ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE
TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Trabajo Fin de Grado

Desarrollo de un sistema de telemetría basado en la plataforma Intel Edison

AUTOR: Andrés Gil Calín

DIRECTOR: Francisco Javier Garrigós Guerrero

Febrero de 2016

Agradecimientos

Me gustaría dedicar este trabajo a mis padres, Encarna y Pepe, por su apoyo incondicional a lo largo de mi vida. También a mi compañero Víctor, por su inestimable amistad. A mis amigos y familia, por estar siempre a mi lado. Y por último a Francisco Javier Garrigós Guerrero, por la ayuda prestada a la hora de realizar este trabajo y por su implicación en el mismo.

Autor	Andrés Gil Calín
E-Mail	sr.calin@hotmail.com
Director	Francisco Javier Garrigós Guerrero
E-Mail Director	javier.garrigos@upct.es
Título del TFG	Desarrollo de un sistema de telemetría basado en la plataforma Intel Edison

Resumen:

La idea fundamental del presente trabajo es la de diseñar un dispositivo electrónico basado en la plataforma Intel Edison, capaz de realizar tanto una telemetría en tiempo real como un almacenamiento de la información obtenida para su posterior procesado.

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, que está desarrollando un prototipo de moto de competición para el campeonato europeo de MotoStudent, para la implementación de un sistema de telemetría en la moto que permita almacenar datos recogidos de los sensores durante la carrera y procesarlos posteriormente.

El pasado año se llevó a cabo el proyecto inicial del sistema de telemetría de la moto. Este primer diseño, basado en la plataforma Arduino Mega, ofrecía ciertos inconvenientes a la hora de ser llevado a la práctica y tan sólo se llegó a realizar un primer prototipo en el laboratorio. Con el desarrollo de este proyecto se pretende dar un salto evolutivo al sistema inicial, utilizando la plataforma Intel Edison, mucho más potente y versátil. Además, se dotará al sistema de nuevas opciones y prestaciones que ofrece el cambio de plataforma, a fin de llevarlo a la práctica y diseñar un sistema de telemetría plenamente funcional.

Finalmente, este trabajo también pretende realizar una comparación en cuestión de prestaciones entre la plataforma Arduino y la Intel Edison, a fin de justificar que esta última ofrece las especificaciones adecuadas para un sistema de este tipo, en el que las velocidades de adquisición y transmisión de datos son cruciales.

Titulación	Grado en Ingeniería de Sistemas de Telecomunicación
Departamento	Electrónica, Tecnología de Computadoras y Proyectos

Índice

Índice de figuras

Capítulo 1: Introducción

1.1 Planteamiento inicial

La temática del presente Trabajo de Fin de Grado es el desarrollo de un dispositivo de telemetría basado en la plataforma Intel Edison [1]. El concepto de telemetría puede ser definido como la capacidad que posee un determinado sistema o dispositivo para medir magnitudes físicas y/o químicas, con el fin de transmitir la información medida de forma remota a un sistema receptor. Debido a la gran utilidad que posee, se ha convertido en una tecnología imprescindible en multitud de campos como la industria, la medicina o el deporte.

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 Moto UPCT, formado por estudiantes de esta universidad y que participa en la competición MotoStudent [2], promovida por la fundación Moto Engineering Foundation (MEF) entre universidades españolas y europeas. Dicha competición consiste en diseñar y desarrollar un prototipo de motocicleta de competición de 250 centímetros cúbicos y 4 tiempos, apta para superar unas determinadas pruebas de evaluación que se llevan a cabo en la Ciudad del Motor de Aragón.

Uno de sus objetivos pendientes es el de implementar un sistema de telemetría plenamente funcional para su motocicleta. El pasado año se llevó a cabo un proyecto inicial de dicho sistema, pero este primer diseño, basado en la plataforma Arduino Mega [3], ofrecía ciertos inconvenientes a la hora de ser llevado a la práctica y tan sólo se llegó a realizar un primer prototipo en el laboratorio. Con el desarrollo de este proyecto se pretende dar un salto evolutivo al sistema inicial, utilizando la plataforma Intel Edison, mucho más potente y versátil. Además, se dotará al sistema de nuevas opciones y prestaciones que ofrece el cambio de plataforma, a fin de llevarlo a la práctica y diseñar un sistema de telemetría plenamente funcional.

Un sistema de telemetría en tiempo real se puede dividir en varias partes. Por un lado, debe de haber un dispositivo capaz de recopilar información de las variables físicas y/o químicas deseadas mediante sensores. Por otro lado, debe de tener también un dispositivo de transmisión de datos que envíe la información recogida por dichos sensores de forma remota a un sistema de recepción, que en sí mismo es otra de las partes del sistema completo. Por último, debe de haber un software que almacene los datos, los procese y los visualice.

Según la normativa vigente de la competición, no está permitido el uso de telemetría en tiempo real. La solución más inmediata es implementar un sistema de almacenamiento de información que trabaje de forma paralela al envío de datos en tiempo real, y que dicha información pueda ser procesada posteriormente por el mismo software que se encarga de la información que se envía en tiempo real.

Ante tal división y normativa, el presente Trabajo de Fin de Grado se centra en la recopilación y almacenamiento de datos y su posterior transmisión a un sistema de recepción.

Finalmente, este trabajo también pretende realizar una comparación en cuestión de prestaciones entre la plataforma Arduino y la Intel Edison, a fin de justificar que esta última ofrece las especificaciones adecuadas para un sistema de este tipo, en el que las velocidades de adquisición y transmisión de datos son cruciales.

1.2 Objetivos

Para la realización de este trabajo resulta necesario definir un conjunto de objetivos específicos que más tarde quedarán desglosados, que son los siguientes:

- Adaptación a la plataforma Intel Edison.
- Estructuración de la rutina de adquisición de datos de los sensores en función de las frecuencias de refresco elegidas para cada uno.
- Diseño óptimo de la trama que contendrá la información adquirida, cumpliendo un compromiso con el software de recepción de datos.
- Implementación física del sistema.

Cabe destacar también que una vez creada la trama, como se ha mencionado con anterioridad, el sistema ha de ser capaz de enviarla a través de internet en tiempo real a un servidor donde se ejecuta un software encargado de la recepción, visualización y análisis de los datos, diseñado expresamente para dicha función. Por otro lado, el sistema de telemetría también debe poder almacenar dichas tramas de forma local, a fin de ser analizadas posteriormente por la misma aplicación o por otro software creado en un futuro.

1.3 Fases

El desarrollo del proyecto se ha llevado a cabo a lo largo de unos catorce meses desde la toma de contacto con el sistema diseñado con anterioridad, en diciembre de 2014, hasta la redacción y entrega de la presente memoria en febrero de 2016.

La primera fase consistió en la lectura y familiarización con los proyectos anteriores para la posterior puesta en marcha del sistema de telemetría inicial. En grupo, junto con los compañeros David Hernández Mejías, Víctor Huéscar López y Francisco Javier Martínez Abellán, se reprodujo el conjunto completo y se realizaron pruebas de funcionamiento (más adelante se explica más detalladamente los pormenores de esta fase y las pruebas llevadas a cabo).

La segunda fase incluyó la primera toma de contacto con la plataforma Intel Edison y la puesta a punto de la misma, partiendo desde cero. Este proceso se demoró en cierta medida debido a que la plataforma era por aquél entonces bastante novedosa, y no había gran cantidad de documentación.

La tercera fase constó del diseño de la etapa de adquisición de datos. Esta fase ha ido evolucionando de forma paralela a las demás, puesto que se ha ido perfeccionando la adquisición de los datos de acuerdo a la inclusión de más o menos sensores, al tiempo de refresco que se les ha querido dar, y a la soltura que se ha ido adquiriendo a la hora de programar la plataforma.

La cuarta fase trató el tema de la comunicación en tiempo real con un sistema de recepción. En este apartado se incluyen tanto la justificación del uso de un protocolo en concreto, como la capacidad de transmisión del sistema en función de la plataforma y la programación misma de la etapa de comunicación.

La quinta fase consistió en la implementación de la etapa de almacenamiento local de la información, estudiando el funcionamiento del proceso a seguir y las posibilidades que ofrece, y la creación de la etapa.

Cabe destacar que, como el trabajo que trata el software de recepción ha ido evolucionando de forma paralela a este trabajo, la creación y el perfeccionamiento de la trama que contiene los datos de interés ha ido evolucionando de forma que el compromiso entre ambos trabajos siempre se mantenga.

La sexta fase comprendió la implementación física y el testeo del sistema completo en una bicicleta de pruebas, adquirida por la Universidad Politécnica de Cartagena para dicho propósito, y la evaluación de los resultados proporcionados por el prototipo final.

La séptima fase se ocupó del estudio de las posibles líneas futuras para este trabajo, mediante el estudio de las posibles funcionalidades que se le podrían añadir al sistema gracias a la versatilidad de la plataforma utilizada.

La octava y última fase consistió en el desarrollo de esta memoria.

La figura 1 expone un pequeño diagrama que representa la evolución temporal a lo largo del desarrollo de este trabajo:

Figura 1 - Diagrama temporal

1.4 Estructura de la memoria

La estructura de esta memoria va a ser la siguiente:

- En primer lugar, una pequeña introducción que incluye un planteamiento inicial, los objetivos principales y las distintas fases del desarrollo del trabajo.
- Un capítulo que explica el estado de la técnica en la telemetría actual y el diseño del sistema de telemetría inicial llevado a cabo con anterioridad.
- Un tercer capítulo que incluye todos los elementos que van a componen el sistema completo, tanto a nivel de hardware como de software.
- El siguiente capítulo hace una breve descripción de los posibles protocolos de comunicación en Internet, acompañada de la justificación de la elección tomada.
- El quinto capítulo detalla la creación del programa que se cargará en la placa.

- En el sexto capítulo se expone la instalación del sistema en la bicicleta de pruebas.
- El séptimo capítulo trata el análisis de prestaciones del sistema completo.
- Tras éste, un capítulo con las conclusiones y las posibles líneas futuras para la mejora del presente sistema.
- Por último, anexos, referencias y bibliografía utilizada.

Capítulo 2: Estado de la técnica

2.1 Introducción a la telemetría

El concepto de telemetría, como ya se ha explicado en el capítulo anterior, engloba un conjunto de procedimientos para medir magnitudes físicas y químicas desde una posición que dista del lugar donde se producen los fenómenos a analizar. Aparte de eso, la telemetría también comprende el posterior envío de la información hacia el sistema de recepción.

El término telemetría procede de las palabras griegas *tele*, que significa remoto, y *metron*, que significa medida. No obstante, cabe destacar que aunque el término telemetría se suele utilizar para hablar de sistemas remotos sin ningún tipo de unión cableada entre emisor y receptor, en algunas bibliografías también es posible encontrarlo para definir sistemas de medición cableados.

Un sistema de telemetría está constituido usualmente por un transductor que cumple la función de dispositivo de entrada, un medio de transmisión, ya sea cableado o inalámbrico, dispositivos de procesamiento de señal, y dispositivos de almacenamiento o visualización de datos.

El dispositivo de entrada puede desglosarse en dos elementos fundamentales. El primero de estos elementos es el sensor, que es el elemento sensible que mide las variaciones de estado de las magnitudes bajo estudio. El segundo es el transductor, que es el encargado de convertir el valor de la magnitud medida en la señal eléctrica correspondiente.

El medio de transmisión, como ya se ha mencionado, puede ser establecido de forma cableada o de forma inalámbrica. Dentro del primer grupo podríamos encontrar desde el cableado de cobre típicamente usado en telefonía hasta uniones de fibra óptica, pasando por uniones de par trenzado como las usadas en redes de computadoras. Dentro del segundo grupo podemos encontrar las ondas de radio o las comunicaciones basadas en estándares de comunicación tan conocidos como Bluetooth o Wi-Fi.

Los dispositivos de procesamiento de señal son la parte principal del sistema de recepción. Son los encargados de realizar el análisis de los datos recibidos y su posterior transformación a la magnitud que más convenga, según si el destino final de la información es ser almacenada o ser visualizada.

Los dispositivos de almacenamiento o visualización de datos son el componente final del sistema. Para el almacenaje de información se suelen utilizar unidades de memoria como las tarjetas microSD que serán usadas en este trabajo. Para la visualización

se puede utilizar tanto la herramienta de software que implementa el sistema de recepción como algún tipo de salida en forma de pantalla, conectada de forma directa al sistema de medida.

2.2 Aplicaciones actuales de la telemetría

En este apartado se van a presentar algunos de los numerosos campos en los que se utiliza de forma continua esta tecnología en la actualidad.

2.2.1 Fórmula 1 y MotoGP

La telemetría llegó a la Fórmula 1 en la década de los años 80 [4], no obstante por aquella época los sistemas de telemetría se encontraban en una fase completamente embrionaria y experimental. Fue ya entrados los años 90 cuando aparecieron los primeros sistemas de telemetría plenamente funcionales. Gracias a esta innovación, los equipos pasaron de no poder obtener ningún dato del coche en tiempo real, a poder ver prácticamente todo lo que sucedía en el coche gracias a los múltiples parámetros de los que podían disponer.

Este avance marcó un antes y un después en la Fórmula 1, y los equipos que no lo usaron se quedaron atrás en la competición. Tanto es así que en las temporadas de los años 1992 y 1993, los famosos Williams (que fueron junto con los McLaren los primeros coches en implementar la telemetría) y su avanzada electrónica consiguieron más del 60% de las victorias de esos años, ganando ambos años 10 de las 16 carreras y bajándose del podio en pocas ocasiones.

Además, gracias a este nuevo sistema los ingenieros no solo recibían información del monoplaza, sino que en 2002 hasta se permitió que en caso de que algo no fuera bien, pudieran variarlo desde el box, por lo que el piloto no tenía que hacer nada al respecto. Pero en 2003, la mejora de estos sistemas llegó a tal punto que la FIA (Federación Internacional del Automóvil) decidió prohibir que los parámetros del monoplaza fuesen manipulados desde el box.

Por este motivo, en la Fórmula 1 actual el ingeniero desde el box se encarga de transmitir información sobre los parámetros del coche al piloto, y es este quien mediante los numerosos botones de su volante varía estos parámetros para intentar ganarle unas décimas al crono.

En estos sistemas en concreto, la telemetría utiliza la banda UHF (banda del espectro electromagnético que ocupa el rango de frecuencias de 300 MHz a 3 GHz) y conexiones punto a punto entre el coche y el receptor. Uno de los problemas de este tipo de comunicación es que no puede haber ningún obstáculo entre ambos puntos, por ello se trabaja con el envío de información a corta distancia mediante el uso de distintas antenas

provistas por McLaren Electronic Systems o MES y situadas en puntos estratégicos del circuito, asegurando así una comunicación constante. Esas antenas actúan como repetidores que redirigen la señal desde el monoplaza hasta el centro de datos de cada equipo, que suele ser un camión situado en el paddock, y desde ahí se manda al muro de boxes para que los ingenieros puedan ver como se está comportando el coche en la pista. Además, hay ciertos casos en los que también se envía información directamente a la fábrica de la escudería vía satélite.

Cada monoplaza lleva incorporada una pequeña y aerodinámica antena situada en el morro, diseñada y situada de la forma más estratégica posible para optimizar el envío de datos teniendo en cuenta numerosos factores . En la figura 2 podemos ver un ejemplo de este tipo de antena.

Figura 2 - Antena de telemetría para monoplaza

La antena receptora va conectada a una unidad principal compuesta por dos subunidades distintas (emisión y recepción), y que actúa como módem y encripta y desencripta los datos que envían y se reciben. 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, propiedad de McLaren Electronic Systems [5], que permite la visualización de los datos mediante el uso de complejas gráficas. En la figura 3 puede verse la unidad receptora CBR-610, uno de los modelos de la marca, y en la figura 4 la unidad transmisora complementaria CBT-610.

Figura 3 - CBR-610

Figura 4 - CBT-610

El tercer elemento que entra en juego es la centralita electrónica o ECU (Electronic Control Unit) de la que tanto se habla año tras año. Es la encargada de recoger todos los datos de los sensores, y además de estándar, es obligatoria para todos los coches de la parrilla y está fabricada por McLaren. A continuación, en la figura 5 se presenta el modelo TAG-320, usado desde su lanzamiento entre los años 2013 y 2014.

Figura 5 - ECU TAG 320

En la figura 6 puede verse un esquema del sistema completo.

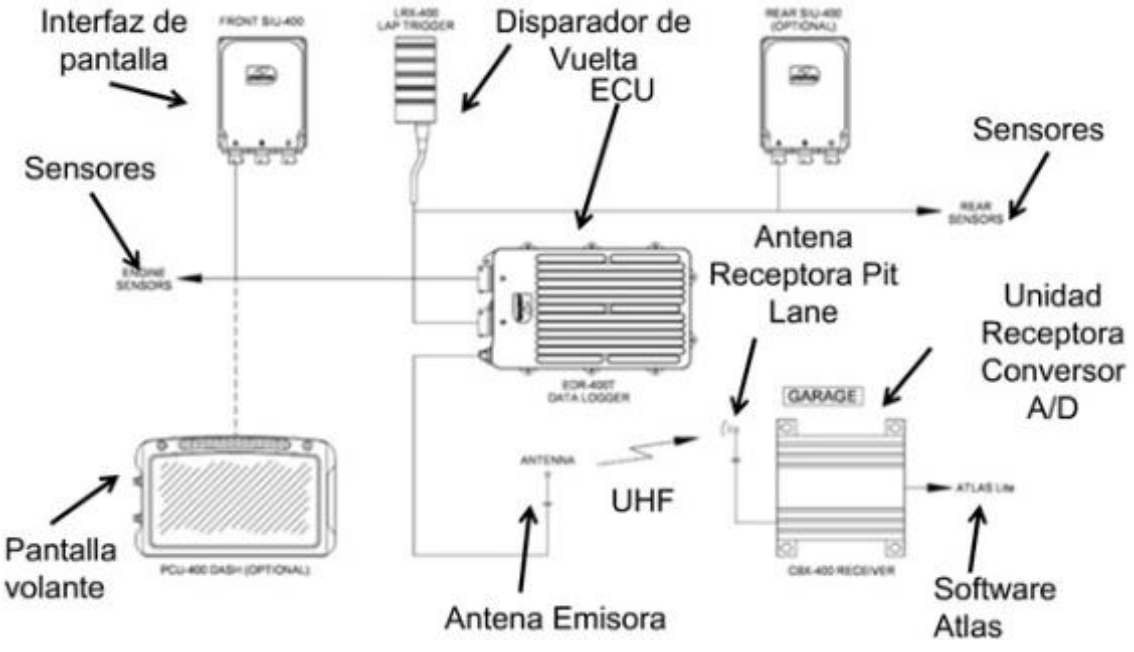

Figura 6 - Esquema del sistema en un monoplaza

En MotoGP, por otro lado, se lleva a cabo un sistema de telemetría totalmente distinto. Por normativa y como ya se ha comentado previamente en este trabajo, la telemetría ha de ser offline, lo que supone la recopilación de información mientras el piloto está en pista para posteriormente descargar la información en el sistema receptor y procesarla o visualizarla.

La información se transmite a través de canales, y 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.

En las figuras 7 y 8 puede verse un ejemplo de visualización de datos obtenidos por telemetría. En concreto se trata de los datos que recogió el sistema Tech Air de Alpinestars y que la misma empresa publicó, a raíz de la caída que sufrió Marc Márquez en los entrenamientos libres en el circuito de Mugello en el año 2013 [6].

