, en el siguiente esquema podemos ver el mapeado de los pines del microcontrolador que incorpora el *Arduino Uno, el Atmega 168:*

Atmega168 Pin Mapping

Arduino function		Pin	Pin		Arduino function
reset	(PCINT14/RESET) PC6	1	28	PC5 (ADC5/SCL/PCINT13)	analog input 5
digital pin 0 (RX)	(PCINT16/RXD) PD0	2	27	PC4 (ADC4/SDA/PCINT12)	analog input 4
digital pin 1 (TX)	(PCINT17/TXD) PD1	3	26	PC3 (ADC3/PCINT11)	analog input 3
digital pin 2	(PCINT18/INT0) PD2	4	25	PC2 (ADC2/PCINT10)	analog input 2
digital pin 3 (PWM)	(PCINT19/OC2B/INT1) PD3	5	24	PC1 (ADC1/PCINT9)	analog input 1
digital pin 4	(PCINT20/XCK/T0) PD4	6	23	PC0 (ADC0/PCINT8)	analog input 0
VCC	VCC	7	22	GND	GND
GND	GND	8	21	AREF	analog reference
crystal	(PCINT6/XTAL1/TOSC1) PB6	9	20	AVCC	VCC
crystal	(PCINT7/XTAL2/TOSC2) PB7	10	19	PB5 (SCK/PCINT5)	digital pin 13
digital pin 5 (PWM)	(PCINT21/OC0B/T1) PD5	11	18	PB4 (MISO/PCINT4)	digital pin 12
digital pin 6 (PWM)	(PCINT22/OC0A/AIN0) PD6	12	17	PB3 (MOSI/OC2A/PCINT3)	digital pin 11(PWM)
digital pin 7	(PCINT23/AIN1) PD7	13	16	PB2 (SS/OC1B/PCINT2)	digital pin 10 (PWM)
digital pin 8	(PCINT0/CLKO/ICP1) PB0	14	15	PB1 (OC1A/PCINT1)	digital pin 9 (PWM)

Digital Pins 11,12 & 13 are used by the ICSP header for MOSI, MISO, SCK connections (Atmega168 pins 17,18 & 19). Avoid low-impedance loads on these pins when using the ICSP header.

3.1 Mapeado pines Atmega168

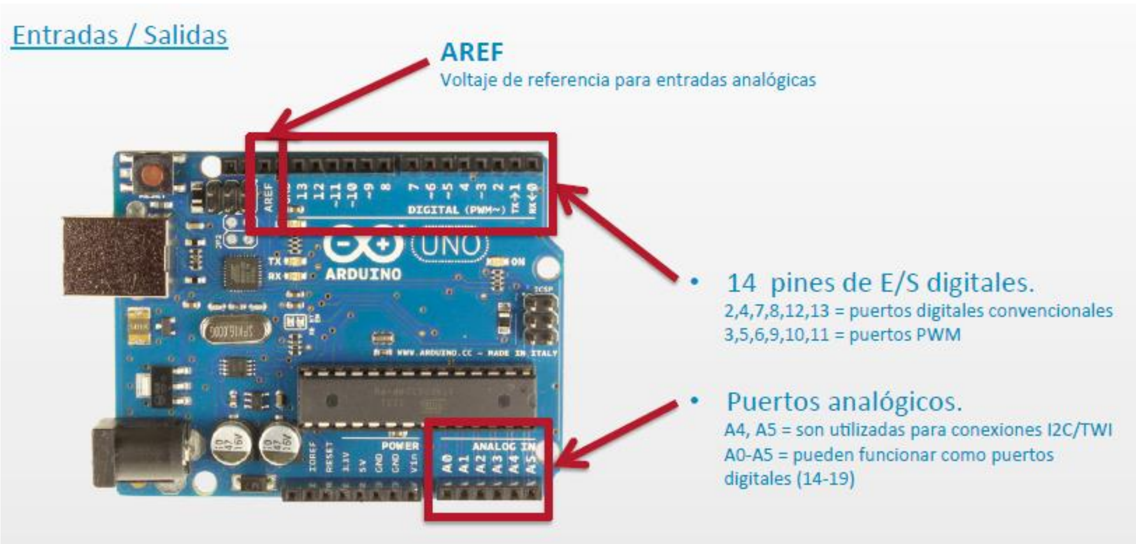
3.2 COMUNICACIÓN MEDIANTE PINES DIGITALES EN ARDUINO

El puerto digital de Arduino está compuesto aquellos pines capaces de recibir y enviar señales digitales a otros dispositivos. Una señal digital es una señal compuesta solo por dos valores “0” y “1” [6][7].

En Arduino la tensión de referencia es 5V, por tanto los valores que sean iguales o superiores a 5V se considerarán un “1” y los que estén por debajo, un “0”.

Los pines de Arduino correspondientes a los pines digitales son aquellos que no llevan prefijo, y comprenden el mayor número del total de pines disponibles. En la

imagen que presentamos a continuación se muestra un ejemplo de la posición de estos pines para la placa *Arduino Uno*



3.2 Pines digitales Arduino

A continuación se muestran las funciones del código Arduino para el uso de estos pines:

-**digitalRead("pin")**. Lee el valor de un pin digital especificado y retorna los valores 0 o 1, que en Arduino también pueden expresarse como HIGH o LOW.

-**digitalWrite("pin", "valor")**. Escribe un valor HIGH o LOW hacia un pin digital. Si el pin ha sido configurado como OUTPUT con pinMode(), su voltaje será establecido al correspondiente valor: 5V para HIGH, 0V (tierra) para LOW.

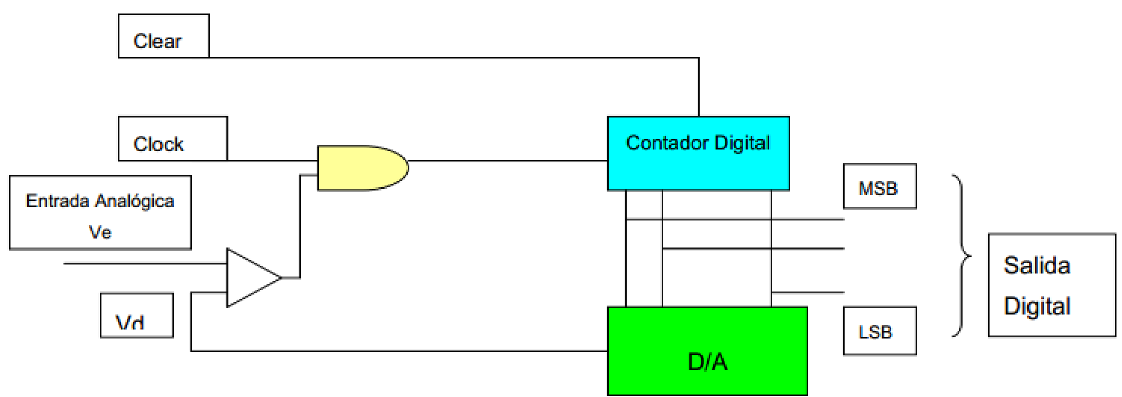
3.2.1 Ejemplo de programa en Arduino para utilización pines digitales

```
int ledPin = 13; // LED conecado al pin digital número 13
int inPin = 7;   // botón (pushbutton) conectado al pin digital número 7
int val = 0;     // variable donde se almacena el valor leído

void setup()
{
  pinMode(ledPin, OUTPUT);      // establece el pin digital número 13 como salida
  pinMode(inPin, INPUT);      // establece el pin digital número 7 como entrada
}

void loop()
{
  val = digitalRead(inPin);   // leer el pin de entrada
  digitalWrite(ledPin, val);    // establece el LED al valor del botón
}
```

3.3 COMUNICACIÓN MEDIANTE PINES ANALÓGICOS EN ARDUINO

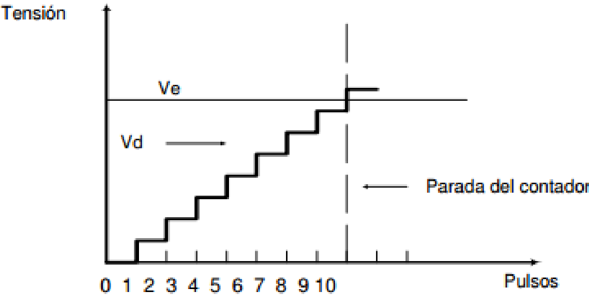
El puerto analógico es aquel que utiliza Arduino para comunicarse mediante señales analógicas con otros dispositivos. Las señalas analógicas son variables eléctricas que evolucionan en el tiempo en forma análoga a alguna variable física, estas variables pueden presentarse en forma de una intensidad de corriente o de una tensión[6][7].

Un ordenador o cualquier sistema de control basado en un microprocesador no puede interpretar señales analógicas, ya que sólo utiliza señales digitales. Es necesario traducir, o transformar en señales binarias, lo que se denomina proceso de digitalización o conversión de señales analógicas a digitales. Existen numerosos métodos para realizar esta conversión, aquí se va a explicar el método que utiliza Arduino, conocido como "convertidor de integración de rampa simple". En la figura siguiente se representa un diagrama en bloques de un conversor A/D de este tipo:



3.3 Esquema convertidor analógico/digital

La línea "clear" se usa para inicializar el contador en 0 (cero). El contador graba en forma binaria el número de pulsos provenientes del "clock". Dado que el número de los pulsos contados aumenta linealmente con el tiempo, la palabra binaria representada al contar es usada en un conversor D/A cuya salida analógica se muestra en el gráfico siguiente:



3.4 Conversión analógico/digital

Esta señal es comparada con la entrada analógica. Mientras la salida del conversor D/A es inferior a la entrada analógica (Ve > Vd) el comparador entrega un uno a la puerta AND que así permite que la señal del “clock” llegue al contador digital. En cuanto la tensión generada en el conversor D/A supera el valor analógico de entrada al comparador (Vd > Ve) éste manda un cero a la puerta AND que detiene la información del “clock”, deteniendo al contador digital. El corte del contador se produce cuando Ve=Vd (o inmediatamente inferior según la escala de error) y este valor es leído a la salida del contador como una palabra que representa en forma digital que representa el valor de la tensión de entrada analógica.

En concreto, el conversor que presentan los microcontroladores ATMega es de 6 canales y tiene una resolución de 10 bits, retornando enteros desde 0 a 1023. En Arduino la tensión de referencia es de 5V. De esta manera una señal de 0V es un valor entero de 0 y una señal de 5V analógica toma un valor de 1023. En la documentación de Arduino podemos ver que el conversor tarda aproximadamente 100 microsegundos (0.0001 segundos) en leer una entrada analógica por lo que se puede llevar una tasa de lectura máxima aproximada de 10.000 lecturas por segundo.

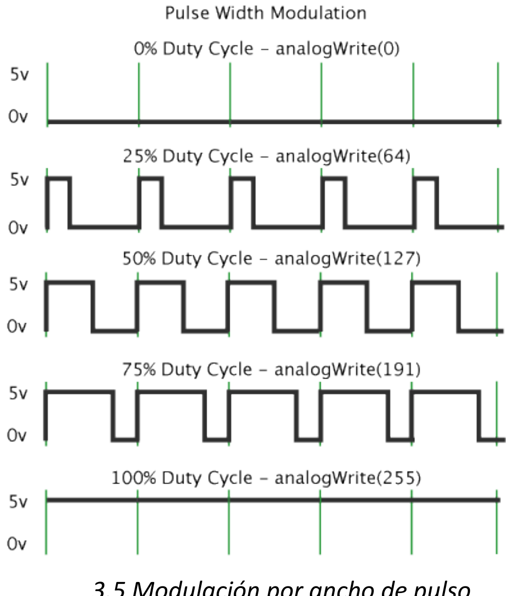
Los puertos analógicos de Arduino se pueden utilizar sólo para leer señales analógicas. Para el proceso contrario, mandar una señal analógica a un dispositivo como un servomotor, se utilizan los puertos PWM, de los que se habla a continuación.

Los pines de Arduino correspondientes a los pines analógicos se nombran con el prefijo “A” (por ejemplo A0) y el número de pines analógicos totales depende del modelo de microcontrolador Atmega y, por tanto, de la placa Arduino que estemos utilizando. Aunque el uso principal de estos pines por los usuarios de Arduino es para la lectura de sensores analógicos, estos pines tienen también toda la funcionalidad de los pines de entrada-salida de propósito general (GPIO), es decir, digitales. Consecuentemente, si un usuario necesita más pines de propósito general de entrada-salida, y no se está usando ningún pin analógico, estos pines pueden usarse como GPIO.

Arduino no dispone de pines de salida analógicas ya que no dispone de un conversor D/A, para realizar este proceso se utilizan los puertos conocidos por PWM. Esta nomenclatura se debe a que utilizan la técnica conocida por el mismo nombre, que en español se traduce en Modulación por ancho de pulso (PWM = *Pulse Width Modulation*). Es una técnica para simular una salida analógica con una salida digital. El control digital se usa para crear una onda cuadrada, una señal que conmuta constantemente entre encendido y apagado. Este patrón de encendido-apagado puede simular voltajes entre 0 (siempre apagado) y 5 voltios (siempre encendido) simplemente variando la proporción de tiempo entre encendido y apagado. A la duración del tiempo de encendido (ON) se le llama Ancho de Pulso (*pulse width*). Para variar el valor analógico cambiamos, o modulamos, ese ancho de pulso. Si repetimos este patrón de

encendido-apagado lo suficientemente rápido por ejemplo con un LED el resultado es como si la señal variara entre 0 y 5 voltios controlando el brillo del LED.

En el gráfico de abajo las líneas verdes representan un periodo regular. Esta duración o periodo es la inversa de la frecuencia del PWM. En otras palabras, con la Arduino la frecuencia PWM es bastante próxima a 500Hz lo que equivale a periodos de 2 milisegundos cada uno. La llamada a la función analogWrite() debe ser en la escala desde 0 a 255, siendo 255 el 100% de ciclo (siempre encendido), el valor 127 será el 50% del ciclo (la mitad del tiempo encendido), etc.



3.5 Modulación por ancho de pulso

Después de llamar a la función correspondiente en el código Arudino para generar la señal analógica, el pin generará una onda cuadrada estable con el ciclo de trabajo especificado hasta que se vuelva a llamar a la función. La frecuencia de la señal PWM es de aproximadamente 490 Hz.

Para utilizar los pines PWM debemos utilizar los puertos digitales que tengan este sufijo, como por ejemplo el pin 3 en *Arduino Uno.* El número de puertos PWM también depende del modelo con el que estemos trabajando.

A continuación se explican las funciones en código Arduino necesarias para utilizar estos pines:

- analogRead("pin"). Lee el valor de tensión en el pin analógico especificado y devuelve un entero (de 0 a 1023).

- analogReference("pin"). Se utiliza para cambiar el valor de referencia o umbral para la conversión A/D.
- analogWrite("pin", "valor"). Escribe un valor analógico (PWM) en un pin. El valor debe estar comprendido entre 0 (siempre apagado) y 255 (siempre encendido).

3.3.1 Ejemplo de programa en Arduino para utilización pines analógicos y PWM

```
int ledPin = 9;      // LED conectado al pin digital 9
int analogPin = 3;   // potenciómetro conectado al pin 3
int val = 0;         // variable en el que se almacena el dato leído

void setup()
{
pinMode(ledPin, OUTPUT);   // indicamos pin como salida
pinMode(analogPin, INPUT); // indicamos pin como entrada
}

void loop()
{
val = analogRead(analogPin);   // lee la tensión en el pin
analogWrite(ledPin, val / 4);  //Escribe este valor en el puerto PWM
}
```

3.4 COMUNICACIÓN SERIE EN ARDUINO

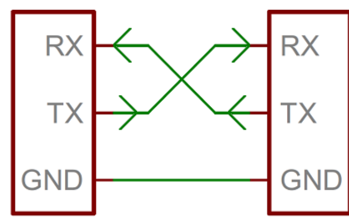
La comunicación serie (o serial) es un protocolo muy común para comunicación entre dispositivos que se incluye de manera estándar en prácticamente cualquier computadora. La comunicación serial es también un protocolo común utilizado por varios dispositivos para instrumentación. Además, la comunicación serial puede ser utilizada para adquisición de datos si se usa en conjunto con un dispositivo remoto de muestreo[6][7].

A nivel físico, la comunicación serie puede establecerse según los protocolos estándares RS-232, RS-422 y RS-485, siendo la primera la que implementa Arduino. RS-232 es la norma más común en este tipo de dispositivos, y se basa en la diferencia de potencial entre dos cables.

El concepto de comunicación serial es sencillo. El puerto serial envía y recibe bytes de información un bit a la vez. Aun y cuando esto es más lento que la comunicación en paralelo, que permite la transmisión de un byte completo por vez, este método de comunicación es más sencillo y puede alcanzar mayores distancias.

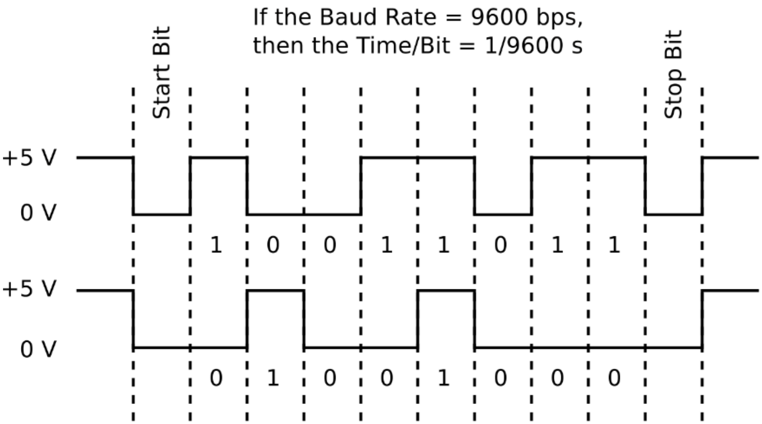
Típicamente, la comunicación serial se utiliza para transmitir datos en formato ASCII. Para realizar la comunicación se utilizan 3 líneas de transmisión: (1) Tierra (o referencia), (TX) Transmitir, (RX) Recibir. Debido a que la transmisión es asíncrona, es posible enviar datos por un línea mientras se reciben datos por otra. Para comunicar dos dispositivos mediante este puerto, solo es necesario conectar el pin RX de uno de ellos

con el puerto TX del otro y viceversa. También es necesario conectar los pines GND (tierra) de cada uno para establecer la misma tensión de referencia.



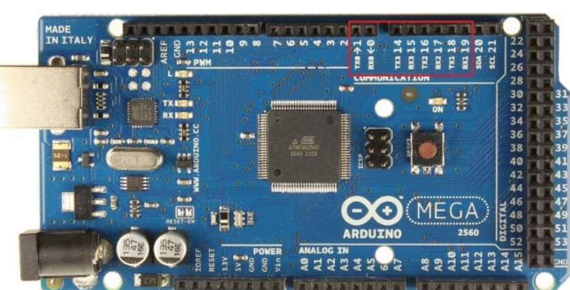
3.6 Conexión puerto serie

A continuación podemos ver un ejemplo de dos tramas enviadas entre dos dispositivos. Ambos dispositivos deben estar configurados ala misma tasa de transmisión de datos (llamada tasa de baudios), en este caso 9600 baudios por segundo. En primer lugar es necasario mandar un bit de comienzo (start), a continuación se envián los datos, y por último un bit de parada(stop), para indicar fin de la transmisión. No se soporta en este caso la tranmisión de bits de paridad para la detección/corrección de errores.



3.7 Ejemplo comunicación puerto serie

En el caso de Arduino, los pines del puerto serie se encuentran en los pines digitales que tienen el sufijo RX o TX. Dependiendo del modelo de la placa, podemos tener entre uno y cuatro puertos serie. Por ejemplo, en el caso de la placa Arduino Uno solo se dispone de un puerto, mientras que en la placa Arduino Mega, se dispone de hasta 4 puertos. En las siguientes imágenes se puede ver su disposición de los pines:



3.8 Puerto serie Arduino Uno y Mega ADK

Para utilizar el puerto serie en Arduino, disponemos de las siguientes funciones del lenguaje Arduino:

- **Serial.begin("bps")**. Establece la velocidad de datos en bits por segundo (baudios) para la transmisión de datos en serie. Para comunicarse con el computador, pueden utilizarse cualquiera de las velocidades definidas por el estándar: 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600 o 115200. Sin embargo, puedes especificar otras velocidades - por ejemplo, para comunicarte a través de los pines 0 y 1 con un componente que requiere una velocidad de transmisión en particular.

- **Serial.end()**. Desactiva la comunicación serie, permitiendo a los pines RX and TX ser usados como entradas o salidas digitales. Para reactivar la comunicación serie, llama al método Serial.begin().

- **Serial.available()**. Devuelve el número de bytes (caracteres) disponibles para ser leídos por el puerto serie. Se refiere a datos ya recibidos y disponibles en el buffer de recepción del puerto (que tiene una capacidad de 128 bytes).

-**Serial.read()**. Lee los datos entrantes del puerto serie.

-**Serial.flush()**. Vacía el búfer de entrada de datos en serie. Es decir, cualquier llamada a Serial.read () o Serial.available () devolverá sólo los datos recibidos después la llamada más reciente a Serial.flush ().

-**Serial.print()**. Imprime los datos al puerto serie como texto ASCII. Este comando puede tomar muchas formas. Los números son impresos mediante un juego de caracteres ASCII para cada dígito. Los valores de tipo "float" son impresos en forma de dígitos ASCII con dos decimales por defecto. Los valores tipo "byte" se envían como un único carácter. Los caracteres y las cadenas se envían tal cual. Por ejemplo:

Serial.print(78) imprime "78"

Serial.print(1.23456) imprime "1.23"

Serial.print(byte(78)) imprime "N" (cuyo código ASCII es 78)

Serial.print('N') imprime "N"

Serial.print("Hello world.") imprime "Hello world."

-**Serial.println("dato")**. Imprime los datos al puerto serie como texto ASCII seguido de un retorno de carro (ASCII 13, o '\r') y un carácter de avance de línea (ASCII 10, o '\n'). Este comando tiene la misma forma que Serial.print ().

-**Serial.write("dato")**. Escribe datos binarios en el puerto serie. Estos datos se envían como un byte o una serie de bytes; para enviar los caracteres que representan los dígitos de un número usar función print() en su lugar.

NOTA1: Si no se especifica puerto, el lenguaje Arduino interpreta que se usa el puerto serie 0. Para tarjetas con más de un puerto serie, especificando el identificador del puerto podemos utilizar todos los puertos. Por ejemplo, para utilizar el puerto serie 1, escribiríamos: Serial1.begin().

NOTA2: La comunicación serie que se establece entre Arduino y el puerto USB de nuestro ordenador siempre se realiza a través del puerto serie "0".

