



**Universidad de Valladolid**



**ESCUELA DE INGENIERÍAS  
INDUSTRIALES**

**UNIVERSIDAD DE VALLADOLID**

**ESCUELA DE INGENIERIAS INDUSTRIALES**

**Grado en Ingeniería Electrónica Industrial y Automática**

**Localizador GPS basado en Arduino. Diseño y  
fabricación de prototipo para su aplicación en  
el análisis de parámetros de rutas**

**Autor:**

**Sastre Sáez, Guillermo**

**Tutora:**

**Pérez Barreiro, Cristina  
Departamento de Tecnología  
Electrónica**

**Valladolid, julio de 2024**



## **Agradecimientos**

A mi madre, mi padre y mi hermana, por estar ahí pase lo que pase, apoyándome incondicionalmente en todas las decisiones de mi vida, aguantándome y deseando siempre lo mejor para mí.

A mi grupo de amigos, por alegrarme el corazón en los días más grises y ser una fuente constante de inspiración. En especial, a mi amigo Alberto, por echarme una mano en la fase final de este proyecto.

A mi amigo Félix, por animarme en los momentos más difíciles y estar siempre a mi lado.

A mi tutora y profesora Cristina Pérez Barreiro, por involucrarse a ser mi tutora y darme la libertad que quería para realizar este TFG.

A mi profesor José Manuel González y a los profesionales Oliver Vicente Cabeza y Enrique Martín Delgado, por compartir generosamente sus valiosos conocimientos y consejos conmigo sin pedir nunca nada a cambio.

A mi amiga Celia, por aportar tanta luz a mi vida y ser una referente para mí, por motivarme a ser un mejor estudiante y, sobre todo, por enseñarme a ser una mejor persona.

**Muchas gracias a todos, sin vosotros llegar al final de este largo y arduo camino hubiera sido mucho más difícil.**







## **RESUMEN**

Arduino es una plataforma de electrónica de código abierto ampliamente reconocida en el ámbito de la ingeniería y la programación. Basada en hardware y software libre, Arduino proporciona a los desarrolladores una vasta gama de herramientas para la creación de innovadores dispositivos electrónicos.

En este proyecto, se explora una de las numerosas aplicaciones de Arduino mediante el desarrollo de un localizador GPS destinado al registro y análisis de datos en actividades al aire libre. La combinación de una placa Arduino Nano, un módulo GPS y un lector de tarjetas microSD permite la captura precisa de coordenadas geográficas y la capacidad de almacenar esta información para su posterior análisis y visualización.

Este trabajo demuestra el potencial de la tecnología para mejorar diversos aspectos de la vida cotidiana, particularmente en el ámbito deportivo y recreativo, a través de un dispositivo eficiente, accesible e innovador.

## **PALABRAS CLAVE**

GPS, Arduino, Power BI, EAGLE, localizador, prototipo.

## **ABSTRACT**

Arduino is an open-source electronics platform widely recognized in the fields of engineering and programming. Based on open-source hardware and software, Arduino provides developers with a vast variety of tools for creating innovative electronic devices.

In this project, one of the many applications of Arduino is explored through the development of a GPS locator designed for logging and analyzing data in outdoor activities. The combination of an Arduino Nano board, a GPS module, and a microSD card reader enables precise capture of geographic coordinates and the ability to store this information for later analysis and visualization.

This work demonstrates the potential of technology to enhance various aspects of everyday life, particularly in the sports and recreational fields, through an efficient, accessible, and innovative device.

## **KEY WORDS**

GPS, Arduino, Power BI, EAGLE, locator, prototype.





## ÍNDICE

ÍNDICE DE FIGURAS	_____
ÍNDICE DE TABLAS	_____
SIGLAS Y ACRÓNIMOS	_____
<b>1. INTRODUCCIÓN Y OBJETIVOS</b>	<b>_____ 1</b>
1.1. INTRODUCCIÓN	_____ 1
1.2. MOTIVACIÓN	_____ 1
1.3. OBJETIVOS	_____ 2
<b>2. CONCEPTOS PREVIOS</b>	<b>_____ 7</b>
2.1. EL SISTEMA GPS	_____ 7
2.1.1. ELEMENTOS DEL GPS	_____ 7
2.1.2. PRINCIPIO DE FUNCIONAMIENTO DEL GPS	_____ 9
2.1.3. APLICACIONES DEL GPS	_____ 11
2.1.4. LATITUD, LONGITUD Y ALTITUD	_____ 13
2.1.5. PROTOCOLO NMEA	_____ 15
2.2. EL ECOSISTEMA ARDUINO	_____ 16
2.2.1. ¿QUÉ ES ARDUINO?	_____ 16
2.2.2. ¿POR QUÉ UTILIZAR ARDUINO?	_____ 17
2.2.3. LA PROGRAMACIÓN EN ARDUINO	_____ 18
2.3. INTERFAZ DE COMUNICACIÓN UART	_____ 18
2.3.1. TEMPORIZACIÓN Y SINCRONIZACIÓN DE DATOS	_____ 19
2.3.2. FORMATO DE TRAMA EN UART	_____ 20
2.3.3. APLICACIONES DE UART	_____ 21
2.4. INTERFAZ DE COMUNICACIÓN SPI	_____ 22
2.4.1. ELEMENTOS DEL SPI	_____ 22
2.4.2. CARACTERÍSTICAS DEL SPI	_____ 23
2.4.3. APLICACIONES DE SPI:	_____ 24
<b>3. DESCRIPCIÓN DEL DISPOSITIVO A DESARROLLAR</b>	<b>_____ 27</b>
3.1. JUSTIFICACIÓN DEL DISEÑO	_____ 27
3.2. ANÁLISIS DE MERCADO	_____ 27
3.3. ELEMENTOS QUE FORMAN EL DISPOSITIVO	_____ 32
3.3.1. MÓDULO GPS GY-GPS6MV2	_____ 32
3.3.2. ARDUINO NANO EVERY	_____ 34



3.3.3. MÓDULO LECTOR TARJETAS MICROSD	37
3.3.4. BATERÍA LIPO 3.7 V / 1000 mAh	41
3.3.5. CARGADOR LiPo	41
<b>3.4. MODO DE OPERACIÓN DEL DISPOSITIVO</b>	<b>45</b>
3.4.1. OBTENCIÓN DE DATOS EN FORMATO NMEA	45
3.4.2. PROCESADO Y CONVERSIÓN DE DATOS	45
3.4.3. CÁLCULO DE PARÁMETROS DE RUTAS	47
3.4.4. ALMACENAMIENTO DE LA INFORMACIÓN	52
3.4.5. UNIFICACIÓN DE LA INFORMACIÓN	55
3.4.6. VISUALIZACIÓN DE LA INFORMACIÓN	55
<b>3.5. PROTOTIPADO INICIAL</b>	<b>57</b>
<b>3.6. COMPROBACIÓN INICIAL DE FUNCIONAMIENTO</b>	<b>59</b>
3.6.1. LIMITACIONES EN LA PRECISIÓN DEL RECEPTOR GPS NEO-M8N	60
<b>3.7. PRIMERA APROXIMACIÓN DEL COSTE DEL EQUIPO</b>	<b>62</b>
<b>4. DISEÑO DEL DISPOSITIVO</b>	<b>67</b>
4.1. REALIZACIÓN DEL DISEÑO DE LA PLACA DE CIRCUITO IMPRESO	67
4.1.1. ANCHURA MÍNIMA DE LAS PISTAS:	69
4.1.2. SEPARACIÓN DE LAS PISTAS	70
4.2. GENERACIÓN DE LA DOCUMENTACIÓN PARA LA FABRICACIÓN DE LA PCB	72
4.2.1. FOTOLITOS	72
4.2.2. FICHEROS DE FABRICACIÓN	73
4.3. DISEÑO DE LA CARCASA	74
<b>5. FABRICACIÓN Y AJUSTE DEL DISPOSITIVO</b>	<b>79</b>
5.1. FABRICACIÓN DE LA PLACA DE CIRCUITO IMPRESO	79
5.2. FABRICACIÓN Y MONTAJE DE LA CARCASA	81
5.3. AJUSTE Y VERIFICACIÓN DEL EQUIPO	82
5.4. CORRECIÓN DE ERRORES	83
5.4.1. ERRORES PCB	83
5.4.2. ERRORES CARCASA	84
<b>6. TRATAMIENTO Y VISUALIZACIÓN DE DATOS</b>	<b>89</b>
6.1. TRATAMIENTO DE LOS DATOS OBTENIDOS	89
6.1.1. CREACIÓN DE ARCHIVO EJECUTABLE	89
6.2. VISUALIZACIÓN DE LOS DATOS TRATADOS	92
6.2.1. CREACIÓN DE PLANTILLA DE INFORME	97



<b>7. COMERCIALIZACIÓN DEL EQUIPO</b>	<b>101</b>
7.1. COSTE TOTAL DEL EQUIPO	101
7.2. CATÁLOGO COMERCIAL DEL EQUIPO	102
7.2.1. ESPECIFICACIONES TÉCNICAS	102
7.2.2. FUNCIONALIDADES	102
7.2.3. APLICACIONES	103
7.2.4. BONDADES	103
7.2.5. MANUAL DE USUARIO	103
<b>8. CONCLUSIONES</b>	<b>109</b>
8.1. IMPACTO DEL PROYECTO EN LOS OBJETIVOS DE DESARROLLO SOSTENIBLE	110
<b>9. LÍNEAS FUTURAS</b>	<b>115</b>
<b>10. BIBLIOGRAFÍA</b>	<b>119</b>
<b>ANEXOS</b>	<b>125</b>
ANEXO I. PLANOS	127
ANEXO II. FOTOGRAFÍAS	133
ANEXO III. HOJAS DE CARACTERÍSTICAS	135
ANEXO V. PROGRAMAS DESARROLLADOS	151