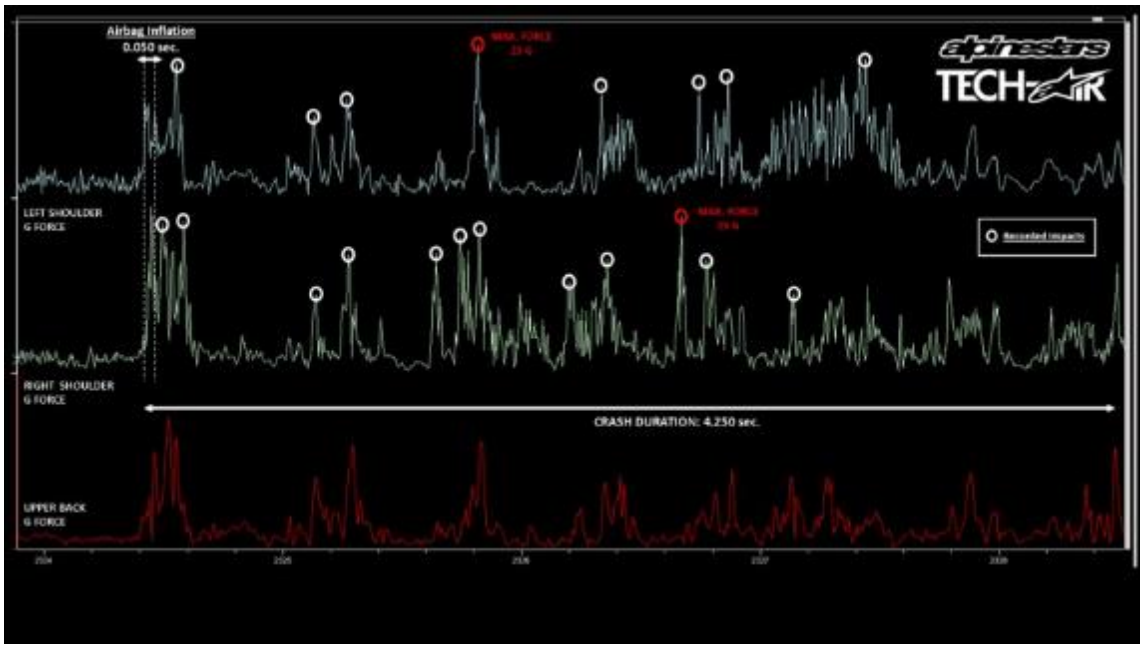

Figura 7 - Gráfica de telemetría 1

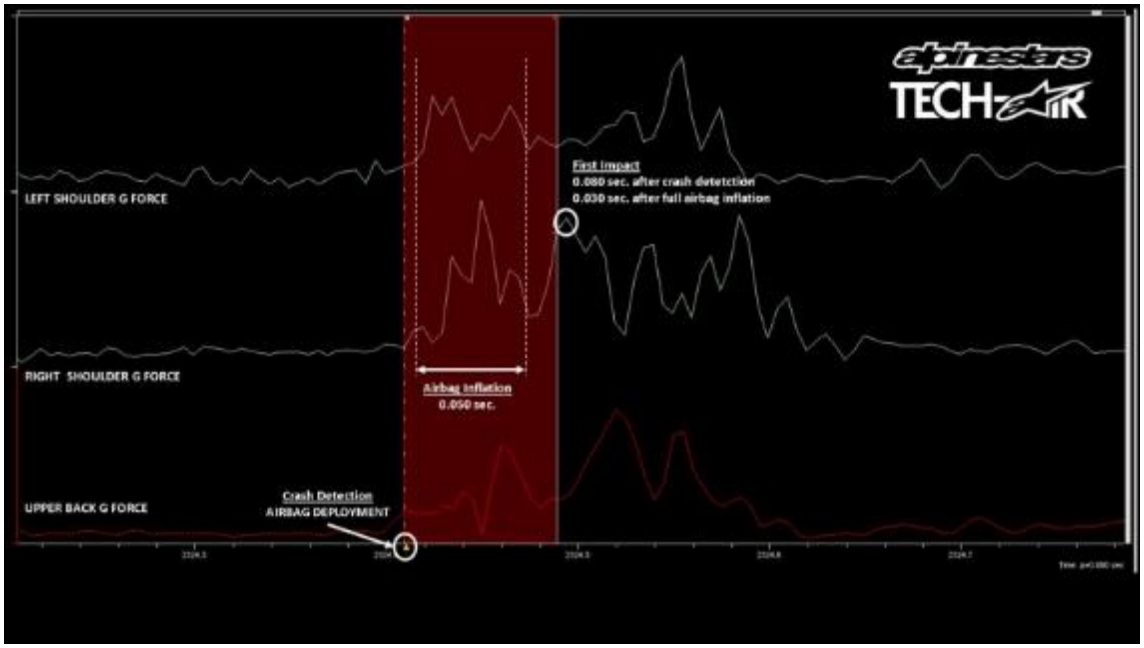

Figura 8 - Gráfica de telemetría 2

2.2.2 Medicina

En el campo de la medicina, la telemetría es utilizada en numerosas y muy variadas situaciones. Está basada en distintas técnicas y tiene el objetivo fundamental de monitorizar constantes vitales u otros parámetros del paciente, de la forma menos invasiva posible, sobre todo cuando el período de observación dura días, semanas o incluso meses.

En algunos casos en particular existen sistemas de telemetría y alarma muy modernos, que hacen uso de comunicaciones móviles y permiten incluso realizar un seguimiento domiciliario de pacientes crónicos o que se encuentren en un largo periodo de convalecencia. De otra forma tendrían que permanecer en entornos hospitalarios de forma permanente para estar al tanto de su estado en todo momento, o en su defecto asistir de forma muy continuada, lo que supondría una discontinuación del control de su evolución. Además, la telemetría supone un ahorro de costes y un incremento de comodidad para el paciente.

Una de las aplicaciones más extendidas es el uso de la telemetría para registrar eventos electrocardiográficos a distancia. Los radiotransmisores se conectan al paciente mediante cinco electrodos que se adhieren a la piel, lo que permite a los pacientes libertad de movimiento. Una unidad central refleja los electrocardiogramas de los pacientes a los que está conectado y guarda los eventos importantes cada 24 horas.

En un estudio realizado por los doctores Pérez Titos y Oliver Ramos del Hospital Universitario Médico-Quirúrgico Virgen de las Nieves de Granada [7], se concluyó que el 80% de los pacientes estudiados registraron eventos importantes, siendo el 23% de estos eventos de gravedad.

En la figura 9 podemos ver un ejemplo de electrocardiograma obtenido por telemetría.

Figura 9 - Ejemplo de ECG obtenido por telemetría

2.2.3 Domótica

Se conoce como domótica al conjunto de sistemas capaces de automatizar una vivienda, aportando servicios de gestión energética, seguridad, bienestar y comunicación, y que pueden estar integrados por medio de redes interiores y exteriores de comunicación, cableadas o inalámbricas, y cuyo control goza de cierta ubicuidad, desde dentro y fuera del hogar.

El término domótica viene de la unión de las palabras *domus* (que significa casa en latín) y *tica* (de automática, palabra en griego que significa 'que funciona por sí sola').

Los inicios de la domótica tuvieron lugar con el protocolo X10 [8]. Este protocolo fue creado por la empresa escocesa Pico Electronics durante los años 1977 y 1978, fruto de un proyecto de control de equipos de audio realizado para BSR (Birmingham Sound Reproducers) en USA, siendo en el año 1978 cuando se introdujo en el mercado americano de la mano de las empresas Radio Shack y Sears.

Fue la primera tecnología domótica en aparecer y sigue siendo la más ampliamente disponible, principalmente por su sencilla instalación, sin necesidad de cableado adicional.

Desde ese entonces hasta nuestros días, las innovaciones en el campo de la domótica son incontables, y pueden encontrarse desde sistemas básicos hasta sistemas increíblemente complejos con el objetivo común de mejorar la calidad de vida del usuario.

En la figura 10 se presentan algunos módulos del mencionado protocolo X10, con los que comenzó a surgir la domótica.

Figura 10 - Módulos X10

2.2.4 Robótica

La robótica es la rama de la ingeniería mecánica, ingeniería eléctrica, ingeniería electrónica y ciencias de la computación que se ocupa del diseño, construcción, operación, disposición estructural, manufactura y aplicación de los robots.

La robótica combina múltiples disciplinas como son la mecánica, la electrónica, la informática, la inteligencia artificial, la ingeniería de control y la física. Otras áreas importantes en robótica son el álgebra, los autómatas programables, la animatrónica y las máquinas de estados.

Algunos de los sistemas robóticos más complejos se encargan de la manipulación de materiales radioactivos o residuos tóxicos, la exploración espacial e incluso de la localización de minas terrestres.

Como puede apreciarse son tareas muy complicadas para las que se requiere una precisión total. Por este motivo, uno de los elementos más utilizados es el sensor láser de telemetría basado en tecnología LIDAR (de los términos *light* y *radar*) [9]. El sensor emite un rayo láser sobre el entorno, que rebota sobre los objetos de manera no especular para determinar la distancia a un objeto. El receptor (dentro del propio sensor) recibe el rayo devuelto, y mediante el tiempo de vuelo se calcula la distancia al objeto apuntado. Dada la gran velocidad de la luz, el rayo es devuelto en muy poco tiempo, lo que permite hacer un barrido 2D o 3D para obtener más datos del entorno.

En la figura 11 puede verse un esquema de un sensor LIDAR, y en la figura 12 un ejemplo de imagen adquirida con dicha tecnología.

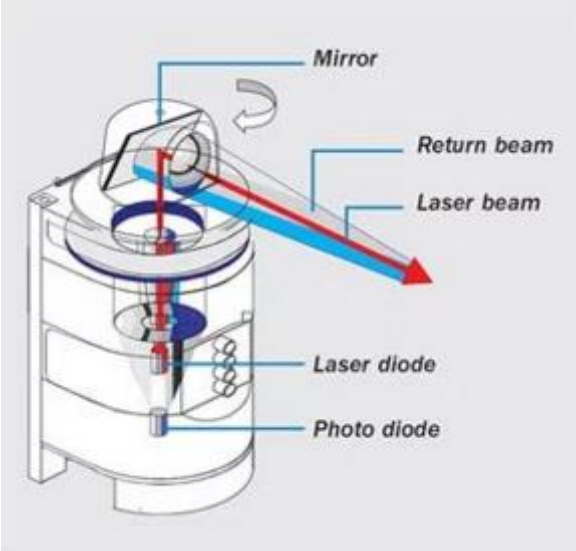

Figura 11 - Esquema de un sensor LIDAR

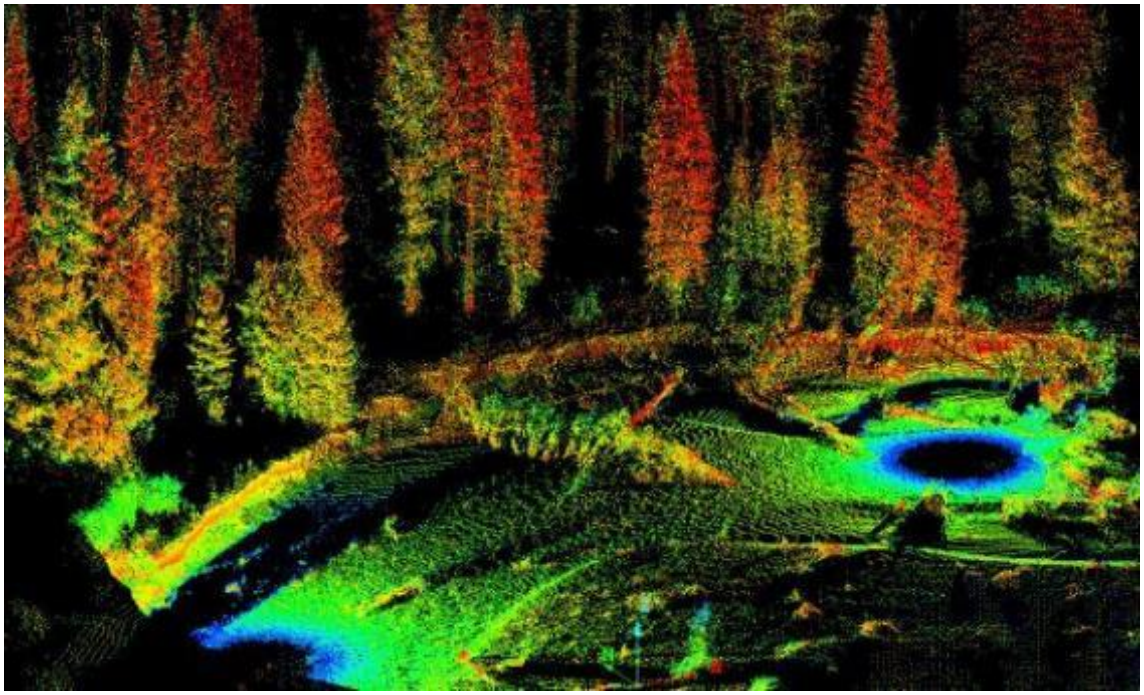

Figura 12 - Imagen 3D obtenida mediante un sensor LIDAR

2.3 Sistema de telemetría inicial

Como ya se ha comentado con anterioridad en esta memoria, con intención de introducir la telemetría en el proyecto de MotoUPCT, el año pasado se llevaron a cabo tres trabajos complementarios que constituían, en conjunto, un sistema completo de telemetría para una motocicleta de competición. Estos tres trabajos abarcaban de forma individual cada uno de los tres grandes componentes de la telemetría, a saber obtención y envío de datos, transmisión a través de un canal, y recepción y visualización de la información.

El primero de ellos, "Desarrollo de un dispositivo de telemetría basado en la plataforma Arduino y Shield 3G+GPS" [10], fue llevado a cabo por Pedro Celestino López Jiménez. En él se describe el diseño de un dispositivo utilizando la plataforma Arduino, capaz de recibir la información de unos sensores instalados en una motocicleta de competición, de procesar dicha información y de enviarla. En este diseño se optó por el hardware libre Arduino por su disponibilidad en el mercado y por poseer la capacidad suficiente para la tarea objetivo, además de disponer de la posibilidad de ampliar su funcionalidad con shields añadidos. La información a enviar era generada por sensores sencillos que fueron elegidos por el autor para este fin, no se trataba de sensores reales para uso en motocicletas. Para el envío de la información y la extracción de datos de posición se optó por un shield de Arduino de 3G+GPS que, gracias al uso de una tarjeta de un operador de telefonía móvil, proporcionaba un ancho de banda suficiente para el propósito del trabajo. Por último se diseñó una trama de datos, en común con el autor del sistema de recepción, que codificaba y albergaba la información de los sensores.

La figura 13 presenta los resultados obtenidos por este proyecto.

Figura 13 - Diseño del sistema de telemetría inicial

Como ya se ha comentado con anterioridad, este trabajo se considera el precedesor del presente TFG.

El segundo de estos trabajos, "Implementación de un sistema de radiocomunicaciones para la transmisión de la telemetría de una moto de carreras" [11], fue realizado por María Belén Pérez Muñoz. Este trabajo tenía como objetivo el uso de la herramienta R&S CMW500 para el despliegue de una red privada de GSM en un circuito que proporcionase soporte físico al canal a través del cual se enviaría la información recopilada por los sensores en la motocicleta. Se llevaron a cabo pruebas utilizando tanto la tecnología GSM como la tecnología UMTS con distintos protocolos de envíos de

paquetes. En las figuras 14 y 15 se presentan tanto el equipo utilizado como una captura de los resultados obtenidos, respectivamente.

Figura 14 - CMW500

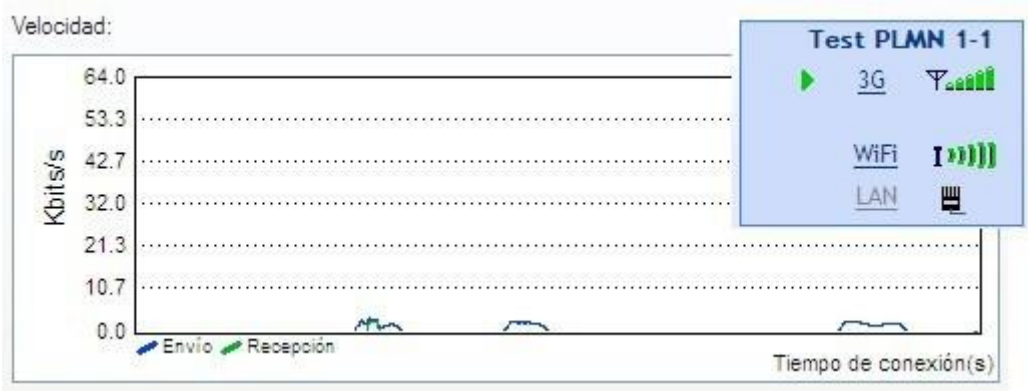

Figura 15 - Resultados obtenidos por el CMW500

Por último, el trabajo bajo el nombre "Desarrollo de un software de telemetría para el control de una moto de carreras" fue llevado a cabo por Pedro José Conesa Sánchez [12]. Tenía como propósito el desarrollar el software que recibiese, decodificase y representase los datos enviados por la telemetría de una motocicleta de competición. Todo el software desarrollado en este proyecto se llevó a cabo usando como lenguaje de programación Java. El resultado fueron dos herramientas, la primera de ellas un programa que se encargaba de recibir los datos, procesarlos, almacenarlos en una base de datos en MySQL y de representarlos, y la segunda era la encargada de llevar a cabo simulaciones.

En las figuras 16 y 17 pueden verse dos representaciones de datos obtenidos, una de cada una de las herramientas.

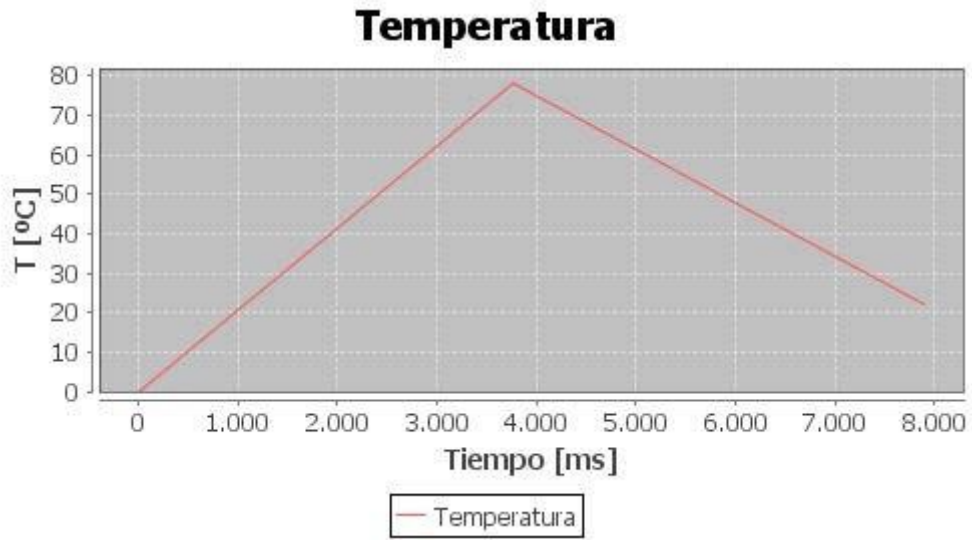

Figura 16 - Representación de datos gracias a la primera herramienta

Figura 17 - Representación de posición gracias a la segunda herramienta

Capítulo 3: Elementos que componen el sistema

Una gran parte del presente trabajo ha consistido en el estudio del funcionamiento y las prestaciones de la plataforma Intel Edison [13], así como los distintos *shields* y sensores que serán utilizados a lo largo del desarrollo del prototipo.

Los elementos que constituyen el sistema se exponen a continuación.

3.1 Intel Edison

El módulo Intel Edison es un SoC (System on Chip) basado en la arquitectura Silvermont de 22 nm [14], que incluye un procesador Intel Atom de doble núcleo a 500 MHz e *Hyper-Threading* [15] doble, y un microcontrolador Intel Quark de 32 bits a 100 MHz. Además, posee conectividad Wi-Fi y Bluetooth LE, que hacen de este módulo un componente ideal para lo que se actualmente se conoce como el Internet de las Cosas (IoT). Está dotado de un conector de 70 pines que permite conectar una amplia variedad de *shields* denominados *blocks*, que pueden ir apilados unos encima de otros y que permiten ampliar la funcionalidad del módulo en base a las necesidades del proyecto a realizar. Esto conlleva un soporte para más de 30 interfaces de entrada/salida estandarizados que se pueden conectar a través de dicho conector.