3.4.1 Ejemplo de programa en Arduino para utilización puerto serie

```
// Arduino Mega usando sus 4 puertos serie
// (Serial, Serial1, Serial2, Serial3),
// con diferentes velocidades de datos:

void setup(){
  Serial.begin(9600); //Inicializamos el Puerto serie "0" con una tasa de 9600 bps
  Serial1.begin(38400); //Inicializamos el Puerto serie "1" con una tasa de 38400 bps
  Serial2.begin(19200); //Inicializamos el Puerto serie "2" con una tasa de 19200 bps
  Serial3.begin(4800); //Inicializamos el Puerto serie "3" con una tasa de 4800 bps

  Serial.println("Hola ordenador"); //Escribimos en el Puerto seie 0 "Hola ordenador"
  Serial1.println("Hola Serial 1");  //Escribimos en el Puerto seie 1 "Hola serial1"
  Serial2.println("Hola Serial 2");  //Escribimos en el Puerto seie 2 "Hola serial2"
  Serial3.println("Hola Serial 3");  //Escribimos en el Puerto seie 3 "Hola serial3"
  Serial.end(); //Terminamos la comunicación serial
  Serial1.end();
  Serial2.end();
  Serial3.end();
  Serial.flush(); // Vaciamos el búfer del Puerto serie 0
 }

void loop() {
```

3.5 COMUNICACIÓN SPI EN ARDUINO

El protocolo de comunicación SPI (del inglés Serial Peripheral Interface) es un estándar de comunicaciones, usado principalmente para la transferencia de información entre circuitos integrados en equipos electrónicos. El bus de interfaz de periféricos serie o bus SPI es un estándar para controlar casi cualquier dispositivo electrónico digital que acepte un flujo de bits serie regulado por un reloj[6][7].

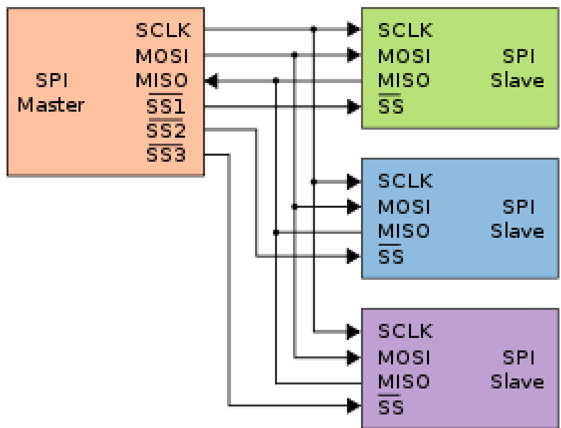
Se define como un protocolo de comunicaciones síncrono, es decir, la sincronización y transmisión de datos es regulada por una señal de reloj. En este protocolo siempre hay un dispositivo maestro (en nuestro caso Arduino), que controla una serie de dispositivos esclavo. En este protocolo se definen típicamente 4 líneas o señales:

-MISO (*Master In Slave Out*). La línea por la cual el esclavo manda datos al maestro.

-MOSI (*Master Out Slave In*). Es la línea por la que el maestro manda datos a los esclavos.

-SCK (*Serial Clock*). Es la señal de reloj que sincroniza la transmisión de datos, y es generada por el maestro.

-SS (*Slave Select*). Es el pin a través del cual el maestro selecciona a un esclavo para comunicarse con él. Cuando el pin SS se encuentra en estado LOW o “0”, se produce la comunicación con el maestro.



3.9 Esquema conexión I2C Maestro-Esclavo

La Cadena de bits es enviada de manera síncrona con los pulsos del reloj, es decir con cada pulso, el Master envía un bit. Para que empiece la transmisión el Master baja la señal SSTE ó SS/Select a cero, con esto el Slave se activa y empieza la transmisión, con un pulso de reloj al mismo tiempo que el primer bit es leído. Nótese que los pulsos de reloj pueden estar programados de manera que la transmisión del bit se realice en 4 modos diferentes, a esto se llama polaridad y fase de la transmisión:

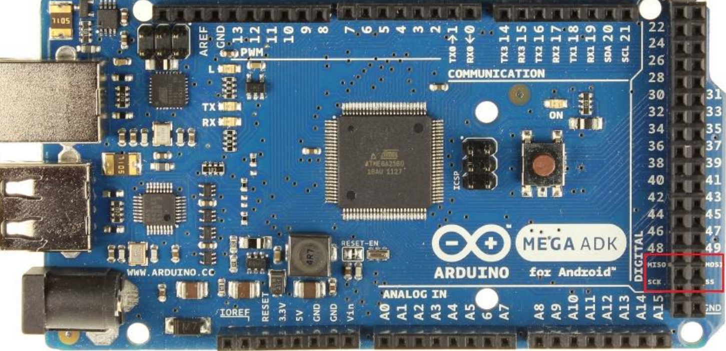
1. Con el flanco de subida sin retraso.

2. Con el flanco de subida con retraso.

3. Con el flanco de bajada sin retraso.

4. Con el flanco de bajada con retraso.

En todos los modelos de Arduino podemos utilizar este protocolo, lo único que cambia entre los modelos es la situación física de estos pines. En el caso de Arduino Uno, estos pines también se pueden utilizar como pines digitales (correspondientes a los pines 11, 12 y 13), mientras que en casos como el de Arduino Mega, existen unos pines dedicados únicamente a SPI. Puesto que pueden utilizarse varios esclavos, Arduino permite utilizar cualquiera de los pines digitales como SS (Slave Select).



3.10 Pines Arduino comunicación I2C

Para utilizar este protocolo con Arduino, el lenguaje de Arduino nos proporciona una librería (SPI.h), con las funciones necesarias para utilizarlo. A continuación se describen estas funciones:

- **SPI.begin()**. Inicializa la comunicación SPI con el dispositivo seleccionado en ese momento mediante el pin SS.

- **SPI.end()**. Finaliza la comunicación SPI con el dispositivo seleccionado en ese momento mediante el pin SS.
- **SPI.setClockDivider("divisor")**. Con este comando se puede configurar la señal de reloj, los divisores que se pueden utilizar son 2, 4, 8, 64 y 128. Por defecto, la señal tiene una frecuencia de 4 MHz.

- **SPI.setDataMode("Modo")**. Configura el modo SPI en cuanto a polaridad y fase. Los modos disponibles son 0, 1, 2 y 3, y se corresponden con los modos indicados anteriormente.

- **SPI.transfer("val")**. Transfiere un byte "val" al dispositivo esclavo seleccionado en ese momento.

Puesto que los programas para la comunicación SPI con otros dispositivos son más complejos que las anteriores, no se mostrará un ejemplo básico. En el capítulo 6 se muestra un programa completo para la utilización de un acelerómetro por SPI, que será más ilustrativo.

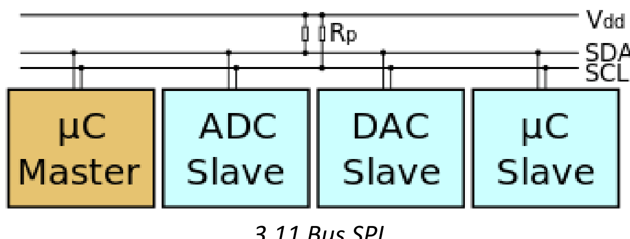
3.6 COMUNICACIÓN I²C EN ARDUINO

I²C (Inter-Intergrated Circuit) es un protocolo de comunicación serie diseñado originalmente por Philips que se utiliza esencialmente entre dispositivos que pertenecen al mismo circuito, por ejemplo, sensores con un microcontrolador. Este bus también es conocido por TWI, esta denominación se utilizaba cuando I2C estaba patentado, pero cayó en desuso ya que las patentes ya han expirado [6][7].

La principal característica de I²C es que utiliza dos líneas para transmitir la información: una para los datos (SDA) y por otra la señal de reloj (SCL). También es necesaria una tercera línea, pero esta sólo es la referencia (masa). Las líneas SDA y SCL están independientemente conectadas a dos resistencias Pull-Up que se encargaran de que el valor lógico siempre sea alto a no ser que un dispositivo lo ponga a valor lógico bajo.

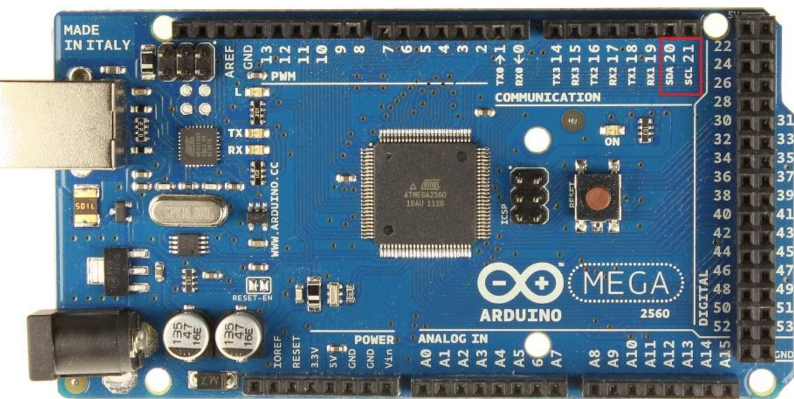
Los dispositivos conectados al bus I²C tienen una dirección única para cada uno, y estos pueden ser maestros o esclavos. El dispositivo maestro inicia la transferencia de datos y además genera la señal de reloj, pero no es necesario que el maestro sea siempre el mismo dispositivo, esta característica se la pueden ir pasando los dispositivos que tengan esa capacidad. Esta característica hace que al bus I²C se le denomine bus multimaestro.

En la siguiente figura podemos ver un ejemplo de conexión de este bus:



3.11 Bus SPI

Para utilizar este bus en Arduino, podemos encontrar en la mayoría de las placas SDA (línea de datos) en el pin analógico 4, y SCL (línea de reloj) está en el pin analógico 5. En el caso de Arduino Mega, SDA se encuentra en el pin digital 20 y SCL en el 21.



3.12 Pines Arduino comunicación SPI

Arduino también nos ofrece una librería denominada "*wire.h*", en la que podemos encontrar todas las funciones necesarias para utilizar este bus. Estas funciones se describen a continuación:

- **Wire.begin("address")**. Esta función inicializa la comunicación I^2C como maestro o esclavo. El parámetro "address" es la dirección asignada al dispositivo. Esta dirección debe ser de 7 bits de esclavo, y si no se especifica, se configura como maestro.

- **Wire.requestFrom("address", "quantity")**. Solicita bytes de otro dispositivo. Los bytes pueden ser recibidos con las funciones available() y receive(). El parámetro "address" es la dirección de 7 bits del dispositivo al que pedir los bytes y "quantity" es el número de bytes a pedir.

- **Wire.beginTransmission("address")**. Comienza una transmisión a un dispositivo I2C esclavo con la dirección dada ("address"). Posteriormente, prepara los bytes a transmitir con la función send() y los transmite llamando a la función endTransmission().

- **Wire.endTransmission()**. Finaliza una transmisión a un esclavo que fue empezada por beginTransmission() y realmente transmite los bytes que fueron preparados por send().

- **Wire.send (“data”)**. Envía datos desde un esclavo en respuesta a una petición de un maestro, o prepara los bytes para transmitir de un maestro a un esclavo (entre llamadas a beginTransmission() y endTransmission()). El parámetro “data” puede ser: un byte para envíar (byte), una cadena de caracteres (o string) para enviar (char *), un vector de datos para envíar (byte *) y en este caso también es necesario enviar el número de bytes de datos para transmitir (byte) de la siguiente manera: Wire.send(data, quantity).

- **Wire.available()**. Devuelve el número de bytes disponibles para recuperar con receive(). Debería ser llamada por un maestro después de llamar a requestFrom() o por un esclavo dentro de la función a ejecutar por onReceive().

- **Wire.onReceive(“handler”)**. Registra una función que será llamada cuando un dispositivo esclavo reciba una transmisión desde un maestro. El parámetro “handler” es la función que será llamada cuando el esclavo recibe datos; esta función debería coger un único parámetro entero (el número de bytes recibidos desde un maestro) y no devolver nada, por ejemplo:
  void myHandler(int numBytes)

- **Wire.onRequest(“handler")**. Registra una función que será llamada por el dispositivo esclavo cuando un maestro solicite datos. El parámetro “handler” es la función a ser llamada, no coge parámetros y no devuelve nada, por ejemplo:
  void myHandler().

NOTA: Hay dos versiones de direccionamiento en I2C 7 y 8 bits. 7 bits identifican al dispositivo, y el octavo bit determina si se está leyendo o escribiendo. La librería Wire usa 7 bits de direccionamiento siempre. Si tenemos un datasheet o un código que usa 8 bits de direccionamiento, tendremos que renunciar al bit más bajo (además debes desplazar el valor un bit a la derecha), cediendo una dirección entre 0 y 127.

Al igual que con el bus SPI, no se mostrará un ejemplo sencillo para esta comunicación, ya que dada su complejidad se entenderá mejor aplicada a un caso concreto de un dispositivo real en el capítulo 6

3.7 COMPARATIVA Y EVALUACIÓN DE LAS PLACAS ARDUINO

En los apartados anteriores hemos visto todas las interfaces de hardware compatibles con Arduino, y en este punto se van a analizar los modelos de Arduino más habituales para elegir el más adecuado para un prototipo de telemetría en función del número del microcontrolador que integran, memoria, número de pines, puertos y precio. A continuación podemos ver una tabla comparativa[6]:

Fabricante	Arduino	Arduino	Arduino	Arduino	Arduino	Arduino	Arduino
Modelo	Pro Mini	Nano	Uno	Mega / Mega 2560	Leonardo	Micro	Due
Microcontrolador	AVR Atmega 168 ó 328 8bits	AVR ATmega 168 ó 328 8bits	AVR ATmega 328 8bits	AVR ATmega2560 8bits	AVR ATmega 32u4 8bits	AVR ATmega 32u4 8bits	ARM SAM3X8E Cortex-M3 32bits
Frecuencia	16Mhz	16Mhz	16Mhz	16Mhz	16Mhz	16Mhz	84Mhz
Memoria RAM	2KiB	2KiB	2KiB	8KiB	2.5KiB	2.5KiB	96KiB (64+32KiB)
Memoria EEPROM	1KiB	1KiB	1KiB	4KiB	1KiB	1KiB	0
Memoria FLASH	16 ó 32KiB	16 ó 32KiB	32KiB	128 ó 256KiB	32KiB	32KiB	512KiB
Pines digitales entradas/salidas	14/14	14/14	14/14	54/54	20/20	20/20	54/54
Tensión/corriente pines digitales	3.3v ó 5v 40mA	5v 40mA	5v 40mA	5v 40mA	5v 40mA	5v 40mA	3.3v 3~15mA (130mA entre todos)
Pines analógicos entradas/salidas	6/0	8/0	6/0	16/0	12/0	12/0	12/2
Tensión/resolución pines analógicos	3.3v ó 5v 10bits (1024 valores)	5v 10bits (1024 valores)	5v 10bits (1024 valores)	5v 10bits (1024 valores)	5v 10bits (1024 valores)	5v 10bits (1024 valores)	3.3v 12bits (4096 valores)
Pines con interrupción externa	2	2	2	6	2	2	-
Pines PWM	6	6	6	15	7	7	12
Conexiones Serial / UART	1	1	1	4	1	1	4
Conexiones I2C / TWI	1	1	1	1	1	1	2
Conexiones ISP / ICSP	1	1	1	1	1	1	1
Conexión USB	No (necesita adaptador externo)	Si	Si, USB-B	Si, USB-B	Si, Nativa, MicroUSB	Si, Nativa, MicroUSB	Si, Nativa, MicroUSB
Conexión USB de depuración	No	No	No	No	No	No	Si, MicroUSB
Conexión Bluetooth	No	No	No	No	No	No	No
Conexión WiFi	No	No	No	No	No	No	No
Conexión Ethernet	No	No	No	No	No	No	No
Conexión USB Host	No	No	No	No	No	No	Si
Almacenamiento por SD	No	No	No	No	No	No	No
Corriente en el pin de 5v	-	500mA	500~800mA	500~800mA	500~800 mA	500mA	800mA
Corriente en el pin de 3.3v	-	50mA	50mA	50mA	50mA	50mA	800mA
Voltaje de alimentación por el USB	3.3v ó 5v (sin usb)	5v	5v	5v	5v	5v	5v
Voltaje de alimentación recomendado por el Jack	3.35 -12 V (modelo 3.3V) ó 5 - 12 V (modelo 5V)	7~12v	7~12v	7~12v	7~12v	7~12v	7~12v
Voltaje de alimentación límite por el Jack	-	6~20v	6~20v	6~20v	6~20v	6~20v	6~20v
Precio oficial	15€	-	20€	40€	18€	18€	39€

En esta tabla podemos ver que los modelos más simples en cuanto a especificaciones son *Arduino Pro Mini, Nano y Micro*, la gama media está formada por *Arduino Uno y Leonardo* y las especificaciones más altas las poseen Arduino Mega y Due, que conforman la gama alta.

En este punto es necesario determinar qué requerimientos debe cumplir nuestra plataforma para elegir la placa de Arduino más apropiada. Puesto que el prototipo que se va a construir en este proyecto es de carácter general y no para una aplicación concreta, es interesante que tenga el mayor número posible de funcionalidades, puertos y pines de los mismos, creando así un dispositivo flexible, que se podrá modificar con un gran margen de disposición de puertos. De esta manera, los modelos más apropiados serían los que pertenecen a la gama alta, *Arduino Mega* y *Due*, ya que ambos disponen del mayor número de pines analógicos (16 en el caso de *Mega* y 12 en el caso de *Due*) y digitales (54 en ambos casos). Además, ambas placas disponen de 4 puertos serie mientras que el resto solo dispone de un único puerto. Esto supone una ventaja, ya que el Shield 3G+GPS utiliza un puerto serie, por lo que podremos conectar hasta tres dispositivos más por este tipo de puerto, cosa que en el resto de placas sería imposible. Por otro lado, todas las placas pueden utilizar los buses I^2C y SPI, lo que no supone ninguna diferenciación. Así, en este proyecto se va a utilizar la placa *Arduino Mega*, aunque *Due* también sería válida siendo el precio similar. Hemos seleccionado *Mega,* porque en primer lugar, *Due* dispone de un microcontrolador de 32 bits, lo que supone mayor complejidad de código ya que requiere librerías de código de Arduino especiales. En segundo lugar, los pines analógicos y digitales en el caso de Mega suministran una tensión de 5V y en el caso de Due de 3.3V, y en muchos de los dispositivos electrónicos como sensores se trabaja a tensiones de 5V.

En el caso de querer realizar una aplicación concreta, fijándonos en los parámetros de los que se ha hablado anteriormente podemos seleccionar la placa que mejor se adapte a nuestra aplicación, pudiendo reducir los costes y tamaño de nuestro dispositivo.

4. SHIELD 3G+GPS. TRANSMISIÓN DE DATOS A TRAVÉS DE INTERNET Y GEOLOCALIZACIÓN

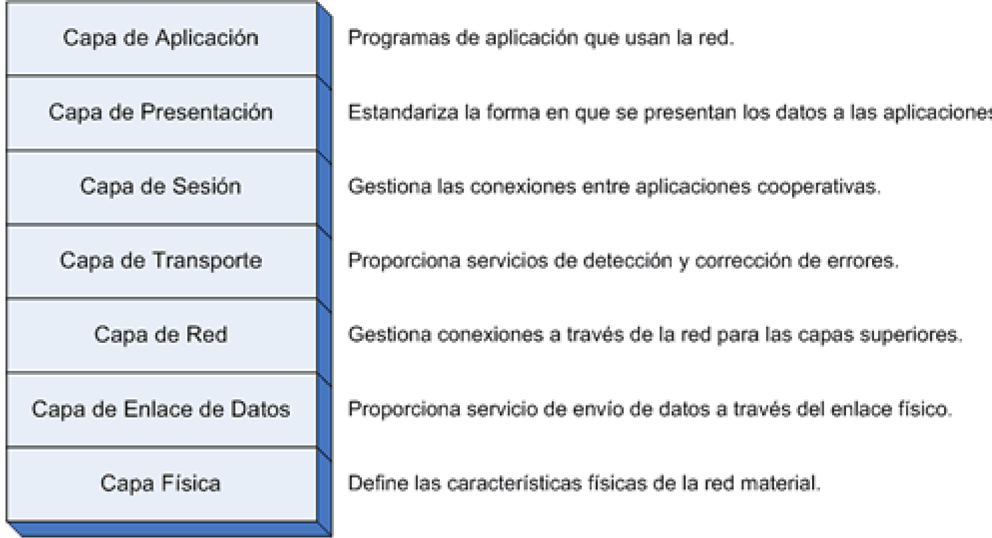
4.1 PRESENTACIÓN SHIELD 3G+GPS. PROTOCOLOS DE INTERNET

En esta sección se presenta un análisis de los protocolos de internet que soporta Shield 3G+GPS para conexiones de datos y métodos de geolocalización a través de satélites GPS. Lo que se pretende es realizar una evaluación de los métodos disponibles para un dispositivo de telemetría en tiempo real.

Previamente a la descripción de los protocolos internet es interesante introducir el Modelo de Referencia OSI (*Open Systems Interconexion*) para entender el funcionamiento de los protocolos. El modelo OSI proporciona una serie de estándares que aseguran la compatibilidad e interoperabilidad entre distintas tecnologías de red. Este modelo se divide en 7 capas, cada una de las cuales ofrece una serie de servicios y se encarga de realizar unas determinadas funciones. Las capas se pueden dividir en dos grupos: por un lado tendríamos los servicios de transporte en los que se encuentran los niveles 1, 2 y 3. Son niveles dependientes de la red de conmutación utilizada para la comunicación entre los 2 sistemas. Por otro lado, están los servicios de soporte al usuario en los que se enmarcan los niveles 5, 6 y 7. Son niveles orientados a la aplicación y realizan funciones directamente vinculadas con los procesos de aplicación que desean comunicarse. El nivel intermedio restante, capa 4, actúa como puente entre cada uno de los dos subgrupos.

La división en capas trae consigo una serie de ventajas: divide la comunicación de red en partes más pequeñas y sencillas, normalizando los componentes de red para permitir el desarrollo y el soporte de los productos de diferentes fabricantes. Permite a los distintos tipos de hardware y software de red comunicarse entre sí de una forma totalmente definida e impide que los cambios en una capa puedan afectar a las demás capas, de manera que se puedan desarrollar con más rapidez. Además, la división en partes más sencillas simplifica el aprendizaje del conjunto.

A continuación se describen las capas del modelo OSI y su función [9]:



4.1 Modelo OSI

1. Capa física: se encarga del medio, de las características del medio y de la forma en que se transmite la información (bits).

2. Capa de enlace de datos: provee acceso a los medios y una transferencia confiable de datos a través de los mismos. Realiza la conexión y selecciona la ruta entre sistemas por medio de un direccionamiento lógico.

3. Capa de red: define el enrutamiento y el envío de paquetes entre redes. Provee una transferencia confiable de datos.

4. Capa de transporte: se encarga de la conexión extremo a extremo ocupándose de aspectos de transporte entre hosts y dotando de confiabilidad al transporte de los datos. Establece, mantiene y termina los circuitos virtuales. Detecta fallas y realiza un control de flujo de la información.

5. Capa de sesión: provee los servicios utilizados para la organización y sincronización del diálogo entre usuarios y el manejo e intercambio de datos. Establece, administra y termina sesiones entre aplicaciones.

6. Capa de presentación: se encarga de la representación de los datos garantizando que sean legibles por parte del sistema receptor. Elige el formato y estructura de los datos y negocia la sintaxis de transferencia de datos para la capa de aplicación.

7. Capa de aplicación: suministra procesos de red a los procesos de aplicaciones, como pueden ser el correo electrónico, la transferencia de archivos o la emulación de terminales.

Se dice que el modelo OSI es un sistema de referencia ya que, en sí mismo, no es considerado una arquitectura al no especificar el protocolo que debe ser usado en cada capa (un modelo de aplicación real es el modelo de Internet, también conocido como modelo TCP/IP al ser éstos los protocolos más importantes que emplea).

A través del modelo OSI podemos caracterizar los protocolos disponibles en el módem, los cuales son[10]:

- TCP (*Transmission Control Protocol*). Según el modelo OSI se encuentra en la capa de transporte (nivel 4).