## ÍNDICE DE FIGURAS

Figura 1. Esquema de funcionamiento inicial del localizador GPS (elaboración propia)	2
Figura 2. Segmento espacial del GPS (extraído de [1])	8
Figura 3. Mapa del segmento de control del GPS (extraído de [1])	8
Figura 4. Ejemplo de segmento de usuario del GPS (extraído de [2])	9
Figura 5. El GPS basa su funcionamiento en la trilateración (extraído de [4])	10
Figura 6. Ejemplos de implementación de un sistema GPS (extraído de [1])	12
Figura 7. Uso de dispositivo GPS en actividades deportivas (extraído de [11])	13
Figura 8. Proyecciones ortográficas de la Tierra con las líneas de paralelos y meridianos. Proyección ecuatorial (izquierda) y proyección oblicua (derecha) (extraído de [13])	14
Figura 9. Diferencia entre altitud, altura y elevación (extraído de [14])	14
Figura 10. Ejemplo de trama GPS en formato NMEA (elaboración propia)	15
Figura 11. Logo de Arduino (extraído de [18])	16
Figura 12. Transmisión de datos en UART (extraído de [21])	19
Figura 13. Formato de trama en UART (extraído de [21])	20
Figura 14. Bits de inicio y parada en UART (extraído de [21])	20
Figura 15. Bits de datos en UART (extraído de [21])	20
Figura 16. Bit de paridad en UART (extraído de [21])	21
Figura 17. SPI bus: un maestro y tres esclavos (extraído de [28])	23
Figura 18. ALLROUND Finder 2.0 de PAJ GPS (extraído de [32])	28
Figura 19. Tracker GPS de Oliver (extraído de [33])	29
Figura 20. Amazfit GTR 3 Pro (extraído de [34])	30
Figura 21. Reloj Garmin Forerunner 165 de Garmin (extraído de [35])	30
Figura 22. Módulo GPS GY-GPS6MV2 (extraído de [37])	33
Figura 23. Chip GPS NEO-M8N (extraído de [39])	33
Figura 24. Placa Arduino Nano Every (extraído de [40])	35
Figura 25. Pinout del Arduino Nano Every (extraído de [40])	37
Figura 26. Tarjeta de memoria microSD (extraído de [41])	40
Figura 27. Módulo lector de tarjetas microSD (extraído de [41])	40
Figura 28. Batería LiPo modelo 603050 (extraído de [42])	41



Figura 29. Curva de carga de una batería LiPo (extraído de [44])	42
Figura 30. Cargador LiPo PowerBoost 500C (extraído de [45])	42
Figura 31. Interruptor tipo Rocker (extraído de [46])	43
Figura 32. Código para el procesado y conversión de datos recibidos del módulo GPS en Arduino IDE (elaboración propia)	47
Figura 33. Distancia ortodrómica entre dos puntos situados sobre una esfera (elaboración propia)	49
Figura 34. Pendiente entre dos puntos (extraído de [49])	51
Figura 35. Diagrama de bloques del sketch de Arduino transformarDatos.ino (elaboración propia)	54
Figura 36. Ejemplo de informe en Power BI (extraído de [52])	56
Figura 37. Esquema final de funcionamiento del localizador GPS (elaboración propia)	57
Figura 38. Prototipado inicial realizado (elaboración propia)	58
Figura 39. Esquema de montaje inicial en Fritzing (elaboración propia)	58
Figura 40. Ejemplo de fichero resultado DATOS.txt (elaboración propia)	60
Figura 41. Placa de expansión para Arduino Nano o Arduino Nano Shield (extraído de [55])	67
Figura 42. Detalle del esquema eléctrico del dispositivo (elaboración propia)	68
Figura 43. Emplazamiento de los componentes en la PCB (elaboración propia)	69
Figura 44. Requisitos de espacio de conductores según norma 2221B (extraído de [56])	71
Figura 45. Rutado de pistas (elaboración propia)	72
Figura 46. Fitolito cara de componentes (elaboración propia)	72
Figura 47. Fitolito cara de soldadura (elaboración propia)	73
Figura 48. Modelo 3D de la PCB diseñada (elaboración propia)	75
Figura 49. Modelo 3D de la carcasa final (elaboración propia)	75
Figura 50. PCB fabricada con componentes (cara superior) (elaboración propia)	80
Figura 51. PCB fabricada sin componentes (cara superior) (elaboración propia)	80
Figura 52. PCB fabricada (cara inferior) (elaboración propia)	81
Figura 53. Vista superior de la carcasa final (elaboración propia)	82
Figura 54. Componentes del localizador GPS montados sobre la carcasa final (elaboración propia)	82



Figura 55. Vista superior. Primera versión de la PCB con componentes (elaboración propia)	84
Figura 56. Vista superior. Primera versión de la PCB sin componentes (elaboración propia)	84
Figura 57. Vista inferior. Primera versión de la PCB (elaboración propia)	84
Figura 58. Vista superior exterior. Primera versión de la carcasa sin componentes (izquierda) y con componentes (derecha) (elaboración propia)	86
Figura 59. Vista superior interior.-Primera versión de la carcasa sin componentes (izquierda) y con componentes (derecha) (elaboración propia)	86
Figura 60. Vista lateral. Primera versión de la carcasa sin componentes (izquierda) y con componentes (derecha) (elaboración propia)	86
Figura 61. Carpeta con unificarDatos.py y 6 ficheros de texto a unificar (elaboración propia)	90
Figura 62. Programa 'Unificación de Rutas' (elaboración propia)	90
Figura 63. Programa 'Unificación de Rutas' sin rutas nuevas (elaboración propia)	90
Figura 64. Carpeta con unificarDatos.py y un nuevo fichero de texto a unificar (elaboración propia)	91
Figura 65. Programa 'Unificación de Rutas' con una nueva ruta (elaboración propia)	91
Figura 66. Ejemplo de historicoDATOS.xlsx (elaboración propia)	91
Figura 67. Página 1 del informe - Parámetros, estadísticas y gráficas (elaboración propia)	93
Figura 68. Perfil de altitud (arriba) y pendiente (abajo) obtenido en una de las rutas realizadas (elaboración propia)	94
Figura 69. Comparativa entre el perfil de altitud obtenido utilizando el localizador GPS (arriba) y utilizando valores reales (abajo) (elaboración propia)	95
Figura 70. Página 2 del informe - Trazado del recorrido (elaboración propia)	96
Figura 71. Detalle trazado del recorrido (elaboración propia)	96
Figura 72. Página 3 del informe - Histórico de rutas (elaboración propia)	97
Figura 73. Plantilla para informe de rutas (elaboración propia)	97
Figura 74. Cuadro de diálogo inicial al crear un informe de rutas (elaboración propia)	98
Figura 75. Ejemplo de informe creado a partir de la plantilla (elaboración propia)	98
Figura 76. Catálogo comercial del localizador GPS desarrollado (elaboración propia)	105





Figura 77. Objetivos de Desarrollo Sostenible (extraído de [57]) \_\_\_\_\_ 111



## ÍNDICE DE TABLAS

Tabla 1. Especificaciones técnicas del chip GPS NEO-M8N (elaboración propia) _	34
Tabla 2. Especificaciones técnicas del Arduino Nano Every (elaboración propia) _	36
Tabla 3. Información a almacenar (elaboración propia) _____	38
Tabla 4. Reparto de consumos estimados (elaboración propia) _____	43
Tabla 5. Conexión de pines entre Arduino Nano y módulo GPS (elaboración propia) _____	59
Tabla 6. Conexión de pines entre Arduino Nano y lector de tarjetas microSD (elaboración propia) _____	59
Tabla 7. Conexión de pines entre Arduino Nano y cargador LiPo (elaboración propia) _____	59
Tabla 8. Lista de componentes inicial (elaboración propia) _____	63
Tabla 9. Códigos LED de diagnóstico del dispositivo (elaboración propia) _____	83
Tabla 10. Datos asociados a las rutas realizadas que aparecen en el informe de Power BI (elaboración propia) _____	93
Tabla 11. Coste total del localizador GPS desarrollado (elaboración propia) _____	101



## SIGLAS Y ACRÓNIMOS

**ANT:** Topología de Red Adaptativa (Adaptive Network Topology)

**ASCII:** Código Estándar Americano para el Intercambio de Información (American Standard Code for Information Interchange)

**B:** Byte

**BeiDou:** Sistema de Navegación por Satélite de China

**BOM:** Lista de Materiales (Bill of Materials)

**CIPO:** Controlador Entrada/Periférico Salida (Controller In/Peripheral Out)

**CNC:** Control Numérico Computarizado (Computer Numerical Control)

**COPI:** Controlador Salida/Periférico Entrada (Controller Out/Peripheral In)

**CS:** Selección de Chip (Chip Select)

**DC:** Corriente Continua (Direct Current)

**EEPROM:** Memoria Programable y Borrable Electrónicamente de sólo Lectura (Electrically Erasable Programmable Read-Only) Memory

**GB:** Gigabyte

**GLONASS:** Sistema Global de Navegación por Satélite de Rusia

**GND:** Tierra (Ground)

**GNGGA:** Datos de Fijación del Sistema Global de Navegación por Satélite (Global Navigation Satellite System Fix Data)

**GNLL:** Posición Geográfica del Sistema Global de Navegación por Satélite - Latitud/Longitud (Global Navigation Satellite System Geographic Position - Latitude/Longitude)

