TRABAJO FINAL DE GRADO

DESARROLLO DE UNA UNIDAD TELEMETRICA PARA MOTOCICLETA BASADA EN ARDUINO

GRADO EN INGENIERIA ELECTRONICA INDUSTRIAL Y AUTOMATICA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DEL DISEÑO

AUTOR: ERNESTO SAN VALENTÍN CALVET

TUTOR: CARLOS DOMINGUEZ

ETSID

UNIVERSITAT POLITECNICA DE VALENCIA

TRABAJO FIN DE GRADO:

DESARROLLO DE UNA UNIDAD TELEMETRICA PARA MOTOCICLETA BASADA EN ARDUINO

DOCUMENTOS:

1. MEMORIA
2. CODIGO
3. PRESUPUESTO
4. PRUEBA DE FUNCIONAMIENTO

AUTOR: ERNESTO SAN VALENTIN CALVET

TUTOR: CARLOS DOMINGUEZ

TITULACION: GRADO EN INGENIERIA ELECTRONICA INDUSTRIAL Y AUTOMATICA, ETSID

CURSO ACADEMICO 2018-2019

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

ETSID

UNIVERSITAT POLITECNICA DE VALENCIA

TRABAJO FIN DE GRADO:

DESARROLLO DE UNA UNIDAD TELEMETRICA PARA MOTOCICLETA BASADA EN ARDUINO

DOCUMENTO:

- MEMORIA

AUTOR: ERNESTO SAN VALENTIN CALVET

TUTOR: CARLOS DOMINGUEZ

TITULACION: GRADO EN INGENIERIA ELECTRONICA INDUSTRIAL Y AUTOMATICA, ETSID

CURSO ACADEMICO 2018-2019

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

INDICE

1. Índice De Ilustraciones 7
2. Introducción 10
3. Objetivos 10
4. Uso/Aplicación 10
   4.1. Esquema de funcionamiento 11
   4.2. Funcionamiento 12
   4.3. Alternativas Generales Para Resolver El Problema 13
5. Telemetría 15
   5.1. ¿Qué es la Telemetría? 15
   5.2. Telemetría aplicada a la competición 15
   5.3. Telemetría en el motociclismo 16
   5.4. Sensores de una MotoGP 16
   5.5. Condicionantes que pueden afectar a este proyecto 17
6. Lenguaje De Programación 19
7. Protocolos De Comunicación Empleados 20
   7.1. Comunicación i2c 20
      7.1.1. Función de Lectura/Escritura 21
   7.2. Protocolo de comunicación UART 21
   7.3. SPI 22
      7.3.1. Ventajas 22
      7.3.2. Desventajas 22
      7.3.3. Bus SPI en Arduino 23
8. NMEA 24
   8.1. Transmisión de datos NMEA 24
9. Hardware Empleado 26
   9.1. Arduino 26
   9.2. Porqué Arduino 29
   9.3. Arduino MEGA 2560 30
      9.3.1. Características Generales 30
      9.3.2. Por qué Arduino Mega 2560 31
      9.3.3. Arduino Mega 2560 Pin Mapping 32
   9.4. MPU6050 35
      9.4.1. Características Técnicas 35
      9.4.2. Conexiones 36

9.4.3. Comunicación 36
9.4.4. Funcionamiento 36
9.4.5. Programación Simplificada 37
9.4.6. Diagrama de Bloques 37
9.5. GPS GY-NEO6MV2 38
9.5.1. Características Técnicas 39
9.5.2. Conexiones 39
9.5.3. Comunicación 39
9.5.4. Programación Simplificada Obtención de Datos 40
9.5.5. Programación Simplificada Detección de Paso por Meta 40
9.5.6. Diagrama de Bloques 41
9.6. Módulo de almacenamiento SD 42
9.6.1. Características Técnicas 43
9.6.2. Conexiones 43
9.6.3. Programación Simplificada 44
9.6.4. Diagrama de Bloques 44
9.7. Pantalla Nextion 45
9.7.1. Características Técnicas 45
9.7.2. Conexiones 46
9.7.3. Programación Simplificada 46
9.7.4. Diagrama de Bloques 47
9.8. Gestor de Tareas 48
10. Software Empleado 51
10.1. Plataforma De Desarrollo Arduino 51
10.2. Plataforma De Desarrollo Nextion 52
10.3. Excel 54
11. Prototipo Final 56
11.1. Funcionamiento 56
11.2. Colocación En La Motocicleta 57
12. Presupuesto 74
12.1. Costes de Desarrollo 74
12.2. Costes por Unidad 75
12.3. Costes Unidad “low cost” 75
13. Prueba Del Prototipo Final 78
14. Conclusiones 83

15. Opciones De Mejora....86
16. Bibliografía....88
16.1. Telemetría....88
16.2. Lenguaje de Programación....88
16.3. Protocolos de Comunicación....88
16.4. NMEA....88
16.5. Arduino/Arduino Mega 2560....88
16.6. MPU6050....88
16.7. GPS GY-NEO6MV2....89
16.8. Modulo de almacenamiento SD....89
16.9. Nextion....89

1. Índice De Ilustraciones

Ilustración 1: Esquema de funcionamiento 11
Ilustración 2: Valoración de características 14
Ilustración 3: Mediciones telemétricas más comunes 15
Ilustración 4: Funcionamiento protocolo i2c 20
Ilustración 5: Funcionamiento protocolo UART 21
Ilustración 6: Tabla de puertos SPI Arduino 23
Ilustración 7: Protocolos GPS 24
Ilustración 8: Logo comunidad Arduino 26
Ilustración 9: Placa Arduino UNO 27
Ilustración 10: Tabla de Características Arduino 28
Ilustración 11: Placa Arduino MEGA2560 30
Ilustración 12: Procesador ATMEGA2560 30
Ilustración 13: Tabla Características Arduino MEGA 2560 30
Ilustración 14: PinOut ATMEGA 2560 32
Ilustración 15: Ejes de coordenadas MPU6050 35
Ilustración 16: Conexiones Arduino-MPU6050 36
Ilustración 17: Diagrama de bloques MPU6050 37
Ilustración 18: Módulo GPS GY-NEO6MV2 38
Ilustración 19: Conexiones Arduino-GPS 39
Ilustración 20: Diagrama de bloques GPS 41
Ilustración 21: Módulo de almacenamiento SD 42
Ilustración 22: Conexiones Arduino-SD 43
Ilustración 23: Puertos SPI Arduino Mega 43
Ilustración 24: Diagrama de bloques SD 44
Ilustración 25: Pantalla Nextion nx32 45
Ilustración 26: Conexiones Arduino-Pantalla Nextion 46
Ilustración 27: Diagrama de bloques Pantalla Nextion 47
Ilustración 28: Caso GPS más desfavorable 48
Ilustración 29: Tabla de equivalencia de coordenadas 48
Ilustración 30: Radio de Cobertura GPS 49
Ilustración 31: Plataforma de desarrollo Arduino 51
Ilustración 32: Plataforma Nextion Editor 52
Ilustración 33: Pantalla Principal del proyecto 53
Ilustración 34: Menú de selección de circuitos 53
Ilustración 35: Menú de exportacion CSV-Excel 54
Ilustración 36: Tablas Excel generadas 54
Ilustración 37: Almacenamiento de datos registrados 55
Ilustración 38: Graficos de datos almacenados por el proyecto 55
Ilustración 39: Prototipo Final I 56
Ilustración 40: Prototipo Final II 56
Ilustración 41: Ejemplo de Colocación 57
Ilustración 42: Costes de Desarrollo 74
Ilustración 43: Costes por Unidad 75
Ilustración 44: Costes por Unidad **low cost** 75
Ilustración 45: Coordenadas “meta” 78
Ilustración 46: Mediciones registradas en la SD 78

Ilustración 47: Gráfico de Velocidades registradas 79
Ilustración 48: Gráfico de ángulos de inclinacion (Roll) registrados 79
Ilustración 49: Gráfico de ángulos de picado (Pitch) registrados 80
Ilustración 50: Registro de tiempos de paso por meta 80

2. Introducción

En el presente proyecto se estudiará la viabilidad y se realizará el desarrollo de una unidad telemétrica basada en Arduino y con componentes de reducido coste, así como su implementación en una motocicleta real para intentar alcanzar unos resultados similares a los que logran obtener los equipos de competición, siempre recordando el carácter amateur del proyecto pero con la finalidad de conseguir unos resultados lo mejores y más similares posibles a los de equipos profesionales manteniendo el coste total del proyecto al mínimo.

3. Objetivos

En este proyecto se busca la creación e implementación de una unidad electrónica de adquisición de datos para una motocicleta. Este sistema estará basado en un microcontrolador Arduino caracterizado por ser una plataforma de software y hardware totalmente libre y con una gran comunidad de usuarios. Además de la medición de los datos telemétricos de la motocicleta, estos serán mostrados en tiempo real al piloto mediante el uso de una pantalla y almacenados en una tarjeta microSD para un posterior estudio y análisis por parte del mismo piloto o de su equipo.

- Búsqueda y estudio de las técnicas telemétricas utilizadas por los equipos de alta competición.
- Búsqueda de alternativas viables y accesibles económicamente para obtener mediciones similares a las obtenidas en competición.
- Búsqueda y elección del microcontrolador principal del proyecto.
- Búsqueda y elección de los componentes y módulos de adquisición de datos los cuales proporcionen unos resultados aceptables sin un coste elevado.
- Montaje, programación y sincronización del microcontrolador y de todos sus módulos.
- Prueba del prototipo final y estudio de los datos obtenidos.

4. Uso/Aplicación

Una vez expuestos los objetivos se continua con la elección de las variables y magnitudes relevantes para la conducción de una motocicleta en circuito:

- Tiempo por vuelta
- Velocidad actual
- Ángulos de inclinación (Roll)
- Ángulos de picado (Pitch)
- Posicionamiento GPS

Con la medición de estas variables se puede conocer los tiempos por vuelta al circuito de la motocicleta, su velocidad, su posición con el plano, así como la posibilidad de realizar una representación de la trazada efectuada por el piloto en la pista.

Además de esto, se registrarán velocidades máximas, así como ángulos máximos de inclinación y picado para conocer las situaciones más extremas que ha alcanzado el piloto.

Para realizar todas estas mediciones se necesitará un módulo GPS con antena propia, una unidad inercial IMU, un módulo de lectura/escritura SD y una pantalla LCD.

4.1.Esquema de funcionamiento

Tal y como se ha explicado en el apartado de objetivos, en el siguiente esquema de funcionamiento se puede apreciar el trabajo conjunto de la unidad telemétrica siendo sus dos módulos de adquisición de datos los encargados de recibir los valores de velocidad, posición, tiempos por vuelta así como ángulos de inclinación y picado, enviando periódicamente estos datos para su almacenamiento en una memoria SD y para mostrarlos también en tiempo real al piloto de la motocicleta, todo ello coordinado por el microcontrolador Arduino.

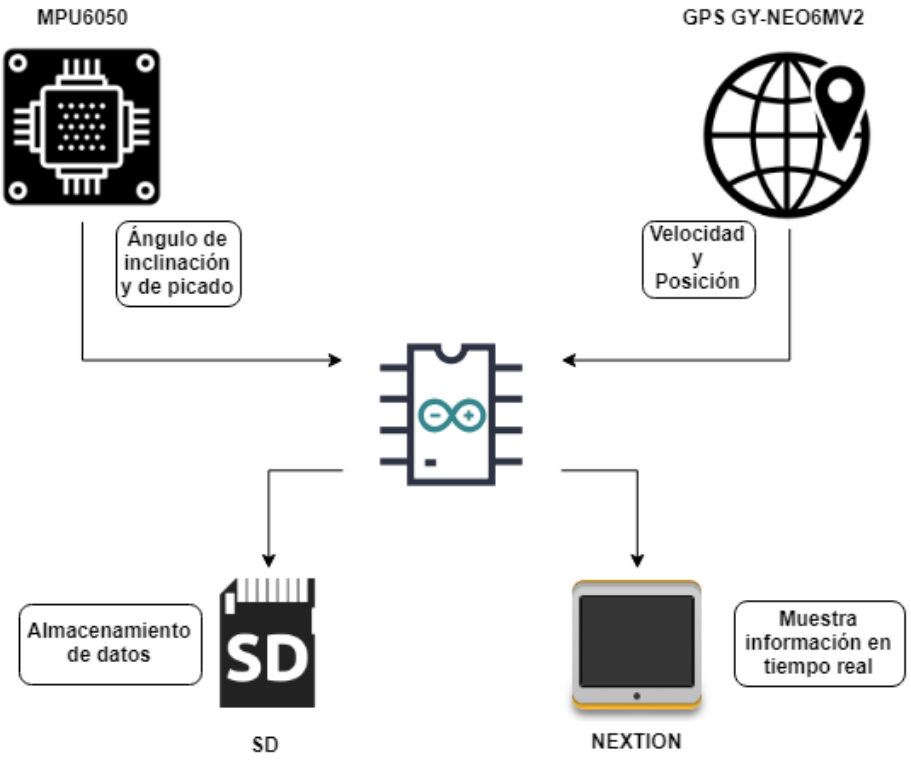


Ilustración 1: Esquema de funcionamiento

4.2.Funcionamiento

El funcionamiento de la unidad telemétrica se basa en un bucle continuo controlado por el Arduino en el cual mediante un gestor de tareas se van lanzando las funciones de medición y guardado de datos que serán detalladas más adelante en sus respectivos apartados. Se ha optado por un gestor de tareas debido a que el microcontrolador no es capaz de realizar una multitarea, es por ello por lo que en base a los tiempos mínimos de cada módulo se realizan las mediciones en cascada, realizando una función en repetidas ocasiones mientras transcurre el tiempo necesario entre las mediciones más lentas.

En este proyecto encontramos un cuello de botella con la lectura de los valores GPS, los cuales pueden ser obtenidos con una velocidad máxima de 1 lectura por segundo, permitiendo realizar por ejemplo varias lecturas de la IMU entre una lectura GPS y la siguiente.

4.3.Alternativas Generales Para Resolver El Problema

Se ha realizado una comparativa basada en las características más importantes para la elección de controlador del proyecto valorando 4 posibles candidatos y puntuando (de 4 a 1) sus características más determinantes.

- COSTE
- CAPACIDAD DE COMPUTO
- DIFICULTAD
- MEMORIA
- NUMERO DE PUERTOS
- INFORMACION DISPONIBLE

Las distintas opciones son:

- **Arduino UNO**
  - **Explicado en su apartado correspondiente más adelante (pag.23).**

- **Arduino MEGA**
  - **Explicado en su apartado correspondiente más adelante (pag.26).**

- **RaspBerry Pi**
  - COSTE: [30-50€] Dependiendo modelo y tienda de compra.
  - CAPACIDAD DE COMPUTO: [700MHz - 1.5GHz]
  - DIFICULTAD: Alta ya que se debe instalar un SO sobre el cual luego se trabaja media otra plataforma con los sensores del proyecto.
  - MEMORIA: [256 MB – 3GB]
  - NUMERO DE PUERTOS: Depende del modelo, pero salvo en Arduino uno hay más que suficientes.
  - INFORMACION DISPONIBLE: Existe bastante información disponible en forma de manuales, tutoriales y ejemplos de proyectos.

- **STM32F4**
  - COSTE: [20-30€]
  - CAPACIDAD DE COMPUTO: [84-180MHz]
  - DIFICULTAD: Muy alta ya que existe poca información de fácil acceso, así como la cantidad de ejemplos también es muy baja en comparación con los otros micros.
  - MEMORIA: [512-2048KB]
  - NUMERO DE PUERTOS:
  - INFORMACION DISPONIBLE: Información muy escasa en comparación con las otras alternativas.

Valorando estas características el resultado inicial que se comprobará finalmente en las conclusiones del proyecto es que el controlador más viable para realizar con éxito el proyecto es el Arduino Mega.

Controlador	Coste	Capacidad de Computo	Dificultad	Memoria	Num. Puertos	Informacion Disp.	Resultado
Arduino UNO	4	1	4	1	2	4	16
Arduino MEGA	3	2	4	2	3	4	18
RaspBerry Pi	1	4	1	3	4	2	15
STM32F4	2	3	1	4	4	1	15

Ilustración 2: Valoración de características

5. Telemetría

5.1.¿Qué es la Telemetría?

La telemetría es la tecnológica que hace posible la medición a distancia de magnitudes físicas que afectan a un objeto o cuerpo con el objetivo de recopilar información para enviarla o almacenarla y permitir su estudio desde un centro de control capacitado.

Por norma general la información telemétrica es enviada utilizando una comunicación inalámbrica, aunque también es posible emplear teléfonos, redes de ordenadores o cableado de distinto tipo.

Sus principales aplicaciones en el mundo actual son la investigación, la medicina, la defensa y los grandes sistemas, ya sean aviones, naves espaciales, redes de suministro o plantas industriales complejas entre muchas otras, todas ellas debido a una necesidad de control de ciertas variables que afectan a tales sistemas con el fin de conseguir el correcto y eficiente funcionamiento y la seguridad de los mismos, permitiendo conocer su estado en todo momento y poder prevenir o corregir fallos y accidentes mediante el envío de alertas a los centros de control.

5.2.Telemetría aplicada a la competición

Ilustración 3: Mediciones telemétricas más comunes

Actualmente la tecnología telemétrica está totalmente vinculada al mundo de la competición ya sean coches, motos, barcos o cualquier tipo de vehículo en los cuales se instalan distintos sensores de adquisición de datos además de una unidad de control. Con el uso de esta tecnología los equipos buscar alcanzar el conocimiento total y en tiempo real de todo lo que sucede en el vehículo para ayudar en la manera posible al piloto que lo controla incluso pudiendo comparar los datos con los de otro piloto para estudiar distintas formas de pilotaje.

Entre muchas otras cosas, mediante la telemetría se puede actualizar los settings de distintos componentes del vehículo, ya sean suspensiones, control de tracción, respuesta del acelerador etc… en tiempo real y conseguir un funcionamiento óptimo del vehículo en todo momento. Por otro lado, es un punto clave a la hora de estudiar errores, caídas o fallos puesto que se puede obtener información punto a punto de cualquier momento pasado.

5.3.Telemetría en el motociclismo

En la competición de motociclismo profesional la telemetría es un elemento totalmente indispensable hoy en día, los equipos reparten sensores de adquisición de datos por toda la moto. Aunque los equipos de competición pueden montar cualquier tipo de sensor para los test, todos los sensores que se empleen en carrera deben estar homologados con el objetivo de equilibrar la competición de los distintos equipos.

5.4.Sensores de una MotoGP

- **Velocidad:** Situados en ambas ruedas, se encargan de recopilar información por separado y en tiempo real desde cada rueda, permitiendo mostrar la velocidad actual del vehículo, así como detectar deslizamientos y perdidas de tracción las cuales son corregidas mediante el trabajo de la ECU y los sistemas de control.
- **Recorrido de suspensión:** Recopilan información sobre el recorrido, la velocidad y la posición de la amortiguación permitiendo controlar los levantamientos involuntarios de las ruedas “antiwheelie” o ajustando la respuesta de la suspensión ante irregularidades del terreno o para una posición determinada de la pista.
- **Posición del Acelerador:** Empleando el sistema “Ride by wire”, se transmite la posición del acelerador a la ECU.
- **Giroscopios y Acelerómetros:** Componen la plataforma inercial del vehículo y obtienen información sobre los ángulos de inclinación, así como su la aceleración o deceleración.

- **Sensores del Motor:** Los sensores que se pueden encontrar en el motor son los de temperatura del refrigerante, presión de aceite, revoluciones, inyección de combustible, temperatura del aire entro otros.
- **Presión de los Neumáticos:** La presión de los neumáticos es registrada como elemento de seguridad para evitar reventones o pinchazos además de ser crucial para el desarrollo de nuevos neumáticos.
- **Posición en el circuito:** Mediante la información de posición generada por el transpondedor de una moto, esta se puede configurar a todos los niveles par un funcionamiento idóneo para las características particulares de ese determinado sector, pudiendo realizar una configuración distinta para cada curva y recta de un circuito.