- UDP (*User Datagram Protocol*). Al igual que el anterior, se encuentra en la capa de transporte (nivel 4) del modelo OSI.

- FTP, FTPS (*File Transfer Protocol*, *File Transfer Protocol Secure*). Se encuentran en la capa de aplicación (nivel 7) según el modelo OSI, utilizando TCP como nivel de transporte. FTPS es similar a FTP pero dispone de funciones de seguridad.

- HTTP, HTTPS (*Hypertext Transfer Protocol*, *Hypertext Transfer Protocol Secure*). Ambos se encuentran en la capa de aplicación (nivel 7) según el modelo OSI y utilizan TCP como nivel de transporte. HTTPS es similar a HTTP pero incluye funciones de seguridad.

De esta manera podemos ver que, con este módem, podemos trabajar tanto a nivel de transporte como a nivel de aplicación. De los protocolos de la capa de aplicación disponibles en este módem, dos de ellos utilizan TCP como protocolo de transporte y ninguno utiliza UDP. Por ello, para trabajar con el protocolo de transporte UDP se hace necesario crear nuestra propia aplicación que tenga la función de servidor y que podamos utilizar desde un PC. En cambio, para FTP y HTTP se puede encontrar software gratuito.

Como las mayores diferencias entre estos protocolos se encuentran en la capa de transporte, es interesante evaluar para este proyecto al menos un protocolo que utilice TCP y otro que utilice UDP. Como se ha dicho anteriormente, para el protocolo de transporte UDP se creará una aplicación específica, y con respecto a TCP, se evaluará FTP por la sencillez de este protocolo y la disponibilidad de servidores gratuitos ya compilados para su uso.

4.2 DESCRIPCIÓN DE LOS PROTOCOLOS DE INTERNET DISPONIBLES EN SHIELD 3G+GPS

4.2.1 Protocolo UDP. Nivel de transporte

User Datagram Protocol (UDP) es un protocolo de nivel de transporte orientado a mensajes, documentado en el RFC 768 de la IETF(*Internet Engineering Task Force*), documento en el que se describen diversos aspectos del funcionamiento de Internet y otras redes de computadoras, como protocolos, procedimientos, etc.

Como ya se ha indicado, es un protocolo del nivel de transporte basado en el intercambio de datagramas (Encapsulado de capa 4 Modelo OSI). Permite el envío de datagramas a través de la red sin que se haya establecido previamente una conexión, ya que el propio datagrama incorpora suficiente información de direccionamiento en su cabecera. Tampoco tiene confirmación ni control de flujo, por lo que los paquetes pueden adelantarse unos a otros; y tampoco se sabe si ha llegado correctamente, ya que no hay confirmación de entrega o recepción. Su uso principal es para protocolos en los que es mayor el tiempo de envío de paquetes que el tiempo de conexión/desconexión, así como para la transmisión de audio y vídeo en tiempo real donde no es posible realizar retransmisiones por los estrictos requisitos de retardo que se tiene en estos casos.

En la familia de protocolos de Internet derivados de UDP se proporciona una sencilla interfaz entre la capa de red y la capa de aplicación. UDP no otorga garantías para la entrega de sus mensajes (por lo que realmente no se debería encontrar en la capa 4) y el origen UDP no retiene estados de los mensajes UDP que han sido enviados a la red. UDP sólo añade multiplexado de aplicación y suma de verificación de la cabecera y la carga útil. Cualquier tipo de garantías para la transmisión de la información deben ser implementadas en capas superiores.

La cabecera UDP consta de 4 campos de los cuales 2 son opcionales (con fondo rojo en la tabla). Los campos de los puertos fuente y destino son campos de 16 bits que identifican el proceso de origen y recepción. Ya que UDP carece de un servidor de estado y el origen UDP no solicita respuestas, el puerto origen es opcional. En caso de no ser utilizado, el puerto origen debe ser puesto a cero. A los campos del puerto destino le sigue un campo obligatorio que indica el tamaño en bytes del datagrama UDP incluidos los datos. El valor mínimo es de 8 bytes. El campo de la cabecera restante es una suma de comprobación de 16 bits que abarca una pseudo-cabecera IP (con las IP origen y destino, el protocolo y la longitud del paquete UDP), la cabecera UDP, los datos y 0's hasta completar un múltiplo de 16. Utilizar *checksum* también es opcional en IPv4, aunque generalmente se utiliza en la práctica (en IPv6 su uso es obligatorio). UDP utiliza

puertos para permitir la comunicación entre aplicaciones. El campo de puerto tiene una longitud de 16 bits, por lo que el rango de valores válidos va desde 0 a 65.535. El puerto 0 está reservado, pero es un valor permitido como puerto origen si el proceso emisor no espera recibir mensajes como respuesta. Entre todos los puertos posibles, los puertos registrados de manera estándar para UDP son los puertos de 1024 a 49151.

bits	0 – 7	8 – 15	16 – 23	24 – 31
0	Dirección Origen			
32	Dirección Destino			
64	Ceros	Protocolo	Longitud UDP	
96	Puerto Origen		Puerto Destino	
128	Longitud del Mensaje		Suma de verificación	
160	Datos			

4.2 Composición trama protocolo UDP

4.2.2 Protocolo TCP. Nivel de transporte

Transmission Control Protocol (TCP), es uno de los protocolos fundamentales de comunicaciones. Fue creado entre los años 1973 y 1974 por Vint Cerf y Robert Kahn y destaca por su implementación en el modelo TCP/IP en el que se basa Internet. Es un protocolo de comunicación orientado a conexión fiable del nivel de transporte (capa 4 del modelo OSI), actualmente documentado por IETF en el RFC 793.

Las conexiones TCP se componen de tres etapas: establecimiento de conexión, transferencia de datos y fin de la conexión. Para establecer la conexión se usa el procedimiento llamado negociación en tres pasos (*3-way handshake*). Para la desconexión se usa una negociación en cuatro pasos (*4-way handshake*). Durante el establecimiento de la conexión, se configuran algunos parámetros tales como el número de secuencia con el fin de asegurar la entrega ordenada de los datos y la robustez de la comunicación.

Los puertos utilizados en TCP son clasificados en tres categorías: bien conocidos, registrados y dinámicos/privados. Los puertos bien conocidos son asignados por la Internet *Assigned Numbers Authority* (IANA), van del 0 al 1023 y son usados normalmente por el sistema o por procesos con privilegios. Las aplicaciones que usan

este tipo de puertos son ejecutadas como servidores y se quedan a la escucha de conexiones. Algunos ejemplos son: FTP (21), SSH (22), Telnet (23), SMTP (25) y HTTP (80). Los puertos registrados son normalmente empleados por las aplicaciones de usuario de forma temporal cuando conectan con los servidores, pero también pueden representar servicios que hayan sido registrados por un tercero (rango de puertos registrados: 1024 al 49151). Los puertos dinámicos/privados también pueden ser usados por las aplicaciones de usuario, pero este caso es menos común. Los puertos dinámicos/privados no tienen significado fuera de la conexión TCP en la que fueron usados (rango de puertos dinámicos/privados: 49152 al 65535

TCP se ocupa de convertir el flujo de datos saliente de una aplicación de forma que se pueda entregar como fragmentos. La aplicación traslada los datos a TCP y éste sitúa los datos en un buffer de envío. TCP toma un trozo de esos datos y le añade una cabecera, creando de esta forma un segmento. Este segmento es trasladado a IP para que lo entregue como un único datagrama. El empaquetado de estos datos en trozos de tamaño adecuado permite usar de una manera eficiente los servicios de transmisión. El formato de la cabecera TCP se detalla a continuación:

<table>
<tr><th>Offsets</th><th>Octeto</th><th colspan="8">0</th><th colspan="8">1</th><th colspan="8">2</th><th colspan="8">3</th></tr>
<tr><th>Octeto</th><th>Bit</th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th><th>16</th><th>17</th><th>18</th><th>19</th><th>20</th><th>21</th><th>22</th><th>23</th><th>24</th><th>25</th><th>26</th><th>27</th><th>28</th><th>29</th><th>30</th><th>31</th></tr>
<tr><td>0</td><td>0</td><td colspan="16">Puerto de origen</td><td colspan="16">Puerto de destino</td></tr>
<tr><td>4</td><td>32</td><td colspan="32">Número de secuencia</td></tr>
<tr><td>8</td><td>64</td><td colspan="32">Número de acuse de recibo (si ACK es establecido)</td></tr>
<tr><td>12</td><td>96</td><td colspan="4">Longitud de Cabecera</td><td colspan="3">Reservado</td><td>NS</td><td>CWR</td><td>ECE</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td><td colspan="16">Tamaño de Ventana</td></tr>
<tr><td>16</td><td>128</td><td colspan="16">Suma de verificación</td><td colspan="16">Puntero urgente (si URG es establecido)</td></tr>
<tr><td>20
...</td><td>160
...</td><td colspan="32">Opciones (Si la Longitud de Cabecera > 5, relleno al final con "0" bytes si es necesario)
...</td></tr>
</table>

4.3 Composición trama protocolo TCP

Los campos de la cabecera son:

- Puerto origen (16 bits): Identifica el puerto emisor.

- Puerto destino (16 bits): Identifica el puerto receptor.

- Número de secuencia (32 bits): Identifica el byte del flujo de datos enviado por el emisor TCP al receptor TCP que representa el primer byte de datos del segmento.

- Número de acuse de recibo (32 bits): Contiene el valor del siguiente número de secuencia que el emisor del segmento espera recibir.

- Longitud de cabecera (4 bits): especifica el tamaño de la cabecera en palabras de 32 bits.

- Reservado (3 bits): para uso futuro. Debe estar a 0.

- Flags (9 bits). Formado por:

- NS (1 bit): *ECN-nonce concealment protection*. Para proteger frente a paquetes accidentales o maliciosos que se aprovechan del control de congestión para ganar ancho de banda de la red.

- CWR (1bit): *Congestion Window Reduced*. El flag se activa por el host emisor para indicar que ha recibido un segmento TCP con el flag ECE activado y ha respondido con el mecanismo de control de congestión.

- ECE (1 bit): Para dar indicaciones sobre congestión.

- URG (1 bit): Indica que el campo del puntero urgente es válido.

- ACK (1 bit): Indica que el campo de asentimiento es válido. Todos los paquetes enviados después del paquete SYN inicial deben tener activo este flag.

- PSH (1 bit): *Push*. El receptor debe pasar los datos a la aplicación tan pronto como sea posible.

- RST (1 bit): *Reset*. Reinicia la conexión.

- SYN (1 bit): *Synchronice*. Sincroniza los números de secuencia para iniciar la conexión.

- FIN (1 bit): El emisor finaliza el envío de datos.

- Tamaño de ventana o ventana de recepción (16 bits): Tamaño de la ventana de recepción que especifica el número máximo de bytes que pueden ser metidos en el buffer de recepción o dicho de otro modo, el número máximo de bytes pendientes de asentimiento. Es un sistema de control de flujo.

- Suma de verificación (16 bits): *Checksum* utilizado para la comprobación de errores tanto en la cabecera como en los datos.

- Puntero urgente (16 bits): Cantidad de bytes desde el número de secuencia que indica el lugar donde acaban los datos urgentes.

- Opciones: Para poder añadir características no cubiertas por la cabecera fija.

- Relleno: Se utiliza para asegurarse que la cabecera acaba con un tamaño múltiplo de 32 bits.

4.2.3 Protocolo FTP. Nivel de aplicación

File Transfer Protocol (FTP) es un protocolo de internet para la transferencia de archivos entre sistemas conectados a través de TCP, basado en la arquitectura cliente-servidor. Desde un equipo cliente se puede conectar a un servidor para descargar archivos desde él o para enviarle archivos, independientemente del sistema operativo utilizado en cada equipo. Un problema básico de FTP es que está pensado para ofrecer la máxima velocidad en la conexión, pero no la máxima seguridad, ya que todo el intercambio de información, desde el *login* y *password* del usuario en el servidor hasta la transferencia de cualquier archivo, se realiza en texto plano sin ningún tipo de cifrado, con lo que un posible atacante puede capturar este tráfico y acceder al servidor. Para solucionar este problema se creó el protocolo FTPS.

La comunicación en este protocolo se origina cuando el cliente FTP envía la petición al servidor para indicarle que requiere establecer una comunicación con él, entonces el cliente FTP inicia la conexión hacia el servidor FTP mediante el puerto 21 el cual establecerá un canal de control. A partir de este punto el cliente FTP enviará al servidor las acciones que este debe ejecutar para poder llevar a cabo el envío de datos. Estas acciones incluyen parámetros para la conexión de datos así como la manera en que serán gestionados y tratados estos datos. Algunos de los parámetros enviados por el cliente FTP para la conexión de datos son: Puerto de datos, modo de transferencia, tipo de representación y estructura. Los parámetros relacionados a la gestión de datos son: Almacenar, recuperar, añadir, borrar y obtener.

El proceso de transferencia de datos desde el servidor hacia el cliente deberá esperar a que el servidor inicie la conexión al puerto de datos especificado (en modo activo) y luego de ello transferir los datos en función a los parámetros de conexión especificados anteriormente.

A diferencia de la mayoría de los protocolos utilizados en Internet, FTP requiere de múltiples puertos de red para funcionar correctamente. Cuando una aplicación cliente FTP inicia una conexión a un servidor FTP, abre el puerto 21 en el servidor — conocido como el puerto de comandos. Se utiliza este puerto para arrojar todos los comandos al servidor. Cualquier petición de datos desde el servidor se devuelve al cliente a través del puerto de datos. El número de puerto para las conexiones de datos y la forma en la que las conexiones son inicializadas varía dependiendo de si el cliente solicita los datos en modo activo o en modo pasivo. A continuación se describen estos modos:

- Modo activo. El modo activo es el método original utilizado por el protocolo FTP para la transferencia de datos a la aplicación cliente. Cuando el cliente FTP inicia una transferencia de datos, el servidor abre una conexión desde el puerto 20 en

el servidor para la dirección IP y un puerto aleatorio sin privilegios (mayor que 1024) especificado por el cliente. Este arreglo implica que la máquina cliente debe poder aceptar conexiones en cualquier puerto superior al 1024. Con el crecimiento de las redes inseguras, tales como Internet, es muy común el uso de cortafuegos para proteger las máquinas cliente. Debido a que estos cortafuegos en el lado del cliente normalmente rechazan las conexiones entrantes desde servidores FTP en modo activo, se creó el modo pasivo.

- Modo pasivo. La aplicación FTP cliente es la que inicia el modo pasivo, de la misma forma que el modo activo. El cliente FTP indica que desea acceder a los datos en modo pasivo y el servidor proporciona la dirección IP y el puerto aleatorio, sin privilegios (mayor que 1024) en el servidor. Luego, el cliente se conecta al puerto en el servidor y descarga la información requerida.

Mientras que el modo pasivo resuelve el problema de la interferencia del cortafuegos en el lado del cliente con las conexiones de datos, también puede complicar la administración del cortafuegos del lado del servidor. Una de las formas de limitar el número de puertos abiertos en el servidor y de simplificar la tarea de crear reglas para el cortafuegos del lado del servidor, es limitando el rango de puertos sin privilegios ofrecidos para las conexiones pasivas.

4.3 UTILIZACIÓN DE LOS PROTOCOLOS DE INTERNET EN SHIELD 3G+GPS

4.3.1 Introducción a los comandos AT

Como se ha dicho anteriormente, la comunicación entre el Shield 3G+GPS y la placa Arduino se realiza a través del puerto serie RX/TX, pero para enviar las órdenes y recibir la información del módem es necesario utilizar unas órdenes concretas. Estas órdenes se conocen como “Comandos AT” o “Comandos Hayes”. Este conjunto de comandos es un lenguaje desarrollado por la compañía *Hayes Communications*, que se convirtió en un estándar abierto de comandos para configurar y parametrizar módems. Los caracteres “AT”, que preceden a todos los comandos, significan “Atención”, e hicieron que se conociera también a este conjunto de comandos como comandos AT.

De esta manera, mediante comandos AT enviados a través del puerto serie, nos comunicaremos con el módem. Existen un gran número de comandos AT, en la página web del fabricante podemos encontrar un manual en el que se detallan todos los comandos AT que son compatibles con el módem SIM5218, corazón del Shield 3G+GPS, así como ejemplos de su uso [8].

En la especificación de los comandos, se determina que deberán ser enviados en mayúsculas. Todos los comandos AT devuelven a través del puerto serie "OK" si se han recibido correctamente. Esto será importante a nivel de programación para comprobar si el módem recibe las órdenes. Los comandos AT que se describen en los siguientes puntos son solo los que han sido necesarios para este proyecto, la lista completa se encuentra en el manual correspondiente.

Para utilizar los comandos AT de manera sencilla, podemos utilizar la siguiente función [10]:

```
int8_t sendATcommand(char* ATcommand, char* expected_answer1,
        unsigned int timeout){
    //inicialiar variables
    uint8_t x=0,  answer=0;
    char response[100];
    unsigned long previous;
    memset(response, '\0', 100);
    //Leer el puerto serie hasta que esté vacio
    while( Serial.available() > 0) Serial.read();
    //Escribir por el puerto serie
    Serial.println(ATcommand);
    x = 0;
    previous = millis();
    //Leer la respuesta del módem por el puerto serie
    do{
        if(Serial.available() != 0){
            response[x] = Serial.read();
            x++;
            if (strstr(response, expected_answer1) != NULL)
            {
                answer = 1;
            }
        }
    //Leer mientras la respuesta no sea nula y no se haya superado timeout
    }while((answer == 0) && ((millis() - previous) < timeout));
    // la función devolve la respuesta del módem
    return answer;
}
```

De esta manera, cuando necesitemos utilizar un comando AT, solo es necesario llamar a esta función indicando el comando AT, la respuesta esperada (suele ser "OK") y un *timeout,*

que es el tiempo que el programa estará esperando una respuesta del módem hasta continuar con al siguiente instrucción en caso de no recibir la respuesta esperada.

4.3.2 Inicialización Shield 3G+GPS

Para iniciar el Shield 3G+GPS es necesario utilizar una serie de comandos AT que se especificarán a continuación. El procedimiento de inicialización del módem empieza por detectar en primer lugar si existe comunicación entre Arduino y Shield y una vez confirmado esto, debemos introducir un número PIN (en caso de estar bloqueada con este sistema) que nos permite acceder a la tarjeta SIM y por tanto, a la red del operador correspondiente. Una vez hecho esto es necesario comprobar si efectivamente, se ha establecido una conexión con la red del operador y por último se establece una dirección APN (*Access Point Name*), para que el módem pueda acceder a la red Internet. Si cualquiera de estos pasos no se supera, no podremos utilizar los protocolos de internet, que es nuestro objetivo. Para utilizar los comandos AT solo es necesario enviar por el puerto serie TX este comando y leer por RX la respuesta del módem. Los comandos necesarios para llevar a cabo esta inicialización son[11]:

Comando AT	Función	Sintaxis	Parámetros
AT	Comprobar comunicación	AT	
AT+CPIN	Desbloquear PIN de la tarjeta SIM	AT+CPIN="pin"	pin: Número PIN de desbloqueo
AT+CREG	Comprobar estado red	AT+CREG	
AT+CGSOCKCONT	Establecer servidor APN	AT+CGSOCKCONT="1", \"IP\",\"APN\"	APN: Dirección operador APN

4.4 Comandos AT inicialización Shield 3G+GPS

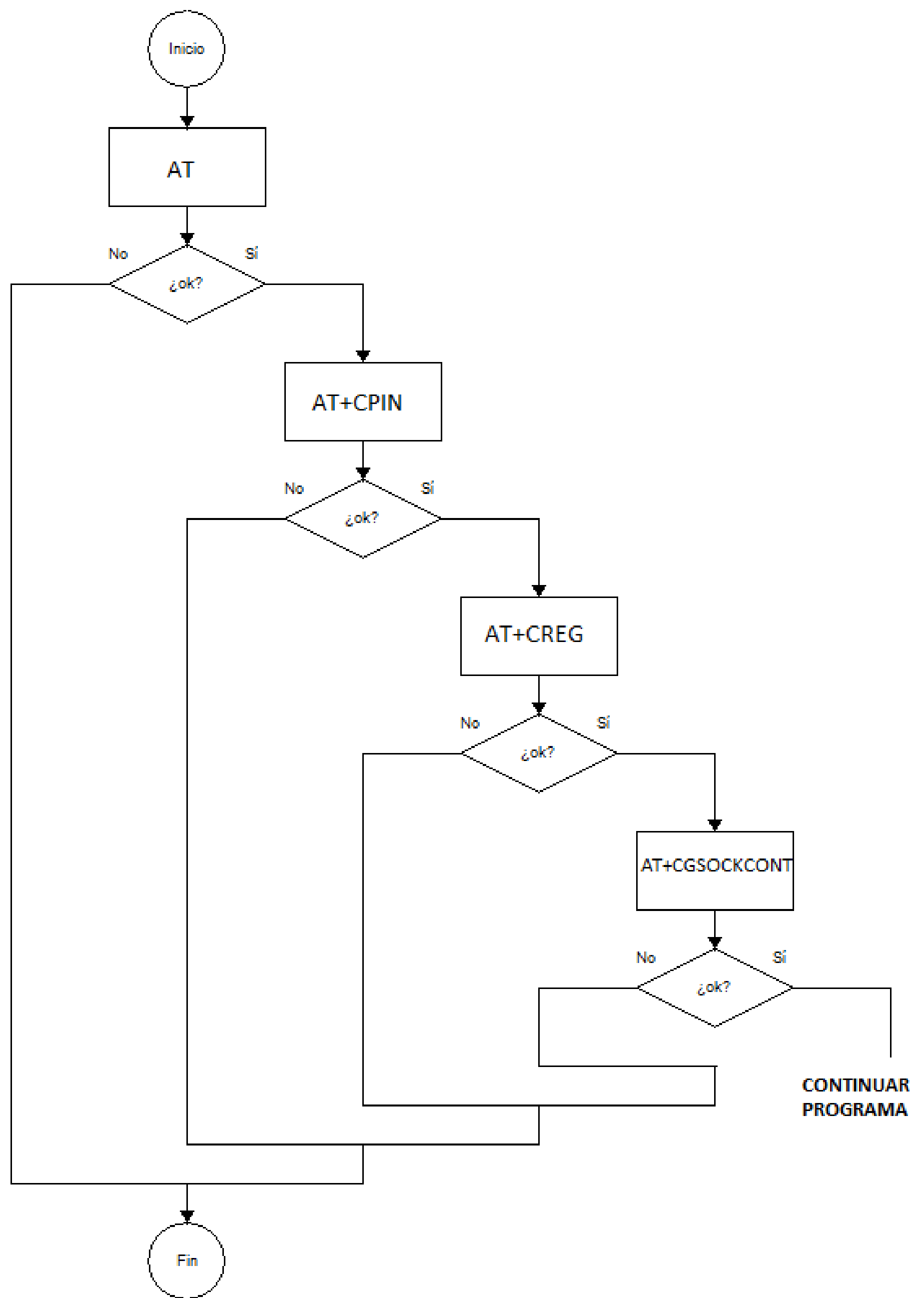
- **AT**. Este es el comando AT más básico, que nos permite comprobar si el módem está comunicándose por el puerto serie.

- **AT+CPIN**. Mediante este comando podemos desbloquear la tarjeta SIM en caso de estar bloqueada por seguridad.