**GNSS:** Sistema Global de Navegación por Satélite (Global Navigation Satellite System)

**GPGSV:** Satélites GPS a la Vista (GPS Satellites in View)

**GPS:** Sistema de Posicionamiento Global (Global Positioning System)

**GSM:** Sistema Global para Comunicaciones Móviles (Global System for Mobile Communications)

**I2C:** Circuito Inter-Integrado (Inter-Integrated Circuit)

**IDE:** Entorno de Desarrollo Integrado (Integrated Development Environment)

**IMU:** Unidad de Medición Inercial (Inertial Measurement Unit)

**iOS:** sistema operativo móvil de Apple (iPhone Operating System)

**IoT:** Internet de las Cosas (Internet of Things)



**IPC:** Comunicación entre Procesos (Inter-Process Communication)

**IVA:** Impuesto sobre el Valor Añadido

**KB:** Kilobyte

**LCD:** Pantalla de Cristal Líquido (Liquid Crystal Display)

**LED:** Diodo Emisor de Luz (Light Emitting Diode)

**LiPo:** Polímero de Litio (Lithium Polymer)

**MISO:** Maestro Entrada/Esclavo Salida (Master In/Slave Out)

**MIT:** Instituto de Tecnología de Massachusetts (Massachusetts Institute of Technology)

**MOSI:** Maestro Salida/Esclavo Entrada (Master Out/Slave In)

**NMEA:** Asociación Nacional de Electrónica Marina (National Marine Electronics Association)

**ODS:** Objetivos de Desarrollo Sostenible

**PCB:** Placa de Circuito Impreso (Printed Circuit Board)

**PLA:** Ácido Poliláctico (Polylactic Acid)

**PVP:** Precio de Venta al Público

**RFID:** Identificación por Radiofrecuencia (Radio-Frequency Identification)

**SaaS:** Software como Servicio (Software as a Service)

**SCK:** Reloj en serie (Serial Clock)

**SD:** Tarjeta Digital Segura (Secure Digital card)

**SIM:** Módulo de Identificación de abonado (Subscriber Identity Module)

**SMBus:** Bus de Administración del Sistema (System Management Bus)

**SPI:** Interfaz Periférica en Serie (Serial Peripheral Interface)

**SQL:** Lenguaje de Consulta Estructurada (Structured Query Language)

**SS:** Selección de Esclavo (Slave Select)

**UART:** Transmisor/Receptor Asíncrono Universal (Universal Asynchronous Receiver-Transmitter)

**USB:** Bus Universal en Serie (Universal Serial Bus)





# CAPÍTULO 1: INTRODUCCIÓN Y OBJETIVOS



## 1. INTRODUCCIÓN Y OBJETIVOS

### 1.1. INTRODUCCIÓN

En el mundo actual, la salud y el bienestar físico se han convertido en prioridades cada vez más importantes para personas de todas las edades. Con la creciente conciencia sobre los beneficios del ejercicio regular y la actividad física, la búsqueda de herramientas y tecnologías que faciliten y mejoren estas prácticas se ha intensificado en los últimos años. En este contexto, la tecnología desempeña un papel crucial al ofrecer soluciones innovadoras que no solo hacen que el ejercicio sea más accesible y efectivo, sino que también permiten un seguimiento y análisis detallados del rendimiento y el progreso personal. Más concretamente, la electrónica, con su capacidad para integrar hardware y software de manera creativa y funcional, se presenta como un recurso súper poderoso para abordar los desafíos relacionados con la práctica deportiva y el seguimiento de la salud. En particular, plataformas de código abierto como Arduino han ganado un significativo reconocimiento, gracias a su enorme versatilidad y accesibilidad.

En este trabajo, se pretende aprovechar algunas de las muchas posibilidades que ofrece Arduino para crear un localizador GPS que se diseñará específicamente para el registro de datos durante la realización de rutas al aire libre, con el fin de fomentar la práctica deportiva. Mediante la combinación de distintos componentes electrónicos, este aparato permitirá la captura precisa y el almacenamiento de información relacionada con la ubicación del usuario para su posterior análisis.

Al poner en práctica esta aplicación concreta de la electrónica, se quiere demostrar el potencial que posee ésta para mejorar aspectos fundamentales de la vida cotidiana, como es el ejercicio físico y la salud.

### 1.2. MOTIVACIÓN

La motivación detrás de este Trabajo de Fin de Grado se origina a partir de varias influencias y aspiraciones multidisciplinares. En primer lugar, me surgió la inquietud de explorar el universo de Arduino, ya que, a pesar de ser una de las plataformas de desarrollo electrónico más prominentes del mundo, yo no era conocedor de su enorme potencial.

En este sentido, este trabajo ha supuesto una excelente oportunidad para sumergirme en un ambiente de aprendizaje autónomo, sin las habituales limitaciones temporales y estructurales que conllevan el entorno académico convencional.

Además, tras haber cursado la asignatura de "Instrumentación Electrónica", he experimentado un renovado interés por la diversidad de sensores disponibles en el ámbito electrónico. Esto ha convertido la búsqueda de nuevas tecnologías



con las que trabajar, como por ejemplo, el GPS en un aspecto fundamental en la concepción del presente trabajo.

A todo esto se suma la idea de desarrollar un dispositivo tangible que no solo sea funcional, sino que también pueda integrarse en la vida cotidiana como un accesorio más, con aplicaciones prácticas y reales, suponiendo un factor determinante en la elección del enfoque deportivo del proyecto.

Por último, se añade el propósito de consolidar y aplicar los conocimientos que he ido adquiriendo a lo largo de la formación académica y poder integrar competencias ligadas al diseño de sistemas electrónicos.

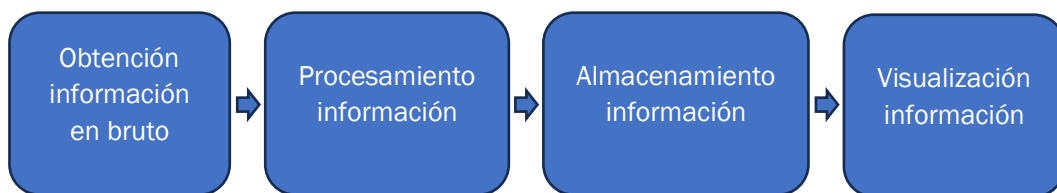
En definitiva, la posibilidad de fusionar teoría y práctica, y de desplegar habilidades aprendidas durante estos años de estudios en un entorno real, ha supuesto un reto muy estimulante para mí, impulsando el compromiso y dedicación hacia este, mi Trabajo de Fin de Grado.

### 1.3. OBJETIVOS

El objetivo principal de este proyecto es diseñar y fabricar un prototipo de localizador GPS funcional y portátil que sirva como dispositivo para registrar la información relacionada con las rutas o recorridos realizados por su portador.

Posteriormente, esta información será tratada en un ordenador con el objetivo de que pueda ser interpretada mediante una herramienta de visualización de datos. El resultado final obtenido será un informe o *dashborad* en el que el usuario podrá observar diversos parámetros, estadísticas y gráficas relacionados con la ruta previamente realizada.

Las etapas que forman parte de la idea inicial de funcionamiento del dispositivo se muestran en la *Figura 1*:



*Figura 1. Esquema de funcionamiento inicial del localizador GPS (elaboración propia)*

Para completar estas etapas, son necesarios distintos componentes, cada uno de los cuáles desempeñará una función específica. El primero de estos componentes debe ser un sensor GPS que capte los datos asociados a la ubicación del dispositivo. No obstante, estos datos son enviados como cadenas de texto en formato NMEA, por lo que, tal y como se explicará más adelante, necesitan ser procesadas y convertidas en información comprensible para los usuarios, esto es, en coordenadas de latitud, longitud y altitud, fecha y hora. Aquí es donde entra en juego la incorporación de un microcontrolador para realizar dicha tarea. Además, el microcontrolador también se encargará de





realizar ciertos cálculos y almacenará la información convertida en una célula de memoria. Finalmente, a través de la herramienta de visualización de datos adecuada, esta podrá ser analizada por el usuario propietario del dispositivo.

Inherente a este objetivo, surgen los siguientes objetivos secundarios:

- ❖ **Comprender conceptos clave:** esto supone entender conceptos esenciales que proporcionen una base sólida para la implementación del localizador GPS propuesto, como son el funcionamiento de la plataforma Arduino, el de la tecnología GPS o el de los interfaces de comunicación necesarios para intercambiar información entre los distintos elementos que formarán parte del dispositivo.
- ❖ **Establecer el modo de operación del dispositivo:** esto implica diseñar y ejecutar cómo va a ser todo el proceso de funcionamiento del localizador GPS desde que se reciben los datos hasta que son visualizados por el usuario.
- ❖ **Aprender a utilizar distintos programas:** para avanzar en el desarrollo del dispositivo será necesario adquirir conocimientos básicos en el manejo de distintas aplicaciones.
- ❖ **Aplicar competencias técnicas obtenidas durante el grado universitario:** para la correcta consecución del proyecto, será necesario repasar y utilizar aquellos conocimientos y capacidades adquiridas en las asignaturas cursadas durante la carrera.

Con el fin de lograr tanto el objetivo principal como los secundarios, se llevarán a cabo los siguientes hitos a lo largo del trabajo:

- ❖ **Análisis de mercado:** se va a realizar una búsqueda de equipos ya fabricados existentes en el mercado similares al dispositivo a desarrollar. Esto ofrecerá una visión global que permitirá comparar y determinar las características que deben adaptarse al prototipo a desarrollar.
- ❖ **Elección de los componentes que formen el dispositivo:** se planteará el tipo de dispositivo a construir, esto es, sus especificaciones técnicas, en qué ámbito se va a utilizar, a qué público va dirigido, que funcionalidades va a presentar, etc. y, a partir de ahí, se seleccionarán aquellos componentes electrónicos que mejor se adecúen a las necesidades específicas del proyecto. Esto implica realizar estudio detallado de las diversas alternativas de microcontroladores, módulos GPS y otros sensores disponibles en el mercado. Se evaluarán diversas opciones en términos de prestaciones, tamaño y precios.
- ❖ **Prototipado inicial:** una vez elegidos los componentes electrónicos y realizados todos aquellos cálculos necesarios para cada parte del



proyecto, se verificará que todo funciona de acuerdo con las especificaciones preestablecidas.