Con la información obtenida por estos sensores y el trabajo realizado por la ECU, se consigue un funcionamiento óptimo de la motocicleta en todo momento actuando activamente en los controles de tracción, deslizamiento, trabajo de la suspensión etc.

5.5.Condicionantes que pueden afectar a este proyecto

Aunque en este proyecto no contamos con las limitaciones impuestas por las direcciones de las distintas competiciones hay que tener en cuenta las distintas limitaciones de nuestro proyecto.

- **Capacidad del procesador:** Con el desarrollo de este proyecto se comprobará si Arduino es capaz de controlar tantos procesos al mismo tiempo o si, por el contrario, sería conveniente un procesador más potente.

- **Tamaño y peso**: estos dos aspectos deben tener unos valores contenidos puesto que la instalación y uso de este dispositivo no debe interferir en el manejo ni el funcionamiento de la motocicleta.

- **Necesidad de una base de datos de coordenadas**: con el objetivo de delimitar la línea de meta para el sistema de crono, se debe realizar una base de datos con las coordenadas de las líneas de meta de los diferentes circuitos en los que se vaya a rodar.

- **Tamaño de la pantalla**: este aspecto es importante ya que hay que conseguir una rápida y correcta visualización de los datos para el piloto con el objetivo de no distraer su atención en exceso, pero también permitir que la lectura de estos se pueda realizar de una forma sencilla y que no afecte al manejo de la motocicleta ni interfiera en el uso de cualquiera de sus mandos.

- **Encapsulado para no interferir en las señales**: es importante realizar un encapsulado del módulo que no interfiera con las señales GPS que se utilizan.

6. Lenguaje De Programación

La programación para los proyectos basados en Arduino se realiza en un lenguaje propio basado en el lenguaje de programación C++ es por ello que es posible realizar una programación en dicho lenguaje sin demasiada complicación ya que esta plataforma es capaz de interpretar todas las funciones y comandos de C++ siendo posible de esta manera una programación sencilla e intuitiva.

La estructura que todos los proyectos Arduino presentan se compone de dos bloques donde el **setup()** es la parte que realiza la configuración previa del sistema y el **loop()** realiza el bucle cíclico de instrucciones implementadas por el programador.

Salvo por esta estructura fija, la programación en Arduino se puede realizar de la misma manera que se programaría cualquier proyecto en C++, implementando sus funciones, sintaxis, variables, librerías o estructuras de control.

7. Protocolos De Comunicación Empleados

7.1.Comunicación i2c

El protocolo de comunicación i2c recibe su nombre del inglés Inter-Integrated Circuit y es uno de los protocolos más utilizados para la comunicación interna de los circuitos, permitiendo el paso de información entre el controlador principal y los distintos periféricos o módulos que integrados.

El I2C está diseñado como un sistema maestro-esclavo donde la transferencia de datos es siempre iniciada por un maestro y se espera la reacción del sistema esclavo. Existe la posibilidad de configurar un sistema multimaestro y el control para el acceso al bus de datos se realiza mediante la programación consiguiendo que todos los maestros puedan turnarse para su uso.

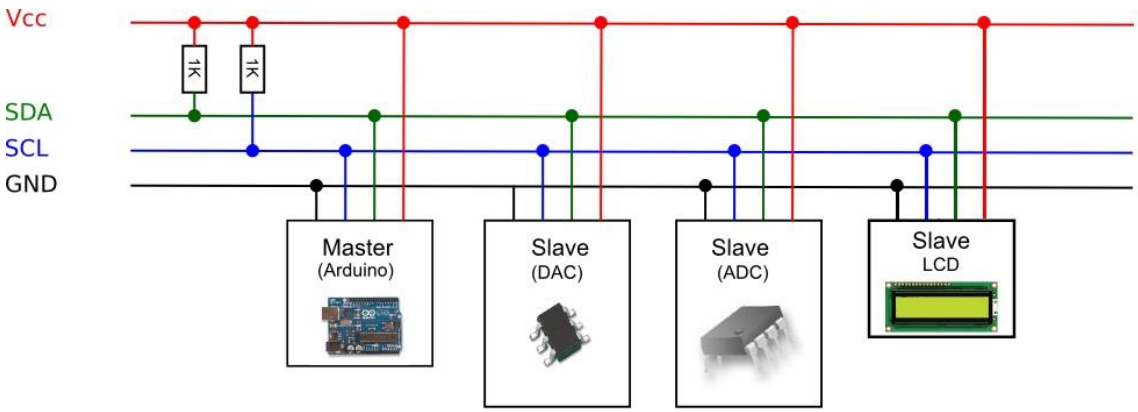


Ilustración 4: Funcionamiento protocolo i2c

El uso del protocolo I2C necesita cuatro líneas de conexión, una de alimentación VCC, otra de masa GND y dos líneas de señal, el reloj CLK (Serial Clock) y la línea de datos SDA (Serial Data) siendo necesaria resistencias pull-up hacia VCC para ambas líneas de señal.

Es importante tener en cuentas que la comunicación mediante el bus I2C funciona con lógica positiva, siendo un nivel alto en la línea de datos un 1 lógico y un nivel bajo correspondería a un 0.

El I2C utiliza un espacio de direccionamiento 1 byte donde los 7 primeros bits representan la dirección permitiendo hasta 112 nodos en un bus y el octavo bit es el que comunica al esclavo si debe recibir datos del maestro (nivel bajo) o enviarlos (nivel alto). Es importante tener en cuenta que todos los circuitos integrados tienen una dirección predeterminada por el fabricante en la cual los últimos tres bits pueden ser fijados como pines de control.

Tal y como se especifica en el párrafo anterior, el ultimo bit de la cadena de datos enviados es el que define la función que desempleará el esclavo.

7.1.1. Función de Lectura/Escritura

Cuando el octavo bit de la cadena es un 1 la orden es recibir o leer una información procedente del módulo esclavo, cuando el bit es un 0 la función es escribir. Para ambas funciones el esclavo devuelve un bit de reconocimiento de solicitud al maestro confirmando que está en condiciones de recibir o proporcionar información, este bit de respuesta se conoce como ACK.

También es importante tener en cuenta que el número de bytes de una transmisión no está restringido de ningún modo y viene dado por las características del esclavo. Cada byte de 8 bites continua con la recepción de un bit de reconocimiento ADK.

7.2.Protocolo de comunicación UART

El nombre de este protocolo de comunicación corresponde a las siglas en ingles de Universal Asynchronous Receiver-Transmitter que es el dispositivo que controla los puertos y dispositivos serie del circuito.

Las funciones principales del UART son manejar interrupciones de dispositivos conectados mediante los puertos serie y convertir esos datos de formato paralelo a formato serie, realizando una transmisión de datos individual la cual es reensamblada en el UART receptor donde estos datos pueden ser utilizados.

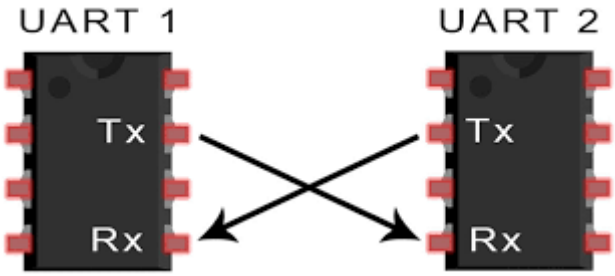


Ilustración 5: Funcionamiento protocolo UART

Como se puede observar en la imagen, estos buses están compuestos por 2 pistas TX y RX (además de la alimentación y la masa) conectado el TX de un módulo al RX del otro y viceversa, consiguiendo de esta manera una comunicación bidireccional de ambos módulos donde ambos son capaces de enviar y recibir información.

Este tipo de protocolo de comunicación es bastante utilizado para la programación de sistemas embebidos debido a que no es necesario que los microcontroladores implicados cuenten con un gestor de arranque o "bootloader" precargado.

7.3.SPI

Arduino dispone de una interfaz SPI de comunicación la cual será empleada con el módulo lector de tarjetas SD. Este bus de datos presente en la placa Arduino, es uno de sus principales medios de comunicación con los periféricos que se pueden utilizar.

El bus SPI emplea una arquitectura tipo Maestro-Esclavo donde el dispositivo maestro es el encargado de iniciar la comunicación con los dispositivos esclavos, los cuales no son capaces de iniciar comunicaciones ni compartir información entre ellos directamente. Otra de sus características es que se trata de un bus asíncrono por lo que se emplea una señal de reloj para mantener el funcionamiento correcto del conjunto de dispositivos.

La conexión SPI necesita únicamente 4 líneas de datos:

- MOSI → Comunicación maestro-esclavo.
- MISO → Comunicación esclavo-maestro.
- SCK → Señal de reloj.
- SS → Selección del dispositivo con el que iniciar comunicación.

7.3.1. Ventajas

- Alta velocidad de transmisión
- Bajo coste
- Posibilidad de envío de secuencias de bit de cualquier tamaño

7.3.2. Desventajas

- Requiere 4 líneas de conexión
- Adecuado solo para cortas distancias
- No dispone de mecanismo de control del envío o recepción correcta del mensaje.

7.3.3. Bus SPI en Arduino

MODELO	SS	MOSI	MISO	SCK
Uno	10	11	12	13
Nano	10	11	12	13
Mini Pro	10	11	12	13
Mega	53	51	50	52

Ilustración 6: Tabla de puertos SPI Arduino

Dado que en este proyecto se emplea un Arduino Mega, los puertos utilizados los D50,D51,D52,D53.

8. NMEA

El protocolo de comunicación NMEA recibe su nombre de la asociación fundada en 1957 National Marine Electronics Association, creada por fabricantes de componentes electrónicos con el objetivo de crear un sistema de comunicación naval estándar.

El primer protocolo recibió el nombre de NMEA 0183 y actualmente sigue en uso para la mayoría de los componentes electrónicos, cuenta con una trasmisión de datos en formato serial con una velocidad de 4800 bits por segundo (baudios).

La versión NMEA 2000 mejora la velocidad de transmisión y permite la conexión de equipos que envíen y reciban información de forma simultánea haciendo innecesario un distribuidor de información como se precisa en el 0183.

8.1. Transmisión de datos NMEA

$GPRMC,044235.000,A,4322.0289,N,00824.5210,W,0.39,65.46,020615,,,A*44

A continuación, se utilizará un ejemplo de una cadena de datos recibida en formato NMEA para explicar a que corresponden los diferentes grupos de caracteres recibidos.

- Todas las transmisiones de este protocolo comienzan por **$.**
- **GPRMC** corresponde al formato de datos mínimos recomendados.

NMEA 2.0

Name	Garmin	Magellan	Lowrance	SiRF	Notes:
GPAPB	N	Y	Y	N	Auto Pilot B
GPBOD	Y	N	N	N	bearing, origin to destination - earlier G-12's do not transmit this
GPGGA	Y	Y	Y	Y	fix data
GPGLL	Y	Y	Y	Y	Lat/Lon data - earlier G-12's do not transmit this
GPGSA	Y	Y	Y	Y	overall satellite reception data, missing on some Garmin models
GPGSV	Y	Y	Y	Y	detailed satellite data, missing on some Garmin models
GPRMB	Y	Y	Y	N	minimum recommended data when following a route
GPRMC	Y	Y	Y	Y	minimum recommended data
GPRTE	Y	U	U	N	route data, only when there is an active route. (this is sometimes bidirectional)
GPWPL	Y	Y	U	N	waypoint data, only when there is an active route (this is sometimes bidirectional)

Ilustración 7: Protocolos GPS

- **044235.000** representa la hora GMT 04:42:35
- **La A** confirma que el dato de posición está fijado y es correcto si por el contrario este no fuera valido aparecería una V.
- **4322.0289** representa la longitud 43º 22.0289´ y **N** representa el Norte.
- **00824.5210** representa la latitud 8º 24.5210´ y **W** representa el Oeste.
- **0.39** representa la velocidad en nudos.
- **65.46** representa la orientación en grados.
- **020615** representa la fecha 2 de Junio del 2015.

9. Hardware Empleado

9.1.Arduino

Ilustración 8: Logo comunidad Arduino

La plataforma Arduino se compone de una placa electrónica con un microcontrolador ATmega integrado y una serie de puertos de conexión que permiten la comunicación entre el controlador y otros dispositivos distintos tales como un ordenador, sensores o Shields.

Es importante remarcar que Arduino se trata de una plataforma para la creación de proyectos electrónicos de código abierto, toda ella basada en el uso de hardware y software libre, sencillo de utilizar y aprender y con una gran comunidad de usuarios detrás que se ayudan e implican en los proyectos que se van presentando.

Existen distintas placas de Arduino las cuales tienen distintas características y se diferencian principalmente unas de otras en el modelo del microcontrolador que integran y los números de puertos de conexión disponibles, aunque todas Comparten una misma arquitectura básica.

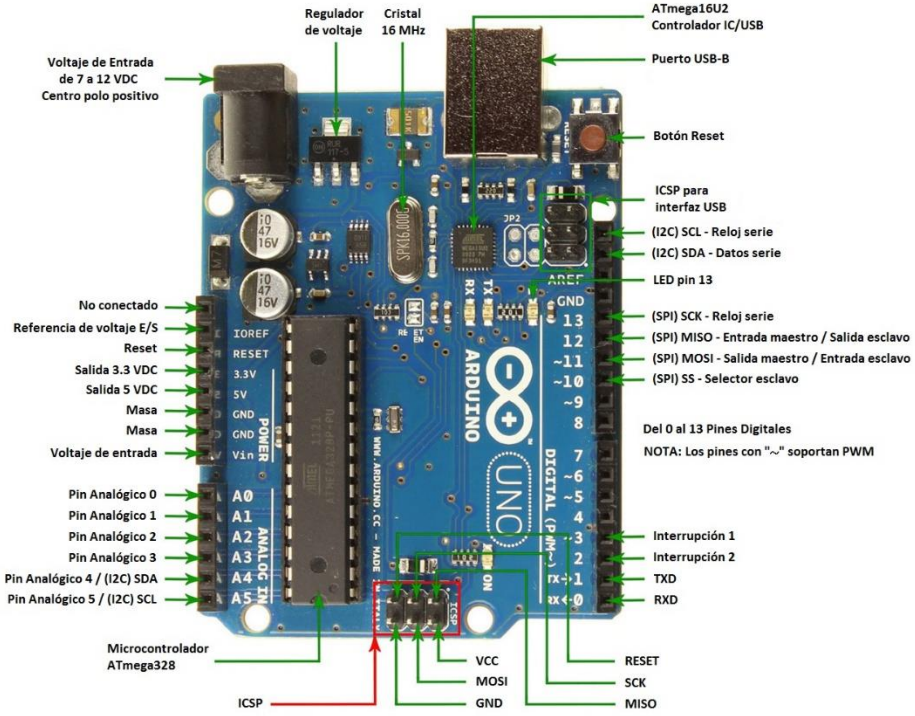


Ilustración 9: Placa Arduino UNO

- **Microcontrolador:** Se trata de un circuito integrado programable el cual es capaz de ejecutar una serie de ordenes previamente programadas en su memoria. Este componente está formado por varios bloques funcionales entre los cuales encontramos una unidad de procesamiento, una memoria y periféricos de entrada/salida de datos.

- **Memoria:** Algo que tienen en común todos los módulos de Arduino es que todos ellos cuentan con un módulo de memoria Flash donde almacenar el código del programador, también cuentan con otro módulo de memoria RAM la cual se emplea durante la ejecución de las acciones del programa y por último un módulo EEPROM el cual mantiene la información al apagar el dispositivo lo cual permite el guardado de datos ante un posible fallo o desconexión del controlador.

- **Alimentación:** Las placas Arduino cuentan con dos puertos distintos de alimentación, una alimentación mediante USB la cual proporciona 5V y también un conector tipo Jack por el que puede recibir un rango de tensión de entre 5 a 15V. Por otro lado, la propia placa es capaz de suministran tensión de alimentación a otros módulos o sensores a través de sus puertos de 3.3V y 5V.

- **Puertos Entrada/Salida:** Todas las placas de Arduino cuentan con un determinado número de pines de entrada analógica y entrada/salida digital por los que reciben y envían información. Algunos de esos pines pueden utilizarse como salidas PWM (simulando una salida analógica) o como puertos serie para utilizar protocolos de comunicación como el i2c y SPI.

- **Comunicaciones:** Las placas Arduino disponen de al menos un puerto USB por el cual se realiza la comunicación con el ordenador, así como un puerto serie para la comunicación con otros dispositivos, sensores o Shields.

- **Reloj:** Estos microcontroladores cuentan con un reloj interno que puede generar frecuencias desde 8 hasta 84 MHz, dependiendo de la placa utilizada con las cuales realizar los ciclos de trabajo del programa.

- **LEDs ON, RX, TX, Botón Reset:** Todas las placas cuentas con tres LEDs los cuales indican si están en funcionamiento o si emiten o reciben datos, así como un botón de Reset manual para poder reiniciar el dispositivo en cualquier momento.

- **Compatibilidad con entorno Arduino:** Todas las placas Arduino son compatibles con un entorno de programación libre e intuitivo, disponible para todas las plataformas (Windows, Mac, Linux), siendo este uno de los puntos fuertes de la programación con este tipo de controladores y también permitiendo la posibilidad de utilizar los mismos códigos, con pequeñas modificaciones, en distintos modelos de Arduino.

	Arduino Uno	Arduino Mega2560	Arduino Leonardo	Arduino Due	Arduino ADK	Arduino Nano	Arduino Pro Mini	Arduino Esplora
								
Microcontrolador	ATmega328	ATmega2560	ATmega32u4	AT91SAM3X8E	ATmega2560	ATmega168 (versão 2.x) ou ATmega328 (versão 3.x)	ATmega168	ATmega32u4
Portas digitais	14	54	20	54	54	14	14	-
Portas PWM	6	15	7	12	15	6	6	-
Portas analógicas	6	16	12	12	16	8	8	-
Memória	32 K (0,5 K usado pelo bootloader)	256 K (8 K usados pelo bootloader)	32 K (4 K usados pelo bootloader)	512 K disponível para aplicações	256 K (8 K usados pelo bootloader)	16 K (ATmega168) ou 32K (ATmega328), 2 K usados pelo bootloader	16 K (2k usados pelo bootloader)	32 K (4 K usados pelo bootloader)
Clock	16 Mhz	16 Mhz	16 Mhz	84 Mhz	16 Mhz	16 Mhz	8 Mhz (modelo 3.3v) ou 16 Mhz (modelo 5v)	16 Mhz
Conexão	USB	USB	Micro USB	Micro USB	USB	USB Mini-B	Serial / Módulo USB externo	Micro USB
Conector para alimentação externa	Sim	Sim	Sim	Sim	Sim	Não	Não	Não
Tensão de operação	5v	5v	5v	3.3v	5v	5v	3.3v ou 5v, dependendo do modelo	5v
Corrente máxima portas E/S	40 mA	40 mA	40 mA	130 mA	40 mA	40 mA	40 mA	-
Alimentação	7 - 12 Vdc	7 - 12 Vdc	7 - 12 Vdc	7 - 12 Vdc	7 - 12 Vdc	7 - 12 Vdc	3.35 - 12 V (modelo 3.3v), ou 5 - 12 V (modelo 5v)	5v

Ilustración 10: Tabla de Características Arduino

9.2.Porqué Arduino

Para la realización de este proyecto se ha optado por utilizar Arduino frente a otras placas similares como la STM32F4 o la Raspberry Pi entre otras, por diversas razones. En comparación con la mayoría de plataformas, Arduino cuenta con una enorme comunidad detrás que crea y modifica proyectos continuamente, también cuenta con un software gratuito y de fácil aprendizaje al alcance de cualquiera, así como una gran flexibilidad en cuanto al lenguaje y arquitectura de programación. Otro de los puntos a favor de la utilización de Arduino es el bajísimo coste comparado con sus rivales, así como la gran compatibilidad con la mayoría de módulos, sensores o shields del mercado. Por último, otro factor determinante para la elección de esta plataforma fue la gran cantidad de bibliotecas y ejemplos creados por la comunidad que se encuentra en internet con un acceso fácil y totalmente gratuito a la enorme cantidad de información que puedes encontrar.