- **AT+CREG**. Este es el comando que nos permite verificar la conexión con la red del operador. Este comando es aconsejable enviarlo mediante un bucle hasta recibir la respuesta esperada, ya que la conexión pude demorarse dependiendo de la cobertura en ese punto.

- **AT+CGSOCKCONT**. Utilizamos este comando para indicar al módem la dirección APN de nuestro operador.

El algoritmo para inicializar el módem mediante el procedimiento descrito es el siguiente presentado en forma de diagrama de flujo:



4.5 Flujograma inicialización Shield 3G+GPS

4.3.2.1 Código Arduino para la inicialización Shield 3G+GPS

```
void setup(){
      Serial.begin(115200);
      //Mandar comando AT hasta verificar comunicación
      while(answer == 0){
            answer = sendATcommand("AT", "OK", 2000);
            }
     //Introducir PIN
     sendATcommand("AT+CPIN=2223", "OK", 2000);
     //Mandar AT+CREG hasta verificar red
     while( (sendATcommand("AT+CREG?", "+CREG: 0,1", 500) ||
             sendATcommand("AT+CREG?", "+CREG: 0,5", 500)) == 0 );
     //Establecer direccion APN
     sendATcommand("AT+CGSOCKCONT=1,\"IP\",\"telefonica.es\"", "OK", 2000);
}
void loop(){
}
```

4.3.3 Protocolo UDP en Shield 3G+GPS

Para poder comunicarnos mediante el protocolo UDP debemos utilizar los siguientes comandos AT[11]:

Comando AT	Función	Sintaxis	Parámetros
AT+NETOPEN	Abrir puerto TCP o UDP	AT+NETOPEN=\”sock_type\”,port	sock_type: tipo de puerto (TCP o UDP) port: puerto
AT+UDPSEND	Enviar una trama a través de UDP	AT+UDPSEND=num,\”IP\”	num: número de caracteres IP: dirección servidor UDP
AT+NETCLOSE	Cerrar puerto TCP o UDP	AT+NETCLOSE=\”sock_type\”,port	sock_type: tipo de puerto (TCP o UDP) port: puerto

4.6 Comandos AT protocol UDP

El procedimiento para la utilización de este protocolo es muy sencillo: En primer lugar es necesario abrir el puerto con el comando AT+NETOPEN indicando tipo UDP y el puerto (este comando solo es necesario utilizarlo una vez), y a continuación debemos utilizar el comando AT+UDPSEND para enviar cada una de las tramas, indicando el tamaño de cada una y la dirección IP que la recibirá. Indicar el tamaño de la trama es importante, ya que la trama se la enviaremos al módem después de recibir "OK" y el módem debe saber exactamente cuántos caracteres pertenecen a la trama, ya que a continuación es posible que se envíe otro comando AT, y si no ha sido enviado el tamaño exacto previamente enviará el comando AT como un mensaje UDP en lugar de interpretarlo y hacer la función correspondiente. En el caso de que fuera necesario cerrar el puerto, se utiliza el comando AT+NETCLOSE.

4.3.3.1 Código Arduino para la utilización del protocolo UDP con Shield 3G+GPS

```
#include <avr/wdt.h>
char aux_str[50];
char server[ ]="212.128.45.104";
char port[ ]="5558";

void setup(){
    Serial.begin(115200);
   while(answer == 0){
            answer = sendATcommand("AT", "OK", 2000);
   }
    sendATcommand("AT+CPIN=2223", "OK", 2000);
    while( (sendATcommand("AT+CREG?", "+CREG: 0,1", 500) ||
            sendATcommand("AT+CREG?", "+CREG: 0,5", 500)) == 0 );
    sendATcommand("AT+CGSOCKCONT=1,\"IP\",\"telefonica.es\"", "OK", 2000);
    sprintf(aux_str, "AT+NETOPEN=\"UDP\",%s", port);//Abrir puerto
    answer = sendATcommand(aux_str, "Network opened", 20000);
    if (answer != 1)//Si el puerto no se ha abierto
    {
      software_Reboot(); //Reiniciar Arduino
    }
}

void loop(){//Enviar trama UDP
    sendATcommand2("AT+UDPSEND=5,\"212.128.45.104\",5558", "OK", 10);
    Serial.println("trama");
    }

void software_Reboot()//Esta función reinicia arduino
{
  wdt_enable(WDTO_15MS);
  while(1)
  {
  }
}
```

4.3.4 Protocolo FTP en Shield 3G+GPS

En la práctica este protocolo se programa de manera similar a UDP, sólo es necesario establecer algunos parámetros más. El orden para utilizarlos es el siguiente: Primero se establece el servidor IP y el puerto con los comandos AT+CFTPSERV y AT+CFTPPORT. A continuación el modo de conexión FTP (pasivo o activo) con AT+CFTPMODE y por último el nombre de usuario y contraseña para acceder al servidor mediante AT+CFTPUN y AT+CFTPPW. Una vez realizado estos pasos correctamente, para enviar un archivo utilizamos el comando AT+CFTPPUT, que creará un archivo en el servidor FTP que contendrá todo lo que se envía a través del puerto serie desde la placa Arduino. En este comando se indica el nombre del archivo, si por ejemplo indicamos como nombre "*file.txt*", creará un archivo de texto [11].

Comando AT	Función	Sintaxis	Parámetros
AT+CFTPSERV	Establecer dirección IP del servidor FTP	T+CFTPSERV=\"IP\"	IP: dirección IP del servidor
AT+CFTPPORT	Establecer puerto del servidor FTP	AT+CFTPPORT=port	port: puerto del servidor
AT+CFTPMODEAT	Establecer modo activo o pasivo	AT+CFTPMODE=mode	mode: 1 (pasivo), 0 (activo)
AT+CFTPUN	Establecer nombre de usuario para acceder al servidor	AT+CFTPUN=\"user\"	user: nombre de usuario
AT+CFTPPW	Establecer contraseña para acceder al servidor	AT+CFTPPW=\"pass\"	pass: contraseña del usuario
AT+CFTPPUT	Enviar un archivo al servidor FTP utilizando los datos recibidos del puerto serie	AT+CFTPPUT=\"file\"	file: nombre del archivo

4.7 Comandos AT protocolo FTP

4.3.3.1 Código Arduino para la utilización del protocolo FTP con Shield 3G+GPS

```
#include <avr/wdt.h>
char aux_str[150];
char file_name[ ]="test.txt";

void setup(){
    Serial.begin(115200);
    while(answer == 0){
            answer = sendATcommand("AT", "OK", 2000);  }
    sendATcommand("AT+CPIN=2223", "OK", 2000);
    while( (sendATcommand("AT+CREG?", "+CREG: 0,1", 500) ||
        sendATcommand("AT+CREG?", "+CREG: 0,5", 500)) == 0 );
    sendATcommand("AT+CGSOCKCONT=1,\"IP\",\"telefonica.es\"", "OK", 20000);
    sendATcommand("AT+NETOPEN=\"TCP\",6669", "OK", 20000);//Abrir puerto
    sendATcommand("AT+CFTPSERV=\"212.128.45.104\"", "OK", 20000);//Servidor
    sendATcommand("AT+CFTPPORT=6669", "OK", 20000);//Puerto
    sendATcommand("AT+CFTPMODE=1", "OK", 20000);//Modo FTP
    sendATcommand("AT+CFTPUN=\"user\"", "OK", 20000);//Usuario
    sendATcommand("AT+CFTPPW=\"1234\"", "OK", 20000);//Contraseña
    sprintf(aux_str, "AT+CFTPPUT=\"%s\"", file_name);//Archivo
    answer = sendATcommand(aux_str, "+CFTPPUT: BEGIN", 100000);
    if (answer != 1)
    {
       software_Reboot();//Reiniciar Arduino
    }
}
void loop()
{
        Serial.println("trama");

}
```

4.4 GEOLOCALIZACIÓN MEDIANTE SATÉLITES GPS

El módem SIM5218 incorporado en el Shield 3G+GPS dispone de un módulo GPS que permite la obtener los datos de geoposicionamiento y hora utilizando lo satélites GPS. Para obtener esta información, el módem nos ofrece tres modos de conexión, que se describen a continuación:

- *Stand-alone mode.* En este modo, solo se utilizan las señales obtenidas de los satélites GPS, sin utilizar servidores de internet, lo que hace que este modo sea el menos rápido y preciso.

- *Mobile-based mode (MB).* Este modo sí utiliza información de servidores de internet, pero es el módulo GPS el que realiza los cálculos de posicionamiento.

- *Mobile-assisted mode (MA).* Este el modo más rápido y preciso, ya que obtiene de los servidores de internet la posición. El sistema de GPS asistido utilizará los datos obtenidos y los combinará con la información de la celda o antena de telefonía móvil para conocer la posición y saber qué satélites puede utilizar. Todos estos datos de los satélites están almacenados en el servidor externo o en el fichero descargado, y según nuestra posición dada por la red de telefonía, el GPS dispondrá de los datos de unos satélites u otros y completará a los que esté recibiendo a través del receptor convencional de GPS, de manera que la puesta en marcha de la navegación es notablemente más rápida y precisa.

Los comandos AT de que disponemos para utilizar el módulo GPS son los siguientes[11]:

Comando AT	Función	Sintaxis	Parámetros
AT+CGPS	Establecer modo GPS	AT+CGPS=modo	modo: 1,1 (modo stand-alone); 1,2 (modo mobile-based); 1,3 (modo mobile assisted)
AT+CGPSURL	Establecer servidor de GPS asistido	AT+CGPSURL=\"direccion\"	direccion: formada por servidor:puerto
AT+CGPSSSL	Establecer certificado seguridad GPS	AT+CGPSSSL= SSL	SSL: 0 (sin certificado), 1 (con certificado)
AT+CGPSINFO	Obtener datos de geoposicionamiento	AT+CGPSINFO	

4.8 Comandos AT GPS

El procedimiento para obtener las geoposición es el siguiente: Primero fijamos de método de obtención de los datos GPS con AT+CGPS. Después, si hemos elegido el modo *Mobile-Assisted* o *Mobile-Based*, debemos indicar la dirección del servidor de GPS asistido con AT+CGPSURL y el certificado de seguridad con AT+CGPSSSL. Una vez fijados estos parámetros, para obtener los datos utilizamos el comando AT+CGPSINFO. El módem enviará “+CGPSINFO:” cuando reciba este comando y a continuación la trama con la información. El formato de las tramas cambia según el modo que utilicemos:

- Si utilizamos los modos *Stand-Alone* o *Mobile-Based*, la información que obtendremos vendrá codificada de la siguiente manera[10]:

[<latitud>],[<N/S>],[<longitud>],[<E/W>],[<fecha>],[<hora_servidor>],[<altitud>],[<velocidad>],[<rumbo>]

Parámetro	Formato	Ejemplo
Latitud	ddmm.mmmmmm d: grados; m: minutos	4140.831527
Longitud	dddmm.mmmmmm d: grados; m: minutos	00053.173495
Fecha	ddmmyy d: dia; m: mes; y: año	020812
Hora_servidor	hhmmss.s h: hora; m: minutos; s: segundos	083418.0
Altitud	Metros	257.00
Velocidad	nudos	2
Rumbo	grados	0

4.9 Trama GPS modo Stand-Alone y Mobile-Based

- En cambio, en modo *Mobile-Assisted*, obtendremos la siguiente trama:

[<latitud>],[<longitud>],[<altitud>],[<fecha>],[<hora_servidor>]

Parámetro	Formato	Ejemplo
Latitud	Grados en 10^8	4168050885
Longitud	Grados en 10^8	88618039
Fecha	ddmmyy d: dia; m: mes; y: año	28092012
Hora_servidor	hhmmss.s h: hora; m: minutos; s: segundos	081611.0
Altitud	Metros	293

4.10 Trama GPS modo Mobile-Assisted

4.4.1 Código Arduino para la utilización del módulo GPS

El código que se muestra a continuación es un ejemplo en el que se utiliza el modo Mobile-Based para obtener la información del GPS. En el caso de utilizar cualquiera de los otros modos de geolocalización GPS disponibles, solo es necesario modificar el comando “AT+CPGPS”, y en el caso de utilizar *Stand-Alone mode,* eliminar los correspondientes comandos para el servidor GPS de internet y de certificado de seguridad.

```
int8_t answer;
char gps_data[50];//Almacenar coordenadas GPS
int counter;

void setup(){
   while(answer == 0){
            answer = sendATcommand("AT", "OK", 2000);//Verificar comunicación
        }
    sendATcommand("AT+CPIN=2223", "OK", 2000);//PIN
    while( (sendATcommand("AT+CREG?", "+CREG: 0,1", 500) ||
   //Verificar Red
    sendATcommand("AT+CREG?", "+CREG: 0,5", 500)) == 0 );
    sendATcommand("AT+CGSOCKCONT=1,\"IP\",\"telefonica.es\"", "OK", 2000);//APN
    sendATcommand("AT+CGPSURL=\"supl.google.com:7276\"","OK",1000);//Servidor GPS Asistido
    sendATcommand("AT+CGPSSSL=0","OK",1000); //Sin certificado
    answer = sendATcommand("AT+CGPS=1,2","OK",1000); //Modo Mobile-Based
    if (answer == 0)
    {
        while(1);//No continuar si no inicia
    }
}
void loop(){
    answer = sendATcommand("AT+CGPSINFO","+CGPSINFO:",1000);//Peticion informacion GPS
    if (answer == 1)//Si la respuesta es afirmativa
    {
        counter = 0;//Leer puerto serial
        do{
            while(Serial.available() == 0);
            gps_data[counter] = Serial.read();
            counter++;
        }
        while(gps_data[counter - 1] != '\r');
        gps_data[counter] = '\0';
        if(gps_data[0] == ',')//Si no se ha recibido nada
        {
             Serial.println("No GPS data available");
        }//Si se han recibido las coordenadas
        else//Enviar por el puerto serie las coordenadas para visualizarlas
        {
            Serial.println(gps_data);
        }
    }
    else//Caso de respuesta negativa
    {
        Serial.println("Error");
    }
}
```

4.5 EVALUACIÓN DE LOS PROTOCOLOS DE INTERNET PARA UN SISTEMA DE TELEMETRÍA EN TIEMPO REAL

Una vez analizados los protocolos que podemos utilizar estamos en disposición de realizar una evaluación para averiguar cuál de ellos es el más eficaz para un dispositivo de telemetría en tiempo real. En la siguiente tabla se muestra una tabla comparativa con las principales diferencias entre los dos protocolos a evaluar [9]:

TCP	UDP
- Orientado a la conexión - Confiabilidad en la entrega de mensajes - Divide los mensajes en datagramas - Hace seguimiento del orden (o secuencia) - Usa *checksum* para la detección de errores - El tamaño de las tramas es mayor que en UDP - La prioridad es la confiabilidad	- Sin conexión - Sin acuse de envío (no comprueba si la trama ha sido recibida) - No se fragmentan los mensajes - No existe sincronización - No hay comprobación de errores - Broadcast disponible (el emisor puede enviar la información a multitud de nodos receptores) - La prioridad es la velocidad

4.11 Comparativa TCP y UDP

Comparando ambos protocolos vemos que la principal diferencia es que TCP garantiza que los datos llegan correctamente, en el orden adecuado y sin errores ni duplicaciones. Estas características son las que a su vez, hacen que la velocidad de transmisión sea menor que en UDP, que no realiza estas comprobaciones. Asimismo en TCP es necesario establecer una conexión, para lo cual se necesita un cierto tiempo de inicialización, en cambio en el protocolo UDP esto tampoco es necesario, haciendo que los tiempos de inicialización sean más rápidos.

Para un dispositivo de telemetría en tiempo real el parámetro más importante es la velocidad de transmisión de datos, no siendo tan importantes la comprobación de errores o el orden de las tramas. El objetivo de la telemetría es recopilar la información de las variables físicas en el mínimo intervalo de tiempo posible, para poder determinar

el comportamiento de la variable. En conclusión, es el protocolo UDP el que proporciona mayor velocidad de transmisión en detrimento de la seguridad y la confiabilidad, que podrán aplicarse en el receptor a nivel de aplicación.

En el siguiente capítulo se mostrarán las pruebas realizadas utilizando ambos protocolos sobre Arduino y Shield 3G+GPS, donde se hará un estudio en profundidad del rendimiento de dichos protocolos sobre el prototipo.

5. PRUEBAS DE CAMPO ARDUINO MEGA ADK Y SHIELD 3G+GPS

5.1 DESCRIPCIÓN DE LAS PRUEBAS DE CAMPO

El objetivo de este capítulo es presentar los resultados de las pruebas realizadas sobre el prototipo de dispositivo de transmisión de datos compuesto por la placa Arduino Mega ADK y el Shield 3G+GPS.

Para realizar las pruebas que se muestran a continuación se ha trabajado en un entorno compuesto por el banco de trabajo descrito en el Capítulo 1. Es de señalar que durante todas las pruebas se ha trabajado sin ruido, y se ha comprobado que el indicador de intensidad de la señal recibida (RSSI) se encuentre entre unos valores de transferencia estables (desde -40 a -60 RSSI). Para comprobar esto, se ha utilizado el Comando AT+CSQ disponible en el manual que permite obtener el indicador de intensidad de la señal recibida y la tasa de error binario. Además, en cada una de las mediciones tomadas se detallará de manera particular la metodología usada para obtener dicha medición.

En último lugar se presenta un análisis de los resultados obtenidos comparando con lo que se describió en el capítulo anterior de manera teórica para cada protocolo.

5.2 TASA DE TRANSMISIÓN DE DATOS

Se entiende por tasa o velocidad de transmisión de datos el número de bytes que el dispositivo es capaz de enviar por segundo a un servidor remoto. Para realizar esta prueba se han utilizado metodologías diferentes para cada protocolo, ya que disponemos de herramientas diferentes en cada uno.

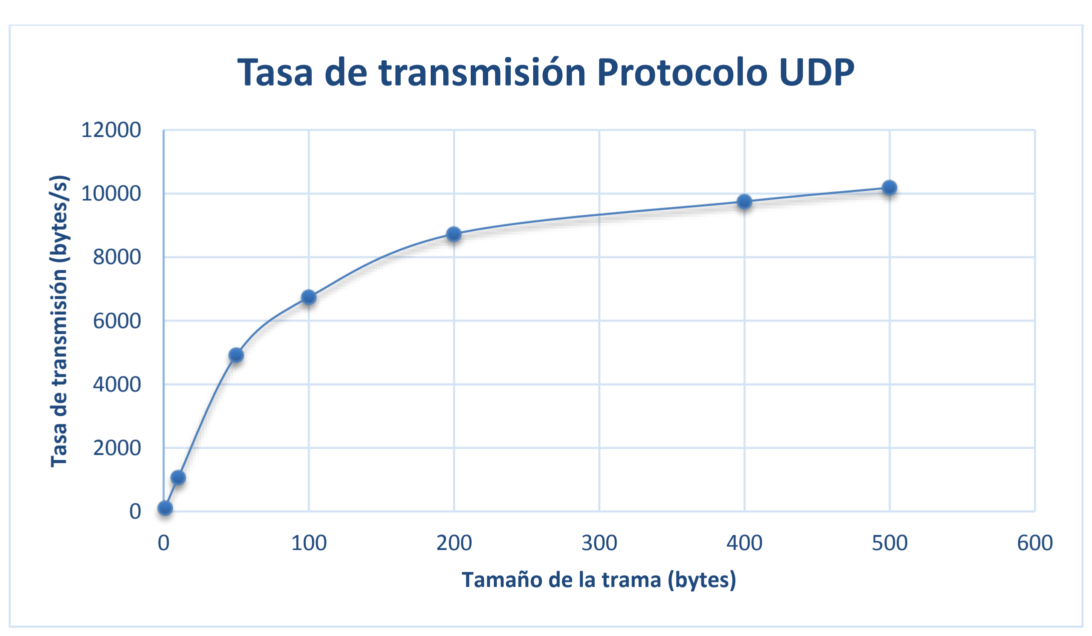
En ambos casos se ha obtenido la velocidad de transmisión de datos en función del tamaño de la trama que se envía, que es el parámetro que podemos modificar.

5.2.1 Tasa de transmisión de datos utilizando protocolo UDP

Para obtener la tasa de transmisión de datos en este protocolo se ha tomado el tiempo que tarda el dispositivo en enviar un número limitado de tramas de igual tamaño. Para medir el tiempo se ha utilizado una función que dispone Arduino para medir el tiempo que el microcontrolador lleva encendido, esta función es *millis()* y se encuentra disponible en la documentación de Arduino. Esta función devuelve el tiempo en milisegundos. El tamaño de trama se ha variado desde 1 byte hasta 500 bytes. No se ha podido superar una tamaño de trama de 500 bytes porque el microcontrolador se saturaba debido al *buffer* de almacenamiento temporal de datos del puerto serie del microcontrolador de Arduino, que tiene una tasa de transmisión de 115200 bits/segundo, que se traduce a 11500 bytes/segundo aproximadamente. Como podemos ver, la velocidad más alta obtenida mediante este protocolo ha sido 10183 bytes/segundo, valor que se encuentra por debajo de los 11500 bytes/segundo teóricos. A continuación se muestra la información obtenida:

Tamaño trama (bytes)	Tasa de transmisión (bytes/segundo)
1	108
10	1061
50	4914
100	6740
200	8729
400	9750
500	10183

5.1 Resultados tasa de transmisión protocolo UDP



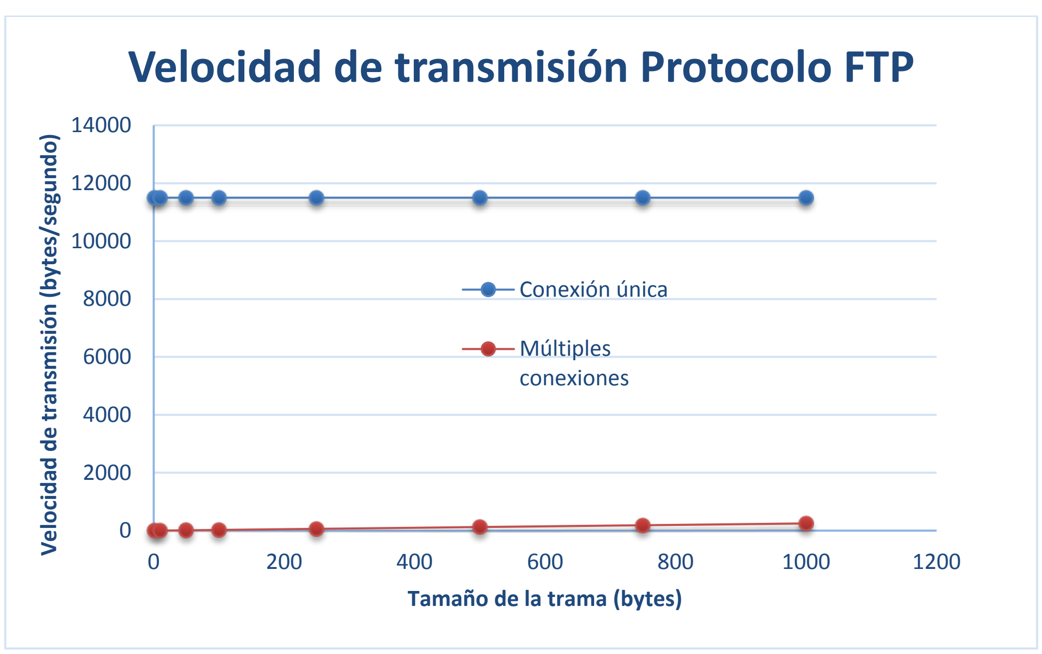
5.2.2 Velocidad de transmisión de datos utilizando protocolo FTP

En el caso de este protocolo la tasa de transmisión de datos se ha obtenido del servidor, cuyo software nos ofrece la posibilidad de medir directamente esta velocidad de transmisión de datos. En este caso, el tamaño de trama se ha podido variar entre 1 byte y 1024 bytes, es decir, un Kilobyte, ya que éste es el tamaño máximo que permite almacenar Arduino en su memoria SRAM. Se presentan dos gráficas, en la primera de ellas se ha realizado una única conexión y se han enviado los datos de manera continua sin cerrar la conexión. Como se esperaba, la velocidad de transmisión ha sido continua y limitada de nuevo por el puerto serie del microcontrolador. En este caso se han obtenido valores en torno a 11500 bytes/segundo, trabajando con la precisión que ofrece el servidor FTP. Como podemos ver estos valores siempre se encuentran muy próximo a 11520 bytes/segundo teóricos del puerto serie.