- ❖ **Diseño del dispositivo:** se realizará el diseño del dispositivo en función de las necesidades funcionales y físicas del proyecto.
- ❖ **Fabricación del dispositivo:** posterior al diseño, se construirá el localizador GPS. Esto incluye desde procesos como el mecanizado de la placa de circuito impreso hasta la confección de la carcasa del dispositivo.
- ❖ **Verificación y ajuste del dispositivo:** se realizarán múltiples pruebas con el fin de asegurar que las mediciones obtenidas estén correlacionadas con los requerimientos inicialmente propuestos.
- ❖ **Tratamiento y visualización de los datos obtenidos:** mediante las herramientas de software adecuadas, se procesará la información almacenada en el localizador GPS, facilitando su interpretación y análisis al usuario final, lo que ayudará a mejorar su rendimiento deportivo.

Estos hitos abarcan el ciclo completo de desarrollo del proyecto, desde la conceptualización y el diseño hasta la implementación y la validación del prototipo, asegurando que cada etapa del proceso esté bien definida y estructurada.



# CAPÍTULO 2: CONCEPTOS PREVIOS





## 2. CONCEPTOS PREVIOS

Antes de describir el localizador GPS que se pretende implementar, se ha creído fundamental explicar en profundidad ciertos conceptos clave como son el sistema GPS, la plataforma Arduino y los interfaces de comunicación UART y SPI, ya que estos suponen los pilares sobre los que se sustentará el dispositivo objeto de este proyecto.

### 2.1. EL SISTEMA GPS

Dado que el GPS se erige como la tecnología central en el desarrollo de este Trabajo de Fin de Grado, resulta esencial entender sus principios de funcionamiento, componentes y aplicaciones. En este apartado, se abordarán los aspectos técnicos y operativos del sistema GPS con el fin de proporcionar una base sólida de conocimiento.

El Sistema de Posicionamiento Global es un sistema basado en el uso de satélites que permite a un dispositivo receptor localizar su propia posición sobre la Tierra con una precisión de unos pocos metros. El sistema fue desarrollado, instalado y empleado por el Departamento de Defensa de Estados Unidos, siendo actualmente propiedad de la Fuerza Espacial de los Estados Unidos. [1]

#### 2.1.1. ELEMENTOS DEL GPS

El GPS se compone de tres elementos: el segmento espacial, formado por satélites que giran alrededor de la Tierra; el segmento de control, formado por las estaciones de seguimiento y control distribuidas por el mundo y el segmento usuario que se compone de los receptores GPS que poseen los usuarios.

##### Segmento espacial

Consiste en una constelación de 24 satélites operativos uniformemente distribuidos en un total de 6 órbitas, de forma que hay 4 satélites por órbita (*Figura 2*). Los satélites GPS orbitan la Tierra a una altitud de unos 20.000 kilómetros y recorren dos órbitas cada día. Cada satélite transmite señales de radio a la Tierra con información acerca de su posición y el instante de tiempo en el que se emite la señal.

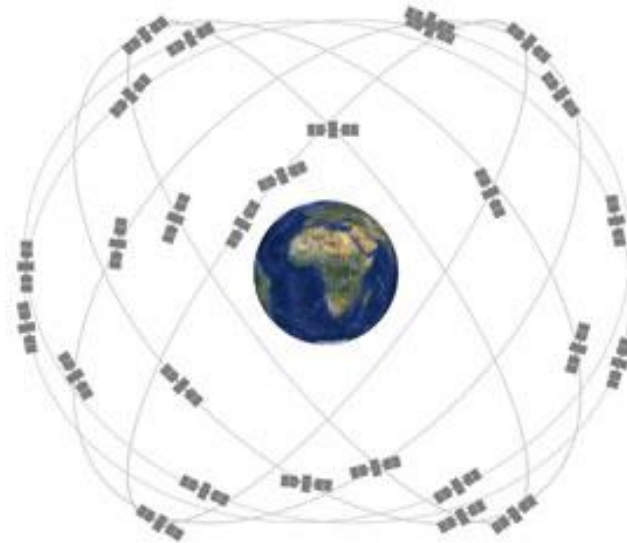


Figura 2. Segmento espacial del GPS (extraído de [1])

Segmento de control:

Está compuesto por estaciones de monitorización, estaciones de control maestras y antenas, todas ellas con base en la Tierra. Su propósito es realizar un seguimiento de los satélites GPS, manteniendo los satélites en la órbita apropiada mediante maniobras de mando, ajustando los relojes satelitales y cargando información actualizada de navegación. Es decir, se encarga de garantizar el correcto funcionamiento del segmento espacial. Como se puede ver en la Figura 3, hay estaciones de control en casi todos los continentes de la superficie terrestre.

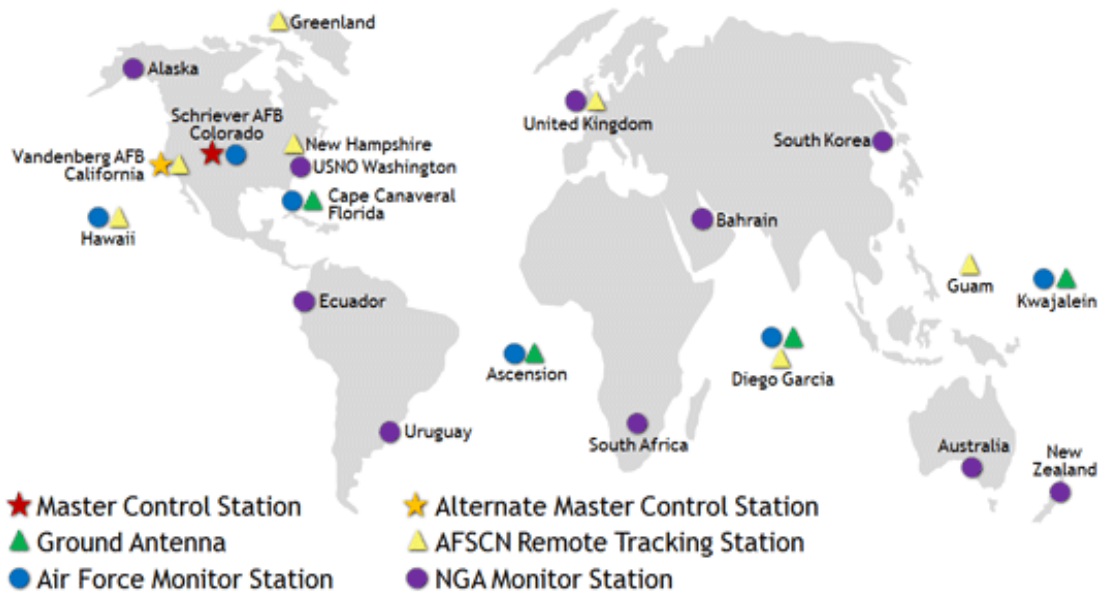


Figura 3. Mapa del segmento de control del GPS (extraído de [1])

### Segmento del usuario

Este tercer y último segmento lo constituyen los receptores GPS que incluyen artículos como relojes o teléfonos inteligentes. Estos aparatos decodifican las señales enviadas por varios satélites de forma simultánea y combinan esta información para calcular su posición tridimensional en la Tierra (latitud, longitud y altitud).



Figura 4. Ejemplo de segmento de usuario del GPS (extraído de [2])

En resumen, desde el espacio, los satélites del GPS transmiten señales que reciben los receptores del GPS; ellos, a su vez, proporcionan por separado sus coordenadas tridimensionales de latitud, longitud y altitud, así como la hora local precisa. [3]

Por último, es de interés mencionar que, actualmente, el GPS no es el único sistema de navegación satelital disponible. A pesar de ser el más conocido, existen otros sistemas como el GLONASS, gestionado por Rusia; el sistema de navegación GALILEO, de la Unión Europea; y el BeiDou, de la República Popular China. Todos estos sistemas ofrecen servicios similares y, a menudo, se integran en dispositivos de navegación para mejorar la cobertura y precisión. La razón por la que todos estos dispositivos se conocen comúnmente como "GPS" se debe a la popularidad y el pionerismo del sistema GPS, que se ha convertido en un término genérico para cualquier aparato que utilice una red satelital para calcular su ubicación.

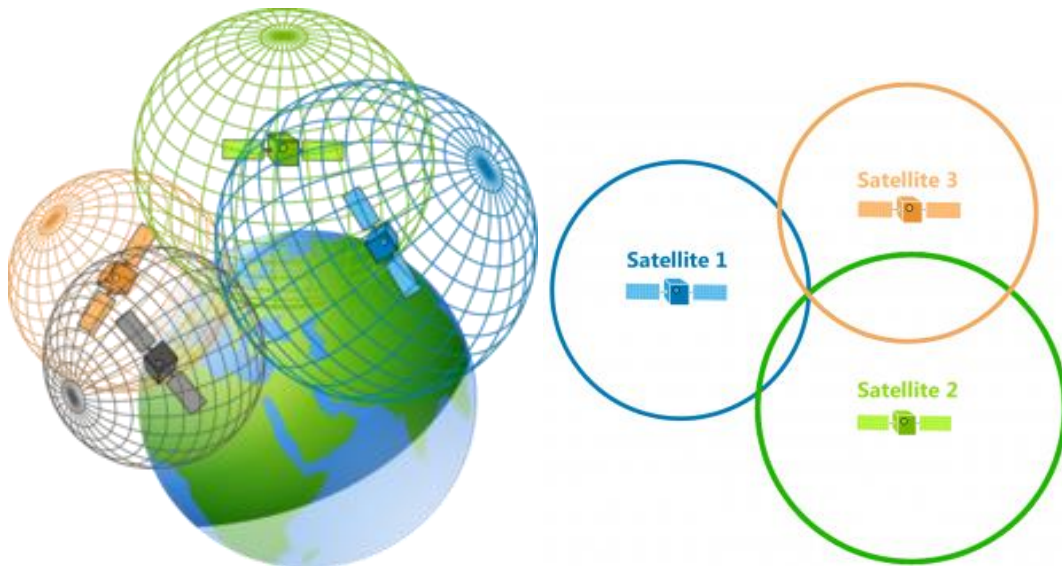
#### **2.1.2. PRINCIPIO DE FUNCIONAMIENTO DEL GPS**

El funcionamiento del GPS se basa en una técnica denominada trilateración, esencial para determinar la posición exacta de un receptor en la Tierra. Cabe destacar que aunque a menudo se confunde con el concepto de triangulación (que mide ángulos) la trilateración se centra en la medición de distancias. Esta se basa en la recepción de señales de al menos tres satélites en órbita alrededor de la Tierra. Cada satélite GPS emite una señal que contiene información precisa sobre su posición y la hora exacta de la transmisión.