9.3.Arduino MEGA 2560

El Arduino mega 2560 es una placa electrónica perteneciente a la familia Arduino la cual está basada en el microcontrolador ATmega2560 fabricado por Atmel.

Ilustración 12: Procesador ATMEGA2560

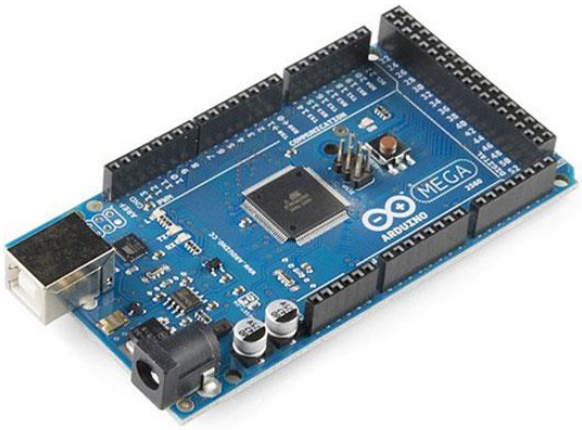


Ilustración 11: Placa Arduino MEGA2560

9.3.1. Características Generales

Microcontrolador	ATmega2560
Tensión de trabajo	5V
Tensión de entrada (recomendada)	7-12V
Tensión de entrada (límite)	6-20V
Pines Digitales I/O	54 (de los cuales 15 proporcionan salida PWM)
Pines de entradas Analógicas	16
DC Corriente por Pin I/O	20 mA
DC Corriente por Pin 3.3V	50 mA
Memoria Flash	256 KB de los cuales 8 KB se usan por el bootloader
SRAM	8 KB
EEPROM	4 KB
Velocidad del reloj	16 MHz
Largo	101.52 mm
Anchu	53.3 mm
Peso	37 g

Ilustración 13: Tabla Características Arduino MEGA 2560

Descripción General: Arduino mega 2560 cuenta con un microcontrolador compuesto por un cristal de reloj con una velocidad de procesamiento de 16MHz, una memoria Flash de 256KB en la cual 8 de ellos correspondiendo al bootloader de la placa. También cuenta con 8KB de memoria estática de acceso aleatorio SRAM, así como 4KB de ROM programable y borrable eléctricamente EEPROM la cual es posible leer y escribir mediante el uso de una biblioteca EEPROM específica.

Puertos: Entre sus características principales se encuentra su elevado número de pines de entrada/salida, con un total de 54 pines donde se pueden utilizar 14 salidas de PWM (Modulación por ancho de pulso), 16 entradas analógicas y 4 puertos serie UART.

Alimentación: su alimentación puede ser realizada a través de la conexión USB presente en la misma placa, pero también es posible realizarla mediante el uso de un adaptador de CA a CC. EL rango de funcionamiento de la tarjeta es de 6 a 20 voltios teniendo en cuenta que con una alimentación menor a 5V podría generar inestabilidad en su funcionamiento y una alimentación superior a 12V podría sobrecalentar el regulador interno de la placa y producir daños. Por ello, el rango de alimentación recomendado es de 7 a 12 voltios.

9.3.2. Por qué Arduino Mega 2560

Inicialmente el proyecto estaba pensado para la utilización de la placa Arduino Uno, por ser el escalón de entrada a la comunidad Arduino y por su sencillez y relación calidad-precio, con el desarrollo del proyecto ha surgido la necesidad de mayor potencia de procesamiento de datos y almacenamiento para la realización de todas las funciones del prototipo además de la necesidad de un mayor número de puertos para la conexión de todos los módulos necesarios para la toma de mediciones, su almacenamiento en un datalog y la posibilidad de mostrar información en tiempo real al usuario final. Ya que la intención era la realización del proyecto con una base Arduino se descarta la posibilidad de la utilización de otro tipo de tarjetas como podrían ser la familia STM32 o la RaspberryPi, es por ello por lo que se ha optado por la utilización de una de las unidades más potentes y completas de la familia Arduino la cual cumple con creces todos los requisitos del proyecto final sin disparar los precios ni elevar la dificultad de programación y conexión de todos sus componentes.

9.3.3. Arduino Mega 2560 Pin Mapping

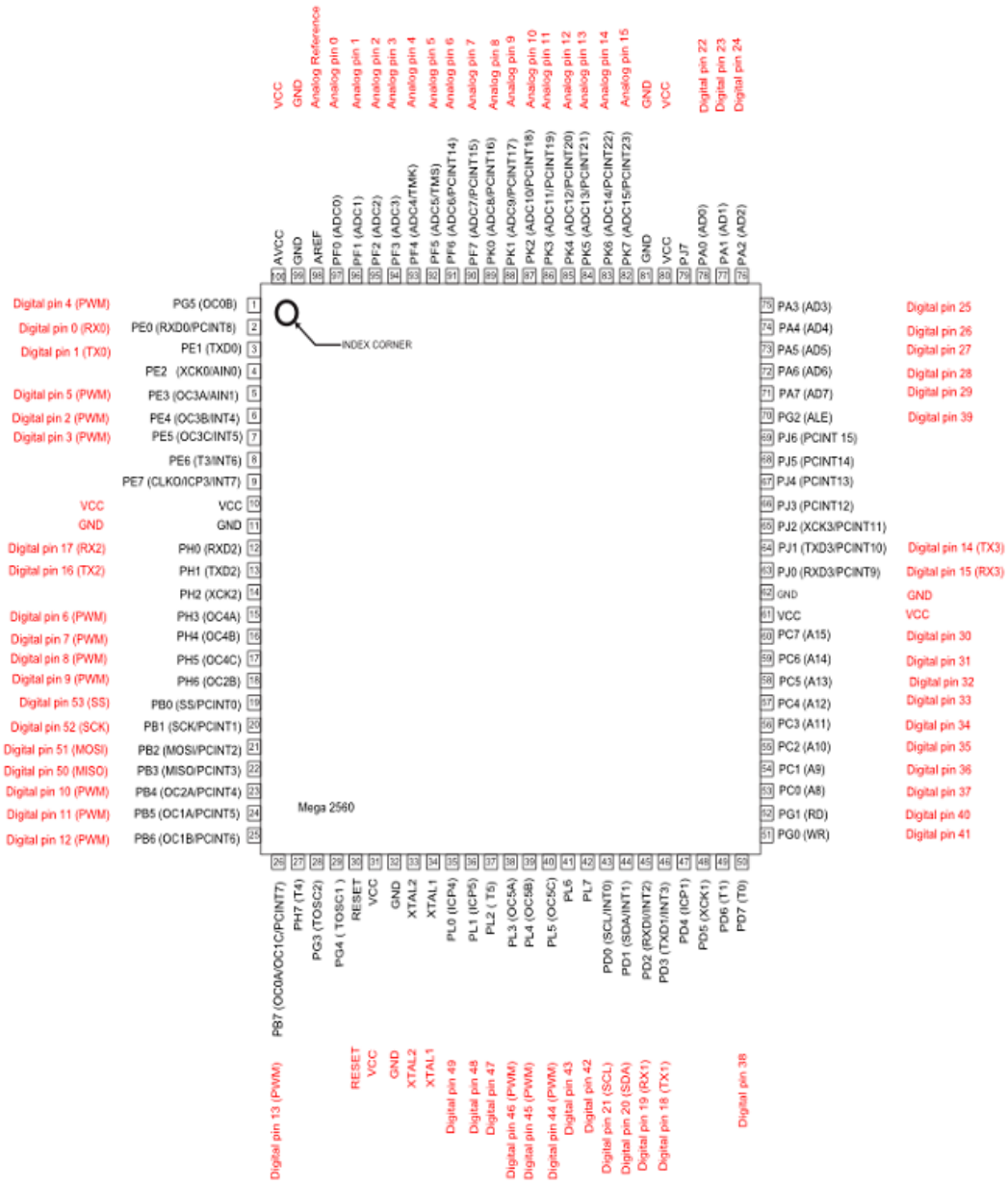


Ilustración 14: PinOut ATMEGA 2560

En la siguiente tabla, se muestran todos los puertos del Arduino 2560 y se marcan en rojo aquellos empleados en el desarrollo de este proyecto.

Número de Pin	Nombre	
1	PG5 (OC0B)	Digital pin 4 (PWM)
2	**PE0 (RXD0/PCINT8)**	**Digital pin 0 (RX0)**
3	**PE1 (TXD0)**	**Digital pin 1 (TX0)**
4	PE2 (XCK0/AIN0)	
5	PE3 (OC3A/AIN1)	Digital pin 5 (PWM)
6	PE4 (OC3B/INT4)	Digital pin 2 (PWM)
7	PE5 (OC3C/INT5)	Digital pin 3 (PWM)
8	PE6 (T3/INT6)	
9	PE7 (CLKO/ICP3/INT7)	
10	**VCC**	**VCC**
11	**GND**	**GND**
12	PH0 (RXD2)	Digital pin 17 (RX2)
13	PH1 (TXD2)	Digital pin 16 (TX2)
14	PH2 (XCK2)	
15	PH3 (OC4A)	Digital pin 6 (PWM)
16	PH4 (OC4B)	Digital pin 7 (PWM)
17	PH5 (OC4C)	Digital pin 8 (PWM)
18	PH6 (OC2B)	Digital pin 9 (PWM)
19	**PB0 (SS/PCINT0)**	**Digital pin 53 (SS)**
20	**PB1 (SCK/PCINT1)**	**Digital pin 52 (SCK)**
21	**PB2 (MOSI/PCINT2)**	**Digital pin 51 (MOSI)**
22	**PB3 (MISO/PCINT3)**	**Digital pin 50 (MISO)**
23	**PB4 (OC2A/PCINT4)**	**Digital pin 10 (PWM)**
24	**PB5 (OC1A/PCINT5)**	**Digital pin 11 (PWM)**
25	PB6 (OC1B/PCINT6)	Digital pin 12 (PWM)
26	PB7 (OC0A/OC1C/PCINT7)	Digital pin 13 (PWM)
27	PH7 (T4)	
28	PG3 (TOSC2)	
29	PG4 (TOSC1)	
30	RESET	RESET
31	VCC	VCC
32	GND	GND
33	XTAL2	XTAL2
34	XTAL1	XTAL1
35	PL0 (ICP4)	Digital pin 49
36	PL1 (ICP5)	Digital pin 48
37	PL2 (T5)	Digital pin 47
38	PL3 (OC5A)	Digital pin 46 (PWM)
39	PL4 (OC5B)	Digital pin 45 (PWM)
40	PL5 (OC5C)	Digital pin 44 (PWM)
41	PL6	Digital pin 43
42	PL7	Digital pin 42
43	**PD0 (SCL/INT0)**	**Digital pin 21 (SCL)**
44	**PD1 (SDA/INT1)**	**Digital pin 20 (SDA)**
45	PD2 (RXDI/INT2)	Digital pin 19 (RX1)
46	PD3 (TXD1/INT3)	Digital pin 18 (TX1)
47	PD4 (ICP1)	
48	PD5 (XCK1)	
49	PD6 (T1)	
50	PD7 (T0)	Digital pin 38
51	PG0 (WR)	Digital pin 41
52	PG1 (RD)	Digital pin 40

53	PC0 (A8)	Digital pin 37
54	PC1 (A9)	Digital pin 36
55	PC2 (A10)	Digital pin 35
56	PC3 (A11)	Digital pin 34
57	PC4 (A12)	Digital pin 33
58	PC5 (A13)	Digital pin 32
59	PC6 (A14)	Digital pin 31
60	PC7 (A15)	Digital pin 30
61	VCC	VCC
62	GND	GND
63	PJ0 (RXD3/PCINT9)	Digital pin 15 (RX3)
64	PJ1 (TXD3/PCINT10)	Digital pin 14 (TX3)
65	PJ2 (XCK3/PCINT11)	
66	PJ3 (PCINT12)	
67	PJ4 (PCINT13)	
68	PJ5 (PCINT14)	
69	PJ6 (PCINT 15)	
70	PG2 (ALE)	Digital pin 39
71	PA7 (AD7)	Digital pin 29
72	PA6 (AD6)	Digital pin 28
73	PA5 (AD5)	Digital pin 27
74	PA4 (AD4)	Digital pin 26
75	PA3 (AD3)	Digital pin 25
76	PA2 (AD2)	Digital pin 24
77	PA1 (AD1)	Digital pin 23
78	PA0 (AD0)	Digital pin 22
79	PJ7	
80	VCC	VCC
81	GND	GND
82	PK7 (ADC15/PCINT23)	Analog pin 15
83	PK6 (ADC14/PCINT22)	Analog pin 14
84	PK5 (ADC13/PCINT21)	Analog pin 13
85	PK4 (ADC12/PCINT20)	Analog pin 12
86	PK3 (ADC11/PCINT19)	Analog pin 11
87	PK2 (ADC10/PCINT18)	Analog pin 10
88	PK1 (ADC9/PCINT17)	Analog pin 9
89	PK0 (ADC8/PCINT16)	Analog pin 8
90	PF7 (ADC7)	Analog pin 7
91	PF6 (ADC6)	Analog pin 6
92	PF5 (ADC5/TMS)	Analog pin 5
93	PF4 (ADC4/TMK)	Analog pin 4
94	PF3 (ADC3)	Analog pin 3
95	PF2 (ADC2)	Analog pin 2
96	PF1 (ADC1)	Analog pin 1
97	PF0 (ADC0)	Analog pin 0
98	AREF	Analog Reference
99	GND	GND
100	AVCC	VCC

9.4.MPU6050

El MPU6050 se trata de un tipo de módulo de adquisición de datos conocidos como sistemas de medida inercial o IMUs (Inercial Measurement Units).

Estos módulos se basan en sistemas de referencia y coordenadas para proporcionar datos sobre la velocidad, orientación y fuerzas gravitacionales de un cuerpo en el espacio, combinando el uso de acelerómetros y giróscopos siendo estos, componentes principales en los sistemas de navegación inerciales de aviones, naves, barcos o misiles.

El MPU6050 está formado por un acelerómetro de tres ejes y un giróscopo también de tres ejes motivo por el cual es un dispositivo con seis grados de libertad. Aunque lo que miden los sensores internos de la IMU son aceleraciones lineales y angulares, el procesador es capaz de realizar los cálculos necesarios para ofrecer datos más útiles como pueden ser por ejemplo ángulos de inclinación con respecto de una posición inicial. Este módulo se comunica mediante protocolo i2c con la placa Arduino y necesita de un filtro complementario para generar medidas correctas.

9.4.1. Características Técnicas

- Salida digital de 6 ejes.
- Giroscopio con sensibilidad de ±250, ±500, ±1000, y ±2000dps
- Acelerómetro con sensibilidad de ±2g, ±4g, ±8g y ±16g
- Algoritmos embebidos para calibración
- Sensor de temperatura digital
- Entrada digital de video FSYNC
- Interrupciones programables
- Voltaje de alimentación: 2.37 a 3.46V
- Voltaje lógico: 1.8V±5% o VDD
- 10000g tolerancia de aceleración máxima

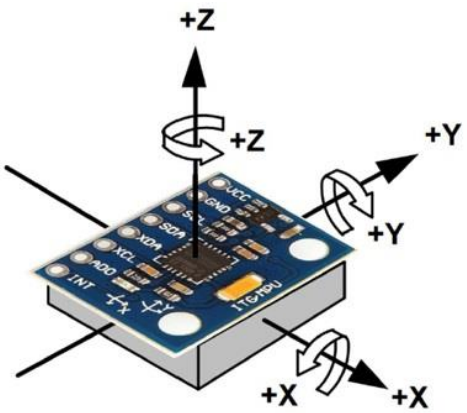


Ilustración 15: Ejes de coordenadas MPU6050

9.4.2. Conexiones

En este proyecto se han realizado la conexión entre el microcontrolador Arduino Mega y la IMU mediante los puertos SDA (pin 20) y SCL (pin 21) de la placa con los respectivos pines del módulo de medición inercial. Además de ello, el Arduino también proporciona la alimentación necesaria al MPU6050 mediante los pines de 5V y GND.

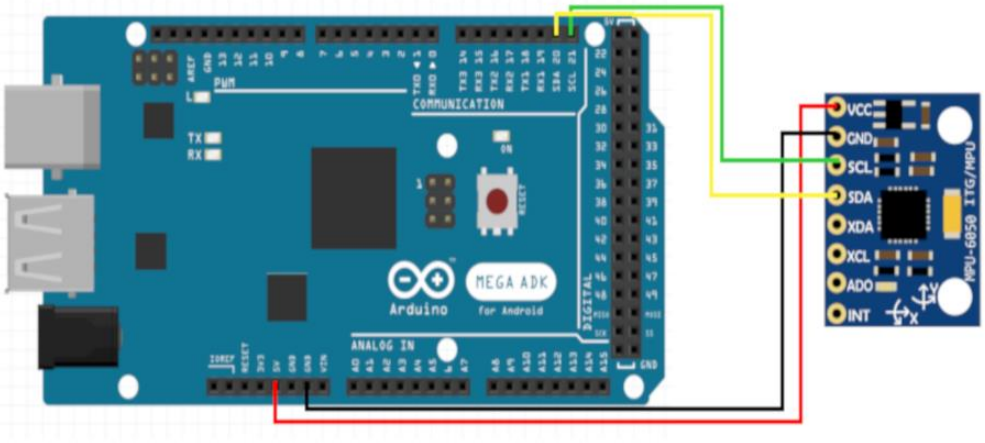


Ilustración 16: Conexiones Arduino-MPU6050

9.4.3. Comunicación

La comunicación de este módulo es por i2c lo que lo hace capaz de conectarse con una amplia gama de microcontroladores, entre ellos Arduino Mega como es el caso particular de este proyecto, utilizando únicamente dos líneas de señal (SDA para datos y SCL para la señal de reloj) y dos de alimentación la cual puede ser conectada a 5V puesto que el módulo cuanta con su propio regulador de corriente para obtener los 3.3V que necesita para funcionar correctamente.

9.4.4. Funcionamiento

Mediante la medición de las aceleraciones y las velocidades angulares detectadas por la IMU, este módulo es capaz de detectar cambios y movimientos en la posición de un objeto en movimiento. Para su utilización es necesario su calibración inicial de offset, así como la programación de los intervalos de medida.

9.4.5. Programación Simplificada

Al igual que para todos los demás módulos, la programación de la IMU se ha organizado de la siguiente manera:

- En primer lugar, se han declarado las variables globales necesarias.
- Se realiza la inicialización de la IMU y la configuración de sus Offset mediante la función **configuracionIMU()** la cual es llamada por la función principal **setup()**. La configuración de la IMU se lleva a cabo realizando 5000 mediciones iniciales con el objetivo de crear unos valores medios permitiendo detectar la posición de la IMU y poder utilizar tales valores como Offset inicial permitiendo colocar el módulo en cualquier posición para su uso.

En el bucle principal el gestor de tareas se encarga de llamar a la función **obtenerDatosIMU()** la cual es la encargada de obtiene los valores de la IMU los cuales son utilizados por la función **updateFilter()** que es la encargada de aplicar un filtro complementario que combina los ángulos obtenidos por el giroscopio y el acelerómetro para no ir acumulando errores (DRIFT) en las mediciones que acaben por corromper las medidas. Los valores calculados son almacenados en variables que luego serán empleadas en las funciones de guardado en la SD así como el visionado en pantalla.