Posee una unidad de almacenamiento flash de 4 GB eMMC, y 1 GB de RAM LPDDR3 POP.

Figura 18 - Módulo Intel Edison

Como puede apreciarse en la figura 18, uno de las mayores ventajas de esta plataforma es su reducido tamaño (35.5 × 25.0 × 3.9 mm).

En el apartado del software, el módulo Intel Edison soporta una distribución de Linux denominada Yocto, Arduino, Python, Node.js y Wolfram, aparte de ser compatible

con el IDE de Arduino, Eclipse y otros entornos de desarrollo denominados Intel SDK. Estas herramientas comunitarias de código abierto facilitan el desarrollo de aplicaciones de terceros en pos de crear aplicaciones de cara al consumidor.

3.2 Intel Edison Kit para Arduino

El Intel Edison Kit para Arduino proporciona los pines de Arduino 1.0 y conectores estándar como lo son un micro USB conectado a una UART, un puerto USB OTG (On-The-Go) que puede ser conmutado con un segundo conector micro USB, un soporte para tarjetas microSD y un puerto de alimentación DC [16]. En la figura 19 puede verse el aspecto que tiene el Intel Edison Kit para Arduino.

Figura 19 - Intel Edison Kit para Arduino

Como un Arduino Uno, el Intel Edison Kit para Arduino permite tener 20 pines digitales de entrada/salida, 6 de los cuáles pueden ser usados como entradas analógicas. El Intel Edison Kit para Arduino tiene 4 salidas PWM (modulación por ancho de pulsos) que pueden ser configuradas mediante el uso de *jumpers* a cualquiera de los 6 pines que soportan PWM en el Arduino UNO (pines 3, 5, 6, 9, 10 y 11).

El Intel Edison Kit para Arduino está diseñado para ser compatible tanto en hardware como en software con los shields de Arduino diseñados para el Arduino Uno R3. Los pines digitales 0 a 13 (y los pines adyacentes de referencia analógica, referencia de entrada/salida y tierra, AREF, IOREF y GND, respectivamente), las entradas analógicas 0 a 5, el cabezal de alimentación, el cabezal ICSP (In-Circuit Serial Programming), y los pines del puerto UART (0 y 1), están en la misma situación que en el Arduino Uno R3.

Las entradas/salidas digitales y los pines analógicos pueden ser configurados para operar tanto a 5 V como a 3.3 V, y las salidas pueden ser fuente o sumidero de 24 mA a 3.3 V y 32 mA a 5 V. El voltaje de entrada va desde 7 V a 15 V.

En la figura 20 aparece esquematizada la distribución de las interfaces y pines en el Intel Edison Kit para Arduino.

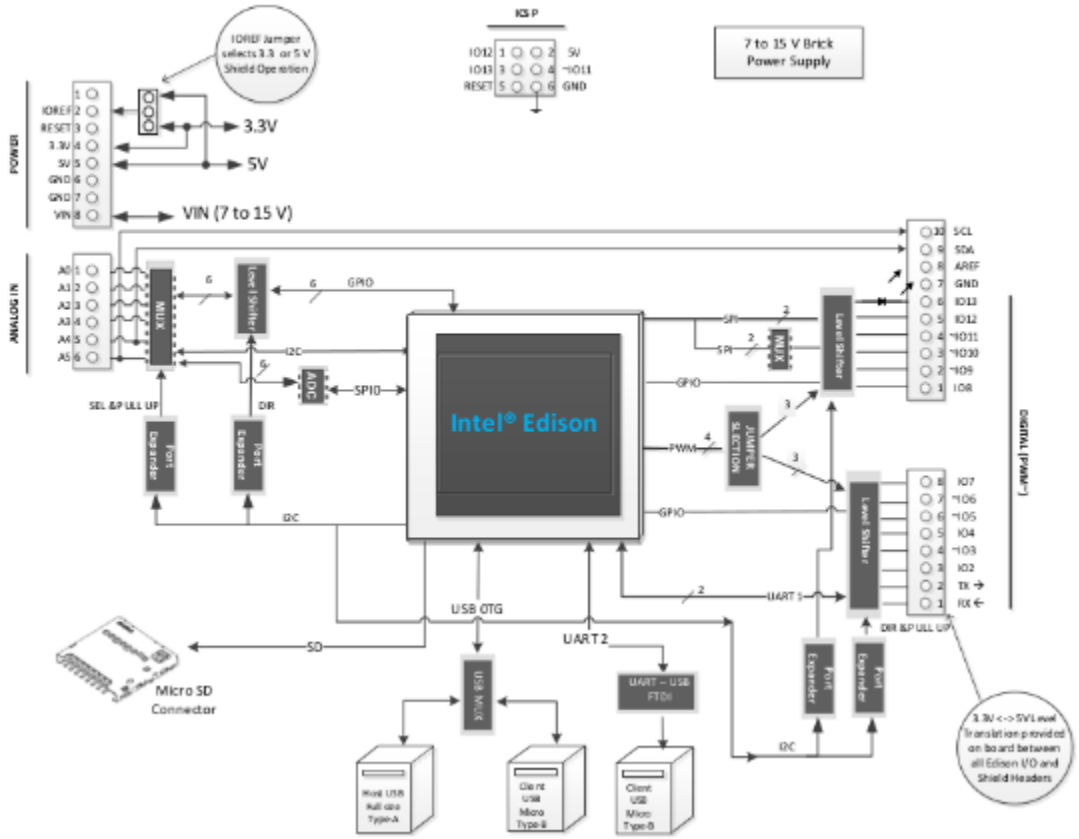

Figura 20 - Esquemático de interfaces del Intel Edison Kit para Arduino

3.2.1 *Jumpers*

Es importante también hablar de algunos de los *jumpers* que posee esta placa de expansión, pues su disposición determina el funcionamiento de la misma.

J1 es el encargado de la potencia principal de entrada. Como ya se ha mencionado, puede se le puede aplicar de 7 V a 15 V y se encuentra físicamente en la entrada VIN, que corresponde con el pin 8 de la interfaz *Power*.

J2 es el conector de la batería. Si se pretende alimentar la placa de expansión mediante una batería de ión litio recargable, hay que conectarla a este *jumper*. Cuando se conecta una batería de ión litio recargable, el Kit para Arduino recargará la batería cuando

se alimenta la placa mediante los *jumpers* J1, J3 o J16, que corresponden a la entrada VIN ya mencionada, al micro USB conectado a la UART y al conector de alimentación, respectivamente.

J8 es el encargado de seleccionar el rango de voltaje de salida que proporciona la placa (0 a 3.3 voltios si seleccionamos la posición de AREF o 0 a 5 voltios si seleccionamos la posición de IOREF).

Los *jumpers* J11 y J12 se utilizan para configurar las 4 GPIO (entradas/salidas de propósito general). Estos *jumpers* permiten a las 4 fuentes PWM ser encaminadas a cualquiera de los 6 pines de Arduino Uno que soportan PWM. Moviendo los *jumpers* de la configuración por defento hacen que los pines de entrada/salida no puedan ser usados.

J6 es un puerto OTG plenamente compatible con USB . Si se conecta un cable tipo A macho en este puerto, el módulo Intel Edison se conectará al dispositivo como anfitrión. Viene conmutado gracias al *switch* SW1 con el *jumper* J16 ya mencionado en este apartado, por lo que si se conecta un cable micro USB en J16, el módulo Intel Edison se conectará al dispositivo como huésped. Esta conmutación debe ser paralela a la posición del *switch* ya que si se conecta el cable en uno de los *jumpers*, pero el *switch* está en la posición contraria, no funcionará.

En la figura 21 puede apreciarse con más claridad y en una vista frontal, la disposición de cada uno de los *jumpers* mencionados.

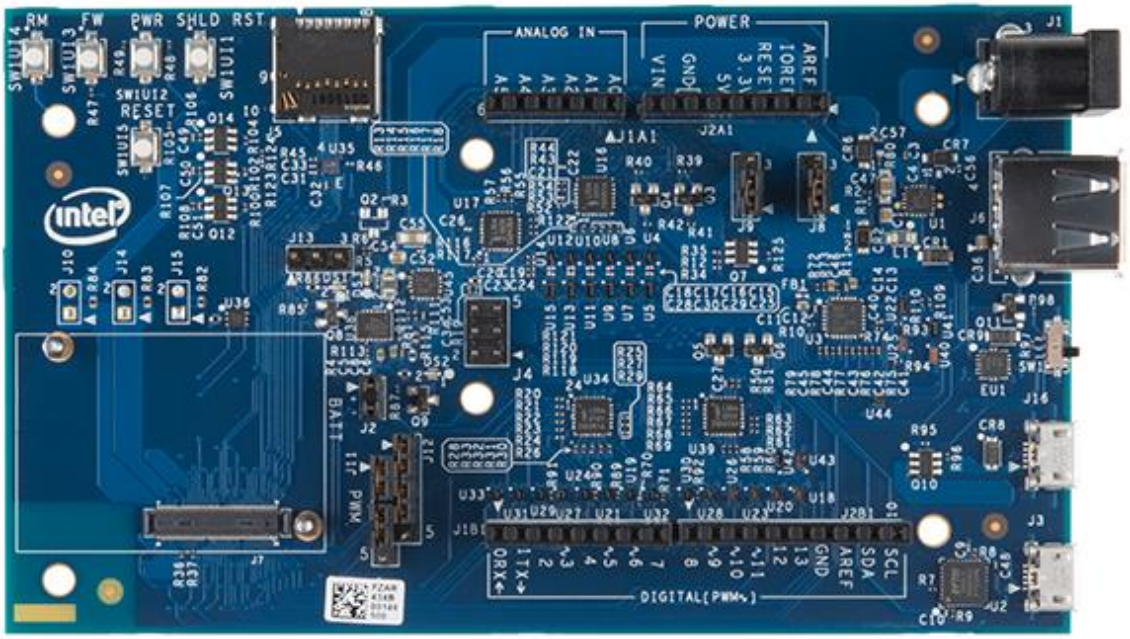

Figura 21 - Frontal del Arduino Breakout Kit o Intel Edison Kit para Arduino

3.2.2 Potencia

Una de las características que convierte esta plataforma en un gran activo es su bajo consumo. En general no consume más de 200 mA con picos cortos de 600 mA durante una transmisión Wi-Fi.

Si el módulo Intel Edison se está alimentando mediante un adaptador externo, la potencia va a un conversor DC-DC y es reducida a 5 V. Esta potencia va a un cargador de batería IC, que limita el voltaje de salida a 4.4 V. Este voltaje se encuentra en el rango seguro para el módulo VSYS, que va de 3.15 V a 4.4 V. Esto permite a la Intel Edison funcionar con una batería de ión litio estándar.

En el Intel Edison Kit para Arduino, el cargador *onboard* está configurado para detectar la fuente de potencia de entrada y limitarla a 500 mA si está conectado al puerto USB micro B o hasta 1 A si está conectado al *jack* DC. El cargador está programado para cargar a 100 mAh y está diseñado para cargar baterías de ión de litio estándar con 4.2 voltios de voltaje de carga máximo.

Para aplicaciones de bajo consumo (*shields* que operan a 3.3 voltios, por ejemplo), se puede conectar a J2 una batería de ión de litio como se ha especificado en el apartado anterior, para proporcionar 100 mA de 3.3 V al *shield*.

Algunas consideracionea a tener en cuenta sobre la distribución de potencia en el Intel Edison Kit para Arduino son:

- Debido a la junta tórica del diodo del conversor DC-DC y a la entrada VBUS, el voltaje real que irá al shield a partir de 5 voltios estará ligeramente por debajo de ese nivel. En el caso del VBUS el voltaje puede ser hasta de 4.4 voltios (4.75 voltios propios del VBUS mas una caída de 0.3 voltios en el diodo). En el caso de un adaptador externo, será de 4.7 V.

- Cuando el sistema esté configurado como anfitrión siempre se requerirá el uso de un adaptador de potencia externo.

La descripción de los pines implicados en la provisión de potencia desde o hacia la placa es la siguiente:

- VIN: El voltaje de entrada que va a la placa. Es posible acceder al voltaje proporcionado mediante el *jack* DC a través de este pin.

- 5V: Este pin saca 5 voltios regulados por el regulador de la placa.

- 3.3V: Este otro pin saca 3.3 voltios regulados por el regulador *onboard*, que también proporciona potencia al microcontrolador Quark.

- GND: Pines de tierra.

- IOREF: Este pin proporciona un voltaje de referencia con el que opera el microcontrolador. Puede ir de 0 a 5 V si el *jumper* J8 está configurado para IOREF.

- AREF: Este otro pin proporciona un voltaje de referencia del conversor ADC. Puede ir de 0 a 3.3 V si el *jumper* J8 está configurado para IOREF.

3.2.3 Botones

Los diferentes botones que contiene este *shield* son:

- *System reset* (SW1UI5): Pulsar este botón hace que el módulo Intel Edison se resetee, junto a los expansores de entrada/salida, configurando todos los pines del shield en estado de alta impedancia sin *pull-up*.

- *Shield reset* (SW1UI7): Pulsar este botón pondrá en bajo la señal de reset del *shield*. Esto no afecta el estado del módulo Intel Edison ni sus entradas/salidas.

- *Power button* (SW1UI2): Este botón está configurado por software. Pulsando y manteniendo el boton producirá distintos efetos dependiendo del estado del módulo Intel Edison y la duración de la pulsación:

  a) Si el dispositivo está completamente apagado, pulsar y mantener el botón durante 3 segundos arrancará el dispositivo y el módulo Intel Edison.

  b) Si el dispositivo está en ejecución, pulsar y mantener el botón durante más de 2 y menos de 7 segundos pondrá el dispositivo en modo AP (access point).

  c) Si el dispositivo está en ejecución, pulsar y mantener el botón durante 10 o más segundos hará que el módulo se apague por completo.

3.2.4 Entrada y salida

En cuanto a las distintas interfaces de entrada y salida que posee el *shield*, tenemos la interfaz serie, la interfaz digital de entrada/salida, la interfaz PWM [17], la interfaz SPI [18], la interfaz LED, la interfaz de entradas analógicas y la interfaz I^2C [19].

3.2.4.1 Interfaz serie

Uno de las características más importantes de esta plataforma es la versatilidad de su interfaz serie [20]. Además, esa versatilidad es uno de los factores que más ha

condicionado su incorporación a este trabajo, en pos de sustituir a la plataforma Arduino y solventar de ese modo las carencias que esta poseía.

La interfaz serie se puede dividir en tres partes. Teóricamente existiría una cuarta que comprendería los puertos virtuales (VCP o *Virtual communications port*), pero no se van a incluir en la descripción ya que no van a ser utilizados en este trabajo.

En primer lugar, está el puerto serie básico, que puede recibir distintos nombres (*Multi-gadget, Firmware Programming* o *Serial console*). Está localizado en el conector micro USB J16, como ya se ha comentado en un apartado anterior de este mismo trabajo. Su nombre en el software de Arduino es `Serial` y su nombre en Linux es `/dev/ttyGS0`.

Este puerto es el que se usa para programar la parte hardware de la plataforma Intel Edison correspondiente a Arduino, además de ser el puerto serie por defecto para la consola serie del IDE de Arduino. Si los drivers están instalados correctamente, este puerto también permite acceder a la partición de almacenamiento USB y a la interfaz de red RNDIS *(Remote Network Driver Interface Specification*). Por último, destacar que sólo funcionará cuando el conmutador SW1 este configurado en modo huésped, como ya se ha comentado previamente.

El segundo puerto serie es el que recibe el nombre de UART1. UART son las siglas en inglés de *Universal Asynchronous Receiver-Transmitter*, que define al dispositivo encargado de controlar los puertos y dispositivos serie. La velocidad máxima que alcanzaba la UART de la plataforma Arduino alcanzaba únicamente los 115200 baudios de velocidad de transferencia, lo que daba lugar a no poder utilizar la velocidad máxima que ofrecía el *shield* 3G+GPS. Esta UART permite alcanzar tasas de hasta 3686000 baudios, lo que elimina el cuello de botella que surgía con la anterior implementación.

Este puerto está localizado en los pines 0 (RX) y 1 (TX) del *shield*, y su nombre en el software de Arduino es `Serial1`. Por su parte, Linux le asigna el nombre `/dev/ttyMFD1`.

En términos generales, es el puerto capaz de crear una interfaz TTL básica de 5 voltios para conectarse a un dispositivo externo.

Por último, tenemos el puerto UART2. Sus características son similares a la UART1. Sus diferencias radican en su localización, ya que se encuentra en el *jumper* J3, y en su función, ya que no es utilizado como puerto TTL de propósito general como el UART1, sino como puerto de depuración Linux.

El nombre que recibe en el software de Arduino es `Serial2`, mientras que en Linux recibe el nombre `/dev/ttyMFD2`.

3.2.4.2 Interfaz digital de entrada/salida

Esta interfaz comprende los pines digitales 0 a 13 y los pines analógicos A0 a A5 del Intel Edison Kit para Arduino. Estos pines pueden ser usados como entradas o salidas digitales, utilizando las funciones `pinMode()`, `digitalWrite()`, y `digitalRead()`.

3.2.4.3 Interfaz PWM

Como ya se ha comentado con anterioridad, hay 4 canales PWM disponibles que pueden ser configurados a través de los *jumpers* a cualquiera de los pines digitales 3, 5, 6, 9, 10 u 11 del Intel Edison Kit para Arduino. La resolución de la interfaz PWM es de 8 bits y los 4 canales mencionados se encuentran en los pines 33, 35, 37 y 39 del módulo Intel Edison.

3.2.4.4 Interfaz SPI

La interfaz SPI (*Serial Peripheral Interface*) se usa principalmente para la transferencia de información entre circuitos integrados en equipos electrónicos. Se trata de un protocolo síncrono, y la sincronización y la transmisión de datos se realiza por medio de 4 señales.

La primera señal es la denominada SCLK o reloj. Es el pulso que marca la sincronización, y con cada pulso se lee o se envía un bit.

La segunda señal es la llamada MOSI (*Master Output Slave Input*). Es la salida de datos del dispositivo maestro y la entrada de datos al esclavo.

La tercera señal se denomina MISO (*Master Input Slave Input*). Es el proceso inverso al que realiza la señal anterior, salida de datos del dispositivo esclavo y entrada de datos al maestro.

La cuarta y última señal es la SS, y su función es seleccionar un esclavo o indicar al maestro que debe decirle al esclavo que se active.

Estas cuatro señales se encuentran en los pines 13 (SCLK), 11 (MOSI), 12 (MISO) y 10 (SS), del Intel Edison Kit para Arduino. Por otra parte, en el módulo Intel Edison, la interfaz SPI está disponible en los pines 51 (SS1), 53 (SS2), 55 (SCLK), 57 (MOSI) y 59 (MISO). Como se aprecia, el módulo Intel Edison puede establecer conexión con dos periféricos SPI distintos de forma simultánea y por ello hay dos SS distintos. En el caso del Intel Edison Kit para Arduino únicamente hay un pin SS1 predefinido, pero se podría estudiar si existe la posibilidad de configurar otro pin como SS2 alternativo.

Cabe destacar también que MOSI, MISO y SCLK están disponibles también de forma física en la cabecera ICSP (*In-Circuit Serial Programming*). Esto puede ser útil si se desea realizar cualquier dispositivo que conlleve unir esta placa con algún otro *shield*.

3.2.4.5 Interfaz LED

Al igual que en las placas Arduino convencionales, hay un LED incorporado al pin digital 13 del Intel Edison Kit para Arduino. Cuando el pin 13 se encuentra en estado alto, el LED se enciende, y cuando se encuentra en estado bajo, se apaga.