Para obtener la segunda gráfica se realizaron múltiples conexiones y desconexiones para la transmisión de cada una de las tramas de información. Este método sería necesario en el caso de que fuera necesario obtener, por ejemplo, el posicionamiento GPS, para el que hay que utilizar Comandos AT y una consecuente desconexión, ya que ello implica multiplexar el puerto serie entre los dos dispositivos, módem y GPS, ya que el microcontrolador comparte el mismo puerto serie para comunicarse con ambos.

Método	**Conexión única**	**Múltiples conexiones**
Tamaño trama (bytes)	**Tasa de transmisión (bytes/s)**	
1	11500	0.25
10	11500	2
50	11500	12
100	11500	24
250	11500	61
500	11500	123
750	11500	184
1024	11500	246

5.2 Resultados tasa de transmisión protocolo FTP



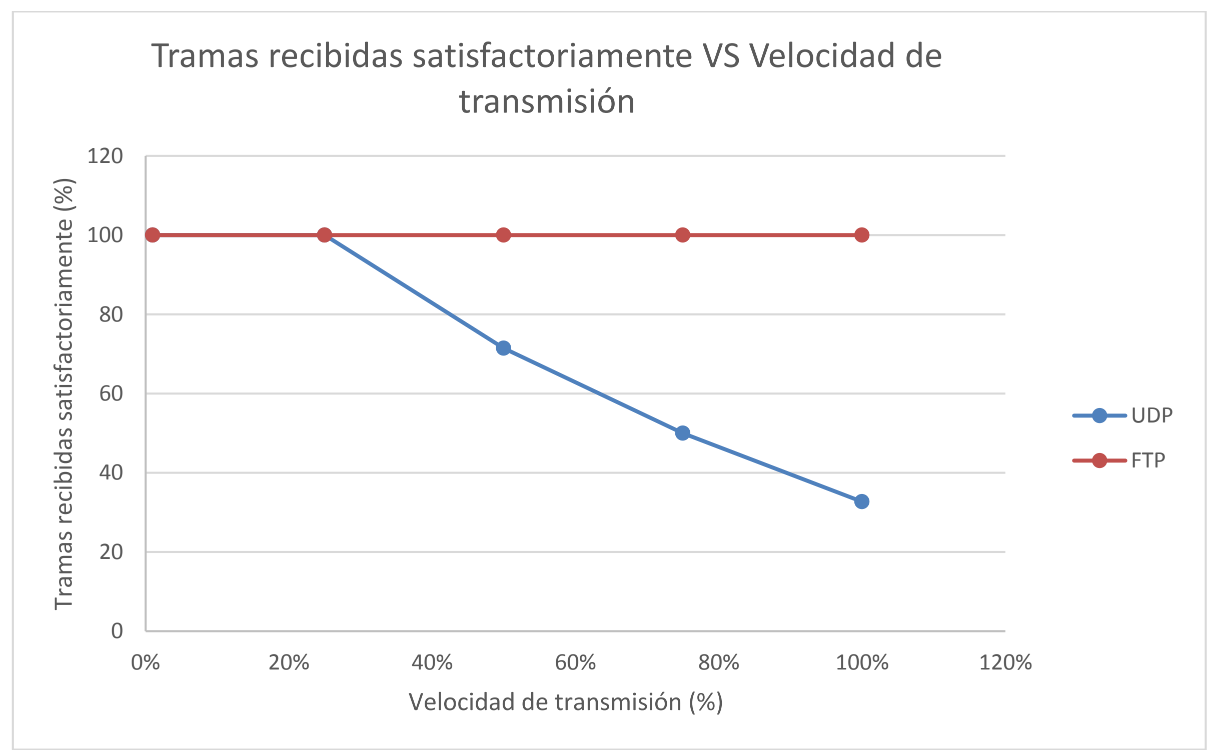
5.3 TASA DE ERROR DE TRANSMISIÓN DE DATOS

La tasa de error de transmisión es una valoración cuantitativa de la cantidad de mensajes que son enviados por el transmisor, en nuestro caso un módem de telefonía móvil, y son recibidos satisfactoriamente por el receptor, en nuestro caso un servidor de Internet. Para realizar esta prueba de campo, se enviaron utilizando el prototipo un número fijo de tramas y se midieron en el servidor el número de tramas recibidas, realizando esto con los dos protocolos que estamos analizando en este trabajo. El parámetro que se modificó en este caso es la velocidad de transmisión para realizar cada medida.

Como era de esperar, mediante el protocolo FTP llegaron satisfactoriamente el 100% de las tramas enviadas, ya que este protocolo implementa corrección de errores y acuse de recibo. De igual manera, para el protocolo UDP se ha obtenido una tasa de error creciente con la velocidad de transmisión. A continuación se presentan los valores obtenidos en función de la velocidad de transmisión de datos (con respecto a la máxima velocidad de transmisión):

5.3 Resultados tasa de error protocolo UDP y FTP

Tasa de transmisión (%tasa máxima)	UDP	FTP
	Tramas recibidas correctamente (%)	
100%	32	100
75%	50	100
50%	73	100
25%	100	100
1%	100	100



5.4 TIEMPO DE EJECUCIÓN DEL PROGRAMA

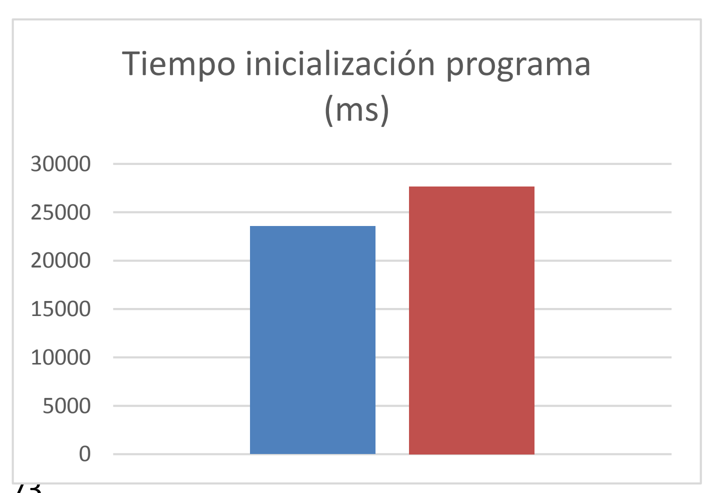
El tiempo de ejecución del programa contempla tanto el tiempo que tarda en inicializar el dispositivo con el programa necesario para utilizar cada protocolo, como el tiempo que tarda el dispositivo en transmitir una trama, una vez inicializado el proceso.

5.4.1 Tiempo inicialización del programa

El tiempo de inicialización del programa comprende el tiempo que transcurre desde que se enciende el dispositivo hasta que transmite la primera trama. Para realizar esta medición se ha utilizado la función millis() descrita anteriormente. Los resultados obtenidos se presentan en la siguiente tabla para los dos protocolos, no existiendo una gran diferencia entre ellos.

Protocolo	Tiempo inicialización programa (ms)
UDP	23563
TCP	27662

5.4 Resultados tiempo de inicialización



5.4.2 Tiempo de bucle del programa

El tiempo de bucle o *loop* del programa es el tiempo que transcurre entre la transmisión de una trama y la transmisión de la siguiente trama. Se ha realizado para ambos protocolos. Para el protocolo UDP podemos ver que el tiempo es despreciable, siendo el resultado obtenido la precisión de la función que utilizamos, 1 milisegundo. Para el protocolo FTP se ha utilizado una metodología de múltiples conexiones/desconexiones. En el caso de conexión única, ya que le proceso de conexión se realiza en la inicialización del dispositivo, el tiempo de blucle sólo dependerá del tamaño el programa, en esta tabla se mostrará el tiempo mínimo, que coincide con el resultado del protocolo UDP.

PROTOCOLO	TIEMPO BUCLE PROGRAMA (ms)
UDP	1
FTP conexión única	1
FTP múltiples conexiones	4065

5.5 Resultados tasa de transmisión protocolo UDP

5.5 UTILIZACIÓN DE MEMORIA SRAM

Un parámetro que concierne a la utilización de recursos del microcontrolador es la cantidad de memoria SRAM (memoria volátil) que utiliza para almacenar las variables utilizadas durante la ejecución del programa, como la lectura de un sensor. Este es un parámetro importante para la elección del microcontrolador y del programa que queramos implementar en él. En el caso de Arduino Mega ADK, se dispone de una memoria SRAM de 8 Kilobytes.

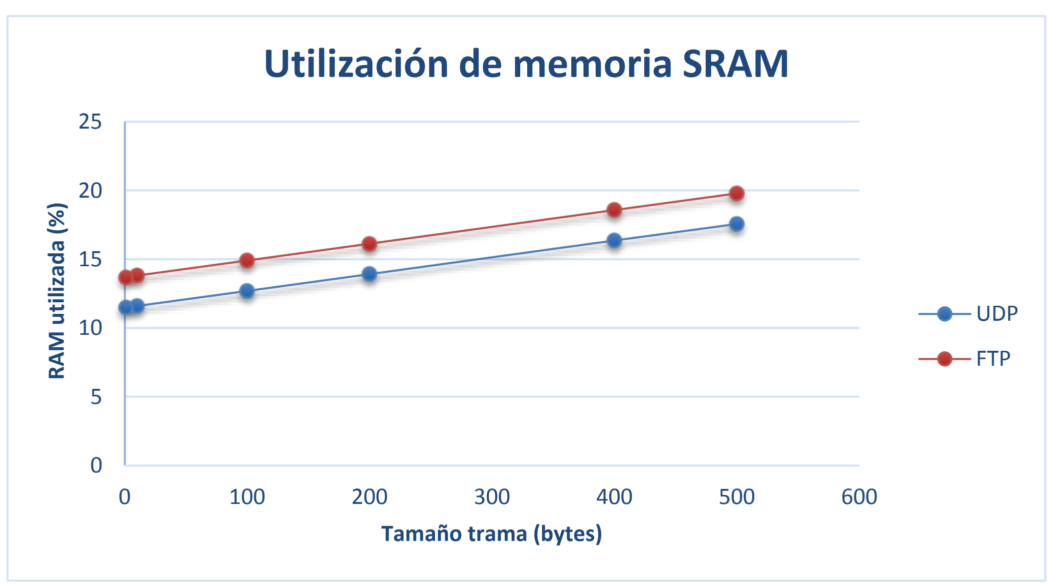
En este apartado se ha medido la cantidad de memoria SRAM que utilizan los programas mínimos necesarios para enviar una trama mediante los protocolos UDP y FTP. Para ello se ha utilizado la siguiente función, que devuelve la cantidad de memoria SRAM libre del microcontrolador:

```
int freeRam()
{
    extern int __heap_start, *__brkval;
    int v;
    return (int) &v - (__brkval == 0 ? (int) &__heap_start : (int) __brkval);
}
```

Los datos recogidos para ambos protocolos sobre un Arduino Mega ADK, en función del tamaño de trama, se muestran en la siguiente tabla:

Tamaño trama (bytes)	SRAM utilizada (bytes)		SRAM utilizada (% respecto total)	
	UDP	FTP	UDP	FTP
500	1438	1621	17,55	19,79
400	1340	1521	16,36	18,57
200	1140	1321	13,91	16,12
100	1040	1221	12,69	14,90
10	950	1131	11,60	13,81
1	940	1122	11,47	13,70

5.6 Resultados utilización de memoria SRAM



Como podemos ver, la cantidad de memoria utilizada es creciente y lineal con el tamaño del *array*, lo que nos indica un comportamiento correcto del compilador.

5.6 TAMAÑO DEL PROGRAMA

El tamaño del programa es el espacio de memoria flash en el que se almacena el código fuente del programa que deseamos ejecutar. En este caso se utiliza la memoria flash que incorpora Arduino (memoria permanente, a diferencia de la memoria SRAM). Es un aspecto importante, ya que la memoria de un microcontrolador es limitada y el espacio libre puede determinar, por ejemplo, el número de instrucciones que podemos incluir en nuestro sistema. Para medir el tamaño del programa se ha utilizado el entorno de programación de Arduino, que ofrece esta posibilidad. En el caso concreto del microcontrolador de Arduino Mega ADK, la memoria flash dedicada al almacenamiento del programa es 251554 bytes.

5.5.1 Tamaño del programa mínimo

El tamaño del programa mínimo es el tamaño del programa necesario para realizar la transmisión de un byte de datos de manera continua, es decir, en un bucle. Estos programas son los que se mostraron como ejemplo en el Capítulo 4. A continuación se muestran los tamaños de estos programas y el espacio libre restante, en función del tamaño de programa máximo que permite la memoria flash (251554 bytes):

UDP	Tamaño Mínimo (bytes)	Espacio libre (bytes)	Espacio libre (%)
	6494	251554	97.5
FTP	**Tamaño Mínimo (bytes)**	**Espacio libre (bytes)**	**Espacio libre (%)**
	6656	251392	97.4

5.7 Resultados tamaño del programa mínimo

5.5.2 Número máximo de sensores que pueden incorporarse

En este apartado se pretende realizar una valoración del máximo número de sensores analógicos o digitales que podríamos incorporar, en el caso de utilizar cualquiera de los programas mínimos necesarios para realizar una transmisión. Se ha tenido en cuenta adicionalmente la posibilidad de utilización del módulo GPS. Para obtener esta aproximación se realizaron programas que incorporaban un sensor, dos sensores, etc. y medido el tamaño necesario para incluir un sensor comparado con el programa que no incorpora sensores. De esta manera se ha podido llegar a la siguiente fórmula, mediante la que podemos obtener una valoración del número de sensores, en función del tamaño del programa que incorpora un sensor, el tamaño mínimo de programa y el espacio libre considerando el tamaño mínimo de programa:

$$Número\ de\ sensores\ (digitales\ o\ analógicos) = \frac{Espacio\ libre\ para\ tamaño\ mínimo}{Tamaño\ con\ 1\ sensor - Tamaño\ mínimo}$$

En la siguiente tabla se muestra el tamaño del programa que incluye la recopilación de los datos del GPS (utilizando el modo *Mobile-Based*) para los protocolos disponibles:

Protocolo	Tamaño mínimo del programa con GPS (Modo BS) (bytes)	Espacio libre (bytes)	Espacio libre (%)
UDP	7038	251010	97.3
FTP	7222	250826	97.2

A continuación se muestran dos tablas en las que se han realizado la valoración del número de sensores, dependiendo de si son analógicos o digitales y si se incorpora las funcionalidades del GPS:

5.8 Número máximo de sensores

Protocolo	Tamaño mínimo con un sensor analógico (bytes)	Numero de sensores totales	Numero de sensores totales con GPS
UDP	7064	444	443
FTP	7222	441	440

Protocolo	Tamaño mínimo con un sensor digital (bytes)	Numero de sensores totales	Numero de sensores totales con GPS
UDP	7048	454	453
FTP	7210	453	452

5.7 TASA DE RECOPILACIÓN DE DATOS GPS

Por último se ha diseñado una prueba para medir el tiempo que tarda Arduino Mega ADK y Shield 3G+GPS en obtener una trama de información de los satélites GPS. Estos datos son importantes por ejemplo, si se quiere incorporar geolocalización por GPS al sistema de telemetría en tiempo real, ya que la tasa de transmisión de datos se verá reducida al aumentar el tiempo de bucle de programa o *loop* para obtener esta información.

Para realizar esta prueba se han utilizado todos los modos disponibles en el módem para este propósito, y que se han podido utilizar satisfactoriamente: *Stand-Alone* y *Mobile-Based*. Aunque la intención era realizar también la prueba para el modo *Mobile-Assisted* (el más rápido y preciso de los tres), este modo no se logró utilizar satisfactoriamente utilizando la documentación que disponemos. En consecuencia, no se han podido obtener datos para este modo.

Para medir el tiempo se ha diseñado una prueba en la que se mide el tiempo que se necesita para obtener un número limitado de tramas con información procedente de satélites GPS. Para ello se ha utilizado la función *millis()* de la biblioteca Arduino. A partir del tiempo en obtener una trama, podemos obtener la tasa de recopilación de datos del

GPS conociendo el tamaño de las tramas, siendo igual en ambos casos y de valor 50 bytes, como se explicó en el capítulo correspondiente.

En la siguiente tabla se muestran la información obtenida:

Modo GPS	Tiempo en obtener 1 trama (ms)	Frecuencia de trabajo (Hz)	Tasa de recopilación (bytes/s)
Stand-Alone	17	59	3013
Mobile-Based	17	59	3013

5.9 Resultados tasa de recopilación datos GPS

5.8 EVALUACIÓN DE LOS RESULTADOS

A partir de los resultados obtenidos en las diversas pruebas realizadas, se pueden obtener una serie de conclusiones para valorar finalmente, de los protocolos de Internet disponibles en el Shield 3G+GPS, el óptimo para un sistema de telemetría en tiempo real.

En primer lugar, la tasa de transmisión de datos es el parámetro más importante para esta decisión. Este parámetro se ve limitado por el proceso más lento dentro de la transmisión, que es el puerto serie del microcontrolador, que se encarga de enviar la información recopilado al módem. En los resultados obtenidos podemos observar que la tasa de transmisión toma el valor más alto para el protocolo FTP con una conexión única, siempre por debajo de la tasa limitadora del puerto serie. Esta sería una opción válida siempre que no se desee implementar geolocalización mediante GPS, ya que una vez realizada la conexión FTP todo lo que se envíe por el puerto serie al módem será transmitido automáticamente al servidor en Internet, imposibilitando la comunicación por comandos AT necesaria para indicarle al modem que se comunique con el GPS. En consecuencia, esta opción queda descartada ya que uno de los objetivos de este proyecto es precisamente, utilizar la información de posicionamiento proporcionada por los satélites GPS. De esta manera será el protocolo UDP el utilizado para la transmisión de datos, presentando una tasa de transmisión máxima de 10 KB/s aproximadamente, ya que como podemos ver, utilizar el protocolo FTP con múltiples conexiones reduce considerablemente la tasa de transmisión hasta en un 98%.

A continuación, la información de la que disponemos es la tasa de error de transmisión de datos. Este es un dato importante para seleccionar la tasa de transmisión de datos. Se puede observar que para el protocolo FTP no existe error para cualquier tasa de transmisión como era de esperar, ya que es una de las cualidades de este protocolo. En cambio, el protocolo UDP no posee esta característica de comprobación de errores. Como podemos ver en los resultados obtenidos, si no seleccionamos una

tasa de transmisión adecuada podemos perder hasta el 68% de las tramas enviadas, inaceptable en un sistema de telemetría en tiempo real. Para una tasa de transmisión de 5 KB/s la tasa de error es del 27%. Este resultado se podría considerar aceptable en sistema en tiempo real, ya que se envían una gran cantidad de información por segundo y se pueden considerar despreciables las pérdidas en comparación con la información disponible.

La siguiente prueba que se realizó fue el tiempo de ejecución del programa. Primero se midió el tiempo de inicialización del programa, que es el tiempo desde que se enciende el dispositivo hasta que envía la primera trama. Para ambos protocolos se obtuvieron tiempo de inicialización similares. La mayor diferencia se encuentra en el tiempo de bucle o *loop* del programa, que es el tiempo que transcurre entre el envío de tramas. En este caso, el tiempo obtenido para el protocolo UDP es a penas despreciable, frente al tiempo necesario en el protocolo FTP para establecer una conexión y enviar una trama, en el que transcurren aproximadamente 4 segundos. En este aspecto, el protocolo UDP supone una gran ventaja frente al protocolo FTP. No se ha considerado el método FTP con conexión única, ya que se ha descartado porque no permite realizar geolocalización, y este es uno de los objetivos de este trabajo.

En cuanto a la utilización de la memoria SRAM y memoria flash, la información obtenida es interesante pero no diferenciadora. Para ambos protocolos se han obtenido valores similares, siendo la utilización de memoria un poco superior para FTP en ambos casos. No obstante, es una información valiosa ya que como se ha visto, el máximo de memoria SRAM utilizado es 20%, lo que supone un 80% de memoria libre, y el espacio de memoria libre *flash* para almacenar programas es aproximadamente el 97% muy útil para la creación de programas más complejos que puedan utilizar estos protocolos. Como información adicional se ha aproximado el número máximo de sensores analógicos o digitales que se podrían incorporar en cuanto a memoria de flash se refiere, obteniendo unos valores muy por encima de las capacidades de Arduino (en el caso de Arduino Mega ADK, el número máximo es 70 sensores digitales y 16 analógicos). En el caso de utilizar el número máximo teórico de sensores que hemos obtenido, la memoria SRAM necesaria para almacenar la información de las variables medidas por estos sensores nunca superará la memoria SRAM total disponible (la información de cada variable de un sensor se almacena como *int* y requiere como máximo de 2 bytes, en total son 880 Kilobytes aproximadamente, en el caso de sensores analógicos sería aún menor). Por tanto podemos afirmar que la memoria flash y SRAM no supondrán un problema para el prototipo.

En último lugar, se ha medido la máxima frecuencia a la que podemos trabajar con el módulo GPS incorporado. Esta información es muy útil a la hora de crear un prototipo de telemetría, ya que limitará la tasa de transmisión y no se podrá obtener la información de los sensores mientras se obtiene la información del GPS.

6. APLICACIÓN CASO PARTICULAR: MOTOCICLETA DE COMPETICIÓN

6.1 DESCRIPCIÓN CASO PARTICULAR

Este capítulo está dedicado a implementar la plataforma Arduino y Shield 3G+GPS para un sistema de telemetría en tiempo real para el caso concreto de una moto de competición. Para ello se hará uso de la información aportada en los capítulos anteriores, que nos sirven para establecer las bases de este sistema.

El procedimiento que se llevará a cabo durante este capítulo para el diseño del sistema de telemetría coincide con el orden establecido en los capítulos anteriores, y se puede resumir en:

- Definir las variables físicas que se medirán con el sistema. Conocido este dato, podremos seleccionar y evaluar el tipo de sensor y modelo que mejor se adapte al caso de una moto de competición. Por último, se diseñarán los programas necesarios para el uso por separado de cada uno de los sensores en Arduino.

- Definir el método de transmisión de datos en tiempo real. Se seleccionará el protocolo de transmisión de datos a través de internet óptimo para este caso particular y se darán las especificaciones concretas del sistema de transmisión.

- Diseño del algoritmo del programa. Una vez estudiado como recopilar los datos y como enviarlos a través de internet, el siguiente paso es diseñar el programa para Arduino que implemente todas estas características de manera óptima. Para ellos se diseñarán los diagramas de flujo correspondiente y finalmente, el código fuente del programa en lenguaje Arduino.

- Pruebas de campo del prototipo completo. Se realizarán pruebas similares a las que se realizaron en el Capítulo 5 para comprobar las especificaciones del prototipo completo.