9.4.6. Diagrama de Bloques

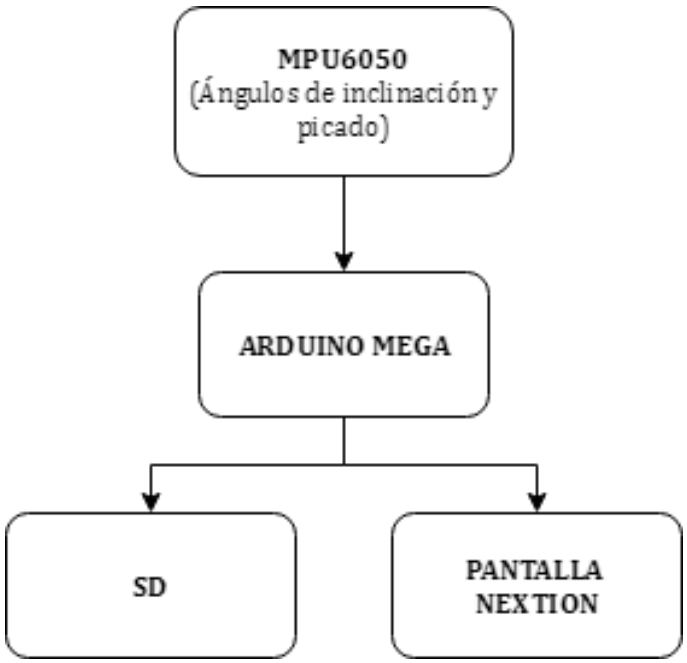


Ilustración 17: Diagrama de bloques MPU6050

9.5.GPS GY-NEO6MV2

Una de las funciones principales de la unidad telemétrica es la medición de la velocidad actual de la motocicleta, así como la localización y trayectoria de esta. Para estas funciones, se ha decidido emplear un módulo GPS el cual recibe la información directamente de los satélites siguiendo el protocolo NMEA (National Marine Electronics Asociation).

Ilustración 18: Módulo GPS GY-NEO6MV2

El módulo GPS empleado en el proyecto es comúnmente utilizado en la creación de drones, aparatos de radiocontrol e incluso sistemas de seguridad por geolocalización debido a su precisión y reducido tamaño y coste.

Este módulo está basado en un receptor de la serie Ublox NEO 6M, una EEPROM con una configuración de fábrica, así como una pila de botón para mantener los datos de la configuración. También cuenta con un LED indicador de la conexión con los satélites, una antena cerámica y cuatro pines de conexión VCC,GND,RX y TX.

9.5.1. Características Técnicas

- Modelo: GY-GPSMV2.
- Receptor: Ublox NEO 6M.
- Voltaje de alimentación: 3.3 - 5V.
- Consumo de corriente: 45 mA.
- Velocidad de comunicación 9600 bps.
- Comunicación UART asíncrona.

9.5.2. Conexiones

En este proyecto se han realizado la conexión entre el microcontrolador Arduino Mega y la unidad GPS mediante los pines TX y RX de la placa con los respectivos puertos 10 y 11 del Arduino. Además de ello, el Arduino también proporciona la alimentación necesaria al GPS GY-NEO6MV2 mediante los pines de 5V y GND.

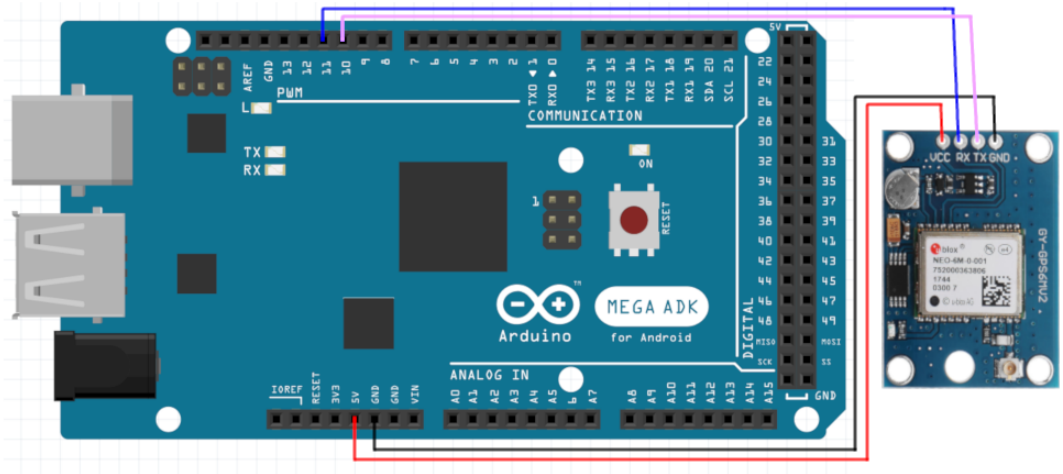


Ilustración 19: Conexiones Arduino-GPS

9.5.3. Comunicación

La comunicación del GPS se realiza siguiendo el protocolo UART donde se realiza la conexión TX - RX y RX -TX entre controlador y modulo de adquisicion. En el caso particular de este proyecto unicamente se necesitan dos lineas de datos y las de alimentacion conectadas a la salida que Arduino Mega proporciona de 5V y GND.

9.5.4. Programación Simplificada Obtención de Datos

Al igual que para todos los demás módulos, la programación del GPS se ha organizado de la siguiente manera:

- En primer lugar, se han declarado las variables globales necesarias.
- Se realiza la inicialización del módulo GPS mediante la función **configuracionGPS()** la cual es llamada por la función principal **setup()**. En la configuración del GPS se configuran los pines 10 y 11, previamente definidos como rx y tx, como entrada y salida de datos además de inicializar el puerto serie.
- En el bucle principal el gestor de tareas se encarga de llamar a la función **obtenerDatosGPS()** la cual es la encargada de obtener los valores de velocidad y posición actual de la motocicleta.
- Por último, la función convierte los datos recibidos directamente del GPS a un formato adecuado para su almacenamiento.

  hora: "123519" → "12:35:19"
  coordenadas: "4807.038" y "N" → "48º 07.038' N" → "48.1173 N"
  velocidad Nudos a Kmh: "022.4" → "41.4848"

9.5.5. Programación Simplificada Detección de Paso por Meta

Mediante la selección del circuito en el menú existente en la pantalla del piloto, las coordenadas de la línea de meta son cargadas al código del programa y mediante una serie de funciones que se explicará a continuación, se compara la posición actual con la de meta para conocer el tiempo en el que la motocicleta ha realizado una vuelta al circuito utilizando el contador interno de milisegundos del microcontrolador.

Con cada recepción de datos de la unidad GPS se genera una alerta o flag que lanza la función **comprobarMeta()** la cual tras comparar la posición actual con las coordenadas de la meta seleccionada separa las unidades de milisegundos obtenidas en un formato mm:ss:cccc (minutos, segundos, centésimas). Esta función también se encarga de generar un archivo csv con todos los tiempos de vuelta y guardarla en la SD.

9.5.6. Diagrama de Bloques

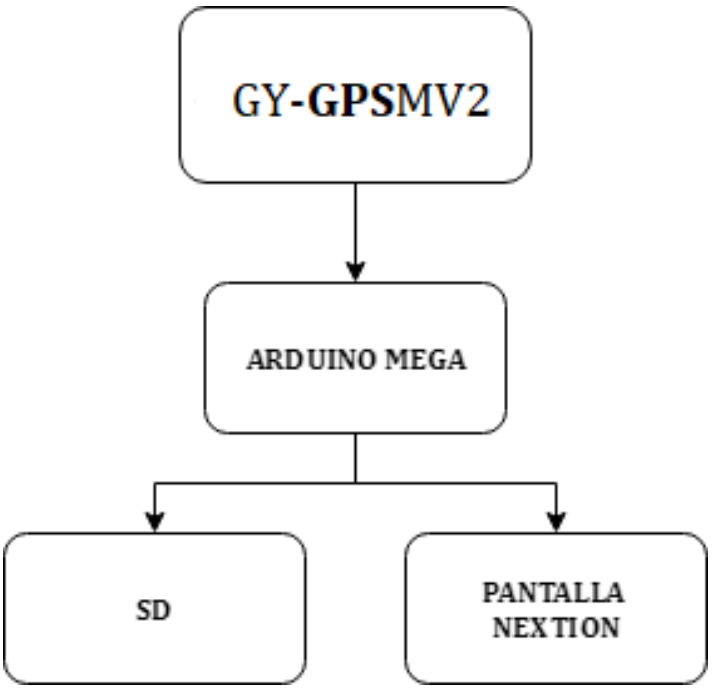


Ilustración 20: Diagrama de bloques GPS

9.6.Módulo de almacenamiento SD

Un lector de memoria SD permite añadir una tarjeta de memoria a los proyectos de Arduino para poder almacenar los datos y valores, en el proyecto que se esta desarrollando, se empleará para almacenar los datos telemétricos de una manera no volátil para su posterior estudio o almacenamiento.

EN los lectores de memoria comúnmente empleados en Arduino, la conexión se puede realizar mediante el bus SPI, aunque también es posible hacerlo mediante protocolos I2C y UART.

La programación de estos lectores resulta bastante pesada para el procesamiento de Arduino es por ello que se trata de una de las razones de más peso por las que se ha optado por emplear un Arduino Mega 2560 para este proyecto.

Entre sus posibles funciones, este módulo puede emplearse para adquirir información almacenada en él o para almacenarla (Datalogger), siendo esta ultima la función que desempeña en la unidad telemétrica desarrollada.

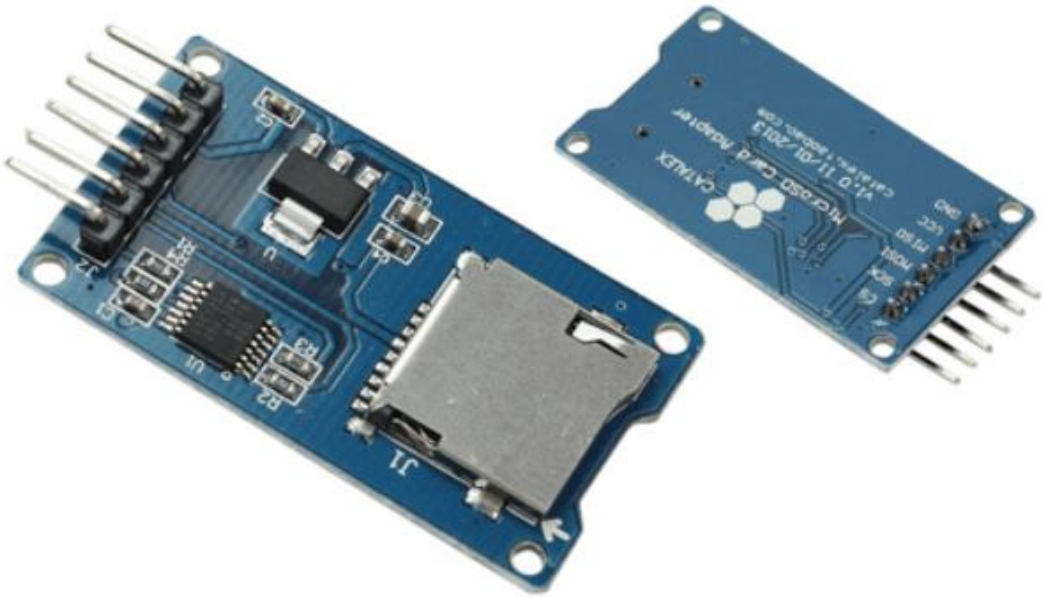


Ilustración 21: Módulo de almacenamiento SD

9.6.1. Características Técnicas

- Almacenamiento de hasta 32GB.
- Voltaje de alimentación: 3.3 - 5V (Regulador interno).
- Interfaz de comunicación SPI estándar.
- 6 pines de conexión (GND, VCC, MISO, MOSI, SCK, CS)

9.6.2. Conexiones

En este proyecto se han realizado la conexión entre el microcontrolador Arduino Mega y el lector SD mediante el bus de conexión SPI (MISO, MOSI, SCK Y CS).

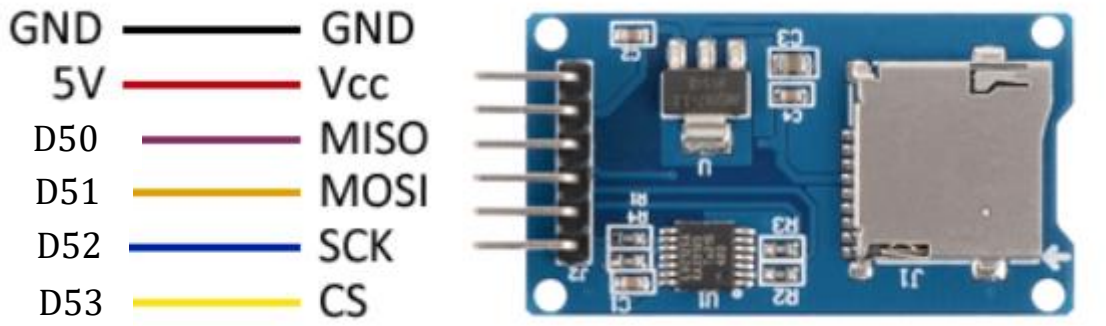


Ilustración 22: Conexiones Arduino-SD

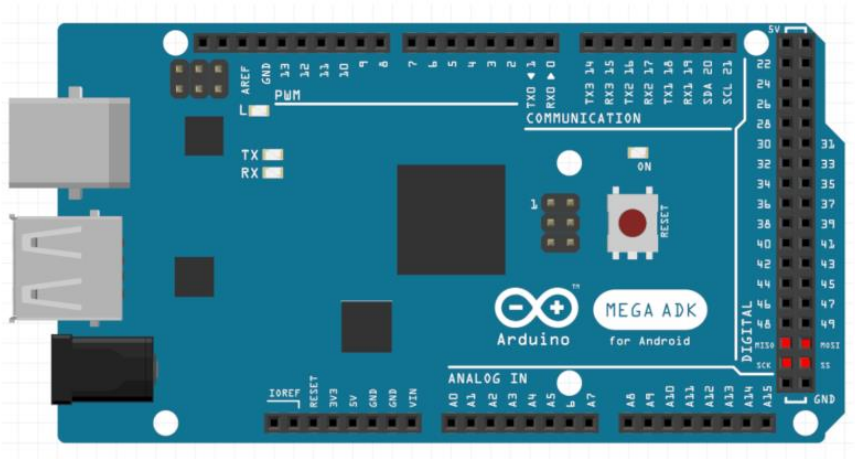


Ilustración 23: Puertos SPI Arduino Mega

9.6.3. Programación Simplificada

Al igual que para todos los demás módulos, la programación del lector SD se ha organizado de la siguiente manera:

- En primer lugar, se han declarado las variables globales necesarias.
- Se realiza la inicialización del módulo GPS mediante la función **configuracionSD()** la cual es llamada por la función principal **setup()**.
- Con el objetivo de optimizar el proceso principal de la unidad, el gestor de tareas lanza la función **guardarDatosSD()** en intervalos de tiempo establecidos, permitiendo que el bucle principal se continúe ejecutando. Esta función la cual se encarga de abrir el archivo csv, almacenar los valores obtenidos y guardados en las variables del programa y de nuevo cerrar el archivo csv para salvar los datos.

9.6.4. Diagrama de Bloques

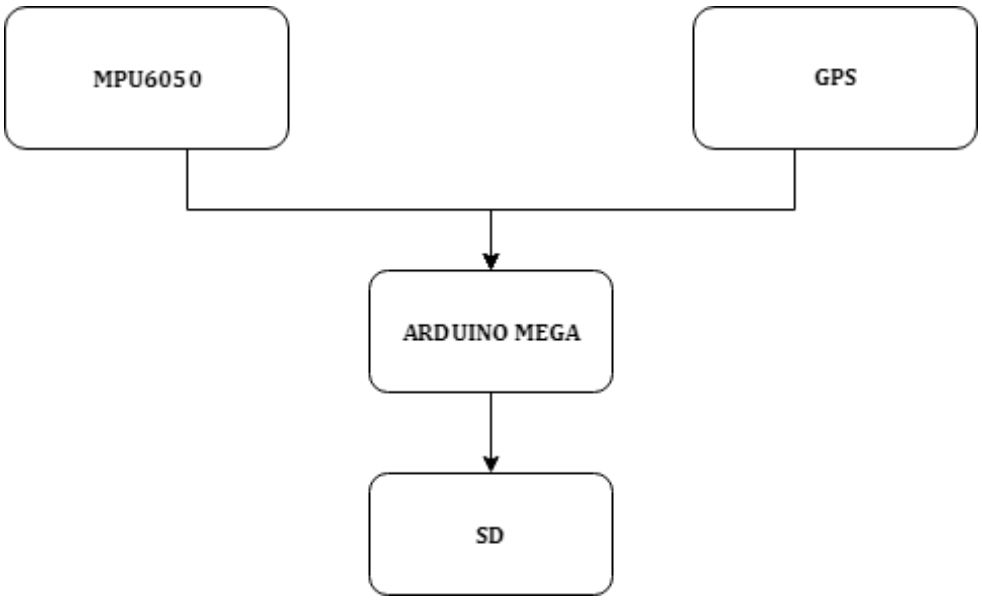


Ilustración 24: Diagrama de bloques SD

9.7.Pantalla Nextion

La pantalla empleada en la realización de este proyecto se trata de una pantalla Nextion nx3224t028 de 2.8 pulgadas en la cual el piloto podrá leer los datos de su motocicleta en tiempo real mediante una interfaz creada específicamente para este proyecto.

El principal motivo por el que se ha elegido la pantalla Nextion nx3224t028 ha sido por su comunicación mediante I2C con el microcontrolador Arduino además de por su excelente relación calidad precio y su editor propio de interfaces de usuario. También es importante destacar que cuenta con un procesador propio además de una memoria flash dedicada al control de la pantalla.

Ilustración 25: Pantalla Nextion nx32

9.7.1. Características Técnicas

- Pantalla de 2.8 pulgadas TFT 320x240
- Comunicación I2C
- Voltaje de alimentación 5V
- Consumo de corriente: 65 mA (Trabajo) 15 mA (Reposo)
- Ratio de baudios típico: 9600
- Memoria Flash: 4 MB
- Memoria RAM: 3584 BYTE

9.7.2. Conexiones

Para la conexión de este módulo se ha creado un puerto Serie virtual en el Arduino utilizando sus pines 3(Rx) y 4(Tx) para realizar la conexión Rx-Tx y Tx-Rx entre la pantalla y el microcontrolador de la siguiente manera.

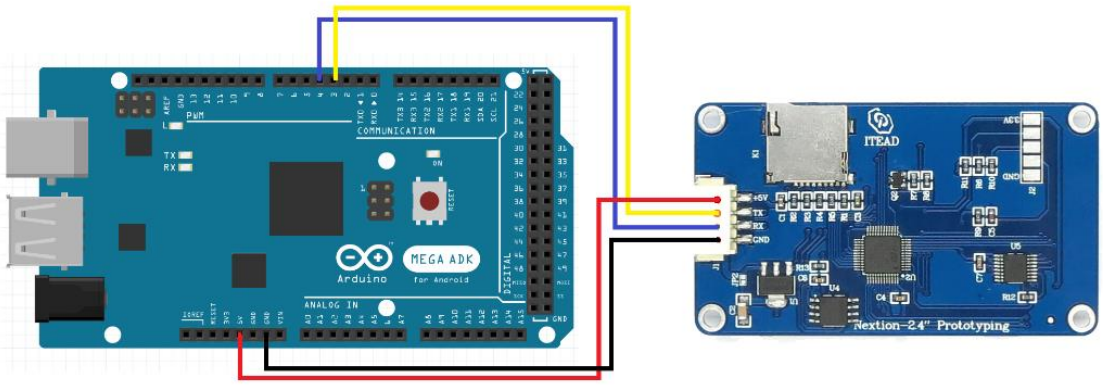


Ilustración 26: Conexiones Arduino-Pantalla Nextion

9.7.3. Programación Simplificada

Al igual que para todos los demás módulos, la programación de la pantalla Nextion se ha organizado de la siguiente manera:

- En primer lugar, se han declarado las variables globales necesarias.
- Se realiza la inicialización la pantalla mediante la función **nexInit()** la cual es llamada por la función principal **setup()**.
- Como última función del bucle principal, el gestor de tareas lanza la función **DatosPantalla()** la cual se encarga de enviar los valores actuales de velocidad, tiempos de vuelta e inclinaciones (pitch y roll) a la pantalla. En esta ocasión la función del gestor de tareas es únicamente lanzar esta función si los valores actuales han variado para así evitar repetir el envío de valores ya mostrados en pantalla y así optimizar el tiempo del bucle.