3.2.4.6 Interfaz de entradas analógicas

De igual forma que se mencionó en apartados anteriores, existe una interfaz de entradas analógicas (pines A0 a A5) en el Intel Edison Kit para Arduino.

3.2.4.7 Interfaz I^2C

I^2C es un bus de comunicaciones serie, así como lo es SPI. Las líneas que gestionan esta interfaz son SDA para datos, SCL para el reloj y GND para tierra, y pueden encontrarse en el Intel Edison Kit para Arduino en los pines con el mismo nombre.

No se hará hincapié en describir esta esta interfaz, ya que la que será usada principalmente será la SPI.

3.3 *Shield* 3G+GPS

Otro de los componentes del presente trabajo es el *Shield* 3G+GPS comercializado por la empresa Cooking Hacks [21]. En la figura 22 podemos ver el aspecto físico de dicho *shield*.

Figura 22 - Shield 3G+GPS de Cooking Hacks

A raíz de lo que se ha comentado anteriormente, sabemos que este *shield* va a estar disponible en una de las dos versiones de este trabajo, ya que no resulta necesario disponer de una conexión 3G en tiempo real en caso de ser usado en una competición, y el sensor GPS que este *shield* incorpora puede ser encontrado de forma independiente.

No obstante, va a ser un elemento importante en el banco de pruebas ya que será usado para justificar el buen funcionamiento de la nueva plataforma, así como la eliminación del cuello de botella que imponía la anterior a la hora de transmitir datos a un servidor remoto. Además, una vez elaborado un dispositivo final, y aunque no pueda ser usado en una competición, sí que sería de gran ayuda en los entrenamientos.

Este *shield* está compuesto principalmente por un módem SIM5218, que no es más que un módem de telefonía 3G convencional. Este módem permite conexiones de hasta 7.2 Mbps de bajada y 5.76 Mbps de subida. Además, este *shield* incluye una UART, un soporte para tarjetas SIM, un puerto USB 2.0, un sensor GPS, una conexión para tarjetas microSD, soporte para cámaras, salida de audio mediante un puerto de 3.5 mm y varios elementos hardware más, que podrían resultar bastante útiles según el uso que se le quiera dar.

Gracias al módem es posible realizar llamadas de voz y vídeo, enviar y recibir mensajes SMS, enviar y recibir datos a través de internet, etcétera. Además, gracias a su sensor GPS puede utilizarse como dispositivo de localización.

El *shield* dispone de muchas de las conexiones principales de una placa Arduino, por lo que puede colocarse encima de una de estas placas y extender su funcionalidad.

A continuación, en la figura 23 podemos ver un diagrama esquematizado de las conexiones con las que está dotado este *shield*.

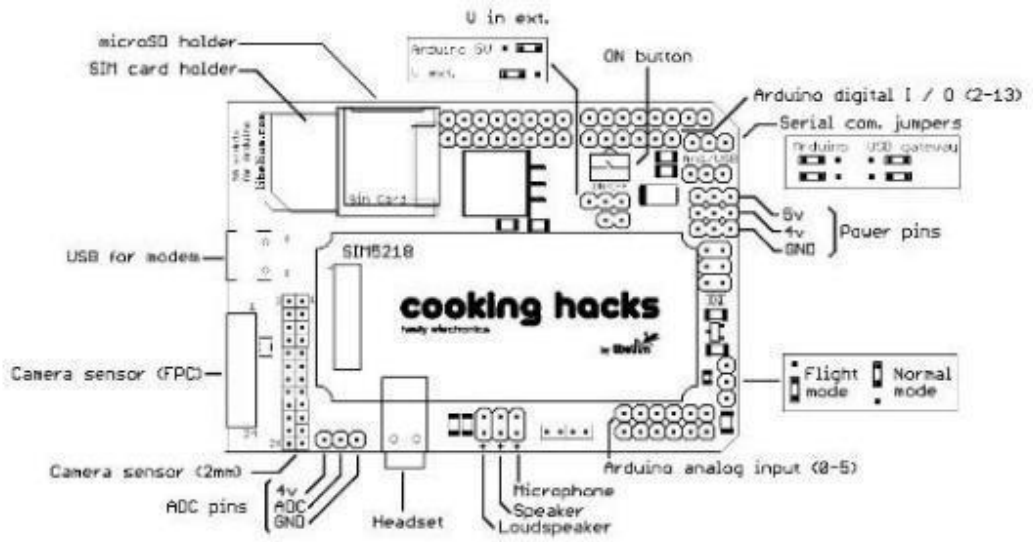

Figura 23 - Esquemático Shield 3G+GPS

El *shield* requiere una alimentación entre 3.4 y 4.2 voltios, y una corriente de entrada de entre 2 y 3 amperios, por lo que no hay problema en alimentarlo de forma directa mediante una placa Arduino o una placa de expansión como el Intel Edison Kit para Arduino que se usará en este trabajo, que emula una de estas placas y se conecta con la Intel Edison. No obstante, también está preparado para alimentación externa a través de un puerto mini USB.

El módem SIM5218 es cuatribanda ya que soporta GSM, GPRS, EDGE y UMTS. Además, soporta protocolos FTP, TCP, UDP, POP3, SMTP y HTTP.

3.4 Sensores

Un sensor es, por definición, un dispositivo que mide una determinada variable química o física y la transmite de forma adecuada a un receptor. Las magnitudes medidas pueden denominarse variables de instrumentación, y para el caso concreto de este trabajo, la forma adecuada de transmitir las mediciones será transmitirlas en forma eléctrica. Esta conversión es posible gracias a un segundo elemento, el transductor, que se obvia en esta descripción debido a que se ha considerado que todo sensor que se va a utilizar ya lo lleva incorporado.

Las variables de instrumentación pueden ser temperatura, intensidad lumínica, distancia, aceleración, inclinación, velocidad, desplazamiento, presión, fuerza, torsión o humedad, entre muchas otras.

Los sensores son la parte primaria y elemental de la telemetría, ya que gracias a ellos es posible obtener información de las variables medidas. En el mercado es posible encontrar una gran variedad de sensores, por lo que es primordial conocer las características que los diferencian, a fin de escoger el sensor más adecuado para cada variable.

La clasificación principal en función de la señal de salida divide a los sensores en dos grandes grupos. Por un lado, están los sensores analógicos, que son aquellos cuya salida es una señal eléctrica analógica, es decir, una tensión o intensidad de corriente que varía con la magnitud medida. Para obtener el valor de la variable que se mide es necesario aplicar una serie de correlaciones a la tensión/corriente de salida, obtenidas de forma empírica y que normalmente suele aportar el fabricante del sensor junto con la documentación. Por otro lado, están los sensores digitales, que son aquellos cuya salida es una señal eléctrica binaria, que no es más que una sucesión de ceros y unos, y que no es necesario transformar. Dentro de este grupo se encuentran los que son de tipo ON/OFF, cuya salida es booleana (0 o 1) y los que ofrecen a la salida una variable binaria perfectamente interpretable, como lo es el valor de una temperatura en forma binaria.

En cuanto a las características más importantes que definen a un sensor tenemos las siguientes:

- Rango de medida: rango en la magnitud medida en el que puede utilizarse el sensor en cuestión.
- Resolución: menor cambio en la magnitud medida que es capaz de detectar.
- Offset: valor de la variable de salida cuando la variable de entrada es nula. Si el rango de medida no llega a valores nulos, habitualmente se establece otro punto de referencia para definir el offset.
- Linealidad o correlación lineal: máxima desviación entre la respuesta real y la puramente lineal.
- Precisión: máxima desviación entre el valor real proporcionado y el teórico según un patrón definido.
- Rapidez de respuesta: puede ser un tiempo fijo o depender de en qué medida varíe la magnitud varíe la magnitud a medir. Depende de la capacidad del sistema para seguir variaciones en la magnitud de entrada.
- Derivas: otras magnitudes, al margen de la medida como magnitud de entrada, que influyen en la variable de salida. Un claro ejemplo de deriva son las condiciones ambientales.
- Repetitividad: error esperado al repetir varias veces la misma medida.
- Sensibilidad: variación de la salida por unidad de magnitud que varía la entrada.
- Histéresis: fenómeno de descompensación existente al realizar una comparación entre la variación de una misma medida tanto a nivel descendente como ascendente, cuando dicha medida debería tener un recorrido idéntico.

3.4.1 Acelerómetro ADXL345

Un acelerómetro es un dispositivo capaz de medir la aceleración y las fuerzas que sufre un cuerpo, inducidas por la gravedad. Existen diversos tipos actualmente, como los acelerómetros mecánicos, los acelerómetros piezoeléctricos, los acelerómetros capacitivos o los acelerómetros basados en el efecto Hall. Otra característica que los diferencia es si son o no capaces de medir aceleraciones en un plano (dos ejes) o en el espacio tridimensional (tres ejes). En este trabajo, este tipo de sensor es idóneo para medir el ángulo de inclinación de la bicicleta y la motocicleta en la que se implante.

El acelerómetro escogido en este caso, ha sido el acelerómetro triaxial ADXL345. Este acelerómetro tiene la particularidad de permitir comunicación tanto SPI como I^2C, aunque sólo se va a utilizar la comunicación SPI. Una de las principales ventajas de este modelo en concreto es su relación calidad/precio, que lo ha llevado a ser el acelerómetro más utilizado en este tipo de sistemas. Además, esto aporta la gran ventaja que supone el disponer de una gran cantidad de programas y tutoriales para facilitar su utilización.

La placa de evaluación ADXL345EB implementa este sensor [22], y está preparada para ser usada directamente con cualquier microcontrolador de un sistema Arduino típico y, por consiguiente, también es compatible con la plataforma Intel Edison.

En la figura 24 se presenta la placa de evaluación mencionada.

Figura 24 - ADXL345EB

3.4.1.1 Especificaciones técnicas de la placa ADXL345EB

Los pines que pueden encontrarse en la placa mencionada, como se ha podido apreciar en la figura anterior, son los siguientes:

- VDD (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)

Como ya se ha comentado, el dispositivo es capaz de utilizar tanto la comunicación SPI como la I^2C. Para utilizar la comunicación SPI son necesarios los pines

SDA, SDO, SCK y CS, que corresponden a los cuatro pines propios de este tipo de comunicación, como ya se adelantó en el apartado correspondiente.

En la tabla expuesta en la figura 25 se muestran las principales especificaciones técnicas de este sensor.

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	±16 g

Figura 25 - Especificaciones técnicas ADXL345

Como añadido, en la figura 26 se muestra la salida que presenta el sensor en función de la orientación del mismo con respecto a la fuerza que ejerce la gravedad. Esto es importante de cara a saber qué respuesta esperar en función de cómo se coloque la placa.

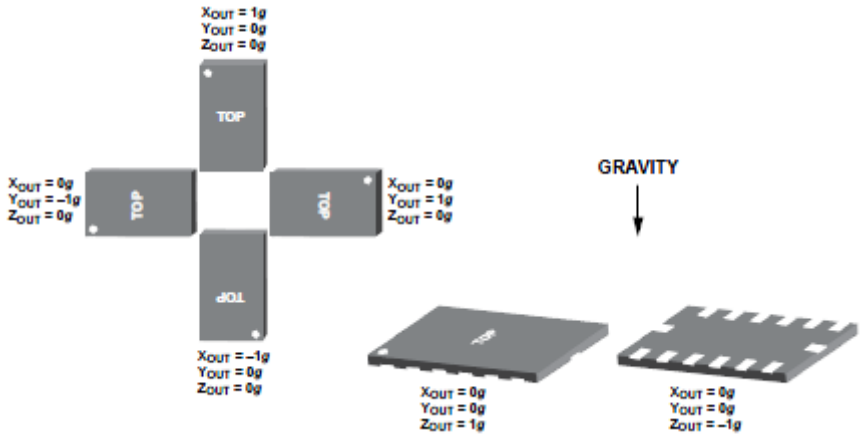

Figura 26 - Salida en función de la orientación con respecto a la gravedad

3.4.2 Sensor de efecto Hall

El sensor de efecto Hall se sirve del efecto bautizado bajo el mismo nombre para la medición de campos magnéticos o corrientes. Esta característica hace posible su utilización para medir la posición de objetos a través de la variación del campo magnético.

En este caso concreto, se va a utilizar un sensor de este tipo para calcular la velocidad de giro de la rueda en la que se instale, añadiendo un pequeño imán en una zona de eje de la misma. A partir de la medición directa de las variaciones del campo magnético que produce dicho imán, y haciendo uso de distintas fórmulas es posible calcular el número de vueltas por unidad de tiempo que da el eje.

Aunque los sensores de efecto Hall pueden tener salidas analógicas, para este caso en concreto se ha elegido un sensor cuya salida es digital. Por supuesto, el sensor trabaja con tensiones de alimentación compatibles con el sistema Intel Edison.

El sensor elegido ha sido el modelo SS495A de Honeywell [23]. Su frecuencia de muestreo característica ha sido clave para su elección, ya que permitiría medir las revoluciones máximas que experimenta una rueda de una motocicleta de competición. Por lo general, una rueda de motocicleta de 55 cm de diámetro a 350 kilómetros por hora alcanza cerca de 57 revoluciones por segundo. Como este sensor tiene una respuesta en frecuencia plana hasta aproximadamente 10 kHz con una alimentación de 5 voltios, no habría problema ninguno en medir la velocidad de una rueda de estas características girando a velocidad máxima, aunque no se situaran los imanes en la parte más exterior de la llanta, y aunque se utilizasen varios imanes para la detección. Por consiguiente, también sería válido para detectar la velocidad de giro de una rueda de bicicleta. Su temperatura máxima de trabajo o su tensión de alimentación también han sido determinantes para su elección.

3.4.2.1 Especificaciones técnicas del sensor SS495A de Honeywell

Este elemento dispone de tres pines para su conexionado. El pin 1 es la alimentación, el pin 2 la tierra y el pin 3 la salida. En la figura 27 puede verse esquematizada esta disposición, y en la figura 28 su circuitería del mismo.

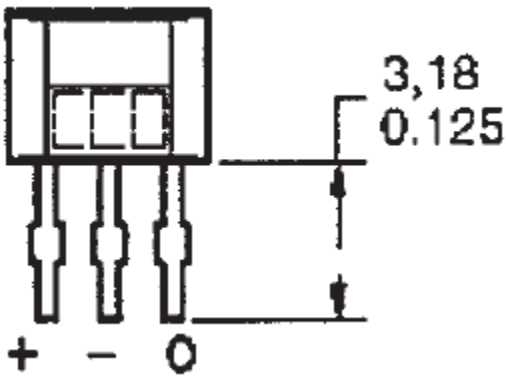

Figura 27 - Disposición de pines del sensor SS495A de Honeywell

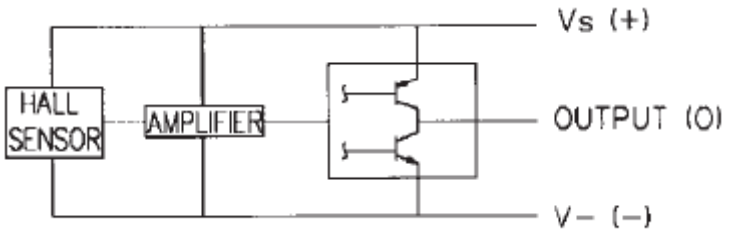

Figura 28 - Circuitería sensor SS495A de Honeywell

Las principales especificaciones técnicas de este sensor se presentan en la tabla de la figura 29.

Tensión de alimentación mínima	4.5 V
Tensión de alimentación máxima	10.5 V
Frecuencia de muestreo máxima (respuesta plana)	10 kHz
Máxima temperatura de operación	125ºC
Mínima temperatura de operación	-40ºC

Figura 29 - Especificaciones técnicas AH182/AH183

3.4.3 Potenciómetro óptico SHARP GP2Y0A51SK0F

Por definición, un potenciómetro no es más que una resistencia variable. Lo que caracteriza a cada uno es la forma en la que hace variar dicha resistencia.

En el trabajo que precede al presente, se utilizó un potenciómetro lineal simple. Este potenciómetro estaba destinado a ser utilizado a modo de prueba, debido a su baja calidad. Por otro lado, los potenciómetros industriales mecánicos son excesivamente caros, por lo que no son aptos para ser usados en un trabajo de este calibre. La solución a ambos inconvenientes es el uso de un potenciómetro óptico.

El sensor elegido, el potenciómetro óptico GP2Y0A51SK0F [24], es una unidad analógica de medición de distancias, compuesta de una combinación integrada de un PSD (*position sensitive detector*), un IR-LED (*infrared emitting diode*) y un circuito de procesado de señal. La variación de la reflectividad del objeto, la temperatura ambiental y la duración de la medición no influyen fácilmente en la detección de la distancia gracias a la adopción de un método de triangulación. Este sensor da a la salida un valor de voltaje dependiente de la distancia detectada de forma similar a la representada en la gráfica que se expone en la figura 30.

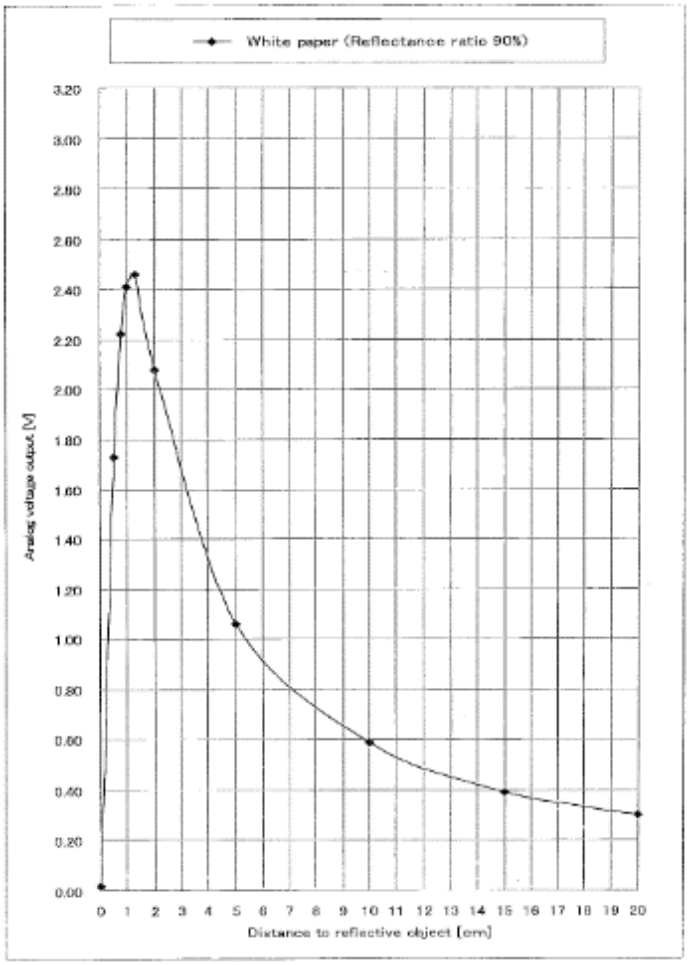

Figura 30 - Gráfica de funcionamiento del sensor SHARP GP2Y0A51SK0F

Como se aprecia, la gráfica de la figura 30 está basada en una medición experimental en la que se usó como objeto reflector un papel blanco. Dado que en este caso no se trata de un cuerpo con las mismas características reflectoras, la variación de la señal de salida no sigue exactamente esa curva, pero sí se ha comprobado experimentalmente que se asemeja lo suficiente en un rango de 2 a 15 centímetros, rango en el que se van a encontrar siempre las medidas tomadas, como para que el error con la gráfica de la figura 30 sea despreciable, por lo que ha sido utilizada para interpretar y escalar los datos recibidos en el software de recepción.

3.4.3.1 Especificaciones técnicas del sensor SHARP GP2Y0A51SK0F

Las principales especificaciones técnicas de este sensor se presentan en la tabla de la figura 31.