6.2 ADQUISICIÓN DE DATOS. SENSORES

En primer lugar se presentan las variables físicas más importantes que se miden en una motocicleta de competición:

- Revoluciones del eje del motor y de las ruedas. Es una variable importante para conocer el comportamiento del motor, de la transmisión y de las ruedas. Con estos datos podemos establecer rendimientos del motor, relacionarlo con la palanca de acelerador y freno, y estudiar el deslizamientos de las ruedas cuando se produzca. Para la medición de las revoluciones, lo más común es utilizar un sensor de efecto Hall.

- Inclinación de la motocicleta. Esta variable nos permite obtener información de la propia conducción del piloto sobre el circuito de carreras. Puede utilizarse para estudiar el grado de inclinación en cada curva, para optimizar el paso por curva, o para detectar una caída y estudiar su causa. Para este fin, los sensores más adecuados son los denominados acelerómetros.

- Temperatura del motor, aceite, neumáticos. Este parámetro también es importante para estudiar el comportamiento de los componentes mecánicos de la motocicleta. Con estos datos podemos obtener información del comportamiento del motor a una carga determinada, pérdidas de rendimiento, correcto funcionamiento del sistema de refrigeración, agarre los neumáticos… Para la medición de temperaturas existen en el mercado un gran número de sensores analógicos y digitales apropiados para este propósito, todos ellos válidos.

- Desplazamientos de la suspensión, acelerador y freno. Los desplazamientos pueden ser lineales (como la suspensión) o radiales (palancas de freno y acelerador). Al igual que las demás variables, medir el desplazamiento de la suspensión y de las palancas de freno y acelerador es vital para conocer el comportamiento de los elementos mecánicos y de su relación con la conducción del piloto de la motocicleta. Para la medición de desplazamientos lo más común es utilización de diversos sensores de desplazamiento analógicos, aunque también pueden utilizarse encoders digitales si se requiriese de más precisión.

En los siguientes puntos de apartado se presentan los sensores escogidos para la medición de estas variables físicas, así como sus especificaciones y los programas necesarios para su uso en Arduino.

6.2.1 Sensor digital de efecto Hall AH183

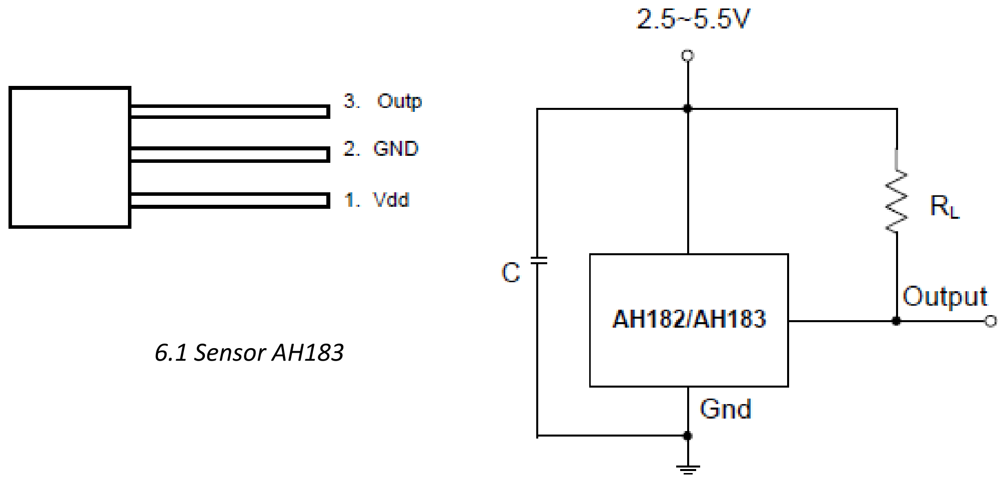
El sensor de efecto Hall se sirve del efecto Hall para la medición de campos magnéticos o corrientes. Esta característica hace posible que sean utilizados para medir la posición de objetos a través de la variación del campo magnético, como es el caso del giro de un eje. Los sensores de efecto Hall se utilizan para medir el número de vueltas del eje, y mediante este dato y el tiempo transcurrido, calcular las revoluciones por unidad de tiempo del eje.

Aunque los sensores de efecto Hall pueden tener salidas analógicas, para el caso que nos concierne utilizaremos un sensor cuya salida sea de tipo digital, es decir “1” o “0”. Además este sensor debe trabajar con una alimentación de 5 V o 3.3 V para que sea totalmente compatible con Arduino.

El sensor de efecto Hall que se ha utilizado para este caso particular es el modelo AH183. La principal característica para su selección ha sido la frecuencia de muestreo de este sensor. Por lo general el motor de una moto de competición alcanza alrededor de 10000 revoluciones por minuto, que son aproximadamente 167 revoluciones por segundo. Para asegurar la medición correcta es necesario que el sensor trabaje a una frecuencia por encima de este número. En el caso del sensor AH183, la frecuencia es 200 Hz, suficiente para detectar la posición de la rueda en cada revolución. También se han tenido en cuenta otras especificaciones como la máxima temperatura de trabajo y la tensión de alimentación.

6.2.1.1 Especificaciones técnicas AH183

Este sensor presenta tres pines para su conexión: 1. Alimentación (*Vdd*), Masa (*GND*) y salida (*Output*), en la disposición que se muestra en las siguiente figura. También se muestra la disposición de su circuito interno [12]:



6.1 Sensor AH183

Las principales especificaciones técnicas de este sensor son:

AH183	
Tensión de alimentación mínima	2,5 V
Tensión de alimentación máxima	5V
Frecuencia de muestreo máxima	200 Hz
Máxima temperatura de operación	85ºC
Mínima temperatura de operación	-40ºC
Tipo de encapsulado	SIP-3L

6.2 Especificaciones técnicas AH183

6.2.1.2 Código Arduino para utilización del sensor AH183

Mediante este programa podemos medir la velocidad instantánea de un eje que gira:

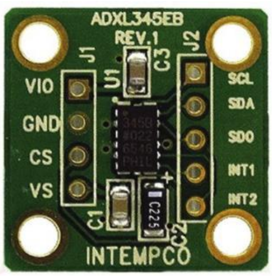
```
int Hall = 3; //Salida sensor conectada al pin digital 3
int tiempo;
int rpm;
int Hallstate;
void setup()
{
  Serial.begin(9600);
  pinMode(Hall, INPUT);//Indicar pin 3 es una salida
  tiempo = millis();//Establecer primera medida de tiempo
}


void loop()
{
  Hallstate = digitalRead(Hall);

  if(Hallstate == true) //Si la salida es "1"
  {
//Calcular tiempo que tarda en dar una vuelta compelta
    tiempo = millis()-tiempo ;//Variación de tiempo entre lecturas "1"
    rpm = (60*1000)/tiempo; //RPM = Revoluciones por minuto
//Multiplicar por mil para pasar a segundos y por sesenta para pasar a minutos
    Serial.print(rpm);//Enviar por el puerto serie "rpm"
  }
  delay(5);//Esperar 5 ms, para fijar una frecuencia de medida de 200 Hz
}
```

6.2.2 Sensor digital SPI acelerómetro ADXL345

Un acelerómetro es un dispositivo capaz medir la aceleración y las fuerzas inducidas por la gravedad. Existen varios tipos de acelerómetros en la actualidad: mecánicos, piezoeléctricos, capacitivos y de efecto Hall. Otra característica diferenciadora es si son capaces de medir aceleraciones en dos ejes (un plano) o tres ejes (espacio tridimensional). Por tanto, el acelerómetro es el sensor idóneo para medir el ángulo de inclinación de una motocicleta.

Para este sistema se ha trabajado con el acelerómetro de tres ejes ADXL345. Este acelerómetro permite la comunicación SPI e I^2C, en este trabajo se utilizará SPI. La gran ventaja de este modelo frente a los demás es su relación calidad/precio, que lo ha llevado a ser el acelerómetro más utilizado para sistemas similares al que nos concierne. Gracias a esta popularidad, se puede encontrar una gran variedad de programas y tutoriales para su uso.

Para facilitar su utilización se adquirió la Placa de Evaluación ADXL345, que implementa este chip y está preparada para su uso directo con el microcontrolador de la placa Arduino. Esta placa se muestra en la siguiente figura:

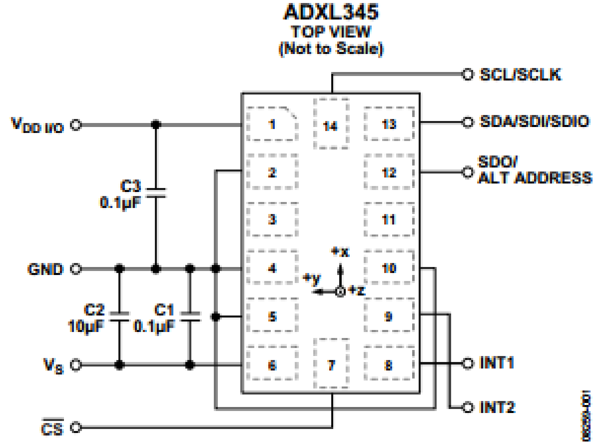


6.3 Sensor ADXL345

6.2.2.1 Especificaciones técnicas ADXL345

La placa de evaluación del acelerómetro ADXL345 dispone el circuito de la figura que se muestra a continuación, donde se pueden visualizar los siguientes pines [13]:

- V_{DD} (*Digital Interface Supply Voltage*)
- GND (*Ground*)
- VS (*Supply Voltage*)
- CS (*Chip Select*)
- SCLK (*Serial Communications Clock*)
- SDA (*Serial Data*)
- SDO (*Serial Data Output*)
- INT1/INT2 (*Interrupt Output*). Pines configurables.



6.4 Esquema conexiones ADXL345

Como se ha dicho anteriormente este dispositivo es capaz de utilizar las interfaces de comunicación SPI e I^2C. Para utilizar la comunicación SPI son necesarios los pines SDA, SDO, SCK y CS como se describió en el Capítulo 3. Estos mismos pines (exceptuando CS) también se pueden utilizar para la comunicación I^2C. Para alimentar el chip se utilizarán los pines VS y GND.

Las principales especificaciones técnicas de este sensor son:

ADXL345	
Tensión de alimentación mínima	3,3 V
Tensión de alimentación máxima	5,5 V
Frecuencia de muestreo máxima	100 Hz
Máxima temperatura de operación	85ºC
Mínima temperatura de operación	-40ºC
Resolución	13 bits
Rango medición	$\pm 16g$

6.5 Especificaciones técnicas ADXL345

Para configurar el sensor podemos encontrar toda la información necesaria en la documentación del chip, en donde podemos encontrar las direcciones de los registros de configuración y de datos, además de los pasos para configurar el chip a través de estos registros.

6.2.2.2 Código Arduino para utilización del sensor ADXL345

A continuación se muestra el programa en código Arduino necesario para obtener el ángulo de inclinación, en este caso, en el plano YZ. Para obtener el resto de ángulos se operaría de manera similar:

```
#include <SPI.h>
int CS=10;//Asignar CS al pin digital 10
//Estos son los registros que vamos a utilizar:
char POWER_CTL = 0x2D;  //Power Control Register
char DATA_FORMAT = 0x31;
char DATAX0 = 0x32;     //X-Axis Data 0
char DATAX1 = 0x33;     //X-Axis Data 1
char DATAY0 = 0x34;     //Y-Axis Data 0
char DATAY1 = 0x35;     //Y-Axis Data 1
char DATAZ0 = 0x36;     //Z-Axis Data 0
char DATAZ1 = 0x37;     //Z-Axis Data 1
//Definimos esta cadena para almacenar los valores de los registros
char values[10];
int x,y,z;
float angle;

void setup(){
  SPI.begin();//Inicializar interfaz SPI
  SPI.setDataMode(SPI_MODE3);//Configurar SPI Modo 3(Ver capítulo 3 - SPI)
  pinMode(CS, OUTPUT);//Indicar CS es una salida
  digitalWrite(CS, HIGH);//CS alto para que no haya comunicación
  writeRegister(DATA_FORMAT, 0x01);//Configuramos Rango +/- 4G range escribiendo 0x01
  writeRegister(POWER_CTL, 0x08);  // Configuramos Measurement mode
}
void loop(){
  //En DATAX0 se encuentran almacenados los valores de aceleración de los ejes x,y,z
  //Estos valores se almacenarán en values
  readRegister(DATAX0, 6, values);
  //En el rango Rango +/- 4G la resolución es 10 bits, almacenados como bytes
  //Para obtener todos los valores, cada eje necesita de 2 bytes
  //Los valores del ejex se en cuentra en values[0] y values[1].Eje Y, Z misma metodología
  x = ((int)values[1]<<8)|(int)values[0];
  y = ((int)values[3]<<8)|(int)values[2];.
  z = ((int)values[5]<<8)|(int)values[4];
 //1 rad = 57.296°
  angle = atan(y/z)*57.296;
 delay(10); //Para establecer frecuencia de trabajo 100 Hz
 }
```

Para facilitar la programación del código se han definido las siguientes funciones para leer y escribir registros:

```
void writeRegister(char registerAddress, char value){
  //Nivel bajo en CS para establecer comunicación
  digitalWrite(CS, LOW);
  //Enviar la dirección del registro en el que queremos escribir
  SPI.transfer(registerAddress);
  //Enviar el valor del registro
  SPI.transfer(value);
  //Nivel alto en CS para terminar comunicación
  digitalWrite(CS, HIGH);
}

void readRegister(char registerAddress, int numBytes, char * values){

  char address = 0x80 | registerAddress;
  if(numBytes > 1)address = address | 0x40;

  //Nivel bajo en CS para establecer comunicación
  digitalWrite(CS, LOW);
  //Enviar la dirección del registro que queremos leer
  SPI.transfer(address);
  //Leer todos los registros hasta el registro especificado
  for(int i=0; i<numBytes; i++){
    values[i] = SPI.transfer(0x00);
  }
  //Nivel alto en CS para terminar comunicación
  digitalWrite(CS, HIGH);
}
```

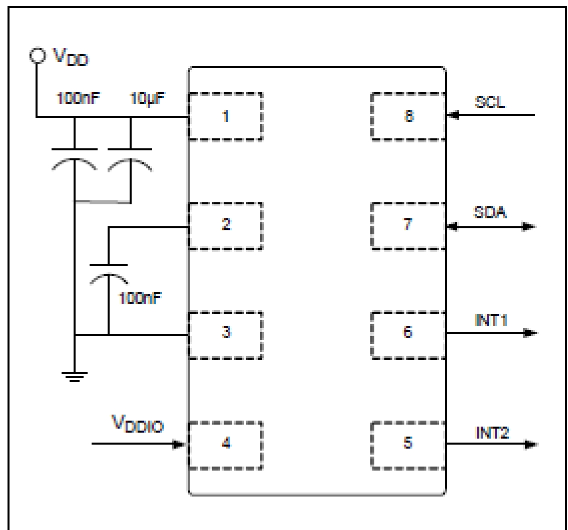
6.2.3 Sensor digital I²C barométrico y de temperatura MPL3115A2

Para la medición de temperatura existen una gran variedad de sensores capaces de medir temperaturas con una alta precisión. En este se ha trabajado hemos trabajado con el sensor MLP3115A2 ya que utiliza el bus de comunicación I²C y aumenta el valor de este trabajo al usar todos los buses de comunicación disponibles en Arduino.

El sensor MLP3115A2 es un altímetro de precisión que permite obtener información de temperatura y presión barométrica, y ha sido implementado en diferentes tipos de sistemas como cuadricópteros. Aunque el dato de la presión y altitud no es relevante en este proyecto, el dato de la temperatura sí es útil.

6.6 Sensor MPL3115A2

Para utilizar este chip es necesario diseñar un circuito integrado con la siguiente configuración:



6.7 Esquema conexiones MPL3115A2

6.2.3.1 Especificaciones técnicas MPL3115A2

En primer lugar se muestra la configuración de los pines de este chip [14]:

Top View

V_{DD}	1	8	SCL
CAP	2	7	SDA
GND	3	6	INT1
V_{DDIO}	4	5	INT2

Pin Connections

6.8 Esquema pines MPL3115A2

- V_{DD} (*Power Supply Connection*)
- CAP (*External Capacitor*)
- GND (*Ground*)
- VS (*Supply Voltage*)
- V_{DDIO} (*Digital Interface Power Supply*)
- SCL (I^2C *Serial Clock*)
- SDA (I^2C *Serial Data*)
- INT1/INT2 (*Interrupt Output*). Pines configurables.

Para conectarlo a Arduino utilizaremos los pines SDA y SCL. Para alimentar el sensor, los pines V_{DD} y GND.

MPL3115A2	
Tensión de alimentación mínima	1,95 V
Tensión de alimentación máxima	3,6 V
Frecuencia de muestreo máxima	100 Hz
Resolución	24 bits
Máxima temperatura de operación	85ºC
Mínima temperatura de operación	-40ºC
Precisión temperatura	$\pm 1ºC$
Rango presión	Máxima: 110KPa Mínima: 50KPa
Tipo de encapsulado	LGA, 5x3x1.1 mm

6.9 Especificaciones técnicasMPL3115A2

6.2.3.2 Código Arduino para utilización del sensor MPL3115A2

A continuación se muestra el código fuente utilizado para obtener únicamente el dato de la temperatura. Previamente a la rutina de monitorización es necesario definir una serie de registros que se utilizarán en este código:

```
#define STATUS     0x00
#define OUT_P_MSB  0x01
#define OUT_P_CSB  0x02
#define OUT_P_LSB  0x03
#define OUT_T_MSB  0x04
#define OUT_T_LSB  0x05
#define DR_STATUS  0x06
#define OUT_P_DELTA_MSB  0x07
#define OUT_P_DELTA_CSB  0x08
#define OUT_P_DELTA_LSB  0x09
#define OUT_T_DELTA_MSB  0x0A
#define OUT_T_DELTA_LSB  0x0B
#define WHO_AM_I   0x0C
#define F_STATUS   0x0D
#define F_DATA     0x0E
#define F_SETUP    0x0F
#define TIME_DLY   0x10
#define SYSMOD     0x11
#define INT_SOURCE 0x12
#define PT_DATA_CFG 0x13
#define BAR_IN_MSB 0x14
#define BAR_IN_LSB 0x15
#define P_TGT_MSB  0x16
#define P_TGT_LSB  0x17
#define T_TGT      0x18
#define P_WND_MSB  0x19
#define P_WND_LSB  0x1A
#define T_WND      0x1B
#define P_MIN_MSB  0x1C
#define P_MIN_CSB  0x1D
#define P_MIN_LSB  0x1E
#define T_MIN_MSB  0x1F
#define T_MIN_LSB  0x20
#define P_MAX_MSB  0x21
#define P_MAX_CSB  0x22
#define P_MAX_LSB  0x23
#define T_MAX_MSB  0x24
#define T_MAX_LSB  0x25
#define CTRL_REG1  0x26
#define CTRL_REG2  0x27
#define CTRL_REG3  0x28
#define CTRL_REG4  0x29
#define CTRL_REG5  0x2A
#define OFF_P      0x2B
#define OFF_T      0x2C
#define OFF_H      0x2D
#define MPL3115A2_ADDRESS
```

```
void setup()
{
  Wire.begin();        //Iniciar bus I2C
  Serial.begin(57600);  //Iniciar puerto serie

  if(IIC_Read(WHO_AM_I) == 196)
    Serial.println("MPL3115A2 online!");
  else
    Serial.println("No response");
 //Configurar el sensor
  setModeAltimeter();
  setOversampleRate(7); //Modo oversample 128
  enableEventFlags(); //Configurar medición temperatura y presión
}
void loop()
{
  temperature = readTemp();
  Serial.print(temperature);
  delay(5);
}
float readTemp()
{
  toggleOneShot();
  int counter = 0;
  while( (IIC_Read(STATUS) & (1<<1)) == 0)
  {
      if(++counter > 100) return(-999); //Error
      delay(1);
  }
  //Leer registros temperatura
  Wire.beginTransmission(MPL3115A2_ADDRESS);
  Wire.write(OUT_T_MSB);  //Direccion del registro
  Wire.endTransmission(false);
  Wire.requestFrom(MPL3115A2_ADDRESS, 2);//Solicitar 2 bytes
  //Esperar a datos disponibles
  counter = 0;
  while(Wire.available() < 2)
  {
    if(++counter > 100) return(-999); //Error
    delay(1);
  }
  byte msb, lsb;
  msb = Wire.read();
  lsb = Wire.read();
  float templsb = (lsb>>4)/16.0; //Temperatura, en grados Celsius
  float temperature = (float)(msb + templsb);

  return(temperature);
```

Donde se han definido las siguientes funciones:

```
//Modo altimetro
void setModeAltimeter()
{
  byte tempSetting = IIC_Read(CTRL_REG1);
  tempSetting |= (1<<7);
  IIC_Write(CTRL_REG1, tempSetting);
}
//Establecer Ratio Oversample
void setOversampleRate(byte sampleRate)
{
  if(sampleRate > 7) sampleRate = 7;
  sampleRate <<= 3;
  byte tempSetting = IIC_Read(CTRL_REG1);
  tempSetting &= 0b11000111;
  tempSetting |= sampleRate;
  IIC_Write(CTRL_REG1, tempSetting);
}
//Borrar el bit de OST
void toggleOneShot(void)
{
  byte tempSetting = IIC_Read(CTRL_REG1);
  tempSetting &= ~(1<<1);
  IIC_Write(CTRL_REG1, tempSetting);
  tempSetting = IIC_Read(CTRL_REG1);
  tempSetting |= (1<<1);
  IIC_Write(CTRL_REG1, tempSetting);
}
//Habilitar la medición de temperatura y presión
void enableEventFlags()
{
  IIC_Write(PT_DATA_CFG, 0x07);
}
//Función que lee en el bus I2C
byte IIC_Read(byte regAddr)
{
  Wire.beginTransmission(MPL3115A2_ADDRESS);
  Wire.write(regAddr);  //Escribir dirección registro
  Wire.endTransmission(false);
  Wire.requestFrom(MPL3115A2_ADDRESS, 1);//Solicitar dato
  return Wire.read();//Devolver valor recibido
}
//Función que escribe en el bus I2C
void IIC_Write(byte regAddr, byte value)
{
  Wire.beginTransmission(MPL3115A2_ADDRESS);//Empezar transmisión
  Wire.write(regAddr);//Escribir dirección registro
  Wire.write(value);//Escribir valor
  Wire.endTransmission(true);//Fin transmisión
}
```

6.2.4 Sensores analógicos para desplazamientos

Para medir desplazamientos y posiciones se pueden encontrar en el mercado multitud de sensores analógicos apropiados para cada tipo de desplazamiento. En concreto, los más utilizados son potenciómetros radiales y lineales. Un potenciómetro es un resistor cuyo valor de resistencia es proporcional al ángulo de giro o al desplazamiento lineal. La medición se obtiene del voltaje o intensidad de salida que es por tanto, proporcional a la resistencia.

En el caso de medir el desplazamiento de la suspensión lo más utilizado es un potenciómetro lineal, mientras que para medir el desplazamiento de freno y acelerador se utilizan potenciómetros radiales.

6.10 Ejemplo potenciómetro lineal y potenciómetro radial

Por norma general, los potenciómetros con salida analógica disponen de tres pines: Alimentación (VS), masa (GND) y salida. Este último pin es el que se debe conectar a una de las entradas analógicas de Arduino.