Cuando un receptor GPS capta la señal de un satélite, utiliza la información de tiempo para calcular la distancia entre el receptor y el satélite. Esta distancia define una esfera con el satélite en su centro y el receptor en cualquier punto de la superficie de esta esfera. Por lo tanto, es evidente que una única esfera no es suficiente para determinar una ubicación. Si el receptor GPS recibe señales de un segundo satélite, se genera una segunda esfera. La intersección de esta con la primera crea un círculo, situándose la ubicación del receptor GPS en algún punto de este círculo. En consecuencia, la posición precisa podría ser cualquier punto de este círculo, por lo que dos satélites siguen resultando insuficientes.

Sin embargo, con la adición de un tercer satélite, en lugar de un círculo, ahora la intersección de tres esferas da como resultado dos posibles puntos. En la práctica, uno de estos puntos se descarta, generalmente el que se encuentra en el espacio, y el otro, más cercano a la superficie terrestre, se acepta como la ubicación final del receptor (ver *Figura 5*).



*Figura 5. El GPS basa su funcionamiento en la trilateración (extraído de [4])*

A medida que el receptor se desplaza, las distancias a los satélites cambian, lo que genera nuevas esferas, actualizándose continuamente la posición del receptor. Este proceso no solo determina la ubicación, sino también la velocidad, la distancia recorrida y el tiempo estimado de llegada a un destino.

Por otra parte, resulta interesante mencionar que el uso de un cuarto satélite proporciona una posición del receptor todavía más exacta. Este cuarto satélite permite corregir posibles errores de tiempo en los cálculos del receptor GPS. Debido a que la exactitud de la trilateración depende en gran medida de la sincronización precisa entre los relojes de los satélites y el receptor, cualquier discrepancia en el tiempo puede introducir errores en la determinación de la posición. El cuarto satélite permite al receptor realizar ajustes finos en su reloj interno, eliminando la necesidad de que el reloj del receptor sea extremadamente preciso. Al proporcionar una





referencia adicional, el cuarto satélite permite al receptor GPS resolver cualquier inconsistencia en las mediciones de tiempo y, por ende, afinar la ubicación calculada.

En conclusión, la trilateración en el GPS implica la recepción de señales de múltiples satélites, la medición de las distancias basadas en el tiempo de viaje de estas señales y la determinación de la posición mediante la intersección de esferas en un espacio tridimensional. [5], [6], [7], [8]

### 2.1.3. APLICACIONES DEL GPS

El GPS, debido a su naturaleza gratuita, fiable y continua, ha permitido el desarrollo de numerosas aplicaciones presentes en casi todas las facetas de la vida moderna. En el ámbito de la navegación y la seguridad, el GPS es indispensable para los sistemas de transporte aéreo, terrestre y marítimo. Los servicios de emergencia y rescate utilizan el GPS para la localización y coordinación en misiones de salvamento, aumentando significativamente la eficiencia y efectividad de las operaciones de respuesta a desastres.

En la agricultura de precisión, el GPS se emplea para planificar cultivos, localizar terrenos, y guiar maquinaria agrícola en condiciones de baja visibilidad, mejorando la eficiencia y reduciendo costos. En la aviación, el GPS proporciona servicios de navegación por satélite que mejoran la seguridad y eficiencia de los vuelos, permitiendo una navegación precisa desde el despegue hasta el aterrizaje.

Además, el GPS ha revolucionado la gestión de flotas, mejorando la asignación de rutas y aumentando la seguridad en las carreteras. También es crucial para la cronometría, ya que permite sincronizar con alta precisión sistemas de comunicación, redes eléctricas y financieras, esenciales para la operación eficiente de estas infraestructuras. [9]



Figura 6. Ejemplos de implementación de un sistema GPS (extraído de [1])

Sin embargo, entre todas las aplicaciones del GPS, aquella en la que se centra este Trabajo de Fin de Grado es en su uso para actividades recreativas, proporcionando a los entusiastas del aire libre la capacidad de determinar su posición exacta y facilitando la navegación en senderismo, ciclismo y otras actividades.

Y es que el GPS ha mejorado significativamente la seguridad en estas actividades al permitir una localización precisa, ampliando el alcance y el disfrute de los usuarios y resolviendo problemas tradicionales como mantenerse en el sendero correcto o encontrar puntos específicos de interés. Las actividades al aire libre conllevan ciertos riesgos, y perderse en territorio desconocido es uno de los más grandes. Por ello, la mayoría de senderistas, ciclistas y aventureros confían ciegamente en el GPS, superando las limitaciones de los mapas impresos, brújulas y puntos de referencia, que pueden ser obsoletos o inexactos.

No sólo eso, sino que el uso de un dispositivo GPS, combinado con mapas electrónicos, supera muchas de las barreras tradicionales de la exploración. Los receptores GPS portátiles permiten a los usuarios seguir rutas con precisión y regresar a su punto de partida con facilidad, almacenando y recuperando puntos intermedios. Además, estos dispositivos tienen la capacidad de intercambiar datos con ordenadores, permitiendo a los usuarios cargar y compartir las coordenadas exactas de sus aventuras. [10]

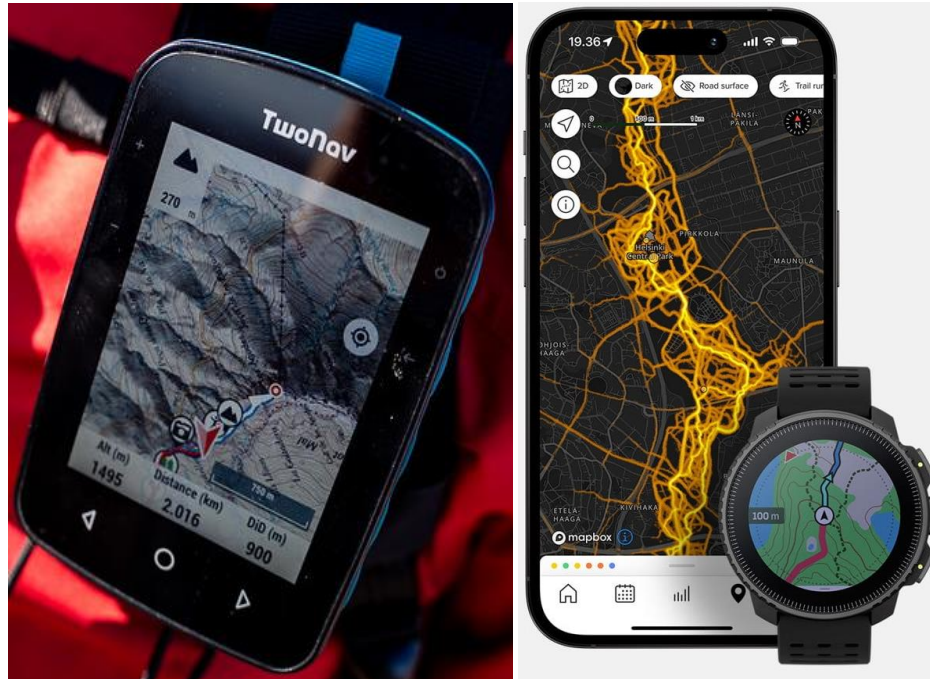


Figura 7. Uso de dispositivo GPS en actividades deportivas (extraído de [11])

#### 2.1.4. LATITUD, LONGITUD Y ALTITUD

Tal y como se comentó en el apartado 2.1.1, el resultado final obtenido de un receptor del GPS son sus coordenadas tridimensionales de latitud, longitud y altitud, así como la hora local.

A continuación, se pretende explicar en qué consisten cada una de estas tres coordenadas.

- ❖ **Latitud:** se simboliza con la letra griega  $\phi$  y se trata del ángulo imaginario que determina la distancia entre un punto cualquiera y el ecuador (la línea imaginaria horizontal que divide al mundo en dos hemisferios: Norte y Sur y cuya latitud es  $0^\circ$ ). En otras palabras, se trata de qué tan lejos o tan cerca está un punto de la referencia del ecuador. De esta forma, si un punto determinado se encuentra en el hemisferio norte (sur), su coordenada de latitud irá acompañada de la letra N (S). Otro tipo de nomenclatura refiere latitudes norte con números positivos y latitudes sur con números negativos.
- ❖ **Longitud:** se simboliza con la letra griega  $\lambda$  y se trata del ángulo imaginario que determina la distancia entre un punto y el meridiano de Greenwich o primer meridiano, que atraviesa la localidad que recibe el mismo nombre en Londres, Inglaterra. Dicho meridiano es vertical y divide al mundo en dos regiones, la occidental (Oeste) y la oriental (Este), y sirve para trazar los demás meridianos que cruzan imaginariamente el globo de manera paralela al meridiano de Greenwich. Si se mide un ángulo al este (oeste) del meridiano

de Greenwich se escribe la letra E (W) acompañando al número que da la longitud. También se pueden utilizar números negativos. [12]

En la *Figura 8*, se muestra un ejemplo visual de la latitud y longitud en la Tierra.