Tensión de alimentación mínima	4.5 V
Tensión de alimentación máxima	5.5 V
Máxima temperatura de operación	60ºC
Mínima temperatura de operación	-10ºC
Máxima distancia detectada	15 cm
Mínima distancia detectada	2 cm

Figura 31 - Especificaciones técnicas del sensor SHARP GP2Y0A51SK0F

En la figura 32 se puede ver el aspecto físico de este sensor y en la figura 33 un esquemático de su circuitería, siendo el pin 1 la alimentación, el pin 2 la masa y el pin 3 la salida.

Figura 32 - Sensor SHARP GP2Y0A51SK0F

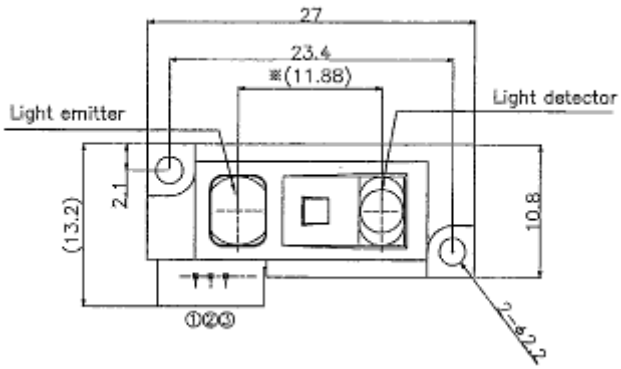

Figura 33 - Esquemático del sensor SHARP GP2Y0A51SK0F

3.4.4 GPS

El sistema de posicionamiento global (GPS) es un sistema que permite determinar en todo el mundo la posición de un objeto con una precisión de hasta centímetros (si se utiliza GPS diferencial), aunque lo habitual son unos pocos metros de precisión. El sistema fue desarrollado, instalado y empleado por el Departamento de Defensa de los Estados Unidos. Para determinar las posiciones en el globo, el sistema GPS está constituido por 24 satélites y utiliza la trilateración [25].

Para este trabajo se va a hacer uso de una antena GPS externa que se conecta al módem 3G+GPS a través de un conector SMA. En la figura 34 se presentan los conectores de los que está dotado dicho módem, siendo el conector de la antena GPS uno de ellos.

Figura 34 - Conectores SMA del módem 3G+GPS

Por otro lado, en la figura 35 se presentan las especificaciones técnicas de la antena en cuestión.

Tensión de alimentación mínima	2.7 V
Tensión de alimentación máxima	5.5 V
Máxima temperatura de operación	85ºC
Mínima temperatura de operación	-40ºC
Ganancia a 3V	26 dB
Ganancia a 5V	28 dB

Figura 35 - Especificaciones técnicas antena GPS

El sensor GPS incorporado tiene tres modos posibles y se configura a través de comandos AT.

El primer modo es el *stand-alone mode*. En este modo sólo se utilizan las señales que se obtienen de los satélites GPS, sin usar servidores de internet, lo que lo convierte en el modo menos rápido y preciso.

El segundo de los modos es el *mobile-based mode*. Este modo sí utiliza información de servidores de internet, pero es el módulo GPS el que realiza los cálculos de posicionamiento.

El último de estos modos es el *mobile-assisted mode*. Este es el modo más rápido y preciso, ya que obtiene la posición de los servidores de internet directamente. El sistema de GPS asistido utilizará los datos obtenidos y los combinará con la información de la antena de telefonía móvil para conocer la posición y saber qué satélites puede utilizar. Todos estos datos están almacenados en el servidor externo y según nuestra posición, el GPS dispondrá de los datos de unos satélites u otros y completará los datos que esté recibiendo a través del propio receptor, de forma que la puesta en marcha de la navegación es notablemente más rápida y precisa.

En cuanto a los comandos AT [26] de los que disponemos para utilizar el módulo GPS son los que se exponen en la tabla de la figura 35.

Comando AT	**Función**	**Sintaxis**	**Parámetros**
AT+CGPS	Establecer modo	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=\"dirección\"	dirección: formada por servidor:puerto
AT+CGPSSSL	Establecer certificado de seguridad GPS	AT+CGPSSSL=SSL	SSL: 0 (sin certificado), 1 (con certificado)
AT+CGPSINFO	Obtener datos de geoposicionamiento	AT+CGPSINFO	

Figura 36 - Comandos AT para GPS

En función de cómo sea configurado, la información que se obtendrá vendrá codificada de una forma u otra.

Si se usan los modos *stand-alone* o *mobile-based*, la información vendrá de la siguiente forma:

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

En la tabla de la figura 37 se presentan los distintos parámetros y su formato.

Parámetro	Formato	Ejemplo
Latitud	ddmm.mmmmmm d: grados; m: minutos	5234.869874
Longitud	dddmm.mmmmmm d: grados; m: minutos	00057.635486
Fecha	ddmmyy d: día; m: mes; y: año	020913
Hora del servidor	hhmmss.s h: hora; m: minutos; s: segundos	092536.0
Altitud	metros	310.00
Velocidad	nudos	3
Rumbo	grados	0

Figura 37 - Trama GPS modos stand-alone y mobile-based

Si se usa el modo *mobile-assisted*, obtendremos la información de la siguiente manera:

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

Del mismo modo, en la tabla de la figura 38 se presentan los distintos parámetros y su formato.

Parámetro	Formato	Ejemplo
Latitud	grados en 10^8	5234.869874
Longitud	grados en 10^8	00057.635486
Altitud	metros	310.00
Fecha	ddmmyy d: día; m: mes; y: año	020913
Hora del servidor	hhmmss.s h: hora; m: minutos; s: segundos	092536.0

Figura 38 - Trama GPS modo mobile-assisted

3.5 *Blocks*

Los *Blocks* de SparkFun para la plataforma Intel Edison [27], en adelante bloques, tal y como ellos mismos indican en su página web, son una forma excelente de explorar todas las posibilidades del conector de 70 pines que ésta posee. Mezclando y conectando distintos bloques, los usuarios pueden configrar a medida su Intel Edison para cualquier aplicación.

En la figura 39 puede verse un ejemplo de pila de distintos bloques.

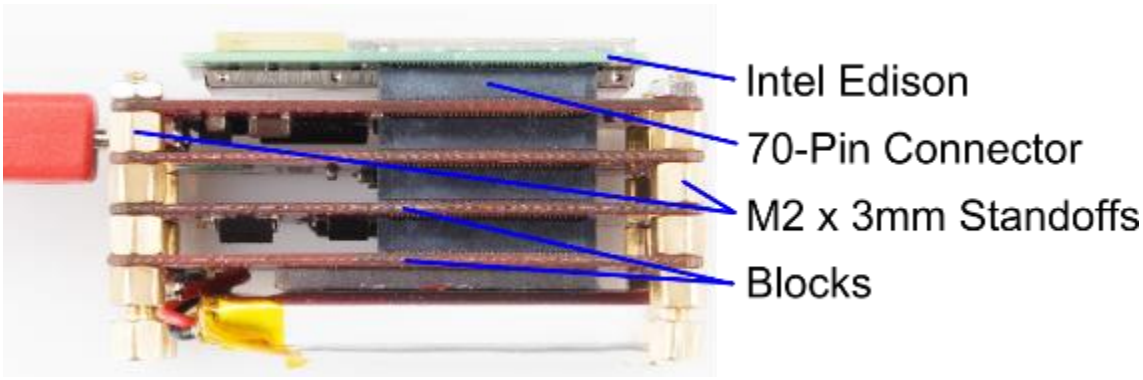

Figura 39 - Pila de bloques

Aparte de los bloques, hay otros elementos que entran en juego en una pila completa.

La Intel Edison es el cerebro de la pila, ya que es la que proporciona el procesamiento y la comunicación.

El conector de 70 pines es la columna vertebral de la pila, ofrece un camino para la alimentación y los datos hacia y desde todos los bloques.

Los separadores proporcionan resistencia mecánica al apilamiento de los bloques.

Los bloques permiten que los diseños, aunque más versátiles, sigan siendo pequeños, gracias al pequeño tamaño de la Edison y su conector, el cual podemos ver más en detalle en la figura 40.

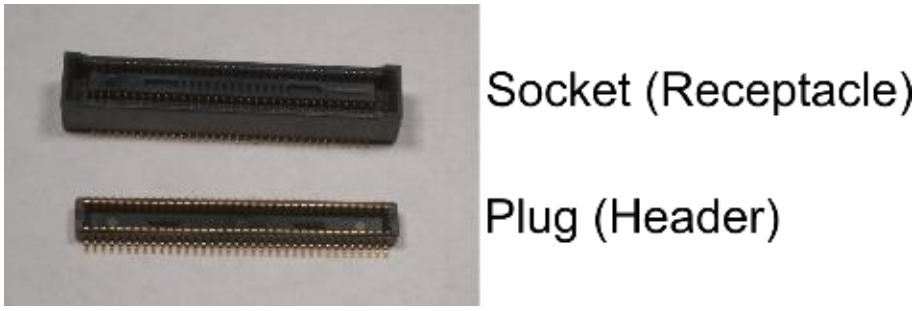

Figura 40 - Conector de 70 pines

El fabricante, Hirose, nombró a los conectores *Headers* y *Receptacles*. Sparkfun, por su parte, los nombra simplemente *Plugs* y *Sockets*, a fin de facilitar un poco la comprensión.

Socket (*Receptacle*): Conector que se encuentra en el bloque. Permite a la Edison o a otro bloque recibir señales.

Plug (*Header*): Conector que se encuentra en la Edison. Permite a las señales ir desde la Edison a los distintos bloques apilados.

Por lo tanto, la mayoría de los bloques tendran dos conectores.

Es el turno de ver cuáles de esos bloques van a formar parte de este trabajo.

3.5.1 Bloque de batería

El bloque de batería lleva una batería de 400 mAh dotada de una única célula de polímero de litio, para suministrar un voltaje de entrada de 3.7 voltios a la Intel Edison y los distintos bloques de expansión [28]. Además, se puede usar con una batería externa para aumentar su autonomía. El interruptor de alimentación desconecta la batería de la Edison mientras permite su carga a través del cable micro USB. También se puede sustituir por una batería de mayor capacidad.

En la figura 41 se muestra el bloque mencionado.

Figura 41 - Bloque de batería

3.5.2 Bloque de GPIO

El bloque GPIO es una placa que extrae los pines de entrada y salida de propósito general de la Intel Edison [29]. Proporciona acceso a los pines básicos de GPIO, PWM y UART2. También proporciona acceso a las tres líneas de voltaje que proporciona la Intel Edison, 3.3 voltios, 2.8 voltios y VSYS, aparte de a la tierra.

Es importante destacar que, si bien es posible alimentar una pila de bloques a través del bloque GPIO, hay riesgos. Hay que suministrar una tensión de entre 3.3 y 4.5 voltios, ya que excediendo este rango puede resultar dañada tanto la Intel Edison como la pila de bloques.

En la figura 42 se puede ver el aspecto físico de este bloque.

Figura 42 - Bloque de GPIO

3.5.3 Bloque de microSD

El bloque de microSD permite equipar a la Intel Edison con una unidad de almacenamiento masivo [30]. De este modo es posible utilizarla para realizar *datalogging* o registro de datos y otros proyectos similares. En lo que a este trabajo respecta, podría ser utilizada en caso de que no fuera posible almacenar la información en la memoria interna de la Intel Edison. Al finalizar este estudio se concluye que sí es posible, por lo que se podría prescindir de este bloque, aunque se ha incluido en la documentación porque puede resultar útil de igual modo.

En la figura 43 se muestra el bloque en cuestión.

Figura 43 - Bloque de tarjeta microSD

3.6 Sistema de recepción

El sistema de recepción es la última parte del sistema que compone este proyecto. En un sistema de telemetría, se define una red formada por dos puntos con una comunicación que puede ser unidireccional en sentido ascendente (sensor a sistema receptor) o incluso bidireccional, dependiendo del protocolo de comunicaciones que se use para enviar la información recogida.

Para el trabajo que precede al presente se utilizó un servidor UDP compilado de forma manual, corriendo en una computadora del laboratorio de la universidad, así como también se usó un conocido servidor FTP de uso gratuito (FileZilla), para comparar ambos métodos de comunicación, para finalmente decantarse por el primero de ellos.

Dado que este trabajo pretende dar un salto evolutivo, se va a utilizar el software de recepción desarrollado por mi compañero Víctor Huéscar López, que ha sido el encargado de llevar a cabo el trabajo paralelo a éste que concierne la parte de recepción de datos.

Se trata de un software de simulación capaz de comprobar el correcto funcionamiento de la aplicación en sus distintos modos. Está programado con herramientas de programación web, y permite hacer representaciones gráficas de los datos recibidos y almacenados en su base de datos.

En la figura 44 se muestra una captura de la interfaz de la aplicación.

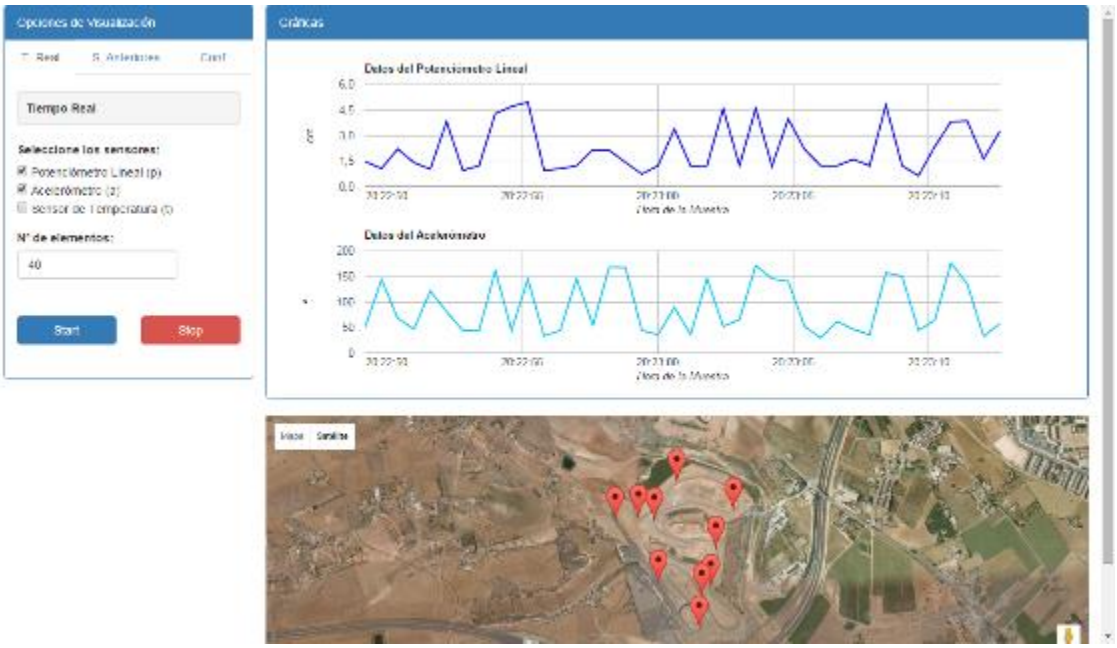

Figura 44 - Interfaz de la aplicación de recepción de datos

3.7 Entorno de trabajo

Por último, pero no por ello menos importante, hay que hacer mención al entorno global en el que se ha desarrollado este trabajo.

En primer lugar, se han utilizado varios ordenadores personales, sobre los cuales está instalado un SO de Windows. En dichos ordenadores se encuentra instalado el software necesario para programar la plataforma Intel Edison. Este software ha pasado por numerosas actualizaciones a lo largo de este proyecto, pero su versión actual puede ser descargada de la página oficial de Intel de forma gratuita, junto con los *drivers* necesarios, en https://software.intel.com/iot/software/installers.

Este entorno de programación utiliza su propio lenguaje de programación para las placas Arduino, compatible también con las distintas plataformas de Intel Edison e Intel Galileo. El lenguaje de programación de Arduino es una implementación de *Wiring*, una plataforma de computación física utilizada en microcontroladores, que a su vez está basada en *Processing*, un entorno de programación multimedia. Toda la información sobre el lenguaje de programación Arduino se encuentra disponible en su página web y además, cuenta con una gran comunidad de profesionales y aficionados que comparten sus programas y librerías, a la vez que intentan ayudar a resolver problemas y dudas a través del foro oficial, disponible en http://forum.arduino.cc/. Del mismo modo, Intel tiene su propio foro, muy útil también para ciertos casos y particularidades propias de las plataformas Intel Edison, y que no podríamos encontrar en el foro de Arduino. Este puede encontrarse en https://communities.intel.com/community/tech/edison.

Algunas de las características del IDE (*Integrated Development Environment*) de desarrollo para Arduino utilizado son las siguientes:

- Editor de textos en el que se escribe el código fuente que será cargado en la plataforma. Los documentos creados se denominan *sketches* y tienen extensión .ino.

- La consola nos muestra por pantalla mensajes de éxito o error acerca de la compilación y la carga del código seleccionado en la placa.

- El monitor del puerto serie, que es una ventana que muestra los datos que envía la placa de Arduino a través del mismo, como ya se ha mencionado con anterioridad. También posee una función de envío, para enviar datos desde la computadora a la placa.

- Programas ejemplo, que utilizan la mayoría de las funciones de este lenguaje y que están a la disposición del usuario para su consulta de forma rápida y sencilla.

En la figura 45 podemos ver una captura del entorno mencionado.

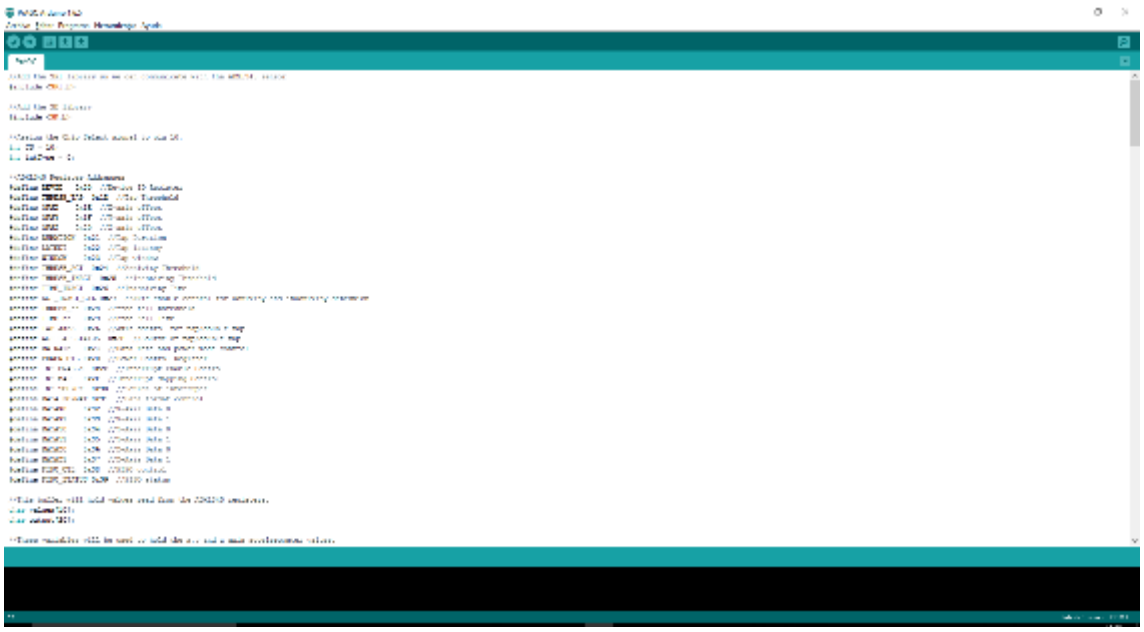

Figura 45 - Captura IDE Arduino 1.6.5

Capítulo 4: Transmisión de datos a través de Internet

En este apartado no se pretende realizar un análisis detallado de los distintos protocolos de internet que soporta el módem 3G utilizado, puesto que esto ya fue objeto de estudio en el trabajo precedente. No obstante, es importante comentar de forma breve estos protocolos y aportar los motivos por los cuáles se han seguido utilizando o no en este trabajo, no sin antes comentar la estructura que conforma los pilares de estos protocolos, el modelo *Open Systems Interconexion* o modelo OSI [31].