Debido al alto coste de los potenciómetros lineales de precisión industriales, no se ha estimado conveniente trabajar con uno de ellos para este primer prototipo. En su lugar se trabajará con un potenciómetro deslizante modelo *Bourns PTE A Series Low Profile Slide Potentiometer*, indicado para sistemas de control de volumen en equipos de audio (con inferiores características en cuanto a tolerancias, repetitividad, estanqueidad y robustez) y se estudiará su validez para medir del desplazamiento de la suspensión. Las rutinas y el esquema de control y conexionado serán idénticas en cualquier caso, por lo que el trabajo desarrollado para este primer prototipo servirá perfectamente en el caso de que se utilice un sensor de mayor calidad en el sistema definitivoAsimismo, se realizará un estudio similar para el potenciómetro radial *Bourns PTV A Series*, para la medición del desplazamiento angular de las palancas de freno y acelerador.

Para la utilización de estos sensores no se mostrará el código del programa de Arduino, ya que se opera de la misma forma que en el ejemplo del Capítulo 3 para sensores analógicos.

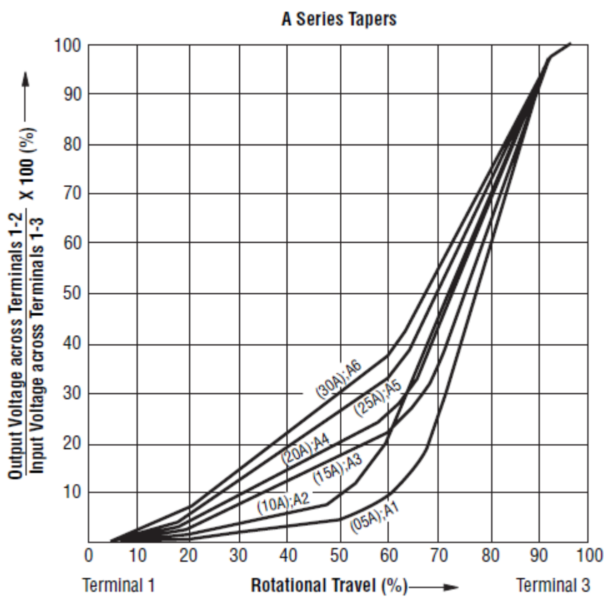
6.2.4.1 Especificaciones técnicas Potenciómetro deslizante Bourns PTE A Series y evaluación

En la hoja de especificaciones de este potenciómetro, proporcionada por el fabricante, podemos ver sus especificaciones más relevantes [15]:

Bourns PTE A Series	
Resistencia máxima	100KΩ
Tolerancia	$\pm 20\%$
Temperatura máxima de operación	55ºC
Temperatura mínima de operación	-10ºC
Longitud	60 mm

6.11 Especificaciones técnicas Bourns PTV A Series

El fabricante también proporciona la siguiente gráfica para el cálculo de la resistencia a través de la relación entre el voltaje de entrada y salida (en %) y el desplazamiento (en %):



6.12 Curva tensión/desplazamiento Bourns PTV A Series

Como podemos ver en esta gráfica, esta relación no es lineal en muchos modelos, y el fabricante no nos aporta una ecuación para obtener la relación de las magnitudes. Este hecho, junto con la tolerancia de fabricación hace difícil obtener una buena aproximación de la verdadera magnitud medida. Por otro lado las temperaturas de operación no son adecuadas para el sistema ni la longitud de deslizamiento del sensor es suficiente para medir el desplazamiento de la suspensión.

En nuestro caso utilizaremos este sensor para simular un potenciómetro lineal en el prototipo, que sería el adecuado en un sistema de estas características, pero que debido a su alto coste no ha podido ser adquirido.

6.2.4.2 Especificaciones técnicas Potenciómetro radial Bourns PTV A series y evaluación

En la siguiente tabla se muestran las especificaciones fundamentales que aporta el fabricante en la hoja de especificaciones de este potenciómetro [16]:

Bourns PTV A Series	
Resistencia máxima	1KΩ
Tolerancia	$\pm 20\%$
Temperatura máxima de operación	50ºC
Temperatura mínima de operación	-10ºC
Nº de vueltas	1

6.13 Especificaciones técnicas Bourns PTE A Series

El fabricante también proporciona la siguiente gráfica para el cálculo de la resistencia a través de la relación entre el voltaje de entrada y salida (en %) y el desplazamiento (en %):

6.14 Curva tensión/rotación Bourns PTE A Series

A Series Tapers

Terminal 1-2 Output Voltage / Terminal 1-3 Input Voltage X 100 (%)

Rotational Travel (%)

En este caso se nos presentan los mismos problemas que en el sensor anterior descrito. En primer lugar, la relación no es lineal en muchos modelos, y el fabricante no nos aporta una ecuación para obtener la relación de las magnitudes. Además, las tolerancias de fabricación son muy altas, lo que nos aportará una mala aproximación de la magnitud medida. A diferencia del sensor anterior, en este caso la temperatura no es un factor determinante, ya que las palancas de freno y acelerador no sufren una alta temperatura, y la longitud del sensor y el número de vueltas es suficiente, ya que estas palancas suelen tener un ángulo de giro de menos de 90º.

Por tanto, podemos afirmar que este sensor no se adecúa al igual que el anterior en para este sistema, en el que se requiere de precisión para ser útil. Debido a esto, aunque se trabajará con este sensor para el prototipo, no se recomiendo su uso final en el sistema.

6.3 TRANSMISIÓN DE DATOS Y GEOLOCALIZACIÓN

Para la transmisión de datos utilizaremos estrictamente las ideas obtenidas en los capítulos anteriores. En cuanto al protocolo de comunicaciones, utilizaremos el protocolo UDP por ser el más eficaz para un sistema de telemetría en tiempo real, como se vio en el capítulo anterior. Para ello se utilizará el tamaño de trama más grande admitido (500 bytes en este caso) y una tasa de transmisión por debajo de 5Kb/s para asegurar que al menos tasa de error este por debajo del 25%.

En cuanto a la geolocalización, el método seleccionado para obtener información de satélites GPS es *Mobile-Based,* ya que no se logró utilizar el modo *Assisted-Mobile* con el Shield 3G + GPS.

6.4 DISEÑO DEL ALGORITMO

6.4.1 Algoritmo para la monitorización de datos

El primer paso para diseñar el algoritmo del programa es estudiar el conjunto de variables físicas y los sensores que se utilizan para cada una para establecer un orden de prioridad de lectura de estas variables. Los factores a tener en cuenta para esto son las frecuencias de trabajo de los sensores y la frecuencia de variación de la magnitud física, esto último se realizará de un modo cualitativo. En algunos casos será interesante leer un sensor por debajo de su frecuencia de muestreo, ya que se puede estimar que la magnitud que mide varía de forma más lenta.

Magnitud	Variación de la magnitud física	Frecuencia de muestreo máxima del sensor
Revoluciones	Rápida	200 Hz
Ángulo de inclinación	Rápida	100 Hz
Temperatura	Lenta	100 Hz
Desplazamiento	Rápida	Analógico
Geolocalización*	Lenta	50 Hz

6.15 Magnitudes motocicleta competición

*Se ha incluido la geolocalización en esta tabla aunque no se trata de un sensor como tal, a efectos prácticos se puede tratar como un sensor si consideramos que obtiene datos de una variable, en este caso, coordenadas geográficas.

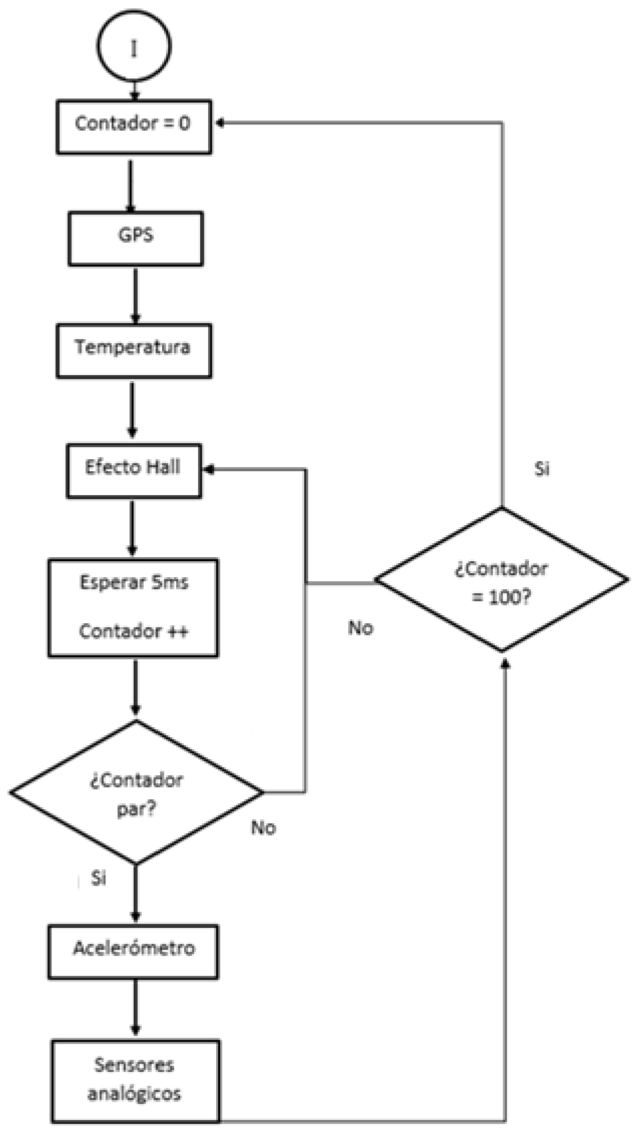
En esta tabla podemos separar en dos grupos, un grupo de magnitudes que se ha considerado que varían de forma rápida en el tiempo, y otro grupo que varían de una forma más lenta con respecto al otro grupo. De esta manera podemos concluir que por cada lectura de una variable "lenta" será necesario realizar varias mediciones del grupo de variables "rápidas". Así se consigue obtener el método más rápido para la recopilación de información, lo que se traduce en optimización del programa.

La frecuencia de muestreo que se va a emplear en este caso para cada sensor se muestra en la siguiente tabla:

Sensor	Frecuencia de muestreo utilizada
Sensor efecto Hall	200 Hz
Acelerómetro	100 Hz
Sensor de temperatura	1 Hz
Sensores analógicos	100 Hz
Geolocalización GPS	1 Hz

6.16 Frecuencias de muestreo sensores motocicleta competición

El esquema del orden de lectura de sensores (por cada segundo) queda de una forma sencilla de la siguiente manera, en forma de diagrama de flujo:



6.17 Flujograma monitorización de datos

6.4.2 Codificación de los datos

El siguiente paso para el diseño del programa es la codificación de datos. En este punto debemos definir el tamaño y la disposición de la información en las tramas. Es un punto clave para la optimización del programa, ya que el protocolo UDP envía mensajes codificados mediante caracteres (compuestos por un byte cada uno), y la cantidad de información que enviemos en cada trama UDP depende de la cantidad de bytes que enviemos por sensor. Por ello, es interesante estudiar los tipos de datos que se han utilizado en los puntos anteriores para cada variable:

Variable	Tipo de dato	Tamaño
Revoluciones	*int*	2 bytes
Ángulo de inclinación	*float*	4 bytes
Temperatura	*float*	4 bytes
Desplazamiento	*int*	2 bytes
GPS	*setring*	50 bytes

6.18 Tipos de datos sensores motocicleta de competición

En esta tabla podemos observar que algunas de las variables podrían reducir su tamaño para aumentar de esta manera la cantidad de información que podemos enviar en cada trama. De forma ideal, el mínimo tamaño que podría tener cada variable es 1 byte, de esta manera la cantidad de información en cada trama UDP sería máxima.

Es posible realizar cambio en el tipo de dato, en detrimento de precisión en la medición de cada variable, para reducir el tamaño hasta 2 byte de cada una si lo guardamos como entero (*int*). En el caso de la trama de información recogida del GPS, también podríamos eliminar información que no es interesante en este sistema, como la altitud, reduciendo así el tamaño de la cadena de caracteres.

Una vez que todas las variables se almacenan como enteros (*int*) que ocupan 2 bytes, para reducir el tamaño hasta 1 byte, que sería lo ideal, podemos utilizar la codificación ASCII. Es un código de caracteres estándar que utiliza 8 bits para representar una serie de caracteres imprimibles y no imprimibles.

Podemos utilizar esta codificación si, previamente, realizamos una serie de operaciones sobre cada variable. Obviamente perderemos precisión de nuevo con cada operación. En concreto, para utilizar esta codificación se deben cumplir estas condiciones:

- Todas las variables deben ser positivas. En este caso solo el ángulo puede tener un valor negativo, y en este caso el signo no es importante en la información necesaria.

- Todas las variables deben tener poseer un valor mínimo de 0 y valor máximo de 255. Las variables que estamos tratando tienen valores muy variados, pero únicamente en el caso de los sensores analógicos se supera el valor 255 (varían desde 0 hasta 1023). Podemos realizar un cambio de escala para que cumplan esta condición utilizando la función *map*() que implementa la biblioteca de programación de Arduino.

Finalmente, después de realizar estas operaciones, cada variable se podrá codificar mediante el estándar ASCII y tendrá un tamaño de un byte. De esta manera, cada variable ocupará el tamaño mínimo y podremos almacenar en cada trama la máxima cantidad de información. Para utilizar la codificación ASCII con Arduino,

podemos emplear la función *Serial.write ("*variable*"),* que escribe "variable" como un byte en el puerto serie, como se vio en el Capítulo 3.

6.4.3 Empaquetamiento y envío de los datos

El último paso consiste en la creación de las tramas que se enviarán a través del protocolo de comunicaciones UDP. Como se ha definido anteriormente, las tramas que se enviarán al servidor para procesar los datos tendrán un tamaño de 500 bytes, tamaño para el cual se obtiene la mayor tasa de transmisión.

En el capítulo 4 se estudió que UDP posee su propia trama que no podemos alterar, pero dentro del espacio reservado en la trama para la información a enviar podemos crear nuestra propia codificación de la trama y definir diferentes parámetros como la cabecera, número de sensores... Que servirá de ayuda para el post-procesamiento de las tramas recibidas en el servidor.

Puesto que las variables que hemos codificado hasta ahora ocupan 1 byte cada una, dentro de una trama UDP de 500 bytes podemos crear una serie de "subtramas" donde se empaquetarán cada uno de los muestreos realizados en cada bucle de programa. De esta manera dentro de una trama de 500 bytes se almacenarán "m" subtramas de "n" bytes cada una. Por otro lado, también es necesario definir en cada subtrama una cabecera si es necesario, para localizar su posición temporal, y un identificador de cada variable o sensor. Aunque en otros sistemas se suele utilizar un byte adicional para indicar el número de bytes que se van a enviar de cada variable, en este sistema concreto no se considera necesario ya que cada variable tiene un tamaño fijo en la trama.

Como cabecera para cada subtrama se ha considerado la mejor opción el tiempo transcurrido entre el muestreo anterior y el actual, ya que la finalidad de los datos enviados es ser almacenados y representados en una gráfica y, por tanto, es necesario conocer la posición del eje temporal para su representación. Además, esta cabecera también se podrá codificar mediante ASCII, ocupando de esta manera 1 byte.

Como identificador de cada variable se utilizará un byte, que corresponderá con uno de los caracteres imprimibles del código ASCII. También se utilizará un byte de identificación para la cabecera, que indica adicionalmente el comienzo de cada subtrama.

En el caso concreto que se está desarrollando en este capítulo, cada subtrama queda definida de la siguiente manera, donde cada campo está compuesto por 2 bytes (el identificador y el dato). En total, si consideramos dos mediciones de las revoluciones de la rueda (una por rueda) y dos sensores analógicos, el tamaño de la subtrama será 12 bytes, en el siguiente orden:

Tiempo	RPM1	RPM2	Ángulo	Analógico1	Analógico2

Adicionalmente, existirán tramas de mayor longitud que integrarán la información de la temperatura y del GPS cuando proceda. Estas tramas de mayor tamaño tendrán la siguiente forma:

GPS	Temperatura	Tiempo	RPM1	RPM2	Ángulo	Analógico1	Analógico2

En este caso, el inicio de la trama se detecta con el byte identificador de los datos recogidos GPS. A continuación la temperatura con 2 bytes al igual que el resto de sensores. En total, la subtrama tendrá un tamaño de 60 bytes.

En resumen, cada trama que se enviará de 500 bytes estará compuesta por los dos tipos de subtramas que se han definido. En el caso de que, para completar la trama de 500 bytes, se enviara una subtrama sin completar, en su lugar se enviará unos caracteres específicos para rellenar los 500 bytes fijos de cada trama (como por ejemplo "#"). De esta manera se evitarán posibles errores en el post-procesamiento de los datos en el servidor.

NOTA: El código fuente del programa diseñado en este capítulo se puede consultar en el ANEXO I.

6.5 PRUEBAS DE CAMPO DEL PROTOTIPO COMPLETO PARA CASO PARTICULAR

En último lugar se realizarán las pruebas realizadas en el Capítulo 5 sobre el prototipo final diseñado para el caso de la aplicación concreta. Estas pruebas utilizan los mismos procedimientos, por lo que no es necesario volver a describir los mismos. También se trabajó en las mismas condiciones de laboratorio, con cobertura de red móvil similar y sin ruido.

6.5.1 Tasa de transmisión de datos

Los resultados obtenidos de esta prueba de campo son:

Tamaño trama (bytes)	**Tasa de transmisión (bytes/s)**
500	1062

6.19 Resultados tasa de transmisión prototipo completo

Como podemos ver en los resultados obtenidos, la tasa de transmisión de datos es en este caso 1062 bytes/s, siendo la tasa máxima permitida 10183 bytes/s, es decir, este programa utiliza solo el 10,44% de la tasa de transmisión máxima. Esta velocidad obtenida es consecuencia del algoritmo para leer los sensores que se ha diseñado.

Por otro lado, se ha realizado una comparación de la información enviada sin codificación de los datos y con la codificación de datos ASCII descrita en el apartado anterior:

Método	Información enviada decodificada (bytes/s)
Sin codificación	1062
Codificación ASCII	2212

6.20 Comparativa codificación de datos

Como podemos observar, si no utilizamos codificación, la cantidad de información enviada coincide con la tasa de transmisión, mientras que realizando una codificación de los datos podemos aumentar en un 208% la información que contienen las tramas enviadas, ya que las tramas ocupan el mismo tamaño pero una vez realizada la descodificación de las mismas se obtiene un 208% más de información.

6.5.2 Tasa de error de transmisión de datos

En esta prueba de campo se obtuvo la siguiente información:

Tasa de transmisión (%tasa máxima)	Tramas recibidas correctamente (%)
10.44%	100%

6.21 Resultados tasa de error prototipo completo

Este resultado concuerda con los resultados obtenidos en el Capítulo 5, en el que se obtuvo que, para una tasa de transmisión por debajo del 25% de la tasa máxima, las tramas recibidas correctamente en el servidor son el 100%, al igual que en los resultados de esta prueba.

6.5.3 Tiempo de ejecución del programa

El tiempo de inicialización y de bucle o *loop* de este programa son:

Tiempo inicialización (ms)	Tiempo bucle (ms)
19835	471

6.22 Resultados tiempo de ejecución prototipo completo

En esta prueba no se puede destacar ninguna información destacable, ya que estos tiempos son consecuencia del programa implementado.

6.5.4 Utilización de memoria SRAM

Los datos recopilados sobre la utilización de los recursos (en este caso memoria SRAM) se presentan a continuación:

SRAM utilizada (bytes)	SRAM utilizada (% respecto total)
1461	17.83

6.23 Resultados utilización de memoria SRAM prototipo completo

Se puede apreciar un valor bajo de la cantidad de memoria SRAM utilizada, que es aproximadamente un 18%. Podemos afirmar que los modelos inferiores en la gama de Arduino también serían válidos para este sistema, siempre y cuando se disponga del número de pines necesarios para los sensores definidos en esta aplicación.

6.5.5 Tamaño del programa

Por último se ha medido el tamaño del programa, que se almacena en la memoria flash de Arduino:

Tamaño (bytes)	Espacio libre (bytes)	Espacio libre (%)
12374	245674	95.2

6.24 Resultados tamaño del programa prototipo completo

En este aspecto podemos afirmar que, al igual que en el caso de la memoria SRAM, solo se utiliza un 4.8% de la memoria flash total. En consecuencia, si nos fijamos en la memoria disponible en los modelos ofrecidos de la gama Arduino, todos podrían almacenar este programa y se podría reducir el presupuesto con un modelo inferior.

6.6 PRESUPUESTO

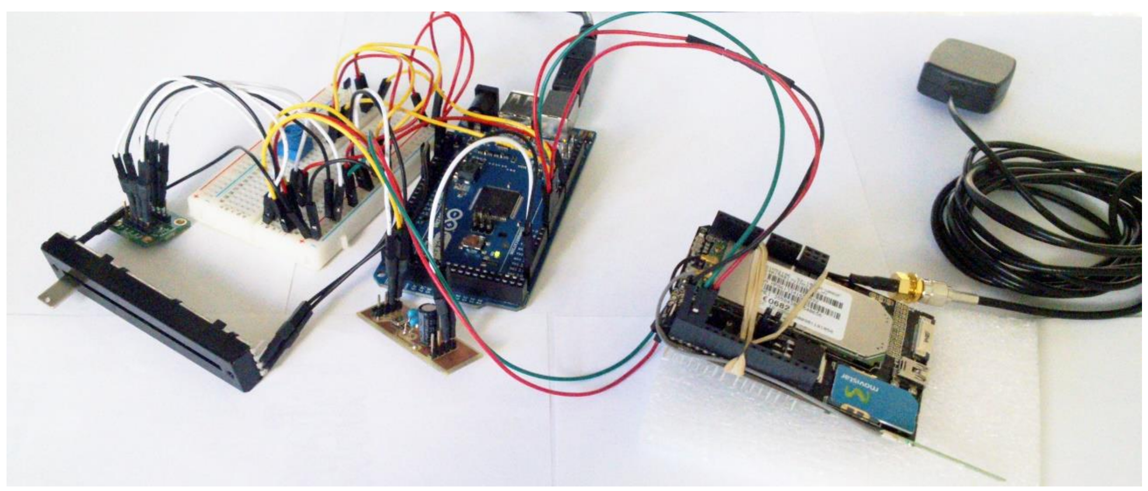
En la siguiente tabla se muestra el presupuesto detallado del sistema de telemetría en tiempo real para una motocicleta de competición diseñado en este capítulo. Los precios de la placa Arduino, Shield 3G+GPS y antenas necesarias han sido recogidos de la página oficial del distribuidor y fabricante *Cooking Hacks* (www.cooking-hacks.com/shop); los sensores utilizados, del distribuidor online de componentes electrónicos *RS-Components* (www.rs-components.com).