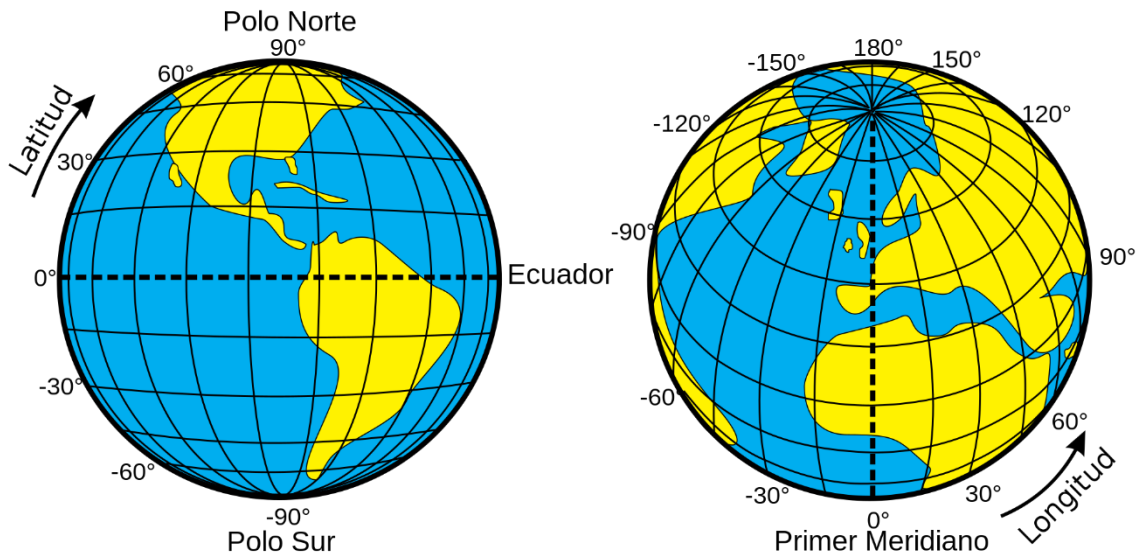


Figura 8. Proyecciones ortográficas de la Tierra con las líneas de paralelos y meridianos. Proyección ecuatorial (izquierda) y proyección oblicua (derecha) (extraído de [13])

- ❖ **Altitud:** la altitud se define como la distancia vertical de un cuerpo hasta el nivel del mar, independientemente de que el objeto esté sobre la superficie de la Tierra o por encima de ella.

Es importante no confundir altitud con los términos altura y elevación. A diferencia de la altitud, la altura es la distancia vertical de un cuerpo hasta la superficie terrestre (ya sea suelo o mar). Por su parte, la elevación es la distancia vertical hasta el nivel del mar de un punto que está sobre la superficie terrestre y en contacto con ella. [14]

Para comprender mejor la diferencia entre estos conceptos se muestra la *Figura 9*.

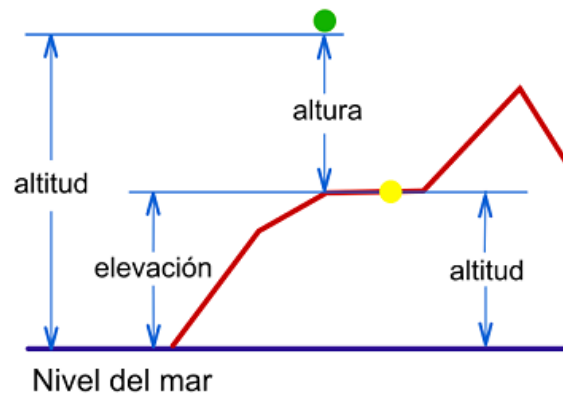


Figura 9. Diferencia entre altitud, altura y elevación (extraído de [14])





### 2.1.5. PROTOCOLO NMEA

Por último, resulta de vital importancia mencionar que un receptor GPS entrega sus datos a través de un estándar mundialmente utilizado para la comunicación entre dispositivos electrónicos llamado NMEA 0183, como ya se comentó en el apartado 1.3. Este protocolo permite la transmisión de datos de posicionamiento y navegación entre equipos como receptores GPS, sistemas de navegación y otros dispositivos a bordo.

Un receptor GPS devuelve una línea de datos, denominada “trama”, junto con sus respectivos separadores reglamentarios (usualmente comas) y letras o símbolos de identificación. Estas tramas se componen de subtramas que contienen información específica del dispositivo. Para simplificar la explicación, se muestra el siguiente ejemplo (*Figura 10*) donde aparecen múltiples tramas NMEA:

```
$GPGSV,3,1,12,01,58,310,31,02,73,009,,03,56,227,28,04,07,176,*75
$GPGSV,3,2,12,08,44,152,,10,01,048,10,14,29,281,29,17,23,314,26*73
$GPGSV,3,3,12,21,65,059,,22,23,304,27,27,09,143,13,32,24,052,17*7B
$GLGSV,1,1,02,78,16,301,14,79,04,348,*6D
$GNGLL,4138.44075,N,00445.64351,W,202849.00,A,A*6C
$GNRMC,202850.00,A,4138.44056,N,00445.64299,W,0.179,,270224,,,A*77
$GNVTG,,T,,M,0.179,N,0.331,K,A*33
$GNGGA,202850.00,4138.44056,N,00445.64299,W,1,05,7.77,742.7,M,50.2,M,,*56
$GNGSA,A,3,03,17,22,14,,,,,,,,,13.68,7.77,11.27*12
$GNGSA,A,3,78,,,,,,,,,13.68,7.77,11.27*1D
$GPGSV,3,1,12,01,58,310,31,02,73,009,,03,56,227,28,04,07,176,*75
$GPGSV,3,2,12,08,44,152,,10,01,048,09,14,29,281,30,17,23,314,26*73
$GPGSV,3,3,12,21,65,059,17,22,23,304,27,27,09,143,13,32,24,052,17*7D
$GLGSV,1,1,02,78,16,301,12,79,04,348,*6B
$GNGLL,4138.44056,N,00445.64299,W,202850.00,A,A*60
```

Figura 10. Ejemplo de trama GPS en formato NMEA (elaboración propia)

Como se puede observar, cada sentencia NMEA comienza con el carácter '\$', seguido del campo de dirección, que consiste en el identificador del emisor (talker ID) formado por 2 letras y el identificador del tipo de mensaje formado por 3 letras. Por ejemplo, una sentencia que empiece por el prefijo "GP" indica que el origen corresponde a un receptor GPS, mientras que aquellas que empiezan por "GN" o "GL" indican que los datos provienen de un sistema de navegación, como los ya mencionados GPS de EEUU, el Galileo de la UE o el GLONASS de Rusia.

Algunos de los tipos de mensajes que se envían son:

- ❖ GNGGA (Global Navigation Satellite System Fix Data) brinda datos detallados sobre la posición del receptor, incluyendo la latitud, altitud, hora y calidad de la señal GNSS.



- ❖ GPGSV (Global Navigation Satellite System Satellites in view) proporciona información el número de satélites que hay disponibles para el receptor.
- ❖ GNGLL (Latitude and Longitude, with time of position fix and status) ofrece información sobre la latitud y longitud de la posición actual, junto con la hora correspondiente.

La siguiente parte de una trama NMEA es la carga útil (*payload*) que lleva el mensaje, es decir, la propia información que se está transmitiendo. Está formada por varios campos de datos separados por comas ','. La carga útil va seguida de un carácter '\*', que indica el inicio de la suma de verificación (*checksum*). Esta tiene como propósito principal detectar cambios accidentales en la secuencia de datos a transmitir para proteger la integridad de estos, verificando que no haya errores entre los valores obtenidos al hacer una comprobación inicial y otra final después de la transmisión. En este caso se transmite con dos caracteres que representan su valor en hexadecimal y comprende todos los caracteres entre los caracteres '\$' y '\*'. [15], [16], [17]

## 2.2. EL ECOSISTEMA ARDUINO

Como ya se mencionó con anterioridad, Arduino supone otra de las bases de este proyecto. Por ello, en este apartado se ahondará en las características que hacen de Arduino una de las plataformas electrónicas más prominentes del mundo en la actualidad.

### 2.2.1. ¿QUÉ ES ARDUINO?

Arduino es una plataforma de electrónica de código abierto que combina hardware y software de manera accesible y fácil de usar. Las placas Arduino son capaces de interpretar entradas, como la luz captada por un sensor, la presión de un botón o un mensaje de WhatsApp, y transformarlas en salidas, como activar un motor, encender un LED o publicar algo en Internet. Se programa mediante el lenguaje de Arduino (basado en Wiring) y el software Arduino IDE (basado en Processing), enviando instrucciones al microcontrolador integrado en la placa.



Figura 11. Logo de Arduino (extraído de [18])

Arduino fue creado en el año 2005 en el Instituto de Diseño de Interacción de Ivrea como una herramienta de prototipado rápido para estudiantes sin experiencia en electrónica y programación. Desde entonces, Arduino ha evolucionado



significativamente. Hoy en día, la plataforma soporta una amplia gama de aplicaciones, desde simples placas de 8 bits hasta dispositivos IoT, *wearables*, impresoras 3D y sistemas embebidos. Esta versatilidad ha permitido que Arduino se convierta en el corazón de innumerables proyectos, abarcando desde objetos cotidianos hasta instrumentos científicos avanzados.