4.1 Modelo OSI

El modelo *Open Systems Interconexion* (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 están 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 dos sistemas. Por otro lado, están los servicios de soporte al usuario en los que se encuentran 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, contenido en la capa 4, actúa como puente entre ambos grupos.

La división en capas trae consigo una serie de ventajas ya que 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. También 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.

En la tabla de la figura 46 se puede ver esquematizada la división de niveles mencionada en orden descendente.

Nivel	Descripción
7 - Aplicación	Programas de aplicación que usan la red
6 - Presentación	Representación de datos
5 - Sesión	Gestión de conexiones entre aplicaciones
4 - Transporte	Servicios de detección y correción de errores
3 - Red	Gestión de conexiones a través de la red para capas superiores
2 - Enlace de datos	Servicio de envío de datos a través del enalce físico
1 - Física	Características físicas de la red

Figura 46 - Capas del modelo OSI

4.2 Protocolos disponibles

A través del modelo OSI se pueden caracterizar los protocolos de transporte disponibles en el módem que a continuación se exponen con brevedad, TCP y UDP [32].

4.2.1 *Transmission Control Protocol* (TCP)

TCP es uno de los protocolos fundamentales en Internet. Fue creado entre los años 1973 y 1974 por Vint Cerf y Robert Kahn.

Muchos programas dentro de una red de datos compuesta por computadoras, pueden usar TCP para crear conexiones entre ellos a través de las cuales puede enviarse un flujo de datos. El protocolo garantiza que los datos serán entregados en su destino sin errores y en el mismo orden en que se transmitieron. También proporciona un mecanismo para distinguir distintas aplicaciones dentro de una misma máquina, a través del concepto de puerto.

TCP da soporte a muchas de las aplicaciones más populares de Internet (navegadores, intercambio de ficheros, programas de mensajería, etc.) y protocolos de aplicación como HTTP, SMTP, SSH y FTP.

El protocolo TCP está documentado en el RFC 793 de la IETF, y sus principales características técnicas son:

- Orientado a la conexión: dos computadoras establecen una conexión para intercambiar datos. Los sistemas de los extremos se sincronizan con el otro para manejar el flujo de paquetes y adaptarse a la congestión de la red.

- Full-dúplex: una conexión TCP consiste en un par de circuitos virtuales, cada uno en una dirección. Sólo los dos sistemas finales sincronizados pueden usar la conexión.

- Comprobación de errores: mediante una técnica de *checksum* se verifica que los paquetes no estén corruptos.

- Acuses de recibo: ante la recepción de uno o más paquetes, el receptor regresa un acuse de recibo al transmisor indicando que recibió los paquetes. Si los paquetes no son notificados, el transmisor puede reenviar los paquetes o terminar la conexión si el transmisor cree que el receptor ya no está en línea.

- Control de flujo: si el transmisor está desbordando el buffer del receptor por transmitir demasiado rápido, el receptor descarta paquetes. Los acuses fallidos que llegan al transmisor le alertan para bajar la tasa de transferencia o dejar de transmitir.

- Recuperación de paquetes: el receptor puede pedir la retransmisión de un paquete. Si el paquete no es notificado como recibido (ACK), el transmisor envía de nuevo el paquete.

La cabecera del protocolo TCP se muestra en la trama de ejemplo de la figura 47.

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

Figura 47 - Ejemplo de trama TCP

4.2.2 *User Datagram Protocol* (UDP)

UDP es un protocolo del nivel de transporte basado en el intercambio de datagramas. Permite el envío de dichos 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. Tampoco se sabe si ha llegado correctamente, ya que no hay confirmación de entrega o recepción. Su uso principal es para protocolos como DHCP, BOOTP, DNS y demás protocolos en los que el intercambio de paquetes de conexión y desconexión superan en número a los paquetes de información, o no resultan estrictamente necesarios, 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.

Las principales características técnicas del protocolo UDP, documentado en el RFC 768 de la IETF, son:

- Orientado a mensajes (datagramas).
- Proporciona una sencilla interfaz entre la capa de red y la capa de aplicación.
- No otorga garantías para la entrega de sus mensajes.

La cabecera del protocolo UDP se muestra en la trama de ejemplo de la figura 48.

+	Bits 0 - 15	16 - 31
0	Puerto origen	Puerto destino
32	Longitud del Mensaje	Suma de verificación
64	Datos	

Figura 48 - Ejemplo de trama UDP

4.3 Protocolo elegido y justificación

Comparando ambos protocolos se puede apreciar que la principal diferencia es que TCP garantiza que los datos llegan correctamente, en el orden adecuado y sin errores ni problemas de duplicación. Estas características son las que a su vez, hacen que la velocidad de transmisión sea menor que en UDP, que no realiza estas comprobaciones. Además, en TCP es necesario establecer una conexión, para lo cual se necesita un cierto tiempo de inicialización para el envío de paquetes de conexión, mientras que en el

protocolo UDP esto tampoco es necesario, haciendo que los tiempos de inicialización sean más cortos.

Para un dispositivo de telemetría en tiempo real, los parámetros más importantes son la velocidad de transmisión de datos y la fidelidad de la misma. En el caso particular que atañe a este trabajo, y haciendo referencia a la teoría del muestreo, al muestrearse a velocidades superiores a la media, puede permitirse una tasa de error o pérdida de datos significativa, y que el sistema siga siendo fiable.

El objetivo de la telemetría para este trabajo en particular es recopilar la información de las variables físicas en el menor 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. Una prueba de ello es lo sucede con el hecho de que las tramas puedan llegar desordenadas, lo que podría suponer un problema, pero al incluir en todas las tramas una muestra de GPS en la que se encuentra la hora exacta, se ha podido utilizar como marca temporal para poder ordenar las tramas en recepción en caso de que lleguen desordenadas y que esto no afecte de forma negativa a la hora de representar la información recibida.

Lo anteriormente mencionado corresponde a la conclusión a la que se llegó en el trabajo que precede al presente. Es en este punto cuando, habiéndose diseñado ya el sistema de recepción, es posible confirmar que efectivamente es dicho sistema el encargado de compensar esos inconvenientes que supone el uso de UDP, para poder aprovechar únicamente sus ventajas frente al protocolo TCP.

Capítulo 5: Programación de la placa

En este apartado se va a exponer, de forma ordenada y detallada, todas las partes que componen el código que va ejecutar la placa.

5.1 Configuración

5.1.1 Inclusión de librerías

Las librerías son archivos escritos en C o C++ (.c, .cpp) que proporcionan a los programas cargados o *sketches* unas funcionalidades adicionales.

En este caso, las librerías necesarias serán la SPI y la SD y las añadiríamos de la siguiente forma:

```
#include <SPI.h>
#include <SD.h>
```

5.1.2 Definición de variables

Acto seguido es el turno de definir todas las variables que van a tomar partido en el programa. Algunas son variables creadas por decisión propia, mientras que otras resultan necesarias para hacer funcionar los sensores. En el caso del acelerómetro, por ejemplo, es el propio distribuidor el que proporciona las variables necesarias.

5.1.3 Subrutina *sendATcommand*

Cabe decir también que la subrutina que se va a utilizar para enviar todos los comandos AT al módem y comprobar si la respuesta recibida es la esperada, es la siguiente:

```
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(Serial1.available() > 0) Serial1.read();
  Serial1.println(ATcommand);
  x = 0;
  previous = millis();
  do{
    if(Serial1.available() != 0){
      response[x] = Serial1.read();
      Serial.write(response[x]);
      x++;
      if (strstr(response, expected_answer1) != NULL){
        answer = 1;
        Serial.write('\n');
```

```
      }
    }
  }
  while((answer == 0) && ((millis() - previous) < timeout));
  return answer;
}
```

5.1.4 Rutina *setup*

Una vez incluidas las librerías y definidas todas las variables, es el momento de elaborar la rutina encargada de configurar el conjunto formado por el hardware de la placa y el *shield* 3G+GPS.

5.1.4.1 Subrutina de encendido

En primer lugar, se ha creado una subrutina que al ser llamada cambia el estado del módem de apagado a encendido y viceversa, utilizando un comando AT. La subrutina en cuestión es la siguiente:

```
void power_on(){
  uint8_t answer=0;
  answer = sendATcommand("AT", "OK", 2000);
  if (answer == 0){
    digitalWrite(onModulePin,HIGH);
    delay(2000);
    digitalWrite(onModulePin,LOW);
    while(answer == 0){
      answer = sendATcommand("AT", "OK", 2000);
    }
  }
}
```

Para invocarla, primero es necesario configurar la variable `onModulePin` como salida:

```
pinMode(onModulePin, OUTPUT);
power_on();
```

5.1.4.2 Tasas de transmisión de los puertos serie

De acuerdo a lo que se ha comentado previamente en este trabajo, existen tres puertos serie distintos en el sistema. Puesto que tan sólo se van a utilizar dos de ellos, hay que configurarlos para que trabajen a la tasa que se haya determinado. Dicho esto, las líneas de código necesarias para configurar los puertos en cuestión son las siguientes:

```
Serial.begin(230400);
Serial1.begin(230400);
```

En caso de que se quisiera utilizar el puerto serie restante, tan sólo habría que añadir la siguiente línea:

```
Serial2.begin(115200);
```

Esto serviría, por ejemplo, para abrir un terminal serie y acceder a la información almacenada en la memoria interna de la Intel Edison mediante comandos Linux.

5.1.4.3 Configuración de parámetros de la red de telefonía

Acto seguido es el turno de configurar los distintos parámetros de los que se sirve el módem para poder conectarse a la red de telefonía. Cabe decir que se ha incluido una línea que cierra automáticamente cualquier conexión abierta previamente, para que sirva de reseteo en caso de querer volver a cargar el código y reiniciar la conexión. El código es el siguiente:

```
sendATcommand("AT+NETCLOSE", "OK", 1000);
sendATcommand("AT+CPIN=2223", "OK", 1000);
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", 1000);
```

5.1.4.4 Configuración de la recepción de datos GPS

También es necesario configurar los parámetros propios de la recepción de datos GPS, para lo cual se han utilizado los siguientes comandos:

```
sendATcommand("AT+CGPSURL=\"supl.google.com:7276\"", "OK", 1000);
sendATcommand("AT+CGPSSSL=0", "OK", 1000);
answerGPS = sendATcommand("AT+CGPS=1,2","OK", 1000);
```

5.1.4.5 Tasa de transmisión del puerto serie del módem

Aunque se haya configurado la tasa de transmisión del puerto serie de la placa que se comunica con el módem, también hay que configurar el propio puerto serie del módem por medio del comando AT destinado a este fin:

```
sendATcommand("AT+IPREX=230400", "OK", 1000);
```

Cabe decir que este comando ha de enviarse de forma independiente al programa principal, por razones que a continuación se exponen.

En este punto es necesario hacer una pausa y explicar los posibles inconvenientes que pueden surgir a la hora de configurar esta tasa. Como ya se ha comentado, lo que se está configurando aquí es la tasa de transmisión del puerto serie del módem, y esto se lleva a cabo mediante el envío de un comando AT desde la placa, y que se recibe a través del mismo puerto serie que se está configurando.

Primeramente, hay que tener en cuenta que para poder indicarle al módem que funcione a una tasa de transmisión distinta a la que actualmente posee, hay que enviarle la instrucción mediante un puerto que funcione a la misma tasa, lo que supone que una vez haya cambiado su tasa de transmisión, ambos puertos dejaran de poder entenderse mutuamente. Eso significa que, una vez se envíe el comando, resulta necesario reconfigurar también la tasa de transmisión del puerto serie de la placa. Teniendo eso presente, existen tres posibilidades.

Por un lado, se puede incluir en el programa principal una línea que configure el puerto serie de la placa a una determinada tasa (la misma a la que esté configurado el puerto serie del módem), después una línea que envíe el comando de configuración al módem, y después otra línea que reconfigure el puerto serie de la placa a la nueva tasa de transmisión. Esto no es eficiente ya que, durante un pequeño lapso de tiempo ambos dispositivos dejan de entenderse, lo que puede acarrear pérdida de información.

Por otro lado, es posible crear un pequeño programa independiente del principal, que sirva únicamente para cambiar la tasa del puerto serie del módem sin importar que después dejen de entenderse, puesto que sólo se usará este programa una vez para configurar el módem. Una vez hecho, ya se sabrá la tasa a la que hay que configurar el puerto serie de la placa en el programa principal para que ambos se entiendan. En un principio se pensó que esta era la solución ideal, pero surgió otro problema cuando el módem no fue capaz de configurar su puerto a la tasa indicada, ya que no hubo manera entonces de poder averiguar a qué tasa había que configurar el puerto serie de la placa para poder establecer contacto, quedando el sistema inutilizable.

En último lugar, y como solución al problema que planteaba la segunda opción, se hizo uso de un cable USB a mini USB para conectar el módem a una computadora, lo que permitió establecer una comunicación serie entre ambos a través de un puerto distinto al que se quería configurar, para hacer uso del conocido programa Hyperterminal y poder enviar a través de él comandos AT para configurar el puerto serie en cuestión y evitar al mismo tiempo cualquier problema de conexión, ya que la comunicación se establecía mediante un puerto independiente que trabajaba a 115200 baudios, y cuya tasa no se modificaba en ningún momento.

Cabe decir en último lugar que la máxima tasa de transmisión a la que ha sido posible configurar el puerto serie del módem ha sido 921600 baudios, a pesar de que

teóricamente podía configurarse también a 3686400 baudios. Esto supone una reducción de la máxima tasa teórica a la que pueden intercambiar datos ambos dispositivos, y en un futuro sería bueno estudiar la razón y concluir si se puede solucionar de tal forma que se pueda aprovechar al máximo la tasa de transmisión.

5.1.4.6 Apertura del puerto UDP

Por último, se abre el puerto que se utilizará para el envío de la trama UDP hasta el servidor que contendrá el sistema de recepción, ya sea el servidor de pruebas o el software de recepción:

```
sendATcommand("AT+NETOPEN=\"UDP\", 5558","OK",1000);
```

Es importante destacar que para que exista comunicación entre módem un servidor, es necesario que dicho servidor también esté conectado a la red a través de un router tenga abierto ese puerto y esté configurado para recibir datos vía UDP. En la figura 49 se presenta una captura de la configuración en cuestión para el router al que está conectado el servidor que se ha utilizado para las pruebas de recepción.

Figura 49 - Configuración del router al que está conectado el servidor

5.1.4.7 Configuración para el acelerómetro ADXL345

Al margen de la configuración ya expuesta, existe una configuración extra que el fabricante del acelerómetro proporciona para la adquisición de los datos que este recoge.

No se va a mostrar en este apartado ya que posteriormente aparece en el anexo, además de ser accesible por cualquiera que lo desee a través de su web, la cual se incluye en la bibliografía de este trabajo.

5.1.5 Rutina *loop*

La rutina *loop* es la encargada de contener el conjunto de instrucciones que la placa ejecutará de forma repetida. Esta rutina contiene la subrutina de adquisición de datos, así como las instrucciones necesarias para el envío de información o su

almacenamiento, en función de si se desea realizar una telemetría en tiempo real o una telemetría local.

5.1.5.1 Registro de datos GPS

En primer lugar, se registran los datos del GPS, que conformarán un vector de 44 caracteres en caso de funcionar bien, o un vector de comas en caso de no tener cobertura y no poder recibir los datos correctamente.

El código utilizado es el siguiente:

```
Serial1.begin(230400);
answerGPS = sendATcommand("AT+CGPSINFO","+CGPSINFO:",10);
if(answerGPS==1){
  counter = 0;
  do{
      gps_data[counter]  = Serial1.read();
      counter++;
  }
  while((counter<44) && (Serial1.available()));
  gps_data[counter] = '\0';
  if(gps_data[0] == ','){
    Serial.println("No GPS data available");
  }
}
```

5.1.5.2 Adquisición de datos del resto de sensores y codificación de la trama

Lo siguiente a realizar es la obtención de los datos provenientes del resto de sensores. Una vez hecho esto, la idea es hacer una codificación previa al envío o almacenamiento de datos. En este caso, en lugar de exponer el código, se va a explicar de forma detallada la forma en la que se va a codificar la trama y se remite al lector al anexo si desea visualizar el código que lleva a cabo esta tarea.

De acuerdo con uno de los objetivos propuestos para este trabajo de fin de grado, se ha rediseñado la estructura de la trama de tal manera que se pueda ampliar el envío de información añadiendo sensores de forma más o menos sencilla y permitiendo la posibilidad de que la trama tenga una longitud variable de forma que se aprovechen mejor los recursos disponibles.

En primer lugar, convendría hablar de la trama diseñada en la solución del curso pasado, diseñada en conjunto por los dos de los tres autores de los trabajos correspondientes. Tal y como se implementó, la trama tenía una longitud fija de 500 bytes que debía ser completada con el signo ‘#’ en caso de que la información no fuese suficiente para llenar una trama completa. El datagrama enviaba siempre la secuencia de

sensores en el mismo orden, precediendo siempre la información de los sensores con un indicador que identificaba de cuál se trataba.

Tomando como referencia este diseño inicial y sabiendo que al menos se va a mantener el mismo GPS, luego los datos de ese sensor tendrán el mismo contenido y longitud, se ha diseñado una nueva trama. En primer lugar, se ha introducido un indicador de comienzo de trama representado con la letra 's', que permitirá desechar tramas o cualquier otra información que pueda llegar al puerto a través del que se reciben los paquetes y que resultará útil. Inmediatamente después viene codificado el número de sensores, incluido el GPS, mediante un carácter ASCII cuya relación es la que se muestra en la figura 50.

ASCII	Símbolo	Número
65	A	0
66	B	1
67	C	2
68	D	3
69	E	4
70	F	5
71	G	6

Figura 50 - Relación valores entre valores ASCII y número de sensores

Esta codificación seguiría siendo así sucesivamente, sin ningún máximo en cuanto al número de sensores que puede albergar una sola trama. La idea de añadir este cambió es la de codificar de forma sencilla y eficiente el número de sensores exacto que va a llevar la trama para poder llevar un mejor control y facilitar la decodificación en el servidor.

Tras la codificación del número de sensores, se envía una secuencia con los indicadores de los sensores que se van a enviar y en el orden en el que posteriormente va a aparecer la información. Hay que tener en cuenta que, si un sensor se envía varias veces en la misma trama, se debe enviar otras tantas veces el indicador de ese sensor ocupando la posición que debe en el orden en que se envía la información. Esta forma de especificar los sensores que contiene la trama permite conocer el orden exacto con que aparece la información al decodificarlo, si a esto añadimos el conocimiento de la longitud de bytes que ocupa cada muestra de sensor, se consigue que el proceso en recepción sea mucho más sencillo y más rápido.

Por último, tras la secuencia de los indicadores de los distintos tipos de sensores aparece la información en sí de cada uno de los sensores, siguiendo el orden antes especificado. Las distintas muestras no van separadas por ningún símbolo ya que, al conocer el orden de la información y la longitud de cada tipo de muestra, el uso de

cualquier separador se hace innecesario. En las figuras 51 y 52 se muestran las dos partes que conforman una trama, en primer lugar, la cabecera, y en segundo lugar el cuerpo que contiene las muestras recogidas por los sensores.