9.7.4. Diagrama de Bloques

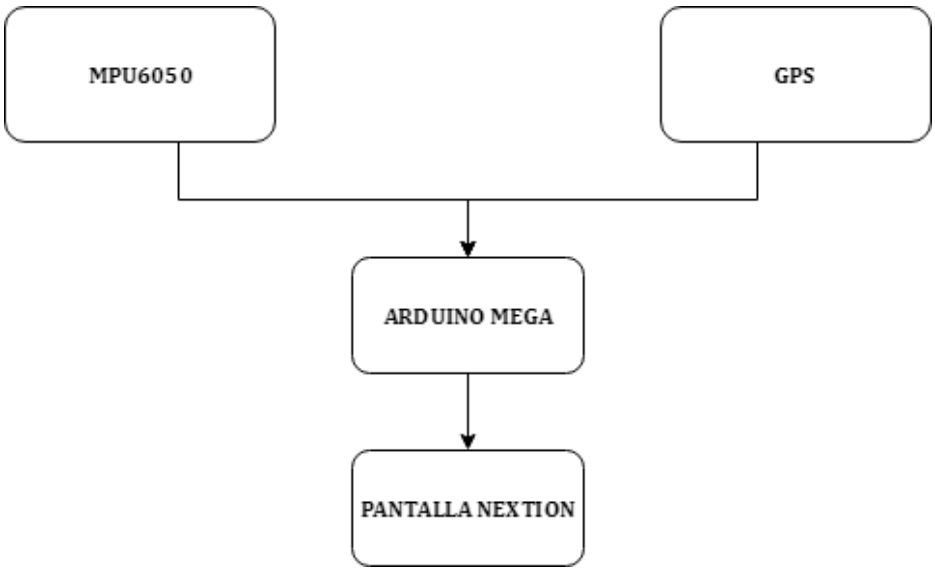


Ilustración 27: Diagrama de bloques Pantalla Nextion

9.8.Gestor de Tareas

Tal y como se ha ido detallando en la programación simplificada de cada bloque, se ha creado un gestor de tareas encargado de organizar la ejecución de las mismas con el objetivo de optimizar el funcionamiento del módulo de adquisición de datos, así como el correcto visionado de la pantalla.

Para el correcto funcionamiento del proyecto hay que priorizar en la función que recibe los datos GPS, la cual se ha conseguido una frecuencia de medición de 1 Hz, por lo que ha sido un dato importante a la hora de detectar el paso por meta.

Se ha estudiado el caso más desfavorable que podría darse, siendo este la recepción de una trama de datos GPS medio segundo antes del paso por meta y la siguiente, medio segundo después. Para este supuesto se ha estimado una velocidad máxima de 250 kmh de paso por meta, por lo que entre mediciones la moto recorrería una distancia inferior a 70m.

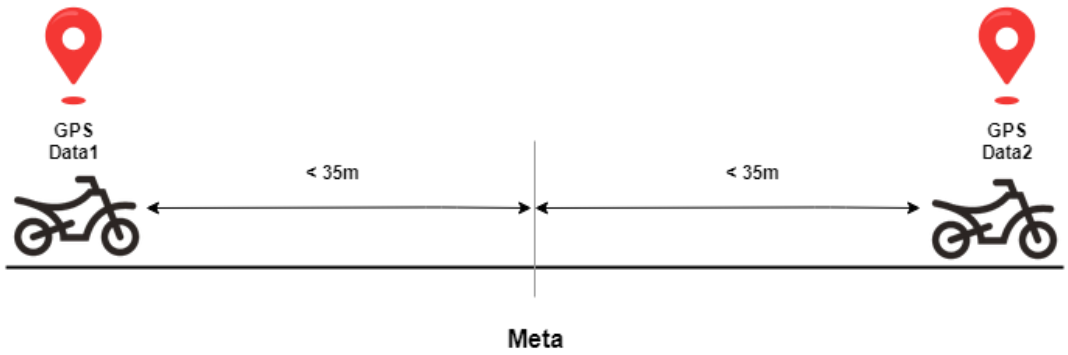


Ilustración 28: Caso GPS más desfavorable

lugares	grados decimales.	grados	distancia
0	1.0	1°0'0"	111.319 km
1	0.1	0°6'0"	11.132 km
2	0.01	0°0'36"	1.113 km
3	0.001	0°0'3.6"	111.3 m
4	0.0001	0°0'0.36"	11.13 m
5	0.00001	0°0'0.036"	1.11 m
6	0.000001	0°0'0.0036"	11.1 cm
7	0.0000001	0°0'0.00036"	1.11 cm

Ilustración 29: Tabla de equivalencia de coordenadas

Por este motivo, la distancia mínima elegida en la comprobación es de 0.0003. De esta forma, queda cubierto cualquier punto en un radio de 33.40m al punto de recepción de la trama GPS donde se comprobará si se ha cruzado la meta, desechando cualquier medición de tiempo inferior a un mínimo establecido en el código (15 segundos) para no realizar varios pasos por meta en una misma vuelta.

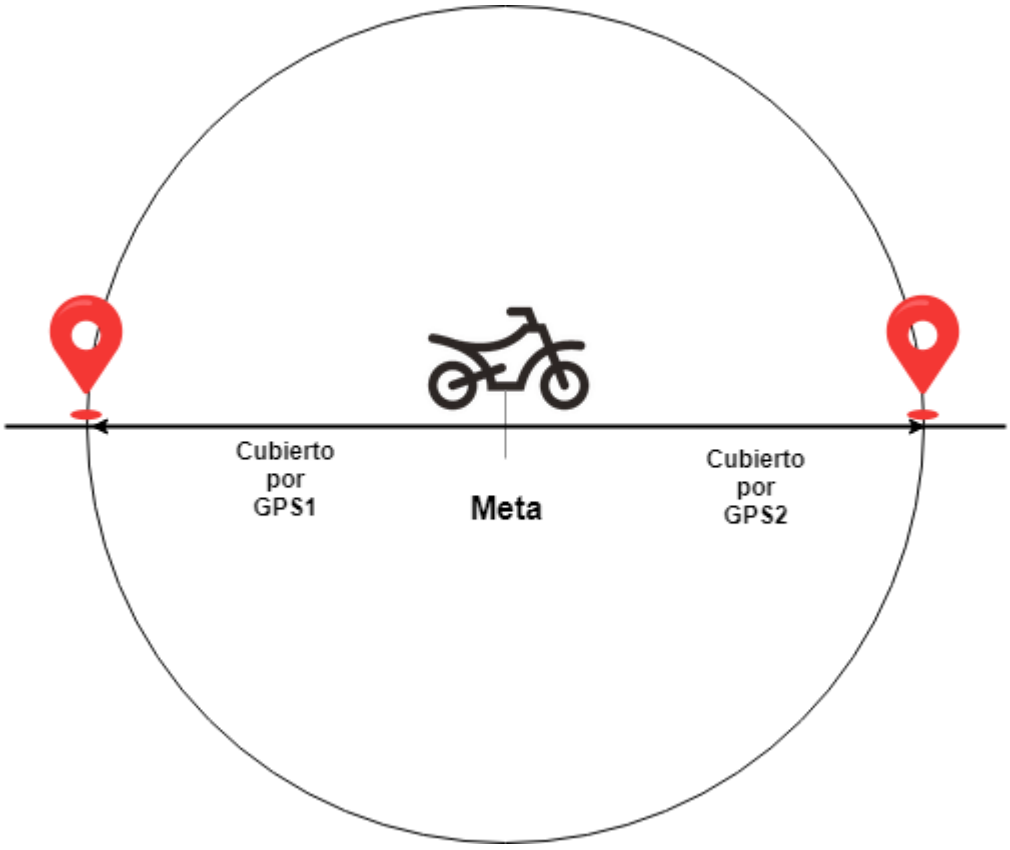


Ilustración 30: Radio de Cobertura GPS

Dada la gran velocidad de ejecución de las demás tareas, estas se ejecutan una vez se ha recibido la trama GPS y el gestor de tareas se encarga de optimizar el proceso realizando un guardado de datos por segundo, así como una actualización de los valores de pantalla que hayan sido modificados, permitiendo así un ahorro de tiempo evitando el envío repetido de variables.

10. Software Empleado

10.1. Plataforma De Desarrollo Arduino

Los microcontroladores Arduino cuentan con su propia plataforma de programación la cual puedes descargar gratuitamente desde su web, así como también existe la posibilidad de utilizar un editor en línea.

En este proyecto se ha utilizado el software propio de Arduino, así como también varias de las librerías que este proporciona. Su funcionamiento es muy sencillo.

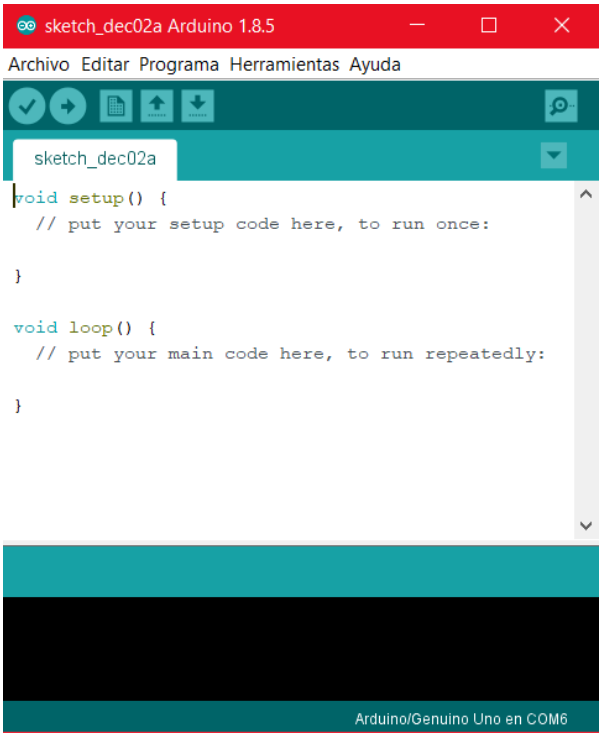


Ilustración 31: Plataforma de desarrollo Arduino

Tal y como puede verse en la imagen, el sketch cuenta con dos módulos, el set up (configuración inicial) y el loop (ciclo de instrucciones que se realizaran) además de eso, lo más destacable son los botones del cick y la flecha horizontal que compilan y envían a la placa el programa respectivamente. También es muy utilizado el botón de la lupa que muestra la salida del puerto serie de nuestra placa para poder ver en tiempo real los resultados que va volcando el controlados con la ejecución del programa.

Esta plataforma permite comunicarse con todos los dispositivos de la familia Arduino mediante la configuración de placa y puerto desde el menú Herramientas.

10.2. Plataforma De Desarrollo Nextion

Las pantallas Nextion cuentan con su propia plataforma de diseño de interfaces, esta plataforma se puede encontrar de forma gratuita en la misma web del fabricante y recibe el nombre de Nextion Editor.

Aunque se trata de una herramienta de trabajo sencilla se pueden conseguir diseños muy completos con los diferentes elementos que permite añadir que podemos encontrar en la sección ToolBox del programa. También es posible añadir fondos e imágenes para completas el diseño. En la sección ToolBox podemos encontrar elementos tales como cuadros de texto y numéricos, pulsadores, barras de progreso, sliders etc.

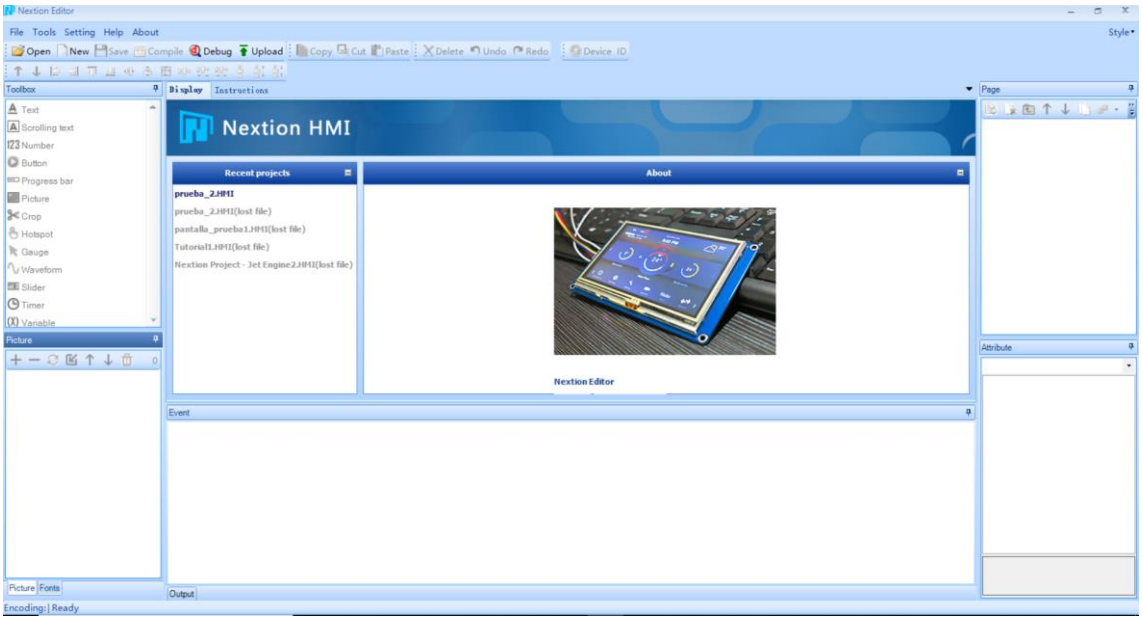


Ilustración 32: Plataforma Nextion Editor

La utilización de este software de diseño es limitada pero sencilla puesto que basta con arrastrar y modificar los diferentes elementos disponibles en el programa para crear la interfaz de usuario, después solo quedará enlazar los valores de dichos elementos con las variables que proporciona el programa para poder observar las actualizaciones de dichas variables en la pantalla.

Este sería un diseño preliminar de la pantalla de la unidad telemétrica realizado con el editor de Nextion, después bastará con vincular las variables que maneja el programa con los distintos campos de la pantalla y configurar un refresco apropiado de la misma.

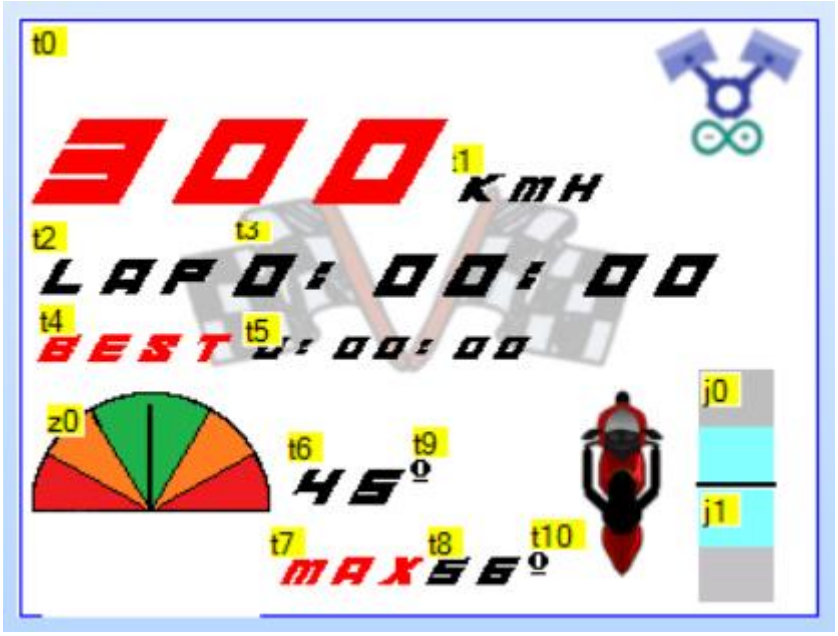


Ilustración 33: Pantalla Principal del proyecto

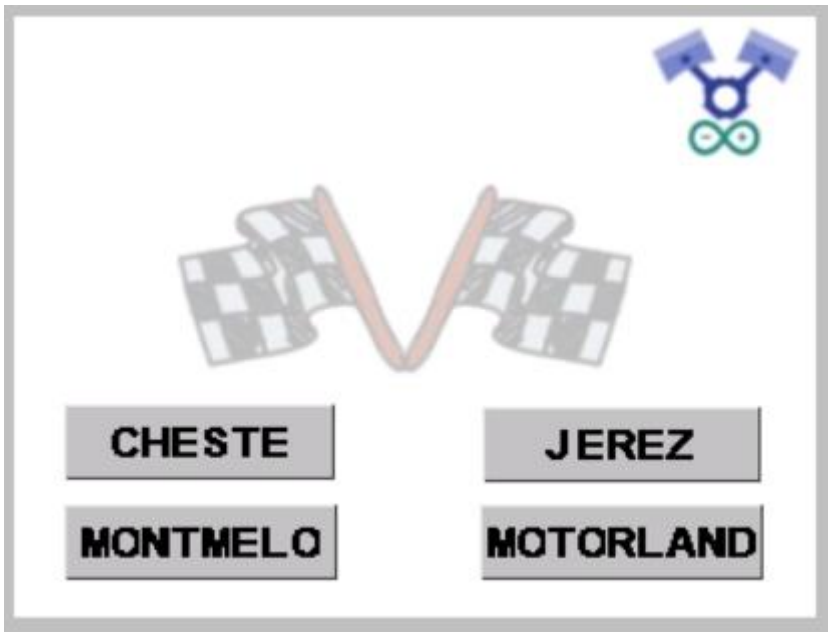
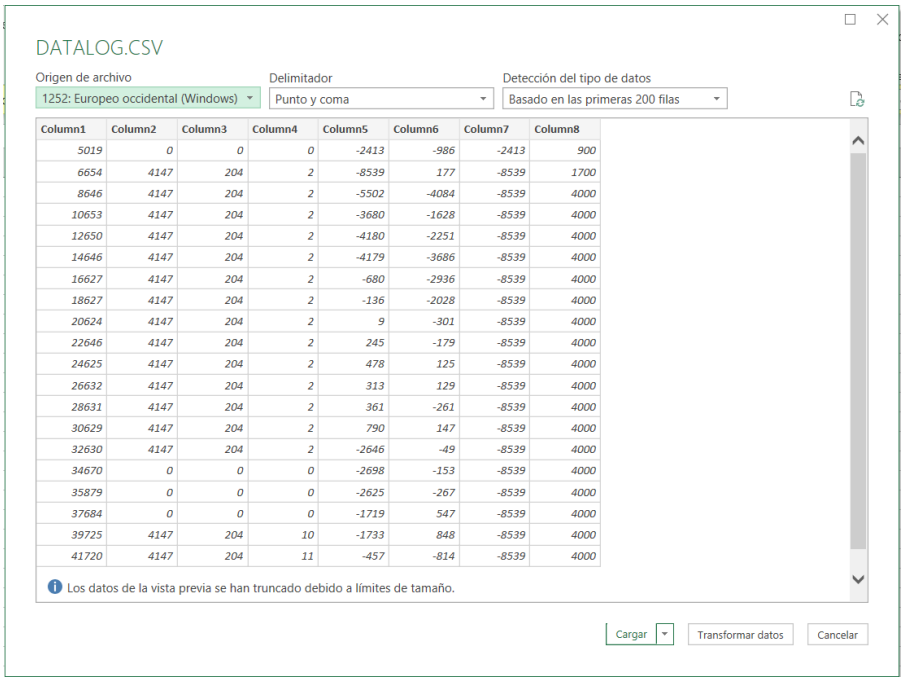


Ilustración 34: Menú de selección de circuitos

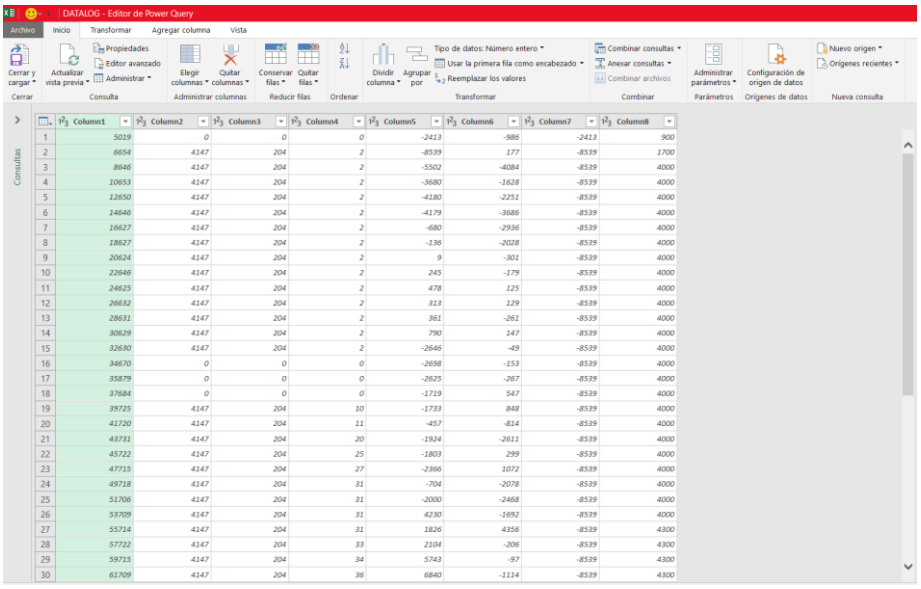
10.3. Excel

Este potente programa de cálculo desarrollado por Microsoft es empleado en este proyecto para el visionado de los datos almacenados. Los archivos CSV generados por el código son fácilmente exportables a esta plataforma para poder emplear su contenido como DATA para la creación de gráficos y tablas útiles para el estudio.