PRESUPUESTO (fecha 8/7/2014)			
Componente	**Cantidad (uds)**	**Precio (euros/ud)**	**Precio (euros)**
Arduino Mega ADK	1	51	51
Shield 3G+GPS	1	149	149
Antena GPRS-GSM-UMTS para exterior	1	13	13
Antena GPS para exterior	1	18	18
Sensor de efecto Hall AH183	2	1,08	2,16
Acelerómetro ADXL345 Evaluation Board	1	28,92	28,92
Sensor de presión y temperatura MPL3115A2	1	1,79	1,79
Potenciómetro deslizante Bourns PTE A Series	1	6,29	6,29
Potenciómetro radial Bourns PTV A Series	2	1,17	2,34
Total			**272,5 euros**

6.25 Presupuesto sistema telemetría moto competición

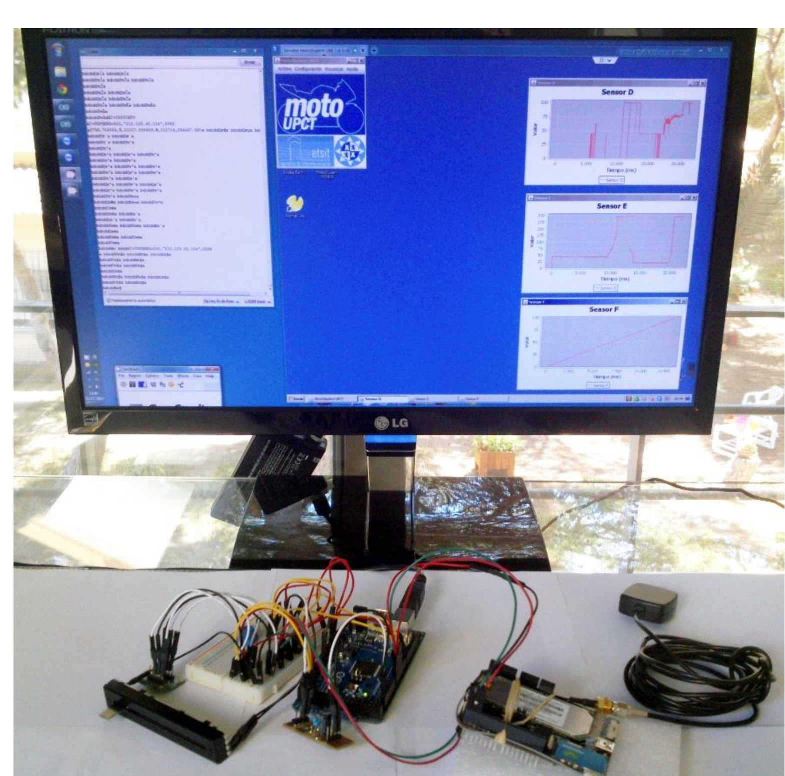
6.7 IMÁGNES DEL PROTOTIPO FINAL

En este punto se muestran algunas imágenes tomadas del banco de pruebas y el prototipo final con el que se ha trabajado durante este trabajo. En la primera imagen se muestra el banco de pruebas, formado por Arduino, Shield 3G+GPS y los distintos sensores que se han descrito en este capítulo:



6.26 Imagen banco de pruebas

A continuación se muestra una instantánea del prototipo conectado a un PC sobre el que se está ejecutando el entorno de programación de Arduino, que nos muestra lo que se está enviando a través del puerto serie, y la pantalla del servidor en el que se reciben los datos, que se muestra con más detalle más adelante:



6.27 Imagen banco de pruebas conectado a PC

La siguiente imagen muestra la pantalla del servidor donde se reciben los datos y son procesados y mostrados a través de gráficas mediante un software que está siendo desarrollado por un estudiante de Ingeniería de Telecomunicaciones, Pedro José Conesa:

6.28 Imagen del servidor que procesa y muestra los datos

7. CONCLUSIONES Y LÍNEAS FUTURAS

Durante este Trabajo de Fin de Grado se ha pretendido estudiar la validez del conjunto Arduino y Shield 3G+GPS para un sistema de telemetría en tiempo real, y la aplicación de esta plataforma al caso concreto de una motocicleta de competición. Se ha estudiado las interfaces de hardware compatibles con Arduino para la adquisición de datos, se han estudiado los protocolos de comunicaciones disponibles en Shield 3G+GPS, y se han realizado una serie de pruebas y evaluaciones sobre el conjunto, con el objetivo de seleccionar los parámetros óptimos de funcionamiento del dispositivo y determinar el alcance de la plataforma. Por último, se han aplicado los conocimientos adquiridos del conjunto para el diseño de un sistema de telemetría en tiempo real para una motocicleta de competición.

Las conclusiones obtenidas de cada uno los estos aspectos de este trabajo son:

- Hasta la fecha, los sistemas de telemetría en tiempo real que se habían diseñado eran de carácter específico, aplicados a casos concretos y que no permitían apenas modificaciones. En este trabajo se ha propuesto utilizar Arduino, de carácter abierto, que permite adaptar la plataforma aquí descrita a diferentes casos particulares. Por otro lado, el uso de redes móviles 3G para un sistema de telemetría en tiempo real también es una novedad con respecto a trabajos anteriores.

- Arduino es una plataforma que implementa las interfaces de *hardware* analógico, digital, puerto serie, I^2C y SPI, lo que hace que sea compatible con la gran mayoría de dispositivos (como sensores o actuadores) que podemos encontrar en el mercado. Estas interfaces están disponibles en todas las placas Arduino y se pueden utilizar de manera sencilla gracias al lenguaje Arduino, un lenguaje de programación común para todos sus microcontrolares. En este trabajo se ha estudiado el funcionamiento de estas interfaces de hardware en Arduino y se aplicado cada una mediante un ejemplo.

- De los protocolos de comunicaciones de Internet disponibles en Shield 3G+GPS que podemos utilizar para el envío de información, el que nos proporciona las mejores especificaciones para este sistema es UDP. En las pruebas de campo realizadas sobre el prototipo, se obtuvo una tasa de transmisión de datos máxima de 10 *Kilobytes/s* aproximadamente, limitada por el puerto serie de Arduino. Frente a la máximas especificaciones de la red móvil 3G (5.5 Mbps

según Cooking-Hacks, que equivalen a *704 KB/s*) el rendimiento obtenido del módem móvil 3G a través de Arduino es muy bajo (aproximadamente 1.4%). Por otro lado, se han diseñado los programas necesarios para utilizar los protocolos de comunicaciones UDP y FTP y geolocalización mediante satélites GPS, y basándonos en las pruebas de utilización de recursos realizadas, podemos afirmar que la capacidad de Arduino Mega 2560/ADK es suficiente para la elaboración de programas complejos de recopilación y posterior envío de datos.

- Se han aplicado los conocimientos adquiridos sobre la plataforma durante el trabajo para el diseño de un sistema de telemetría en tiempo real y geolocalización de una motocicleta de competición. Se han evaluado los sensores necesarios para medir los parámetros físicos principales del sistema utilizando todas las interfaces de *hardware* disponibles, y se ha diseñado un programa que realice de manera óptima la recopilación y envío de los datos a un servidor de Internet. A través de las pruebas de campo realizadas sobre el prototipo completo, podemos afirmar que la capacidad de la plataforma es válida para esta aplicación y otras más complejas, habiendo requerido solamente un 10% del a tasa de envío de datos máxima y una utilización de los recursos del microcontrolador baja.

Por ello, en vista de los resultados y las conclusiones obtenidas a lo largo del proyecto, puede concluirse el mismo, apostando por la viabilidad de continuar con el desarrollo de un sistema de telemetría en tiempo real mediante esta plataforma para la motocicleta de competición del equipo Moto UPCT.

En cuanto a las líneas futuras de este proyecto, están relacionadas con la mejora del rendimiento de la plataforma Arduino y Shield 3G+GPS. Sería interesante el estudio de la nueva placa Intel Galileo, creada por la unión de las empresas Intel y Arduino. Esta placa incorpora un procesador de 32 bits que es capaz de aumentar notablemente la capacidad de transmisión del puerto serie, que es la mayor limitación encontrada en las placas Arduino para esta aplicación. En consecuencia, sería posible aumentar el rendimiento de Shield 3G+GPS y alcanzar tasas de transmisión más altas.

También es interesante el estudio de la mejora de las funcionalidades que ofrece Arduino para un sistema de telemetría, como el almacenamiento en paralelo de los datos recopilados en una memoria extraíble SD. Aunque Shield 3G+GPS dispone de un lector de tarjetas micro-SD, este no es posible utilizarlo para este propósito ya que no existen comandos AT para realizar esta tarea.

Otro punto importante podría ser modificar la configuración de los sensores digitales a través de un servidor de Internet y estudiar si es posible la utilización de *shields* adicionales junto a Shield 3G+GPS para emplear otros buses de campo como el bus CAN, ampliamente utilizado en automoción.

ANEXO 1. CÓDIGO FUENTE SISTEMA DE TELEMETRÍA EN TIEMPO REAL PARA MOTOCICLETA DE COMPETICIÓN

```
//Bibliotecas necesarias
#include <SPI.h>
#include <Wire.h>
#include <avr/wdt.h>
//Direcciones registros MLP3115A2
#define STATUS 0x00
#define OUT_P_MSB 0x01
#define OUT_P_CSB 0x02
#define OUT_P_LSB 0x03
#define OUT_T_MSB 0x04
#define OUT_T_LSB 0x05
#define DR_STATUS 0x06
#define OUT_P_DELTA_MSB 0x07
#define OUT_P_DELTA_CSB 0x08
#define OUT_P_DELTA_LSB 0x09
#define OUT_T_DELTA_MSB 0x0A
#define OUT_T_DELTA_LSB 0x0B
#define WHO_AM_I 0x0C
#define F_STATUS 0x0D
#define F_DATA 0x0E
#define F_SETUP 0x0F
#define TIME_DLY 0x10
#define SYSMOD 0x11
#define INT_SOURCE 0x12
#define PT_DATA_CFG 0x13
#define BAR_IN_MSB 0x14
#define BAR_IN_LSB 0x15
#define P_TGT_MSB 0x16
#define P_TGT_LSB 0x17
#define T_TGT 0x18
#define P_WND_MSB 0x19
#define P_WND_LSB 0x1A
#define T_WND 0x1B
#define P_MIN_MSB 0x1C
#define P_MIN_CSB 0x1D
#define P_MIN_LSB 0x1E
#define T_MIN_MSB 0x1F
```

```
#define T_MIN_LSB 0x20
#define P_MAX_MSB 0x21
#define P_MAX_CSB 0x22
#define P_MAX_LSB 0x23
#define T_MAX_MSB 0x24
#define T_MAX_LSB 0x25
#define CTRL_REG1 0x26
#define CTRL_REG2 0x27
#define CTRL_REG3 0x28
#define CTRL_REG4 0x29
#define CTRL_REG5 0x2A
#define OFF_P 0x2B
#define OFF_T 0x2C
#define OFF_H 0x2D
#define MPL3115A2_ADDRESS 0x60

int8_t answer,answerGPS;
int onModulePin= 2;
char aux_str[50];
char port[ ]="5558";
char gps_data[50];
int CS=8;
int angle;
//Direcciones registros ADXL345
char POWER_CTL = 0x2D;
char DATA_FORMAT = 0x31;
char DATAX0 = 0x32;
char DATAX1 = 0x33;
char DATAY0 = 0x34;
char DATAY1 = 0x35;
char DATAZ0 = 0x36;
char DATAZ1 = 0x37;
int pot1 = A0;
int pot2 = A1;
int analog1,analog2;
int temperature;
char values[10];
int y,z;
int temp,difftemp;
int counter;
int Ha1 = 4;
int Ha2 = 5;
int tempHa1,tempHa2;
int StateHa1, StateHa2;
int rpm1=100;
int rpm2=100;
intcounterchar,counterchartemp;
intcountersec,counterfreq;
void setup(){
pinMode(onModulePin, OUTPUT);
pinMode(CS, OUTPUT);
```

```
Serial.begin(115200);
power_on();
delay(10);
sendATcommand("AT+CPIN=2223", "OK", 2000);
while( (sendATcommand("AT+CREG?", "+CREG: 0,1", 500) ||
sendATcommand("AT+CREG?", "+CREG: 0,5", 500)) == 0 );
delay(10);
sendATcommand("AT+CGSOCKCONT=1,\"IP\",\"telefonica.es\"",
"OK", 2000);
sendATcommand("AT+CGPSURL=\"supl.google.com:7276\"","OK",10
00);
sendATcommand("AT+CGPSSSL=0","OK",1000);
answerGPS = sendATcommand("AT+CGPS=1,2","OK",1000);
if (answerGPS == 0)
{
Serial.println("Error starting the GPS");
}
sprintf(aux_str, "AT+NETOPEN=\"UDP\",%s", port);
answer = sendATcommand(aux_str, "Network opened", 20000);
if (answer == 1)
{
Serial.println("Network opened");
}
else
{
Serial.println("Error opening the nertwork");
software_Reboot();
}
//Bus SPI
SPI.begin();
SPI.setDataMode(SPI_MODE3);
digitalWrite(CS, HIGH);
writeRegister(DATA_FORMAT, 0x01);
writeRegister(POWER_CTL, 0x08);
//Bus I2C
if(IIC_Read(WHO_AM_I) == 196)
Serial.println("MPL3115A2 online!");
else
Serial.println("No response MPL3115A2");
setModeAltimeter();
setOversampleRate(7);
enableEventFlags();
//Inicializar
temp = millis();
tempHa1 = millis();
tempHa2 = millis();
countersec=5;
counterfreq=0;
}
void loop(){
Serial.flush();
```

```
counterchar=500;
//Sensores lentos, 1 vez por segundo temp y gps
if(countersec==5)
{
//GPS
if(answerGPS==1)
{
answer = sendATcommand2("AT+CGPSINFO","+CGPSINFO:",10);
if (answer == 1)
{
counter = 0;
do{
while(Serial.available() == 0);
gps_data[counter] = Serial.read();
counter++;
}
while((gps_data[counter          -          1]          !=
'\r')&&(counter<44));//Recortamos los
gps_data[counter] = '\0';
}
}
}
//Mandamos comando AT para UDP
sendATcommand2("AT+UDPSEND=500,\"212.128.45.104\",5558",
"OK", 10);
//Si hemos leido el gps y la temperatura, la enviamos
if(countersec==5)
{
countersec = 0;
if(gps_data[0] == ',')
{
}
else
{
Serial.print("g");
Serial.print(gps_data);
counterchar = counterchar - 45; //Tamaño gps = 44 + tamaño
'g' = 45
}
temperature = readTemp();
Serial.print("t");
Serial.write(temperature);
counterchar = counterchar - 2;
}
//Sensores rápidos
do
{
//Sensores Hall, 200 mediciones por segundo
StateHa1 = digitalRead(Ha1);
StateHa2 = digitalRead(Ha2);
if(StateHa1 == true)
```

```
{
tempHa1 = millis()-tempHa1 ;
rpm1 = 60000/difftemp+0;
}
if(StateHa2 == true)
{
tempHa2 = millis()-tempHa2;
rpm2 = 60000/difftemp+0;
}
delay(5);
if(counterfreq==2)
{
difftemp = millis()-temp;
temp =millis();
counterfreq=0;
//Potenciometros
analog1 = map(analogRead(pot1),0,1023,1,255);
analog2 = map(analogRead(pot2),0,1023,1,255);
//Acelerometro
readRegister(DATAX0, 6, values);
y = ((int)values[3]<<8)|(int)values[2];
z = ((int)values[5]<<8)|(int)values[4];
angle = atan(y/z)*57.296;
if(angle<0)
{
angle=-angle;
}
//Trama sensores - escribimos en puerto serie
counterchartemp = counterchar - 10;
//¿Cabe la trama?
if(counterchartemp>=0)
{
Serial.print("a");
Serial.write(difftemp);
Serial.print("b");
Serial.write(rpm1);
Serial.print("c");
Serial.write(rpm2);
Serial.print("d");
Serial.write(angle);
Serial.print("e");
Serial.write(analog1);
Serial.write("f");
Serial.write(analog2);
counterchar = counterchar - 12;
}
//Si no cabe se envían # hasta completar el máximo
else
{
do
{
```

```
Serial.print("#");
counterchar = counterchar - 1;
}while(counterchar>0);
}
}
counterfreq++;
}while(counterchar >= 0);
countersec++;
}
void writeRegister(char registerAddress, char value){
digitalWrite(CS, LOW);
SPI.transfer(registerAddress);
SPI.transfer(value);
digitalWrite(CS, HIGH);
}
void readRegister(char registerAddress, int numBytes, char
* values){
char address = 0x80 | registerAddress;
if(numBytes > 1)address = address | 0x40;
digitalWrite(CS, LOW);
SPI.transfer(address);
for(int i=0; i<numBytes; i++){
values[i] = SPI.transfer(0x00);
}
digitalWrite(CS, HIGH);
}
void power_on(){
uint8_t answer=0;
answer = sendATcommand("AT", "OK", 2000);
if (answer == 0)
{
digitalWrite(onModulePin,HIGH);
delay(3000);
digitalWrite(onModulePin,LOW);
while(answer == 0){
answer = sendATcommand("AT", "OK", 2000);
}
}
}
int8_t      sendATcommand(char*      ATcommand,      char*
expected_answer1,
unsigned int timeout){
uint8_t x=0, answer=0;
char response[100];
unsigned long previous;
memset(response, '\0', 100);
delay(5);
while( Serial.available() > 0) Serial.read();
Serial.println(ATcommand);
x = 0;
previous = millis();
```

```
do{
if(Serial.available() != 0){
response[x] = Serial.read();
x++;
if (strstr(response, expected_answer1) != NULL)
{
answer = 1;
}
}
}while((answer == 0) && ((millis() - previous) < timeout));
return answer;
}
int8_t      sendATcommand2(char*      ATcommand,      char*
expected_answer1,
unsigned int timeout){
uint8_t x=0, answer=0;
char response[100];
unsigned long previous;
memset(response, '\0', 100);
while( Serial.available() > 0) Serial.read();
Serial.println(ATcommand);
x = 0;
previous = millis();
do{
if(Serial.available() != 0){
response[x] = Serial.read();
x++;
if (strstr(response, expected_answer1) != NULL)
{
answer = 1;
}
}
}while((answer == 0) && ((millis() - previous) < timeout));
return answer;
}
float readTemp()
{
toggleOneShot();
int counter = 0;
while( (IIC_Read(STATUS) & (1<<1)) == 0)
{
if(++counter > 100) return(-999); //Error
delay(1);
}
//Leer registros temperatura
Wire.beginTransmission(MPL3115A2_ADDRESS);
Wire.write(OUT_T_MSB); //Direccion del registro
Wire.endTransmission(false);
Wire.requestFrom(MPL3115A2_ADDRESS, 2);//Solicitar 2 bytes
//Esperar a datos disponibles
counter = 0;
```

```
while(Wire.available() < 2)
{
if(++counter > 100) return(-999); //Error
delay(1);
}
byte msb, lsb;
msb = Wire.read();
lsb = Wire.read();
float templsb = (lsb>>4)/16.0; //Temperatura, en grados
Celsius
float temperature = (float)(msb + templsb);
return(temperature);
}
//Modo altimetro
void setModeAltimeter()
{
byte tempSetting = IIC_Read(CTRL_REG1);
tempSetting |= (1<<7);
IIC_Write(CTRL_REG1, tempSetting);
}
//Establecer Ratio Oversample
void setOversampleRate(byte sampleRate)
{
if(sampleRate > 7) sampleRate = 7;
sampleRate <<= 3;
byte tempSetting = IIC_Read(CTRL_REG1);
tempSetting &= 0b11000111;
tempSetting |= sampleRate;
IIC_Write(CTRL_REG1, tempSetting);
}
//Borrar el bit de OST
void toggleOneShot(void)
{
byte tempSetting = IIC_Read(CTRL_REG1);
tempSetting &= ~(1<<1);
IIC_Write(CTRL_REG1, tempSetting);
tempSetting = IIC_Read(CTRL_REG1);
tempSetting |= (1<<1);
IIC_Write(CTRL_REG1, tempSetting);
}
//Habilitar la medición de temperatura y presión
void enableEventFlags()
{
IIC_Write(PT_DATA_CFG, 0x07);
}
//Función que lee en el bus I2C
byte IIC_Read(byte regAddr)
{
Wire.beginTransmission(MPL3115A2_ADDRESS);
Wire.write(regAddr); //Escribir dirección registro
Wire.endTransmission(false);
```

```
Wire.requestFrom(MPL3115A2_ADDRESS, 1);//Solicitar dato
return Wire.read();//Devolver valor recibido
}
//Función que escribe en el bus I2C
void IIC_Write(byte regAddr, byte value)
{
Wire.beginTransmission(MPL3115A2_ADDRESS);//Empezar
transmisión
Wire.write(regAddr);//Escribir dirección registro
Wire.write(value);//Escribir valor
Wire.endTransmission(true);//Fin transmisión
}
void software_Reboot()
{
wdt_enable(WDTO_15MS);
while(1)
{
}
}
```

BIBLIOGRAFÍA

[1] Arduino, documentación, [Consulta:21 Junio 2014], disponible en: http://arduino.cc

[2] Siquier Batler, Guillem; Sistema de Adquisición de datos de una motocicleta de competición, [Consulta:22 Junio 2014], disponible en: http://upcommons.upc.edu/handle/2099.1/11136

[3]Cárdenas Valencia, Álvaro Hernán; Prototipo de un sistema de telemetría y control para seguridad en vehículos, soportado en redes móviles, [Consulta:22 Junio 2014], disponible en: http://200.21.98.67:8080/jspui/bitstream/handle/10785/1761/CDMIST20.pdf?sequence=3

[4] Cornejo Ortega, Ángel Danilo; Sistema de telemetría en tiempo real mediante redes móviles GSM para el monitoreo de los parámetros básicos de un vehículo, [Consulta:22 Junio 2014], disponible en: http://www.dspace.ups.edu.ec/bitstream/123456789/1114/23/UPS-CT001987.pdf

[5] Casillas, Sonia; Telemetría inalámbrica por red celular GSM, [Consulta:22 Junio 2014], *disponible en:* http://www.uaz.edu.mx/eniinvie/old/eninvie2008/INST_2.pdf

[6] Arduino, documentación interfaces de hardware, [Consulta:23 Junio 2014], disponible *en:* http://playground.arduino.cc/Main/InterfacingWithHardware

[7] Arduino, tutoriales, [Consulta:23 Junio 2014], disponible en: http://arduino.cc/en/Tutorial/HomePage

[8] Atmel, documentación Atmega2560, [Consulta:23 Junio 2014], disponible en: http://www.atmel.com/Images/doc2549.pdf

[9] Doménech Asensi, Ginés; García-Bravo García, José; Zapata Pérez, Juan; Redes de Comunicaciones Industriales, 2014, Universidad Politécnica de Cartagena

[10] Cooking-Hacks, documentación Shield 3G+GPS, [Consulta:26 Junio 2014], disponible en: http://www.cooking-hacks.com/documentation/tutorials/arduino-3g-gprs-gsm-gps

[11] SIMCOM, Manual comandos AT SIM52118, [Consulta:26 Junio 2014], disponible en: http://www.cooking-hacks.com/skin/frontend/default/cooking/pdf/SIM5218_AT_command_manual.pdf

[12] Diodes, AH182/83 datasheet, [Consulta:28 Junio 2014], disponible en: http://diodes.com/datasheets/AH182_AH183.pdf

[13] Analog, ADXL345 datasheet, [Consulta:28 Junio 2014], disponible en: http://www.analog.com/static/imported-files/data_sheets/ADXL345.pdf

[14] Freescale,MPL3115A2 datasheet, [Consulta:28 Junio 2014], disponible en: http://cache.freescale.com/files/sensors/doc/data_sheet/MPL3115A2.pdf?pspll=1&Parent_nodeId=1307914182184706466838&Parent_pageType=product

[15] Bourns, PTV A series datasheet, [Consulta:28 Junio 2014], disponible en: http://www.bourns.com/data/global/pdfs/pte.pdf

[16] Bourns, PTE A Series datasheet, [Consulta:28 Junio 2014], disponible en: *http://www.bourns.com/data/global/pdfs/ptvptt.pdf*