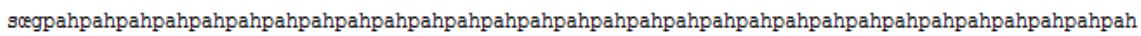

Figura 51 - Cabecera de una trama recibida

Figura 52 - Cuerpo de una trama recibida

5.1.5.3 Transmisión de datos vía UDP

Una vez elaborada la trama, y habiendo configurado previamente el módem para que sea capaz de realizar una transmisión vía UDP a una dirección IP determinada, es posible enviar dicha trama mediante una sencilla instrucción, que se muestra a continuación:

```
Serial1.println("AT+UDPSEND=250,\"79.110.172.71\",5558");
delay(10);
Serial1.print(trama);
```

5.1.5.4 *Datalogging*

Ya se ha dejado claro cómo podría realizar una telemetría en tiempo real, pero en la competición MotoStudent, al igual que en la alta competición de motociclismo, no está permitida. Por este motivo una de las líneas futuras del trabajo que precede al presente era implementar un método de anotación de información similar al que sigue un registrador de datos, comúnmente conocido por su definición anglosajona *datalogger*. Un *datalogger* es un dispositivo electrónico capaz de realizar un registro de información proveniente de instrumentos o sensores propios o conectados de forma externa.

La idea que se ha querido llevar a cabo es que el sistema fuera capaz de almacenar la información obtenida de los sensores en una tarjeta microSD conectada de forma externa al dispositivo principal (la Intel Edison en este caso), a través de una placa de expansión como el Arduino Breakout Kit con el que se ha trabajado en el banco de pruebas, o a través del *block* correspondiente que se instalará en el dispositivo final.

Cabe destacar cómo funciona este registro de datos y qué es lo que lo diferencia del que podría implementarse con cualquier otra placa anterior a la Intel Edison que estuviera basada en Arduino.

Primeramente, comentar que la librería `SD` será la que permitirá leer y escribir en tarjetas microSD externas [33]. Esta librería está basada en la librería `sdfatlib`

implementada por William Greiman. Soporta los formatos FAT16 y FAT32 en tarjetas tipo SD y SDHC y utiliza nombres de archivo del tipo 8.3 (nombres cortos o *short filename*), a menudo mencionados como SFN, convención usada en antiguas versiones de DOS y que implica un nombre de fichero de 8 caracteres o menos y un nombre de extensión de 3 caracteres, normalmente .txt o .log [34]. Estos nombres de archivo se pasan como argumentos para las funciones de la librería `SD` y pueden incluir rutas separadas por / como por ejemplo `Registro/log.txt`, siendo `Registro` una carpeta situada en la raíz de la microSD. Además, como el directorio de trabajo es siempre la raíz de la tarjeta, `/log.txt` es equivalente a `log.txt`.

La comunicación entre el microcontrolador que puede encontrarse en las placas Arduino que preceden a la Intel Edison y la tarjeta microSD utiliza SPI (*Serial Peripheral Interface*), que tiene lugar en los pines digitales 11, 12 y 13 en la mayoría de ellas salvo en la Arduino Mega que ocupa los pines digitales 50, 51 y 52. Adicionalmente, otro pin tiene que ser usado para seleccionar la tarjeta microSD. Este pin puede ser el pin 10 o el 53 en función de si se trata de una placa Arduino convencional o de una Mega, respectivamente, o cualquier otro pin especificado en la llamada a la función `SD.begin()`. Recalcar que, aunque no se use el pin por defecto (10 o 53), ha de ser configurado como pin de salida o la librería `SD` no funcionará correctamente.

Como puede apreciarse, esto puede suponer un inconveniente a la hora de ser implementado ya que se perderían dos pines que en un futuro podrían limitar la expansión del sistema.

La ventaja que ofrece la Intel Edison en este aspecto es que, en lugar de utilizar la interfaz SPI, el lector *on-board* de tarjetas microSD en el Arduino Breakout Kit se maneja mediante un controlador SD integrado.

Esto supone en primer lugar que no es necesario configurar de ningún modo los pines de la placa, basta con incluir la librería correspondiente e introducir nuestra rutina de registro de información. Además, significa que la interfaz SPI puede dedicarse a otras tareas de forma paralela mientras almacenamos información sin tener que dedicarse ella misma a dicha tarea.

Una vez explicado el mecanismo, es hora de llevar esta idea a la práctica implementando la rutina que se encargará de esta tarea. Al inicio del programa hay que incluir la librería `SD`, como ya se ha explicado con anterioridad en este capítulo.

Luego sólo tenemos que añadir la rutina en cuestión, que será la que se expone a continuación [35]:

```
File dataFile = SD.open("info.txt", FILE_WRITE);
  delay(1000);
```

```
  // if the file is available, write to it:
  if (dataFile) {
    dataFile.println(trama);
    dataFile.close();
    delay(1000);
  }
  // if the file isn't open, pop up an error:
  else {
    Serial.println("Error opening info.txt");
  }
}
```

Como puede apreciarse, el funcionamiento es sencillo. Primero se define el archivo mediante la función `SD.open()` y se le da el nombre `info.txt`. Además, se indica que va a escribirse en él añadiendo `FILE_WRITE` como segundo argumento de la función. Acto seguido se comprueba si el archivo se ha abierto correctamente y está disponible, en caso afirmativo se envía el String `trama` que contiene la información de los sensores que quiere guardarse y se cierra el fichero, en caso negativo se imprime por pantalla un mensaje de error.

Ya se ha explicado cómo puede almacenarse la información en una microSD externa, pero no es la única opción.

Una de las ventajas que ofrece la plataforma Intel Edison y que a su vez la hace más versátil, es su capacidad de almacenamiento interno, accesible tanto desde la parte Arduino, como desde la parte Linux. Esto podría resultar muy útil si en un futuro se desea acceder a esa información desde una aplicación que trabaje con bases de datos o con envío de datos vía Bluetooth o Wi-Fi, por ejemplo.

Para indicar al sistema dónde tiene que guardar el fichero con las tramas de información, es necesario acceder a la librería `SD.cpp` que se encuentra en la ruta AppData\Roaming\Arduino15\packages\Intel\hardware\i686\1.6.2+1.0\libraries\SD\src, y modificar la línea 37 para que luzca de la siguiente manera:

```
const char* SD_MOUNT_PATH = "/home";
```

Si la carpeta “home” no existe, se crea de forma automática. De igual forma, podríamos seleccionar como carpeta de destino la carpeta “update”, que es la única accesible por el explorador de Windows cuando conectamos la placa al ordenador. De una forma u otra ya se tendría la información almacenándose en el almacenamiento interno del sistema, siendo accesible por el mismo cuando fuese requerido [36].

Capítulo 6: Implementación física del sistema en una bicicleta de pruebas

En este apartado se va a tratar la implementación del primer prototipo sobre una bicicleta de pruebas, incluyendo los elementos necesarios para adaptar el sistema a dicho fin y comprobar su viabilidad de cara a una posible implementación final.

6.1 Bicicleta de pruebas

La Escuela Técnica Superior de Ingeniería de Telecomunicación de la Universidad Politécnica de Cartagena ha adquirido una bicicleta todo terreno, destinada a su uso en la realización de este trabajo.

6.2 Soportes para los sensores

Una vez se adquirió la bicicleta resultó necesario idear la manera de colocar los sensores sobre la estructura de la bicicleta y mantenerlos fijados.

Para esta tarea se han diseñado una serie de soportes impresos en 3D, tarea que ha corrido a cargo de Francisco Javier Martínez Abellán, a quien hay que agradecer su ayuda desinteresada diseñando los soportes con la herramienta SolidWorks y su posterior impresión.

En total se han diseñado 3 soportes, uno para el potenciómetro óptico, otro para el acelerómetro, y otro para el sensor de efecto Hall, que se presentan en las figuras 53, 54 y 55 respectivamente, ya colocados en la bicicleta y con sus sensores correctamente colocados.

Figura 53 - Soporte y potenciómetro óptico colocados en bicicleta de pruebas

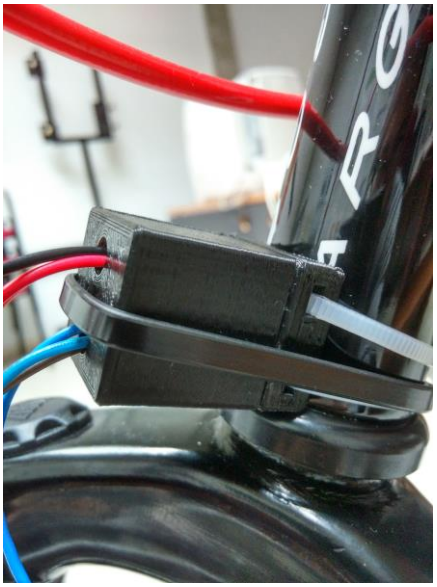

Figura 54 - Soporte y acelerómetro colocados en bicicleta de pruebas

Figura 55 - Soporte y sensor de efecto Hall colocados en bicicleta de pruebas

6.3 Prototipo de pruebas completo

Finalmente, el aspecto del prototipo de pruebas completo es el que se muestra en la figura 56.

Figura 56 - Prototipo completo para pruebas en laboratorio

Capítulo 7: *Benchmarking* del sistema completo

En este capítulo, y una vez mostrado como quedaría el prototipo final, es el turno de realizar una serie de análisis con la finalidad de comprobar el rendimiento real del sistema.

Por un lado, se va a realizar un análisis de la transmisión de datos de la que se encarga el módem para comprobar que efectivamente se ha conseguido eliminar el cuello de botella que sufría el sistema anterior.

Por otro lado, se va a analizar la velocidad con la que el sistema completo es capaz de almacenar los datos de forma local para concretar su potencial en este aspecto.

Además, se indicará el tamaño total del programa completo.

En definitiva, estos análisis se realizarán para poder concluir si el sistema es realmente viable para el uso para el que se ha ideado.

7.1 Análisis de la tasa de tansmisión de datos

Primeramente, cabe decir que se entiende por tasa de transmisión como el número de bytes que el dispositivo es capaz de enviar por segundo a un servidor remoto, por lo que el sistema de envío de información quedará caracterizado en función del tamaño de las tramas que se van a enviar y el tiempo medio aproximado que tarda en enviar cada grupo de tramas de un tamaño concreto.

Para la obtención de ese tiempo se han utilizado las funciones `micros()` y `millis()`, que devuelven el tiempo de ejecución del programa en microsegundos y milisegundos respectivamente. La tasa de transmisión se ha configurado a 230400 baudios y el tamaño de las tramas se ha ido variando desde 1 byte hasta 900 bytes.

Una vez superado el umbral de los 900 bytes, el sistema tiene problemas a la hora de cargar el código en él debido a un cierto límite que tiene el *buffer* de almacenamiento temporal de datos del puerto serie. Al ser la tasa de transmisión de 230400 baudios, es de esperar que la tasa de transmisión real se acerque a su máximo teórico de 23040 bytes/segundo.

En la figura 57 se presenta una tabla que relaciona los distintos tamaños de trama que se han ido seleccionando con la respectiva tasa de transmisión obtenida.

Tamaño de la trama (bytes)	Tasa de transmisión (bytes/segundo)
1	4255
10	10526
50	12500
100	14285
200	16666
300	18750
400	20000
500	20833
600	21428
750	21875
900	22500

Figura 57 - Tasa de transmisión UDP en función del tamaño de la trama enviada (I)

En la figura 58 se expone esta relación en forma de gráfica.

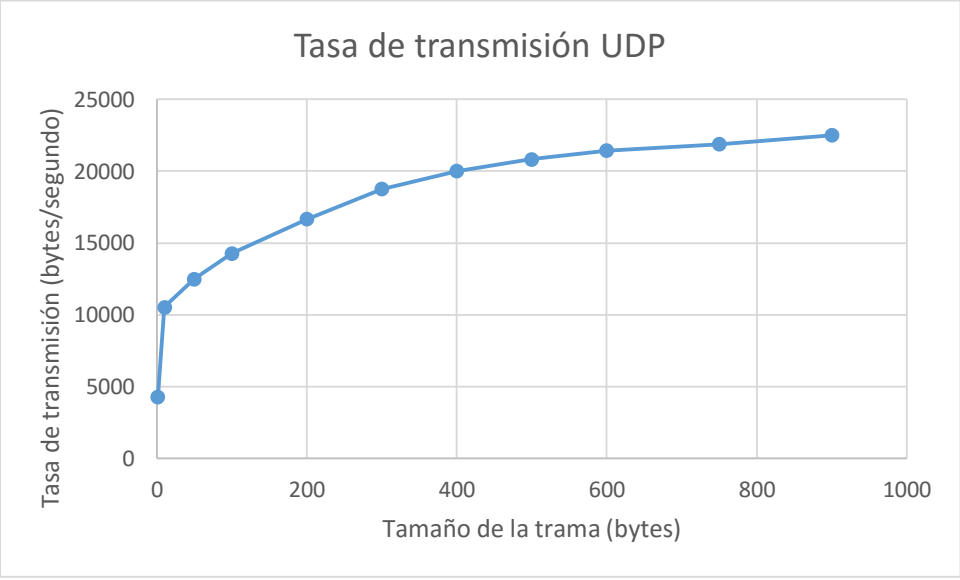

Figura 58 - Tasa de transmisión UDP en función del tamaño de la trama enviada (II)

Como se puede apreciar, esta plataforma no sólo elimina el límite de 500 bytes que poseía el sistema anterior en el *buffer* de almacenamiento temporal del puerto serie, sino que además es capaz de funcionar a mayor tasa de transmisión, aprovechando así el rendimiento del módem 3G+GPS.

Además de a 230400, se ha testeado el sistema configurado a tasas de 460800 y 921600 baudios, y se ha comprobado que efectivamente, la tasa de transmisión real aumenta conforme lo hace la tasa configurada. No obstante, sigue existiendo un límite cercano a los 900 bytes en el *buffer*, por lo que no es posible aumentar más el tamaño de las tramas enviadas. Aun así, se podría estudiar en un futuro si se puede reducir este límite mediante la modificación de alguna librería.

En cualquier caso, y dado que en tramas de menos de 900 bytes es posible codificar (de la forma que se ha hecho en este trabajo y que ya se ha mencionado) una gran cantidad de información, podemos concluir que el sistema cumple los requisitos esperados y subsana las deficiencias que el anterior imponía.

7.2 Análisis de la creación y almacenamiento de una trama de tamaño definido

Para este segundo análisis se ha utilizado la función `millis()` que se ha mencionado en el anterior apartado. En este caso se ha evaluado el tiempo total de creación y almacenamiento de una trama de tamaño constante, consistente en una muestra proveniente del GPS (44 caracteres), junto con treinta muestras de cada uno de los demás sensores, junto con sus respectivos marcadores en la cabecera, lo que equivale a un tamaño total de trama de 227 bytes.

Como resultado de esta prueba, se han obtenido unos tiempos que se sitúan entre los 300 y los 320 milisegundos. Esto se traduce en la adquisición y almacenamiento de tres tramas completas de 227 bytes, que a su vez supone tres posiciones de GPS y 90 muestras de cada uno de los sensores restantes en un segundo.

A la vista de los resultados, se concluye que el funcionamiento del sistema es muy positivo, al mismo tiempo que satisface el objetivo perseguido de almacenar las tramas de forma local.

Cabe destacar que también existe la posibilidad de remodelar el código creado de forma que sea más eficiente, con lo que se podría reducir el tiempo de ejecución del bucle y aumentar así el rendimiento del sistema.

7.3 Tamaño del programa completo

El *sketch* completo ocupa 86921 bytes en total, que no llega a ser ni un 23% del tamaño máximo de almacenamiento de programa en la memoria flash del microcontrolador, que es de 384000 bytes. Por lo tanto, existiría un sinfín de mejoras y añadidos posibles en cuanto al código se refiere.

7.4 Resultados

En este último apartado se van a presentar, de la forma más visual posible, los resultados que se han obtenido tanto enviando datos a un servidor remoto, como almacenando los datos de forma local.

En la figura 59 se observa una captura de la recepción de los datos recogidos por el potenciómetro, el acelerómetro y el sensor de efecto Hall, por parte del software de recepción diseñado por Víctor Huéscar López.

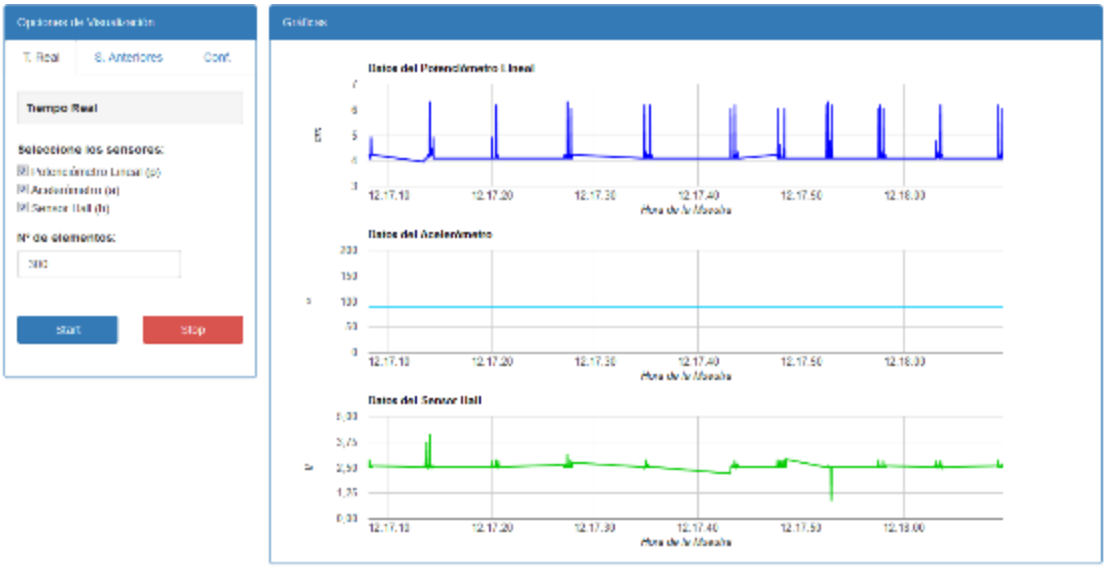

Figura 59 - Ejemplo de recepción de datos con el software de Víctor Huéscar López

Como se ve, la recepción y decodificación de los datos es correcta.

En la figura 60, por otra parte, se presenta el modo en el que el usuario puede acceder al documento de *datalogging* creado y almacenado en la memoria de la Intel Edison, mediante el uso de comandos Linux y un programa que permita establecer una comunicación serie.

```
Poky (Yocto Project Reference Distro) 1.7.2 edison ttyMFD2

edison login: root
root@edison:~# cd /home
root@edison:/home# ls -a
.           ..          info.txt    lost+found  root
root@edison:/home#
```

Figura 60 - Acceso al documento de datalogging

Por último, en las figuras 61 y 62 se presenta el aspecto final del dispositivo que podría ser instalado en la motocicleta, compuesto únicamente por la Intel Edison y los

blocks de alimentación y expansión de pines GPIO, que actuaría como cerebro del sistema.