Tanto es así, que una extensa comunidad global de creadores, que incluye estudiantes, aficionados, artistas, programadores y profesionales, se ha formado en torno a Arduino. Las contribuciones de esta comunidad han generado una vastísima cantidad de conocimiento accesible, útil tanto para principiantes como para expertos. Esta colaboración enriquece continuamente el ecosistema de Arduino, facilitando la creación y el intercambio de proyectos innovadores. [18], [19]

### 2.2.2. ¿POR QUÉ UTILIZAR ARDUINO?

Arduino destaca por su sencillez y accesibilidad, lo que lo convierte en una herramienta idónea para una amplia variedad de proyectos y aplicaciones. El software Arduino es intuitivo para los principiantes, pero suficientemente potente para aquellos usuarios más avanzados. Además, a diferencia de muchas otras plataformas de microcontroladores cuyo uso está limitado a Windows, Arduino es compatible con sistemas operativos Mac, Windows y Linux.

Existen otros microcontroladores y plataformas para la computación física, como Parallax Basic Stamp, BX-24 de Netmedia, Phidgets y Handyboard del MIT. Aunque todos estos sistemas facilitan la programación de microcontroladores, Arduino ofrece ventajas específicas:

- ❖ **Económico:** las placas de Arduino son relativamente baratas. Incluso las versiones preensambladas suelen costar menos de 50€.
- ❖ **Multiplataforma:** como ya se ha mencionado, el software de Arduino (IDE) es compatible con Windows, Mac OS X y Linux.
- ❖ **Entorno de programación sencillo:** el Arduino IDE, basado en el entorno de programación Processing, es fácil de usar para principiantes, pero también es lo suficientemente flexible para usuarios avanzados.
- ❖ **Software de código abierto:** el software de Arduino es de código abierto y extensible, permitiendo a los programadores experimentados añadir bibliotecas de C++ o integrar código AVR-C.
- ❖ **Hardware de código abierto:** los planos de las placas de Arduino se publican bajo una licencia Creative Commons, permitiendo a los diseñadores crear y modificar sus propias versiones del hardware.

En definitiva, Arduino se destaca por ser una herramienta accesible para aprender y experimentar, con kits y recursos disponibles que permiten a cualquier persona, empezar a desarrollar sus propias ideas. [18]



### 2.2.3. LA PROGRAMACIÓN EN ARDUINO

En este apartado se pretende explicar en qué consisten y cómo funcionan los denominados *sketches* en Arduino.

Un *sketch* es el nombre que Arduino utiliza para referirse a un programa. Es la unidad de código que se carga y ejecuta en una placa de Arduino.

Como es típico en un entorno de programación, en Arduino se emplean variables para almacenar fragmentos de datos, las cuales se componen de un nombre, un tipo y un valor asignado. Estos elementos son fundamentales para el desarrollo de cualquier proyecto, ya que permiten gestionar información de manera eficiente. Además, Arduino ofrece la posibilidad de trabajar con diferentes tipos de datos, como enteros, decimales y caracteres, lo que proporciona flexibilidad al programador en la manipulación de la información. Junto con las variables, las funciones son otra herramienta esencial en la programación en Arduino. Estas son bloques de código que realizan tareas específicas y pueden ser llamadas desde cualquier parte del *sketch*. Esta modularidad facilita la organización del código y permite reutilizar segmentos de código, lo que resulta en programas más legibles y sostenibles a lo largo del tiempo. Además de las variables y las funciones, Arduino también permite la declaración de constantes, que son valores fijos que no cambian durante la ejecución del programa y se utilizan para representar elementos invariables, como pines de hardware específicos o valores de configuración predeterminados.

No obstante, lo que distingue a un programa de Arduino es la presencia de dos funciones especiales que forman parte de cada *sketch*: *setup()* y *loop()*.

La función *setup()* se llama una sola vez, al inicio del *sketch*. Es la parte del *sketch* habilitada para realizar tareas de configuración (p.ej.: inicializar bibliotecas) que deben llevarse a cabo antes de que comience la ejecución principal. Por su parte, la función *loop()* se llama repetidamente y es el corazón de la mayoría de los *sketches*, donde se coloca el código principal del programa que se repetirá en un ciclo infinito siempre y cuando la placa Arduino esté encendida. Es obligatorio incluir ambas funciones en todos los programas de Arduino, incluso si no se necesitan para nada en particular.

Esta estructura básica de *setup()* y *loop()* es fundamental en la programación de Arduino y permite crear proyectos que respondan de manera predecible a las entradas y realicen acciones específicas de forma repetida. [20]

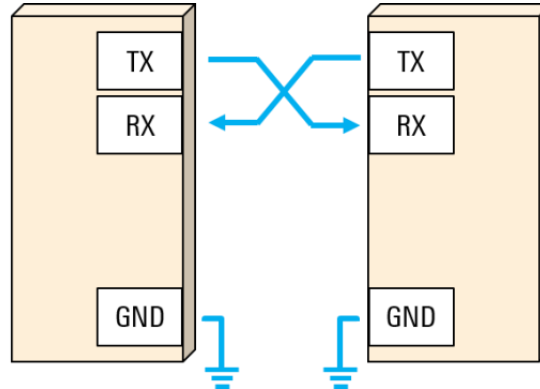
### 2.3. INTERFAZ DE COMUNICACIÓN UART

El protocolo Universal Asynchronous Receiver-Transmitter (UART) será uno de los interfaces de comunicación utilizados por el localizador GPS. En este apartado, se explicará en qué consiste este protocolo, así como sus características y elementos básicos.





El UART es un conjunto de reglas que rigen el intercambio de datos en serie entre dos dispositivos. Es un protocolo simple que emplea únicamente dos cables para la transmisión y recepción de datos en ambas direcciones, con ambos dispositivos conectados a tierra (*Figura 12*):



*Figura 12. Transmisión de datos en UART (extraído de [21])*

Funciona de manera asíncrona, lo que significa que no utiliza un reloj compartido entre el transmisor y el receptor. En cambio, utiliza líneas separadas para transmitir y recibir datos, así como para sincronizar la comunicación.

La comunicación UART puede ser de tres tipos: simplex, donde los datos fluyen en una sola dirección; semidúplex, donde cada extremo se comunica, pero solo uno a la vez; y dúplex completo, que permite la transmisión simultánea desde ambos extremos. En la actualidad, UART se utiliza en aplicaciones de baja velocidad y bajo rendimiento, puesto que es muy sencillo y económico de integrar.

Además, en UART, los datos se organizan en tramas, cuyo formato y contenido se describen un poco más adelante.

### 2.3.1. TEMPORIZACIÓN Y SINCRONIZACIÓN DE DATOS

Una de las principales características de UART radica en su carácter asíncrono, lo que significa que el transmisor y el receptor no necesitan sincronizarse mediante una misma señal de reloj. Aunque esta peculiaridad simplifica el protocolo, impone ciertos requisitos en los dispositivos emisores y receptores.

Dado que no comparten un reloj común, ambos extremos deben “acordar” previamente una velocidad de transmisión idéntica para garantizar la correcta sincronización de los bits. Las velocidades de baudios más comunes en UART incluyen 4800, 9600, 19,2 K, 57,6 K y 115,2 K. Además de la velocidad de baudios, ambos extremos de una conexión UART deben estar configurados con la misma estructura y parámetros de trama. [21]

Esto se puede entender fácilmente al observar una trama UART (*Figura 13*).



### 2.3.2. FORMATO DE TRAMA EN UART

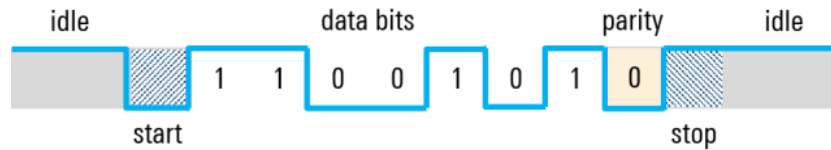


Figura 13. Formato de trama en UART (extraído de [21])

Como en la mayoría de los sistemas digitales, en UART se emplea un nivel de tensión alto para representar un estado lógico de "1", mientras que un nivel de tensión bajo indica un estado lógico de "0". Es interesante notar que durante el estado de reposo (idle), cuando no hay transmisión de datos, la línea permanece en el estado alto, lo cual facilita la detección de fallos en la línea o en el transmisor.

#### Bits de inicio y de parada

Para señalar la llegada de los bits de datos en una comunicación asíncrona, el transmisor utiliza un bit de inicio, que es una transición del estado alto de reposo (idle) al estado bajo, seguido inmediatamente de los bits de datos. Cuando finalizan los bits de datos, el bit de parada indica el final de la transmisión (Figura 14). Este bit puede ser una transición de regreso al estado alto o bien mantenerse en alto durante un tiempo adicional, y en algunos casos se puede configurar un segundo bit de parada para permitir al receptor prepararse para la próxima trama.



Figura 14. Bits de inicio y parada en UART (extraído de [21])

#### Bits de datos

Los bits de datos constituyen la información útil de la comunicación y se transmiten después del bit de inicio. La cantidad de bits de datos puede variar de 5 a 9, aunque generalmente se utilizan 7 u 8 bits. Es común transmitir los bits de datos con el bit menos significativo primero (Figura 15).



Figura 15. Bits de datos en UART (extraído de [21])

Por ejemplo, si se quiere enviar el carácter "B" en ASCII de 7 bits, la secuencia de bits sería 1 0 0 0 0 1 0 (66 en decimal). Primero, se invierte el orden de los bits para empezar la secuencia por el bit menos significativo, es decir, 0 1 0 0 0 0 1, antes de



enviarlos. Una vez transmitidos los bits de datos, el bit de parada finaliza la trama y la línea vuelve al estado de reposo.