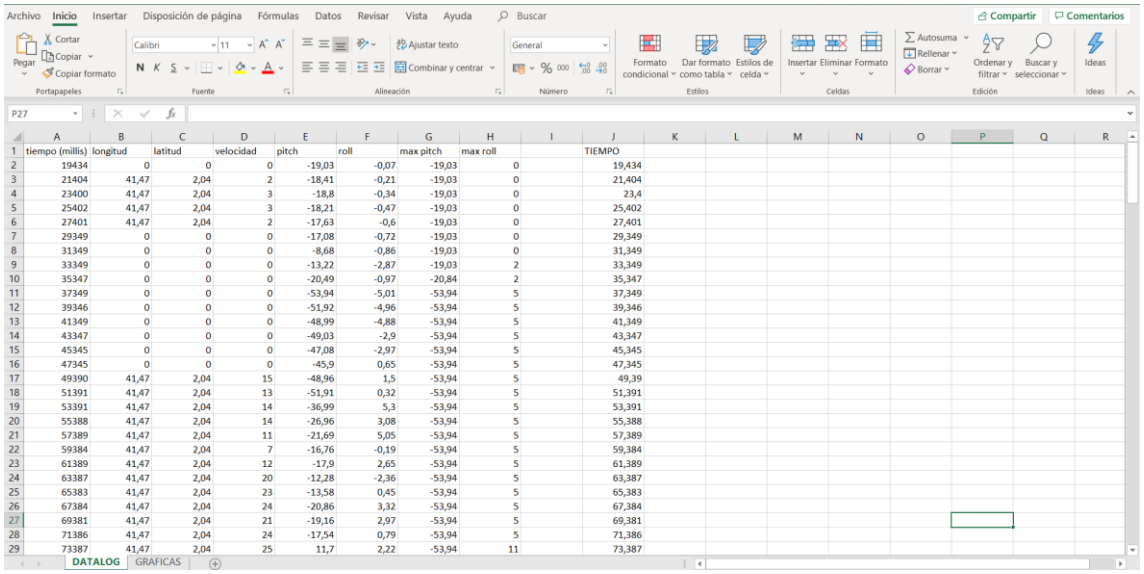
Column1	Column2	Column3	Column4	Column5	Column6	Column7	Column8
5019	0	0	0	-2413	-986	-2413	900
6654	4147	204	2	-8539	177	-8539	1700
8646	4147	204	2	-5502	-4084	-8539	4000
10653	4147	204	2	-3680	-1628	-8539	4000
12650	4147	204	2	-4180	-2251	-8539	4000
14646	4147	204	2	-4179	-3686	-8539	4000
16627	4147	204	2	-680	-2936	-8539	4000
18627	4147	204	2	-136	-2028	-8539	4000
20624	4147	204	2	9	-301	-8539	4000
22646	4147	204	2	245	-179	-8539	4000
24625	4147	204	2	478	125	-8539	4000
26632	4147	204	2	313	129	-8539	4000
28631	4147	204	2	361	-261	-8539	4000
30629	4147	204	2	790	147	-8539	4000
32630	4147	204	2	-2646	-49	-8539	4000
34670	0	0	0	-2698	-153	-8539	4000
35879	0	0	0	-2625	-267	-8539	4000
37684	0	0	0	-1719	547	-8539	4000
39725	4147	204	10	-1733	848	-8539	4000
41720	4147	204	11	-457	-814	-8539	4000

Ilustración 35: Menú de exportacion CSV-Excel



	Column1	Column2	Column3	Column4	Column5	Column6	Column7	Column8
1	5019	0	0	0	-2413	-986	-2413	900
2	6654	4147	204	2	-8539	177	-8539	1700
3	8646	4147	204	2	-5502	-4084	-8539	4000
4	10653	4147	204	2	-3680	-1628	-8539	4000
5	12650	4147	204	2	-4180	-2251	-8539	4000
6	14646	4147	204	2	-4179	-3686	-8539	4000
7	16627	4147	204	2	-680	-2936	-8539	4000
8	18627	4147	204	2	-136	-2028	-8539	4000
9	20624	4147	204	2	9	-301	-8539	4000
10	22646	4147	204	2	245	-179	-8539	4000
11	24625	4147	204	2	478	125	-8539	4000
12	26632	4147	204	2	313	129	-8539	4000
13	28631	4147	204	2	361	-261	-8539	4000
14	30629	4147	204	2	790	147	-8539	4000
15	32630	4147	204	2	-2646	-49	-8539	4000
16	34670	0	0	0	-2698	-153	-8539	4000
17	35879	0	0	0	-2625	-267	-8539	4000
18	37684	0	0	0	-1719	547	-8539	4000
19	39725	4147	204	10	-1733	848	-8539	4000
20	41720	4147	204	11	-457	-814	-8539	4000
21	43731	4147	204	20	-1924	-2611	-8539	4000
22	45722	4147	204	25	-1803	299	-8539	4000
23	47715	4147	204	27	-2366	1072	-8539	4000
24	49718	4147	204	31	-704	-2078	-8539	4000
25	51706	4147	204	31	-2000	-2468	-8539	4000
26	53709	4147	204	31	4230	-1692	-8539	4000
27	55714	4147	204	31	1826	4356	-8539	4300
28	57722	4147	204	33	2104	-206	-8539	4300
29	59715	4147	204	34	5743	-97	-8539	4300
30	61709	4147	204	36	6840	-1114	-8539	4300

Ilustración 36: Tablas Excel generadas



	A	B	C	D	E	F	G	H	I	J
1	tiempo (millis)	longitud	latitud	velocidad	pitch	roll	max pitch	max roll		TIEMPO
2	19434	0	0	0	-19,03	-0,07	-19,03	0		19,434
3	21404	41,47	2,04	2	-18,41	-0,21	-19,03	0		21,404
4	23400	41,47	2,04	3	-18,8	-0,34	-19,03	0		23,4
5	25402	41,47	2,04	3	-18,21	-0,47	-19,03	0		25,402
6	27401	41,47	2,04	2	-17,63	-0,6	-19,03	0		27,401
7	29349	0	0	0	-17,08	-0,72	-19,03	0		29,349
8	31349	0	0	0	-8,68	-0,86	-19,03	0		31,349
9	33349	0	0	0	-13,22	-2,87	-19,03	2		33,349
10	35347	0	0	0	-20,49	-0,97	-20,84	2		35,347
11	37349	0	0	0	-53,94	-5,01	-53,94	5		37,349
12	39346	0	0	0	-51,92	-4,96	-53,94	5		39,346
13	41349	0	0	0	-48,99	-4,88	-53,94	5		41,349
14	43347	0	0	0	-49,03	-2,9	-53,94	5		43,347
15	45345	0	0	0	-47,08	-2,97	-53,94	5		45,345
16	47345	0	0	0	-45,9	0,65	-53,94	5		47,345
17	49390	41,47	2,04	15	-48,96	1,5	-53,94	5		49,39
18	51391	41,47	2,04	13	-51,91	0,32	-53,94	5		51,391
19	53391	41,47	2,04	14	-36,99	5,3	-53,94	5		53,391
20	55388	41,47	2,04	14	-26,96	3,08	-53,94	5		55,388
21	57389	41,47	2,04	11	-21,69	5,05	-53,94	5		57,389
22	59384	41,47	2,04	7	-16,76	-0,19	-53,94	5		59,384
23	61389	41,47	2,04	12	-17,9	2,65	-53,94	5		61,389
24	63387	41,47	2,04	20	-12,28	-2,36	-53,94	5		63,387
25	65383	41,47	2,04	23	-13,58	0,45	-53,94	5		65,383
26	67384	41,47	2,04	24	-20,86	3,32	-53,94	5		67,384
27	69381	41,47	2,04	21	-19,16	2,97	-53,94	5		69,381
28	71386	41,47	2,04	24	-17,54	0,79	-53,94	5		71,386
29	73387	41,47	2,04	25	11,7	2,22	-53,94	11		73,387

Ilustración 37: Almacenamiento de datos registrados

Pudiendo generar fácilmente gráficas útiles para el estudio de la telemetría almacenada.

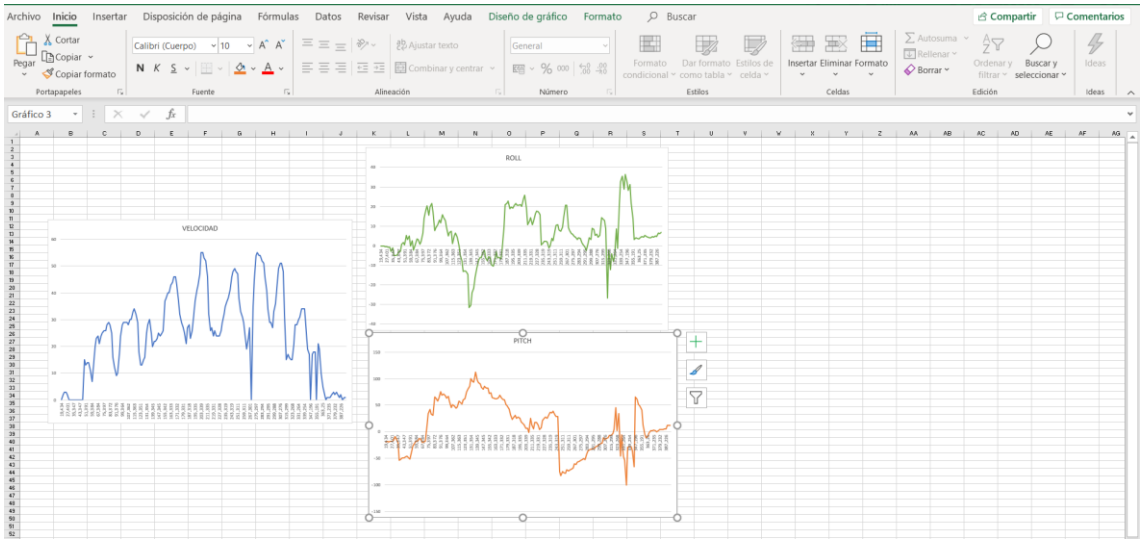


Ilustración 38: Graficos de datos almacenados por el proyecto

11. Prototipo Final

Ilustración 39: Prototipo Final I

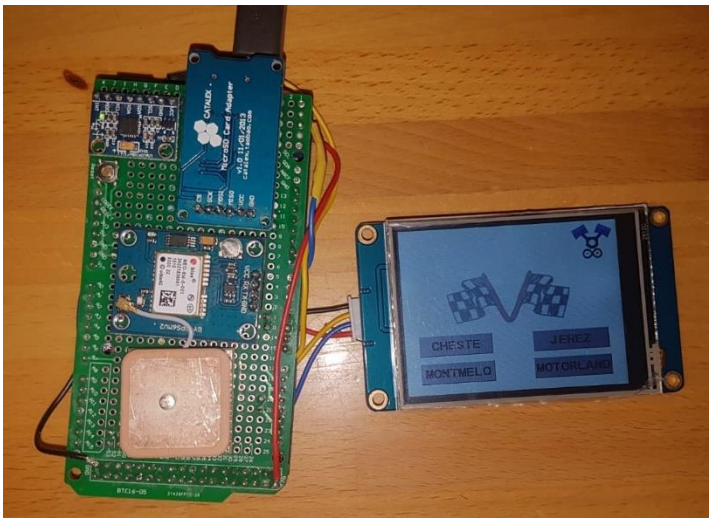


Ilustración 40: Prototipo Final II

11.1. Funcionamiento

Como se detalla en apartados posteriores, se ha logrado un funcionamiento del conjunto entero, realizando mediciones de ángulos de inclinación y picado, velocidad y posición GPS así como el guardado de estos datos en una tarjeta SD y el visionado de los mismos a través de la pantalla. Como se detallará en el apartado de conclusiones, el proyecto se ha dejado en estado de prototipo y no se ha llegado a implementar directamente en una motocicleta.

11.2. Colocación En La Motocicleta

En un principio el proyecto estaba pensado para realizarse junto con un encapsulado para poder fijarse al manillar de la motocicleta mediante un soporte tipo “GoPro” que permitiera quitarlo y ponerlo de manera rápida y sencilla. Este sería un ejemplo de cómo podría quedar integrado para su utilización.

Ilustración 41: Ejemplo de Colocación

ETSID

UNIVERSITAT POLITECNICA DE VALENCIA

TRABAJO FIN DE GRADO:

DESARROLLO DE UNA UNIDAD TELEMETRICA PARA MOTOCICLETA BASADA EN ARDUINO

DOCUMENTO:

- CODIGO

AUTOR: ERNESTO SAN VALENTIN CALVET

TUTOR: CARLOS DOMINGUEZ

TITULACION: GRADO EN INGENIERIA ELECTRONICA INDUSTRIAL Y AUTOMATICA, ETSID

CURSO ACADEMICO 2018-2019

```
////////////////////////////////////   LIBRERIAS   ////////////////////
/////////////////

#include <SoftwareSerial.h>
#include <TinyGPS.h>
#include <SD.h>
#include <VirtualWire.h>
#include <SPI.h>
#include "I2Cdev.h"
#include "MPU6050.h"
#include "Wire.h"
#include <Nextion.h>
#include <math.h>

#define rxPin 10
#define txPin 11

int num_loop = 0, minutos, segundos, milis, mejorminutos,
mejorsegundos, mejormilis, ultimociclo = 0;

int ciclo = 0; //guardado en SD cada
boolean flagIMU = false, flagSD = false, flagPantalla = false;

long vuelta_anterior = 0, tiempo_vuelta, mejorvuelta = 9999999;

double meta[2], posActual[2], posAnterior[2];
float lati, lon;
boolean nuevoGPS;

// Tiempos de ejecucion
long lastTimeIMU = 0,
     lastTimeSD  = 0,
     lastTimePantalla = 0;

long refrIMU = 100,
     refrSD  = 1000,
     refrPantalla = 250;


////////////////////////////////////  VARIABLES
SD   ////////////////////////////////

File dataFile, dataFile2;


////////////////////////////////////  VARIABLES
PANTALLA   ////////////////////////////////

int velocidad_en_pantalla = 1;
int ang_y_en_pantalla;
int ang_x_en_pantalla;
int mayory_en_pantalla;
int minutos_en_pantalla, segundos_en_pantalla, milis_en_pantalla,
mejorminutos_en_pantalla, mejorsegundos_en_pantalla,
mejormilis_en_pantalla;

//NexButton bcircuito = NexButton(0, 15, "bcircuito"); //boton
CIRCUITO
```

```
NexButton bcircuito = NexButton(0, 11, "bcircuito"); //boton CIRCUITO
NexButton bcheste = NexButton(1, 1, "bcheste"); //boton CHESTE
NexButton bmontmelo = NexButton(1, 2, "bmontmelo"); //boton MONTMELO
NexButton bjerez = NexButton(1, 3, "bjerez"); //boton JEREZ
NexButton bmotorland = NexButton(1, 4, "bmotorland"); //boton
MOTORLAND

NexTouch *nex_listen_list[] =       //lista de botones
{
  &bcircuito,
  &bcheste,
  &bmontmelo,
  &bjerez,
  &bmotorland,
  NULL
};

////////////////////////////////////  DECLARACION DE OBJETOS
//////////////////////////////

// (page id, component, name)
NexNumber n0 = NexNumber(0, 10, "n0"); //velocidad

NexNumber n3 = NexNumber(0, 19, "n3"); //Vuelta actual MINUTOS
NexNumber n4 = NexNumber(0, 20, "n4"); //Vuelta actual SEGUNDOS
NexNumber n5 = NexNumber(0, 21, "n5"); //Vuelta actual MILIS
NexNumber n6 = NexNumber(0, 22, "n6"); //Mejor Vuelta MINUTOS
NexNumber n7 = NexNumber(0, 23, "n7"); //Mejor Vuelta SEGUNDOS
NexNumber n8 = NexNumber(0, 24, "n8"); //Mejor Vuelta MILIS

NexNumber n1 = NexNumber(0, 13, "n1"); //Angulo actual
NexNumber n2 = NexNumber(0, 14, "n2"); //Angulo maximo

NexProgressBar j1 = NexProgressBar(0, 1, "j1"); //Picado total
NexGauge z0 = NexGauge(0, 2, "z0"); //Angulo de inclinacion
NexProgressBar j15 = NexProgressBar(0, 15, "j15"); //Calibracion OK

////////////////////////////////////   VARIABLES
IMU   ////////////////////////////////////

const int mpuAddress = 0x68;
MPU6050 mpu(mpuAddress);
MPU6050 accelgyro(0x68);

const int buffersize = 500;
const byte acel_deadzone = 8,////////////////porque const byte?
           giro_deadzone = 1;

int mean_ax, mean_ay, mean_az, mean_gx, mean_gy, mean_gz;
int ax, ay, az,                 // Variables acelerometro
    gx, gy, gz;                 // Variables giroscopio

float offset_ang_x, offset_ang_y;

long tiempo_prev;
float dt;
float ang_x, ang_y;
float ang_x_prev, ang_y_prev;

float mayorx = 0;
```

```
float mayory = 0;

int ZAxGForce;

float pitch, roll;

////////////////////////////////  VARIABLES
GPS   ////////////////////////////////

TinyGPS gps;
SoftwareSerial puertoGPS(rxPin, txPin);

String latitud, longitud, hora;
int velocidad = 100;

////////////////////////////////   CONFIGURACION
IMU   ////////////////////////////////

void meansensors() {

  int i = 0;
  long buff_ax = 0, buff_ay = 0, buff_az = 0, buff_gx = 0, buff_gy =
0, buff_gz = 0;
  ////Serial.println("Leyendo sensores por primera vez...");

  while (i < (buffersize + 101)) {

    accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);             //
lectura raw de accel/gyro

    if (i > 100 && i <= (buffersize + 100)) { //Descartar 100 primeras
mediciones
      buff_ax = buff_ax + ax;
      buff_ay = buff_ay + ay;
      buff_az = buff_az + az;
      buff_gx = buff_gx + gx;
      buff_gy = buff_gy + gy;
      buff_gz = buff_gz + gz;
    }
    if (i == (buffersize + 100)) {
      mean_ax = buff_ax / buffersize;
      mean_ay = buff_ay / buffersize;
      mean_az = buff_az / buffersize;
      mean_gx = buff_gx / buffersize;
      mean_gy = buff_gy / buffersize;
      mean_gz = buff_gz / buffersize;
    }
    if (i % 50 == 0) {
      Serial.println(i);
    }

    i++;
    delay(2); //Necesario para no repetir medidas
  }
  Serial.println("END Minsensor");
}

void calibration() {
```

```
  //offset_ang_x = -atan2(mean_ay, mean_az) * (180/PI);    //Angulo de
rotacion, no elevacion de un plano
  //offset_ang_y = atan2(mean_ax, mean_az) * (180/PI);
  double gravedad = 16384;
  offset_ang_x = -asin(mean_ay / gravedad) * (180 / PI);  //Angulo de
rotacion, no elevacion de un plano
  offset_ang_y = asin(mean_ax / gravedad) * (180 / PI);
}

// FILTRO COMPLEMENTARIO

void updateFiltered()
{
  dt = (millis() - tiempo_prev) / 1000.0;
  tiempo_prev = millis();

  //Calcular los ángulos con acelerometro
  float accel_ang_x = atan(ay / sqrt(pow(ax, 2) + pow(az, 2))) *
(180.0 / 3.14);
  float accel_ang_y = atan(-ax / sqrt(pow(ay, 2) + pow(az, 2))) *
(180.0 / 3.14);

  //Calcular angulo de rotación con giroscopio y filtro complementario
  ang_x = 0.98 * (ang_x_prev + (gx / 131) * dt) + 0.02 * accel_ang_x;
  ang_y = 0.98 * (ang_y_prev + (gy / 131) * dt) + 0.02 * accel_ang_y;

  pitch = ang_x;
  roll = ang_y;

  ang_x_prev = ang_x;
  ang_y_prev = ang_y;
}

void configuracionIMU() {

  Wire.begin();
  TWBR = 24; // 400kHz I2C clock (200kHz if CPU is 8MHz).
  mpu.initialize();
  accelgyro.initialize();
  Serial.println(mpu.testConnection() ? F("IMU iniciada
correctamente") : F("Error al iniciar IMU"));

  Serial.println(accelgyro.getFullScaleAccelRange());

  // reset de los offsets

  // Iniciamos las calibraciones:

  meansensors();
    // State 0
  //Serial.println("Calculando offsets...");
  calibration();
   // State 1

  Serial.print("mean_ax:");
  Serial.print(mean_ax);
  Serial.print("mean_ay:");
  Serial.print(mean_ay);
  Serial.print("mean_az:");
  Serial.println(mean_az);
```

```
  Serial.println("IMU configurada correctamente");
}

////////////////////////////////////   CONFIGURACION
GPS   ////////////////////////////////

void configuracionGPS() {

  pinMode(rxPin, INPUT);      //puerto donde se recibe señal gps
  pinMode(txPin, OUTPUT);
  puertoGPS.begin(9600);     // Puerto serie con el GPS
  Serial.println("GPS configurado correctamente");

}

////////////////////////////////////   CONFIGURACION
SD   ////////////////////////////////////

void configuracionSD() {
  if (!SD.begin(53))
  {
    Serial.println("Error al iniciar SD");
    return;
  }
  Serial.println("SD configurada correctamente");
}

////////////////////////////////////   FUNCIONES
BOTONES   ////////////////////////////////////

void bchestePushCallback(void *ptr) //funcion seleccionada al pulsar
boton cheste
//Pop para evento al soltar - Push para evento al pulsar
{
  //Serial.print("Envio coordenadas Cheste"); //enviar matriz formada
por coordenadas de la linea de meta
  //meta[0] = 39.483554;
  //meta[1] = -0.631065;

  //coordenadas prueba sant cugat 41.471052, 2.043695
  meta[0] = 41.471052;
  meta[1] = 2.043695;
  configuracionOK();
}
void bchestePopCallback(void *ptr)
{
  //Serial.print("volver pagina inicio");
}

void bjerezPushCallback(void *ptr)
{
  //Serial.print("Envio coordenadas Jerez");
  meta[0] = 36.709683;
  meta[1] = -6.032505;
  configuracionOK();
}
void bjerezPopCallback(void *ptr)
{
  //Serial.print("volver pagina inicio");
}
```

```
void bmontmeloPushCallback(void *ptr)
{
  //Serial.print("Envio coordenadas Montmelo"); //enviar matriz
formada por coordenadas de la linea de meta
  meta[0] = 41.570029;
  meta[1] = 2.261223;
  configuracionOK();
}
void bmontmeloPopCallback(void *ptr)
{
  //Serial.print("volver pagina inicio");
}

void bmotorlandPushCallback(void *ptr)
{
  //Serial.print("envio coord Motorland");
  meta[0] = 41.078288;
  meta[1] = -0.197821;
  configuracionOK();
}
void bmotorlandPopCallback(void *ptr)
{
  //Serial.print("volver pagina inicio");
}

////////////////////////////////////   FUNCIONES
IMU   ////////////////////////////////////

void obtenerDatosIMU() {
  //Serial.println("FUNCION IMU");
  mpu.getAcceleration(&ax, &ay, &az);    // Leer las aceleraciones
  mpu.getRotation(&gx, &gy, &gz);        // Leer las velocidades
angulares
  updateFiltered();                      // Aplicar filtro de datos
leidos

  if (abs(pitch) > abs(mayorx)) {        // Comprobar si tenemos un
maximo nuevoGPS
    mayorx = pitch;
  }
  if (abs(roll) > abs(mayory)) {
    mayory = abs(roll);
  }
}


////////////////////////////////////   FUNCIONES
GPS   ////////////////////////////////////

void obtenerDatosGPS() {

  //Serial.println("FUNCION GPS");

  String tramaGPS;

  if (puertoGPS.available()) {

    puertoGPS.read();
    int inicioGPS = millis();
```

```
    if (puertoGPS.find("$GPRMC,")) {

      tramaGPS = puertoGPS.readStringUntil('\n');

      //Serial.println(tramaGPS);

      String datoshora = tramaGPS.substring(0, 6);
      String datoslati = tramaGPS.substring(12, 22);
      String datoslon = tramaGPS.substring(25, 36);
      String datosvelocidad = tramaGPS.substring(39, 44);


      float minutosLa = tramaGPS.substring(14, 23).toFloat();
      int gradosLa = tramaGPS.substring(12, 14).toInt();
      lati = gradosLa + minutosLa / 60.0;

      float minutosLo = tramaGPS.substring(28, 37).toFloat();
      int gradosLo = tramaGPS.substring(25, 28).toInt();
      lon = gradosLo + minutosLo / 60.0;

      velocidad = round(datosvelocidad.toFloat() * 1.852);

      //      Serial.print("latitud ");
      //      Serial.println(lati, 4);
      //
      //      Serial.print("longitud ");
      //      Serial.println(lon, 4);
      //
      //      Serial.print("velocidad ");
      //      Serial.println(velocidad);


      nuevoGPS = true;
      return;
    }

  }
  else {
    nuevoGPS = false;
    return;
  }
}

////////////////////// FUNCION COMPROBAR PASO POR META
///////////////////////
void comprobarMeta() {

  //Serial.println("FUNCION META");

  posActual[0] = lati; //meta(A,B), posAct(C,D)
  posActual[1] = lon;

  double d = sqrt(pow((meta[0] - posActual[0]) , 2) + pow((meta[1] -
posActual[1]), 2));  //Calcula la distancia entre la posicion actual y
la meta registrada

  if (d < 0.0003) {
```

```
    tiempo_vuelta = millis() - vuelta_anterior; //CALCULO DEL TIEMPO
DE LA VUELTA

    if (millis() - vuelta_anterior > 15000) { //Descarta vueltas
inferiores a 15seg

      minutos = tiempo_vuelta / 60000; //SEPARACION DEL TIEMPO
      segundos = tiempo_vuelta / 1000;
      milis = tiempo_vuelta % 1000;

      //Mejor vuelta
      if (tiempo_vuelta < mejorvuelta) {
        mejorvuelta = tiempo_vuelta;
        mejorminutos = mejorvuelta / 60000;
        mejorsegundos = mejorvuelta / 1000;
        mejormilis = mejorvuelta % 1000;
      }
      vuelta_anterior = millis(); //ACTUALIZACION DEL TIEMPO ANTERIOR


      //abre el dataFile csv donde almacena los datos
      dataFile2 = SD.open("laplog.csv", FILE_WRITE);

      if (dataFile2) {
        //Serial.println("VUELTA
GUARDADAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA");
        dataFile2.print("vuelta: ");
        dataFile2.println(tiempo_vuelta);
        dataFile2.close();
      }
      else {
        Serial.println("Error al abrir el dataFile");
      }
    }
  }
  return;
}

////////////////////////////////////   FUNCIONES
SD   ////////////////////////////////////

void guardarDatosSD() {

  //Serial.println("FUNCION SD");

  //abre el dataFile csv donde almacena los datos
  dataFile = SD.open("datalog.csv", FILE_WRITE);

  if (dataFile) {
    dataFile.print(millis());
    dataFile.print(";");
    dataFile.print(lati);          //latitud
    dataFile.print(";");
    dataFile.print(lon);           //longitud
    dataFile.print(";");
    dataFile.print(velocidad);     //Velocidad
    dataFile.print(";");           //Angulo en X PITCH
    dataFile.print(pitch);
    dataFile.print(";");           //Angulo en Y ROLL
    dataFile.print(roll);
```

```
    dataFile.print(";");          //Mayor angulo alcanzado en X PITCH
    dataFile.print(mayorx);
    dataFile.print(";");          //Mayor angulo alcanzado en Y ROLL
    dataFile.print(mayory);
    dataFile.println();
    dataFile.close();
    //Serial.println("Datos guardados!!!!!!!!!!");
  }
  else {
    //Serial.println("Error al abrir el dataFile");
  }
}

void DatosPantalla() {
  //ENVIO DE VALOR DE VELOCIDAD A LA PANTALLA
  //Serial.println("FUNCION PANTALLA");
  if (velocidad != velocidad_en_pantalla) {
    n0.setValue(velocidad);
    velocidad_en_pantalla = velocidad;
  }
  //comprobar angulo de inclinacion correspondiente a Y(roll) e
X(pitch)

  if ((int)roll != ang_y_en_pantalla) {
    z0.setValue(roll + 90);
    n1.setValue(abs(roll));
    ang_y_en_pantalla = roll;
  }

  if (pitch != ang_x_en_pantalla) {
    j1.setValue(50 - pitch);
    ang_x_en_pantalla = pitch;
  }

  if ((int)mayory != mayory_en_pantalla) {
    n2.setValue((int)mayory);
    mayory_en_pantalla = mayory;
  }
  //envio tiempo de vuelta

  if (minutos != minutos_en_pantalla) {
    n3.setValue(minutos);
    minutos_en_pantalla = minutos;
  }
  if (segundos != segundos_en_pantalla) {
    n4.setValue(segundos);
    segundos_en_pantalla = segundos;
  }
  if (milis != milis_en_pantalla) {
    n5.setValue(milis);
    milis_en_pantalla = milis;
  }

  //envio mejor vuelta

  if (mejorminutos != mejorminutos_en_pantalla) {
    n6.setValue(mejorminutos);
    mejorminutos_en_pantalla = mejorminutos;
  }
  if (mejorsegundos != mejorsegundos_en_pantalla) {
```

```
    n7.setValue(mejorsegundos);
    mejorsegundos_en_pantalla = mejorsegundos;
  }
  if (mejormilis != mejormilis_en_pantalla) {
    n8.setValue(mejormilis);
    mejormilis_en_pantalla = mejormilis;
  }

}

void configuracionOK() {
  int OK = 100;
  j15.setValue(OK);
  j15.setValue(OK);
  num_loop = 1;

}
//////////////////////////   SETUP   ////////////////////////////

void setup() {
  Serial.begin(9600);

  bcheste.attachPush(bchestePushCallback);
  bcheste.attachPop(bchestePopCallback);
  bmontmelo.attachPush(bmontmeloPushCallback);
  bmontmelo.attachPop(bmontmeloPopCallback);
  bjerez.attachPush(bjerezPushCallback);
  bjerez.attachPop(bjerezPopCallback);
  bmotorland.attachPush(bmotorlandPushCallback);
  bmotorland.attachPop(bmotorlandPopCallback);

  Serial.println("Configurando...");


  nexInit();
  nexSerial.begin(9600);

  configuracionIMU();
  configuracionSD();
  configuracionGPS();

  nuevoGPS = false;

  pinMode(53, OUTPUT);

  Serial.println("configuracion completada");

}

//////////////////////////   LOOP   ////////////////////////////

void loop() {

  if (num_loop == 0) {
    Serial.println("Esperando circuito???");
    nexLoop(nex_listen_list);
  }

  else if (num_loop == 1) {
```

```
    //Serial.println("INICIO LOOP");

    updateFlags();

    if (flagIMU) {
      obtenerDatosIMU();            // Leer los datos del IMU
      lastTimeIMU = millis();
      flagIMU = false;
    }
    if (flagSD) {
      guardarDatosSD();
      lastTimeSD = millis();
      flagSD = false;
    }
    if (flagPantalla) {
      lastTimePantalla = millis();
      DatosPantalla();            //Muestra valores por pantalla
      flagPantalla = false;
    }

    if (nuevoGPS) {
      comprobarMeta();
    }
    obtenerDatosGPS();           // Leer los datos del GPS
    //obtenerDatosIMU();
    //guardarDatosSD();
    //DatosPantalla();
  }
}

static void updateFlags() {
  //Serial.println("ACTUALIZAR FLAGS");
  if (millis() - lastTimeIMU > refrIMU) {
    flagIMU = true;
    //Serial.println("FLAG IMU TRUE");
  }
  if (millis() - lastTimeSD > refrSD) {
    flagSD = true;
    //Serial.println("FLAG SD TRUE");
  }
  if (millis() - lastTimePantalla > refrPantalla) {
    flagPantalla = true;
    //Serial.println("FLAG PANTALLA TRUE");
  }
}
```

ETSID

UNIVERSITAT POLITECNICA DE VALENCIA

TRABAJO FIN DE GRADO:

DESARROLLO DE UNA UNIDAD TELEMETRICA PARA MOTOCICLETA BASADA EN ARDUINO

DOCUMENTO:

- PRESUPUESTO

AUTOR: ERNESTO SAN VALENTIN CALVET

TUTOR: CARLOS DOMINGUEZ

TITULACION: GRADO EN INGENIERIA ELECTRONICA INDUSTRIAL Y AUTOMATICA, ETSID

CURSO ACADEMICO 2018-2019

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

12. Presupuesto

Como se detallará a continuación, se han elaborado distintos presupuestos correspondiéndose con el desarrollo del producto, la fabricación por unidad y por último la fabricación de una unidad aún más "low cost" a la conseguida con el desarrollo de este proyecto. Debido al tipo de materiales empleados, no se han encontrado grandes empresas que suministren al por mayor estos componentes por lo que no se puede estimar una reducción adecuada del coste total obtenida por descuentos aplicados con la compra de grandes cantidades.

Es importante remarcar que no se han incluido los costes del encapsulado y la batería que debería llevar el módulo de adquisición de datos en caso de considerarse viable su implementación en una motocicleta.

12.1. Costes de Desarrollo

En los costes de desarrollo se han incluido además de las horas estimadas de trabajo que ha llevado el proyecto, la compra repetida de algunos módulos, así como el microcontrolador Arduino UNO que se empleó en un primer intento de desarrollo.

Costes de Desarrollo			
Producto	Unidades	Coste	Total
Arduiono UNO	1	8,99 €	8,99 €
Arduino MEGA	1	11,99 €	11,99 €
MPU6050	2	1,60 €	3,20 €
GPS	2	14,99 €	29,98 €
Modulo SD + MicroSD	1	4,99 €	4,99 €
ProtoBoard	2	8,29 €	16,58 €
Pantalla Nextion	1	28,32 €	28,32 €
Conexiones	1	10,16 €	10,16 €
Horas de trabajo	300	35,00 €	10.500,00 €
Total			10.614,21 €

Ilustración 42: Costes de Desarrollo

12.2. Costes por Unidad

En los costes por unidad se han incluido el precio de los componentes, así como el tiempo necesario para el montaje y programación del módulo.

Coste por unidad			
Producto	Unidades	Coste	Total
Arduino MEGA	1	11,99 €	11,99 €
MPU6050	1	1,60 €	1,60 €
GPS	1	14,99 €	14,99 €
Modulo SD + MicroSD	1	4,99 €	4,99 €
ProtoBoard	1	8,29 €	8,29 €
Pantalla Nextion	1	28,32 €	28,32 €
Conexiones	1	10,16 €	10,16 €
Horas de trabajo	1	35,00 €	35,00 €
Total			115,34 €

Ilustración 43: Costes por Unidad

12.3. Costes Unidad "low cost"

Por último, se ha realizado un presupuesto del proyecto empleando materiales comprados directamente en China, los cuales son de muy reducido coste, pero, por el contrario, cuentan con inconvenientes tales como la dificultad de hacer uso de garantía si fuera necesario, largos periodos de espera para la recepción de los envíos, así como posibles sobrecostes por el paso de los paquetes por aduanas.