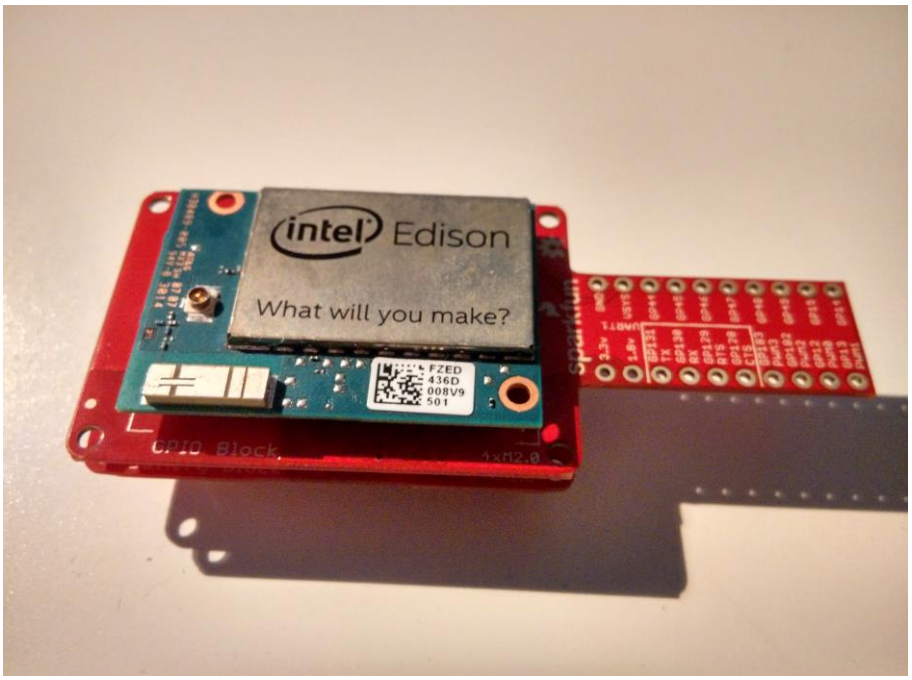

Figura 61 - Aspecto del dispositivo final (I)

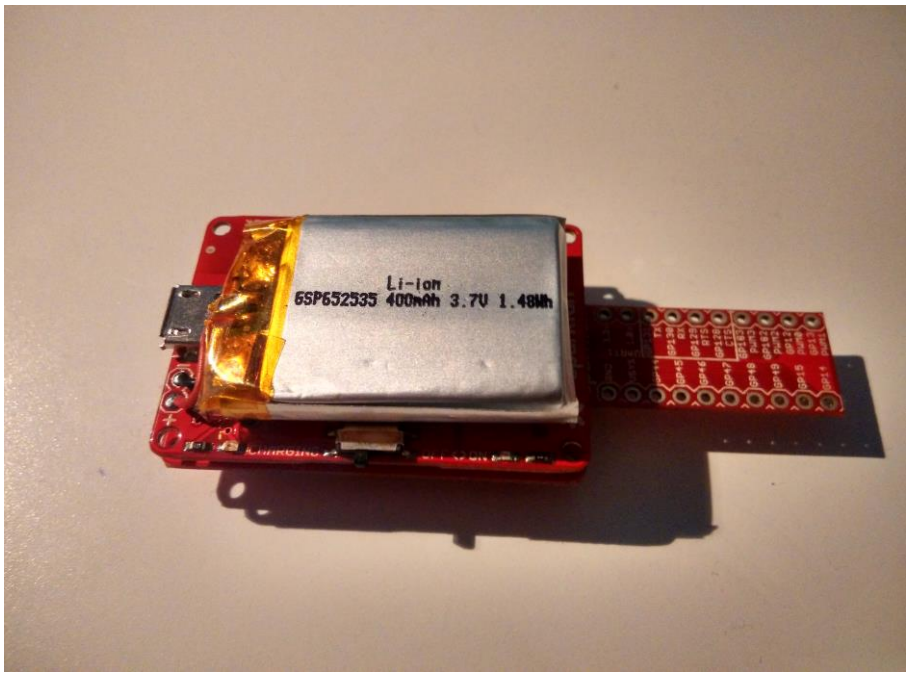

Figura 62 - Aspecto del dispositivo final (II)

Capítulo 8: Conclusiones y líneas futuras

8.1 Conclusiones

Durante el desarrollo de este trabajo se han llevado a cabo los objetivos propuestos inicialmente de forma satisfactoria.

En primer lugar, se han analizado en profundidad todas las capacidades de las que la plataforma Intel Edison está dotada, y se han llevado a cabo una serie de pruebas y experimentos que han corroborado que la elección inicial de la plataforma, a fin tanto de cumplir con una serie de ambiciones como de solventar los problemas que planteaba el sistema inicial, ha sido exitosa.

Hasta el momento, gran parte de los sistemas de telemetría existentes se han diseñado para una aplicación específica y por tanto, no admiten demasiadas modificaciones. Este sistema, no solo se basa en la plataforma Arduino para explotar su versatilidad y poder crear distintas variables del mismo en función del fin al que se quiera destinar, sino que además aumenta sus prestaciones y elimina ciertas limitaciones de Arduino gracias al uso de la plataforma Intel Edison.

El gran abanico de posibilidades que ya de por sí ofrece la plataforma de Intel, unida a la posibilidad de combinar los procesos llevados a cabo por el microcontrolador que dirige la parte de Arduino con el potente hardware que maneja el microprocesador principal, y que permite utilizar un entorno Linux para acceder a la información recogida, hace de esta una plataforma con la que se pueden realizar una gran cantidad de proyectos distintos, de forma muy económica pero dotados de gran potencia y versatilidad.

Se ha diseñado un prototipo real, gracias al que se ha podido comprobar y verificar el funcionamiento real y aplicado del sistema. Del mismo modo, se ha asegurado la posibilidad de portarlo a la motocicleta del equipo MotoUPCT de cara a la próxima competición, gracias a la posibilidad de almacenar los datos en tiempo real.

Por otro lado, y gracias a que la plataforma está dotada de Bluetooth y Wi-Fi, es posible complementar el sistema con algún tipo de aplicación compatible con Android, lo cuál aportaría gran valor al mismo.

8.2 Posibles líneas futuras

Ya es sabido que esta plataforma ofrece una amplia variedad de posibilidades a la hora de realizar algún tipo de proyecto similar al presente, pero si hay que concretar, existen seis pautas principales que podrían marcar seis líneas futuras muy a tener en cuenta.

En primer lugar, podría desarrollarse algún tipo de aplicación móvil Android o IOS que funcione con tecnología Bluetooth o Wi-Fi, mediante la que se pudieran visualizar los datos recogidos por el sistema en tiempo real en cualquier lugar, sin necesidad de estar delante de un ordenador.

En segundo lugar, y focalizando el sistema para su utilización en la próxima competición de MotoStudent, podría realizarse un estudio de posibles sensores industriales que permitan obtener datos de distintas variables extra de forma más precisa, así como llevar a cabo su instalación en la propia motocicleta para añadirlos al sistema y aumentar su valor.

En tercer lugar, se podría desarrollar una aplicación que funcione en el entorno Linux de la Intel Edison, que sea capaz de almacenar la información de forma ordenada en bases de datos compatibles con dicho entorno.

En cuarto lugar, sería interesante estudiar la forma más precisa de interpretar y extrapolar la información obtenida. Esto aportaría un gran valor al sistema, sobre todo a la hora de darle un uso puramente práctico y hacer que cualquier usuario sea capaz de interpretar los datos que está visualizando.

En quinto lugar, y vista la viabilidad del prototipo inicial, sería bueno diseñar un sistema definitivo, capaz de ser instalado en un vehículo de dos ruedas y que permitiera su uso mientras se monitorizan de forma remota las distintas variables a medir.

Por último, una buena línea futura a seguir sería la de estudiar otras aplicaciones prácticas para las cuáles esta plataforma podría ser útil como cerebro de un sistema de telemetría, y que permita estudiar y aprovechar todo su potencial de cara a conseguir aportar un valor comercial real al sistema desarrollado.

Bibliografía

[1] Intel Edison, información disponible en: https://www.arduino.cc/en/ArduinoCertified/IntelEdison

[2] Competición MotoStudent, información disponible en: http://www.MotoStudent.com/

[3] Arduino Mega, información disponible en: https://www.arduino.cc/en/Main/arduinoBoardMega

[4] Telemetría en la F1, información disponible en: http://www.vavel.com/es/formula1/366137-guia-tecnica-de-formula-1-vavel-capitulo-8-la-telemetria.html

[5] McLaren Applied Technologies, información disponible en: http://www.mclaren.com/appliedtechnologies/products/list/#!/electronics/telemetry

[6] Telemetría de la caída de Márquez en Mugello, información disponible en: http://www.motogp.com/es/noticias/2013/06/01/la-ca-da-de-m-rquez-analizada-desde-la-telemetr-a/161522

[7] Telemetría en cardiología, información disponible en: http://www.enfermeriaencardiologia.com/revista/res2904.htm

[8] Protocolo X10, información disponible en: http://www.eumed.net/rev/cccss/2015/01/domotica.html

[9] Telemetría en robótica, información disponible en: http://wiki.robotica.webs.upv.es/wiki-de-robotica/sensores/sensores-proximidad/sensor-laser/

[10] TFG de Pedro Celestino López Jiménez, disponible en: http://repositorio.upct.es/bitstream/handle/10317/4593/tfg338.pdf?sequence=1

[11] TFG de María Belén Pérez Muñoz, disponible en: http://repositorio.upct.es/bitstream/handle/10317/4365/tfg476.pdf?sequence=1

[12] TFG de Pedro José Conesa Sánchez, disponible en: http://repositorio.upct.es/bitstream/handle/10317/4171/pfc5824.pdf?sequence=1

[13] Módulo Intel Edison, información disponible en: http://download.intel.com/support/edison/sb/edisonmodule_hg_331189004.pdf

[14] Arquitectura Silvermont de Intel, información disponible en: http://www.pcactual.com/articulo/actualidad/noticias/12970/intel_silvermont_ofrece_bajo_consumo_alto_rendimiento.html

[15] Tecnología Hyper-Threading de Intel, información disponible en: http://www.intel.es/content/www/es/es/architecture-and-technology/hyper-threading/hyper-threading-technology.html

[16] Intel Edison Kit para Arduino, información disponible en: http://download.intel.com/support/edison/sb/edisonarduino_hg_331191007.pdf

[17] Interfaz PWM, información disponible en: https://www.arduino.cc/en/Tutorial/PWM

[18] Interfaz SPI, información disponible en: https://www.arduino.cc/en/Reference/SPI

[19] Interfaz I^2C, información disponible en: http://i2c.info/

[20] Interfaz serie de Intel Edison, información disponible en: https://communities.intel.com/thread/54236

[21] Módem 3G+GPS de Cooking Hacks, información disponible en: https://www.cooking-hacks.com/3g-gprs-shield-for-arduino-3g-gps

[22] Placa de evaluación con acelerómetro ADXL345, información disponible en: http://www.element14.com/community/docs/DOC-50973/l/analog-devices-eval-adxl345z-accelerometer-evaluation-board

[23] Sensor de efecto Hall SS495A de Honeywell, información disponible en:

http://es.rs-online.com/web/p/sensores-de-efecto-hall/2166231/

[24] Potenciómetro óptico GP2Y0A51SK0F, información disponible en: http://www.dema.net/pdf/sharp/GP2Y0A51SK0F_ED-08G027A.pdf

[25] Definición de trilateración, información disponible en: http://catedras.fcaglp.unlp.edu.ar/geofisica/referenciacion-en-geofisica/teoria/instrumental-y-tecnicas-topograficas/trilateracion

[26] Comandos AT para el módem SIM5218, información disponible en: http://www.mt-system.ru/sites/default/files/documents/simcom_sim5218_serial_atc_en_v1.38.pdf

[27] *Blocks* para Intel Edison, información disponible en: https://learn.sparkfun.com/tutorials/general-guide-to-sparkfun-blocks-for-intel-edison

[28] *Block* de batería, información disponible en:
https://www.sparkfun.com/products/13037?_ga=1.17454713.1977363858.1449576624

[29] *Block* GPIO, información disponible en:
https://www.sparkfun.com/products/13038?_ga=1.21698299.1977363858.1449576624

[30] *Block* microSD, información disponible en:
https://www.sparkfun.com/products/13041?_ga=1.93462237.1977363858.1449576624

[31] Modelo OSI, información disponible en:
http://www.exa.unicen.edu.ar/catedras/comdat1/material/ElmodeloOSI.pdf

[32] Protocolos UDP y TCP, información disponible en:
http://www.it.uc3m.es/lpgonzal/protocolos/transporte.php

[33] Librería SD, información disponible en: https://www.arduino.cc/en/Reference/SD

[34] Especificaciones para el uso de tarjetas microSD, información disponible en:
https://www.arduino.cc/en/Reference/SDCardNotes

[35] Rutina de *datalogging*, información disponible en:
https://www.arduino.cc/en/Tutorial/ReadWrite

[36] Modificación de la librería SD.cpp, información disponible en:
https://communities.intel.com/message/362361#362361

Anexo I: Código completo

```
//Add the SPI library so we can communicate with the ADXL345 sensor
#include <SPI.h>

//Add the SD library
#include <SD.h>

//Assign the Chip Select signal to pin 10.
int CS=10;

//This is a list of some of the registers available on the ADXL345.
//To learn more about these and the rest of the registers on the
ADXL345, read the datasheet!
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

//This buffer will hold values read from the ADXL345 registers.
char values[10];
//These variables will be used to hold the x,y and z axis
accelerometer values.
int x,y,z;

int8_t answer, answerGPS;       //Variables we are going to use to
check the answer received from the modem for each AT command sent.
int counter;
char gps_data[50];              //Array in which we will store the
information of the GPS
int analogPin = A0;             //Input pin of the signal that stores
the measure of the optical encoder
int analogPin1 = A1;            //Input pin of the signal that stores
the measure of the Hall sensor
int pot;                        //Variable that stores the analog
measure of the encoder
int angle;                      //Variable that stores the inclination
angle of the acelerometer
int hall;                       //Measured value from the Hall sensor
String trama;
char c[91];
String mark;
String G;
String info;
int onModulePin = 2;
int i;
int num;

void setup(){
  pinMode(onModulePin, OUTPUT);
  power_on();
  delay(1000);
  Serial1.begin(230400);
  Serial2.begin(115200);
  Serial.begin(230400);
//Configuration of the transmission rate of both serial ports
```

```
  sendATcommand("AT+NETCLOSE", "OK", 1000);
  sendATcommand("AT+CPIN=2223", "OK", 1000);
//Introduction of the SIM card PIN
  while((sendATcommand("AT+CREG?", "+CREG: 0,1", 500) ||
sendATcommand("AT+CREG?", "+CREG: 0,5", 500)) == 0 );
  delay(10);
//While no receiving a response from the modem that indicates that we
are linked to the network, wait
  sendATcommand("AT+CGSOCKCONT=1,\"IP\",\"telefonica.es\"", "OK",
1000);    //Configuramos el APN de Telefonica
  sendATcommand("AT+CGPSURL=\"supl.google.com:7276\"", "OK", 1000);
//Configuration of the GPS server
  sendATcommand("AT+CGPSSSL=0", "OK", 1000);
//Configuration of the non certified use of the GPS
  answerGPS = sendATcommand("AT+CGPS=1,2","OK", 1000);
//Configuration of the GPS for an assisted-mode
  sendATcommand("AT+NETOPEN=\"UDP\", 5558","OK",1000);
//Open the UDP port

  //Initiate an SPI communication instance.
  SPI.begin();
  //Configure the SPI connection for the ADXL345.
  SPI.setDataMode(SPI_MODE3);
  //Set up the Chip Select pin to be an output from the Arduino.
  pinMode(CS, OUTPUT);
  //Before communication starts, the Chip Select pin needs to be set
high.
  digitalWrite(CS, HIGH);

  //Put the ADXL345 into +/- 4G range by writing the value 0x01 to the
DATA_FORMAT register.
  writeRegister(DATA_FORMAT, 0x01);
  //Put the ADXL345 into Measurement Mode by writing 0x08 to the
POWER_CTL register.
  writeRegister(POWER_CTL, 0x08);  //Measurement mode
}

void loop(){
  answerGPS = sendATcommand("AT+CGPSINFO","+CGPSINFO:",10);
  if(answerGPS==1){
    counter = 0;
    do{
        gps_data[counter]  = Serial1.read();
        counter++;
    }
    while((counter<44) && (Serial1.available()));
    gps_data[counter] = '\0';
    if(gps_data[0] == ','){
      Serial.println("No GPS data available");
    }
  }

  //Frame creation

  trama = "s";
  mark = "g";
  i = 0;
  while(i<89){

    //Reading 6 bytes of data starting at register DATAX0 will
retrieve the x,y and z acceleration values from the ADXL345.
```

```
    //The results of the read operation will get stored to the
values[] buffer.
    readRegister(DATAX0, 6, values);

    //The ADXL345 gives 10-bit acceleration values, but they are
stored as bytes (8-bits). To get the full value, two bytes must be
combined for each axis.
    //The X value is stored in values[0] and values[1].
    x = ((int)values[1]<<8)|(int)values[0];
    //The Y value is stored in values[2] and values[3].
    y = ((int)values[3]<<8)|(int)values[2];
    //The Z value is stored in values[4] and values[5].
    z = ((int)values[5]<<8)|(int)values[4];

    if(z == 0){
      if(y>=0){
        angle = 90;
      }
      else{
        angle = -90;
      }
    }
    else{
      angle = abs(atan(y/z)*57.296);
    }
    angle = map(angle, -90, 90, 33, 126);
    pot = map(analogRead(analogPin), 0, 1023, 33, 126);
//Analog reading from de optical encoder
    hall = map(analogRead(analogPin1), 0, 1023, 33, 126);
//Analog reading from de Hall sensor

    c[i] = (char) pot;
    mark = mark + "p";

    c[i+1] = (char) angle;
    mark = mark + "a";

    c[i+2] = (char) hall;
    mark = mark + "h";

    i = i+3;

  }

  c[90] = '\0';

  //Must convert 'c' from 'string' to 'String' in order to concatenate
it with other data
  info = (String) c;
  G = String (gps_data);
  delay(10);
  num = 65+sizeof(c);
  trama = trama + (char) num + mark + G + info;
  Serial.println(trama);
  delay(10);
  Serial1.println("AT+UDPSEND=250,\"79.109.172.71\",5558");
  delay(10);
  Serial1.println(trama);
  delay(10);

  //Datalogging
```

```
  File dataFile = SD.open("info.txt", FILE_WRITE);
  delay(100);
  // if the file is available, write to it:
  if (dataFile) {
    dataFile.println(trama);
    dataFile.close();
    delay(100);
  }
  // if the file isn't open, pop up an error:
  else {
    Serial.println("Error opening info.txt");
  }
}

//This function will write a value to a register on the ADXL345.
//Parameters:
//  char registerAddress - The register to write a value to
//  char value - The value to be written to the specified register.
void writeRegister(char registerAddress, char value){
  //Set Chip Select pin low to signal the beginning of an SPI packet.
  digitalWrite(CS, LOW);
  //Transfer the register address over SPI.
  SPI.transfer(registerAddress);
  //Transfer the desired register value over SPI.
  SPI.transfer(value);
  //Set the Chip Select pin high to signal the end of an SPI packet.
  digitalWrite(CS, HIGH);
}

//This function will read a certain number of registers starting from
a specified address and store their values in a buffer.
//Parameters:
//  char registerAddress - The register addresse to start the read
sequence from.
//  int numBytes - The number of registers that should be read.
//  char * values - A pointer to a buffer where the results of the
operation should be stored.
void readRegister(char registerAddress, int numBytes, char * values){
  //Since we're performing a read operation, the most significant bit
of the register address should be set.
  char address = 0x80 | registerAddress;
  //If we're doing a multi-byte read, bit 6 needs to be set as well.
  if(numBytes > 1)address = address | 0x40;

  //Set the Chip select pin low to start an SPI packet.
  digitalWrite(CS, LOW);
  //Transfer the starting register address that needs to be read.
  SPI.transfer(address);
  //Continue to read registers until we've read the number specified,
storing the results to the input buffer.
  for(int i=0; i<numBytes; i++){
    values[i] = SPI.transfer(0x00);
  }
  //Set the Chips Select pin high to end the SPI packet.
  digitalWrite(CS, HIGH);
}

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(Serial1.available() > 0) Serial1.read();
  Serial1.println(ATcommand);
  x = 0;
  previous = millis();
  do{
    if(Serial1.available() != 0){
      response[x] = Serial1.read();
      Serial.write(response[x]);
      x++;
      if (strstr(response, expected_answer1) != NULL){
        answer = 1;
        Serial.write('\n');
      }
    }
  }
  while((answer == 0) && ((millis() - previous) < timeout));
  return answer;
}

void power_on(){
  uint8_t answer=0;
  answer = sendATcommand("AT", "OK", 2000);
  if (answer == 0){
    digitalWrite(onModulePin,HIGH);
    delay(2000);
    digitalWrite(onModulePin,LOW);
    while(answer == 0){
      answer = sendATcommand("AT", "OK", 2000);
    }
  }
}
```