Coste por unidad (Low Cost)			
Producto	Unidades	Coste	Total
Arduino MEGA	1	5,61 €	5,61 €
MPU6050	1	0,55 €	0,55 €
GPS	1	2,47 €	2,47 €
Modulo SD + MicroSD	1	4,99 €	4,99 €
ProtoBoard	1	1,08 €	1,08 €
Pantalla Nextion	1	16,85 €	16,85 €
Conexiones	1	10,16 €	10,16 €
Horas de trabajo	1	35,00 €	35,00 €
Total			76,71 €

*Ilustración 44: Costes por Unidad **low cost***

ETSID

UNIVERSITAT POLITECNICA DE VALENCIA

TRABAJO FIN DE GRADO:

DESARROLLO DE UNA UNIDAD TELEMETRICA PARA MOTOCICLETA BASADA EN ARDUINO

DOCUMENTO:

- PRUEBA DEL PROTOTIPO FINAL

AUTOR: ERNESTO SAN VALENTIN CALVET

TUTOR: CARLOS DOMINGUEZ

TITULACION: GRADO EN INGENIERIA ELECTRONICA INDUSTRIAL Y AUTOMATICA, ETSID

CURSO ACADEMICO 2018-2019

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

13. Prueba Del Prototipo Final

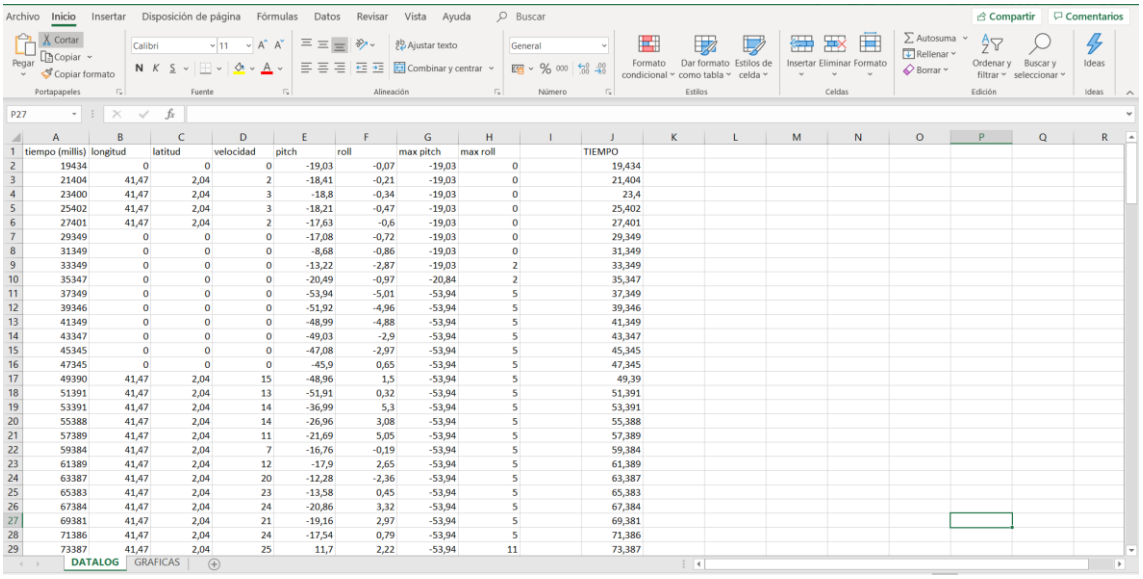
Para la prueba del prototipo final se ha realizado una pequeña simulación de lo que sería el funcionamiento normal del proyecto en una motocicleta, dentro de las limitaciones de no disponer de la posibilidad de rodar en un circuito.

Para la prueba se han introducido las coordenadas de un punto en concreto de mi localidad y se han realizado diversas pasadas en coche para el almacenamiento de datos.

```
//coordenadas prueba sant cugat 41.471052, 2.043695
meta[0] = 41.471052;
meta[1] = 2.043695;
```

Ilustración 45: Coordenadas "meta"

Tras algunos minutos de prueba, se han registrado 198 mediciones distintas en la SD



	A	B	C	D	E	F	G	H	I	J
1	tiempo (millis)	longitud	latitud	velocidad	pitch	roll	max pitch	max roll		TIEMPO
2	19434	0	0	0	-19,03	-0,07	-19,03	0		19,434
3	21404	41,47	2,04	2	-18,41	-0,21	-19,03	0		21,404
4	23400	41,47	2,04	3	-18,8	-0,34	-19,03	0		23,4
5	25402	41,47	2,04	3	-18,21	-0,47	-19,03	0		25,402
6	27401	41,47	2,04	2	-17,63	-0,6	-19,03	0		27,401
7	29349	0	0	0	-17,08	-0,72	-19,03	0		29,349
8	31349	0	0	0	-8,68	-0,86	-19,03	0		31,349
9	33349	0	0	0	-13,22	-2,87	-19,03	2		33,349
10	35347	0	0	0	-20,49	-0,97	-20,84	2		35,347
11	37349	0	0	0	-53,94	-5,01	-53,94	5		37,349
12	39346	0	0	0	-51,92	-4,96	-53,94	5		39,346
13	41349	0	0	0	-48,99	-4,88	-53,94	5		41,349
14	43347	0	0	0	-49,03	-2,9	-53,94	5		43,347
15	45345	0	0	0	-47,08	-2,97	-53,94	5		45,345
16	47345	0	0	0	-45,9	0,65	-53,94	5		47,345
17	49390	41,47	2,04	15	-48,96	1,5	-53,94	5		49,39
18	51391	41,47	2,04	13	-51,91	0,32	-53,94	5		51,391
19	53391	41,47	2,04	14	-36,99	5,3	-53,94	5		53,391
20	55388	41,47	2,04	14	-26,96	3,08	-53,94	5		55,388
21	57389	41,47	2,04	11	-21,69	5,05	-53,94	5		57,389
22	59384	41,47	2,04	7	-16,76	-0,19	-53,94	5		59,384
23	61389	41,47	2,04	12	-17,9	2,65	-53,94	5		61,389
24	63387	41,47	2,04	20	-12,28	-2,36	-53,94	5		63,387
25	65383	41,47	2,04	23	-13,58	0,45	-53,94	5		65,383
26	67384	41,47	2,04	24	-20,86	3,32	-53,94	5		67,384
27	69381	41,47	2,04	21	-19,16	2,97	-53,94	5		69,381
28	71386	41,47	2,04	24	-17,54	0,79	-53,94	5		71,386
29	73387	41,47	2,04	25	11,7	2,22	-53,94	11		73,387

Ilustración 46: Mediciones registradas en la SD

Consiguiendo gráficas de velocidades, pitch y roll como se pueden ver a continuación.

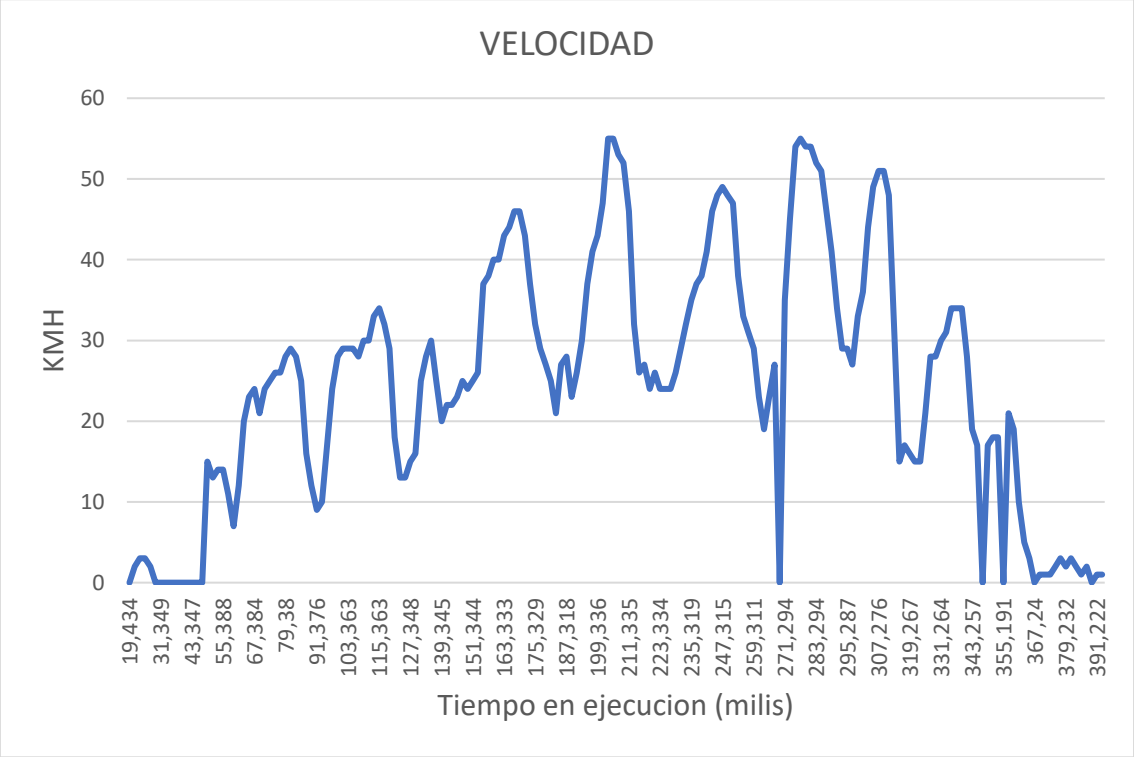


Ilustración 47: Gráfico de Velocidades registradas

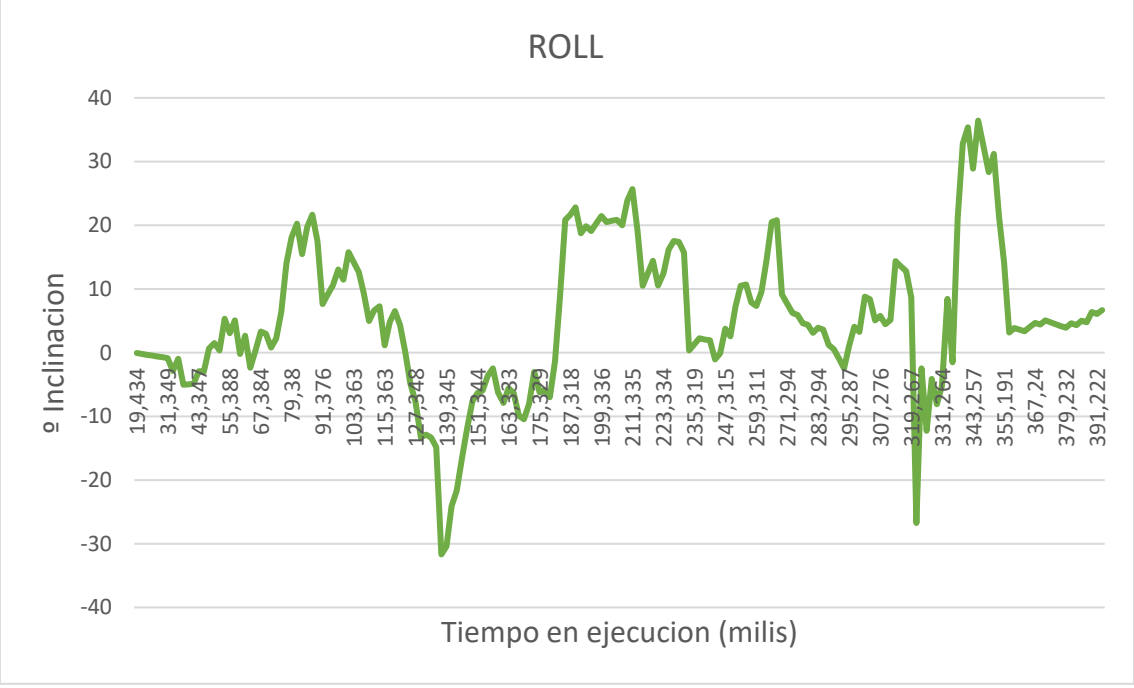


Ilustración 48: Gráfico de ángulos de inclinacion (Roll) registrados

Ilustración 49: Gráfico de ángulos de picado (Pitch) registrados

También se crea otro archivo CSV donde se registran los pasos por "meta" en milisegundos.

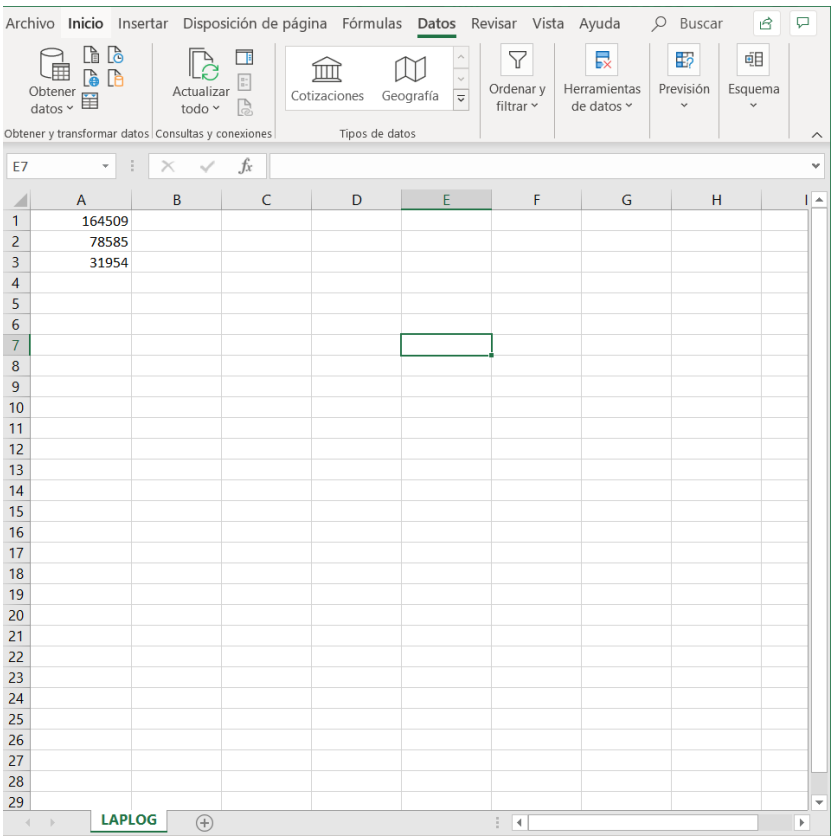


Ilustración 50: Registro de tiempos de paso por meta

ETSID

UNIVERSITAT POLITECNICA DE VALENCIA

TRABAJO FIN DE GRADO:

DESARROLLO DE UNA UNIDAD TELEMETRICA PARA MOTOCICLETA BASADA EN ARDUINO

DOCUMENTO:

- CONCLUSIONES

AUTOR: ERNESTO SAN VALENTIN CALVET

TUTOR: CARLOS DOMINGUEZ

TITULACION: GRADO EN INGENIERIA ELECTRONICA INDUSTRIAL Y AUTOMATICA, ETSID

CURSO ACADEMICO 2018-2019

14. Conclusiones

Con las pruebas finales realizadas con el prototipo, se puede observar que se ha conseguido acercarse al objetivo inicial del proyecto, aunque hay varios puntos a tener en cuenta que se detallaran a continuación por los que se acaba concluyendo que no es totalmente viable el uso de Arduino y componentes low-cost si se busca conseguir unos resultados útiles y cercanos a los necesitados para un equipo de motociclismo y por lo que finalmente no se ha añadido la creación de un encapsulado para su uso.

La unidad telemétrica queda totalmente condicionada por la incapacidad del microcontrolador Arduino para realizar multitarea ya que como se ha podido comprobar en el desarrollo del proyecto, la ejecución de las distintas funciones por separado proporciona unos resultados fluidos y en tiempo real que permiten un visionado por pantalla perfecto y sin ningún tipo de interrupción. El problema reside cuando se intentan implementar todas las funciones de manera simultánea ya que algunas de ellas necesitan periodos de tiempo bastante amplios (hasta 1 segundo) para completarse y esto ocasiona que todo lo demás se detenga, llegando incluso a producir un desbordamiento del código que arroja medidas ilógicas.

Las principales causantes de este problema son la función GPS, la cual para funcionar correctamente debe estar continuamente en ejecución para no perder ninguna trama de datos recibida. La función que guarda los datos en la tarjeta SD también es crítica ya que, aunque se ha intentado optimizar mediante la programación el guardad de datos, la acción de abrir, escribir y cerrar el archivo lleva un tiempo en el que el microcontrolador pierde distintas mediciones.

Las conclusiones del proyecto son que, aunque para un uso de recreo, este sería viable, si se busca alcanzar una adquisición de datos útil para un equipo de competición, se necesita mínimo, una unidad de procesamiento de datos más potente y con la opción de realizar multitarea.

ETSID

UNIVERSITAT POLITECNICA DE VALENCIA

TRABAJO FIN DE GRADO:

DESARROLLO DE UNA UNIDAD TELEMETRICA PARA MOTOCICLETA BASADA EN ARDUINO

DOCUMENTO:

- OPCIONES DE MEJORA

AUTOR: ERNESTO SAN VALENTIN CALVET

TUTOR: CARLOS DOMINGUEZ

TITULACION: GRADO EN INGENIERIA ELECTRONICA INDUSTRIAL Y AUTOMATICA, ETSID

CURSO ACADEMICO 2018-2019

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

15. Opciones De Mejora

Tal y como se ha expuesto en las conclusiones, las carencias del proyecto son claras y se detallaran las posibles soluciones a continuación.

- Emplear un procesador con varios núcleos para conseguir una multitarea capaz de ejecutar todas las funciones al mismo tiempo.
- GPS con mayor precisión y capacidad de conexión para una configuración más rápida y unos datos almacenados más precisos.
- Creación de una PCB específica para el montaje de los componentes y así poder evitar posibles fallos causados por los cableados, conexiones y vibraciones, así como disminuir el grosor del conjunto.
- Creación de un encapsulado para todo el conjunto que incluya una batería para la alimentación del dispositivo, así como un sistema de sujeción a la motocicleta, empleando como se había pensado en un inicio, un acople para cámaras de acción tipo GoPro.
- Implementación de una conexión con el BUS CAN de la ECU de la motocicleta para obtener parámetros de esta, tales como velocidades, rpm, porcentaje del acelerador empleado muy utilizadas en el estudio de telemetría y necesarias para generar canales calculados que devuelvan información de marcha engranada, sobre régimen del motor etc.

16. Bibliografía

16.1. Telemetría

- https://es.wikipedia.org/wiki/Telemetr%C3%ADa
- http://www.aficionadosalamecanica.com/sistemas-telemetria-en-la-competicion/
- https://www.boxrepsol.com/es/motogp/la-telemetria-explicada-todos-la-entendamos/
- https://www.boxrepsol.com/es/motogp/mejoran-los-sensores-la-electronica-rendimiento-una-motogp/

16.2. Lenguaje de Programación

- https://es.wikibooks.org/wiki/Lenguaje_de_programaci%C3%B3n_Arduino
- https://playground.arduino.cc/ArduinoNotebookTraduccion/Structure

16.3. Protocolos de Comunicación

- https://www.diarioelectronicohoy.com/blog/introduccion-al-i2c-bus
- https://es.wikipedia.org/wiki/I%C2%B2C
- https://es.wikipedia.org/wiki/Universal_Asynchronous_Receiver-Transmitter
- https://www.rinconingenieril.es/funciona-puerto-serie-la-uart/
- https://www.luisllamas.es/arduino-spi/

16.4. NMEA

- https://www.gpsinformation.org/dale/nmea.htm#nmea
- http://www.informaticaabordo.com/2011/10/%C2%BFque-es-el-standar-nmea/

16.5. Arduino/Arduino Mega 2560

- https://www.xataka.com/basics/que-arduino-como-funciona-que-puedes-hacer-uno
- https://es.wikipedia.org/wiki/Arduino
- https://www.luisllamas.es/tag/generico/
- http://manueldelgadocrespo.blogspot.com/p/arduino-mega-2560.html
- http://www.electrontools.com/Home/WP/2018/06/19/arduino-mega-2560-caracteristicas/
- https://www.arduino.cc/en/Hacking/PinMapping2560

16.6. MPU6050

- https://naylampmechatronics.com/blog/45_Tutorial-MPU6050-Aceler%C3%B3metro-y-Giroscopio.html
- https://robologs.net/2014/10/15/tutorial-de-arduino-y-mpu-6050/
- https://www.luisllamas.es/medir-la-inclinacion-imu-arduino-filtro-complementario/

16.7. GPS GY-NEO6MV2

- https://leantec.es/blog/54_Tutorial-Arduino--Modulo-GPS-GPS6MV2.html
- https://naylampmechatronics.com/blog/18_Tutorial-M%C3%B3dulo-GPS-con-Arduino.html
- https://www.nextiafenix.com/producto/modulo-gps-gy-neo6mv2/

16.8. Modulo de almacenamiento SD

- https://www.luisllamas.es/tarjeta-micro-sd-arduino/
-

16.9. Nextion

- https://www.rinconingenieril.es/pantalla-nextion-de-itead-y-arduino/
- https://nextion.itead.cc/resources/datasheets/nx3224t028_011/