- ❖ Letra B en ASCII con 7 bits (0x42) = 1 0 0 0 1 0
- ❖ Orden de bit menos significativo = 0 1 0 0 0 1

### Bit de paridad

Adicionalmente, una trama UART puede incluir un bit opcional de paridad, el cual se emplea para la detección de errores. Este bit se sitúa entre los bits de datos y el bit de parada (*Figura 16*). Su valor depende del tipo de paridad seleccionado (par o impar):

- ❖ En la paridad par, se ajusta de manera que el número total de unos en la trama sea par.
- ❖ En la paridad impar, se ajusta para que el número total de unos en la trama sea impar.

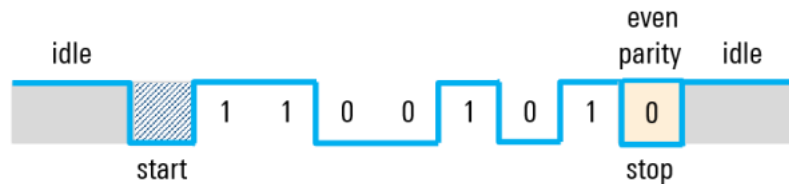


Figura 16. Bit de paridad en UART (extraído de [21])

Siguiendo con el mismo ejemplo de antes, se vuelve a considerar la letra "B" mayúscula (1 0 0 0 1 0), que contiene 3 ceros y 2 unos en total. Si aplicamos paridad par, el bit de paridad sería cero, ya que el número de unos es par. Sin embargo, si optamos por paridad impar, el bit de paridad deberá ser uno para mantener un número impar de unos en la trama.

Es importante destacar que el bit de paridad solo puede detectar un único bit invertido. Si hay más de un bit invertido, la detección no será fiable utilizando un único bit de paridad. [21], [22], [23], [24], [25]

### 2.3.3. APLICACIONES DE UART

El protocolo UART es ampliamente utilizado en diversos dispositivos y aplicaciones electrónicas debido a su simplicidad y efectividad para enviar pequeñas cantidades de datos. Algunas de sus aplicaciones más destacadas incluyen:

- ❖ Receptores GPS: Utilizados para la comunicación de datos de ubicación.
- ❖ Módulos Bluetooth: facilitan la comunicación inalámbrica entre dispositivos.
- ❖ Aplicaciones basadas en RFID: para la lectura y transmisión de datos de identificación.



- ❖ Programación y depuración de microcontroladores: herramientas para cargar firmware y realizar pruebas.
- ❖ Sistemas de navegación y telemetría: en aplicaciones de largo alcance donde SPI e I2C no son opciones viables.

Estas aplicaciones resaltan la versatilidad de UART en la comunicación serie de datos en una variedad de entornos y dispositivos electrónicos. [24], [26]

## 2.4. INTERFAZ DE COMUNICACIÓN SPI

Además del UART, el localizador GPS también empleará el Serial Peripheral Interface (SPI) como otro interfaz de comunicación. A continuación, se describirá el funcionamiento del SPI, sus componentes, características y aplicaciones más comunes.

El SPI es un protocolo de comunicación síncrono que facilita la transferencia de datos entre un microcontrolador y dispositivos periféricos, como sensores, memorias EEPROM, tarjetas SD, pantallas LCD, entre otros. Este protocolo ofrece una manera eficiente de intercambiar datos en serie entre un dispositivo maestro (microcontrolador) y múltiples dispositivos esclavos mediante un conjunto de líneas de señal específicas. [27]

### 2.4.1. ELEMENTOS DEL SPI

- ❖ Maestro (Master): es el dispositivo principal que controla la comunicación y genera el reloj de sincronización.
- ❖ Esclavo (Slave): son los dispositivos periféricos controlados por el maestro. Puede haber uno o varios esclavos conectados al mismo bus SPI.
- ❖ Líneas de señal: a diferencia de UART, SPI es un protocolo síncrono. La sincronización y transmisión de datos se realizan mediante 4 señales concretas:
  - **SCK** (Serial Clock): es el reloj de sincronización. Con cada pulso de este reloj, se lee o envía un bit.
  - **MOSI** (Master Output Slave Input): transfiere datos desde el maestro al esclavo.
  - **MISO** (Master Input Slave Output): transfiere datos desde el esclavo al maestro.
  - **SS** (Slave Select): esta línea selecciona el esclavo con el cual el maestro desea comunicarse, permitiendo activar o desactivar un esclavo específico.

La secuencia de bits se envía de manera síncrona con los pulsos del reloj; es decir, con cada pulso, el dispositivo maestro envía un bit. La transmisión comienza cuando

el maestro baja la señal de selección de esclavo (SS) a cero, activando el dispositivo esclavo y leyendo el primer bit. [28], [29]

En la *Figura 17*, se muestra de forma gráfica los elementos que componen el SPI. Además, se observa como un único dispositivo maestro puede intercambiar datos con varios dispositivos esclavos.

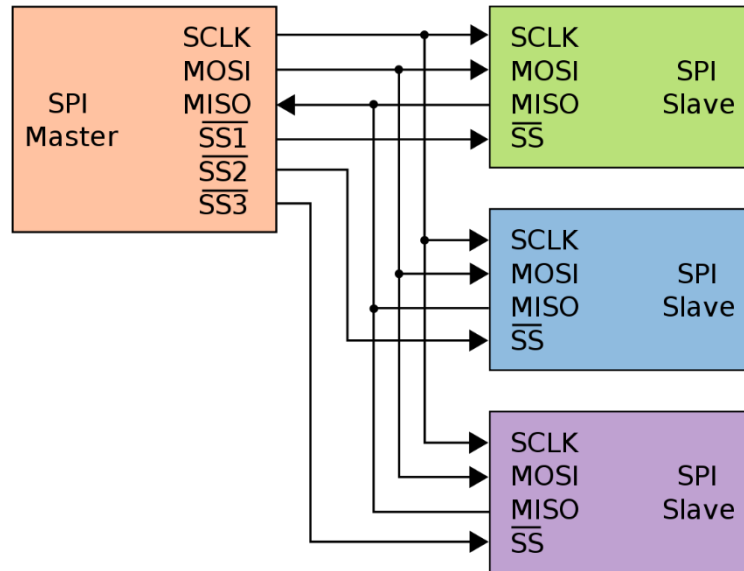


Figura 17. SPI bus: un maestro y tres esclavos (extraído de [28])

Cabe destacar que, en la actualidad los términos MISO/MOSI han sido sustituidos por CIPO (Controller In Peripheral Out) /COPI (Controller Out Peripheral In),y SS ha sido sustituido por el término CS (Chip Select). [30]

#### 2.4.2. CARACTERÍSTICAS DEL SPI

- ❖ Comunicación síncrona: la comunicación SPI utiliza un reloj compartido entre el maestro y los esclavos para sincronizar la transferencia de datos, lo que permite una alta velocidad de transferencia.
- ❖ Full-Duplex: el SPI admite comunicación full-duplex, lo que significa que los datos se pueden enviar y recibir simultáneamente. El maestro envía datos al esclavo a través de la línea MOSI, al mismo tiempo que el esclavo envía datos al maestro a través de la línea MISO.
- ❖ Configurabilidad: es un interfaz altamente configurable y flexible, permitiendo ajustes en parámetros como la velocidad de transferencia, el formato de los datos (MSB-first o LSB-first), y el modo de operación (polaridad y fase del reloj).
- ❖ Múltiples dispositivos esclavos: un único bus SPI puede admitir múltiples dispositivos esclavos (*Figura 17*), ya que cada esclavo tiene su propia línea de selección de esclavos (SS). El maestro activa una línea SS específica para comunicarse con un esclavo particular. [31]



### 2.4.3. APLICACIONES DE SPI:

El SPI es ampliamente utilizado en diversas aplicaciones debido a su alta velocidad y flexibilidad, tales como:

- ❖ Comunicación con tarjetas de memoria: como SD, microSD, y TF.
- ❖ Interacción con pantallas LCD y OLED: facilitando la transmisión rápida de datos gráficos.
- ❖ Control de sensores: incluyendo acelerómetros, giroscopios, y magnetómetros, entre otros. [29]



# CAPÍTULO 3: DESCRIPCIÓN DEL DISPOSITIVO A DESARROLLAR







### 3. DESCRIPCIÓN DEL DISPOSITIVO A DESARROLLAR

Una vez comprendidos los conceptos esenciales que sustentarán el funcionamiento del localizador objeto de este proyecto, en el presente apartado se pretende describir dicho dispositivo y cómo se van a integrar todas las herramientas mencionadas en el apartado 2 en su desarrollo.

#### 3.1. JUSTIFICACIÓN DEL DISEÑO

Tal y como se ha comentado anteriormente, el equipo que se pretende diseñar tiene su aplicación en aquellas personas que quieran registrar la información relacionada con las rutas que lleven a cabo al aire libre. Estas pueden ser realizadas haciendo deporte (andando, corriendo, montando en bicicleta) o incluso en un vehículo motorizado como una moto, un coche, etc.

Por lo tanto, su diseño debe caracterizarse por:

- ❖ Portabilidad y ergonomía: el dispositivo debe ser cómodo de llevar y utilizar durante las ya mencionadas actividades al aire libre. Por ello, su tamaño debe ser lo más reducido posible y su forma debe favorecer su portabilidad.
- ❖ Durabilidad y resistencia a condiciones externas: el dispositivo debe ser resistente al agua, polvo y golpes para soportar condiciones adversas durante las actividades al aire libre.
- ❖ Batería de larga duración: el dispositivo debe ser capaz de funcionar durante varias horas sin necesidad de recarga frecuente.
- ❖ Capacidad de almacenamiento suficiente: el dispositivo debe ser capaz de almacenar información relativa a múltiples rutas.
- ❖ Interfaz de usuario intuitiva y fácil de usar: la visualización de los parámetros y estadísticas debe ser fácil de acceder y manejar para cualquier usuario, independientemente de su condición y/o conocimientos.

#### 3.2. ANÁLISIS DE MERCADO

En este punto, se realiza una búsqueda de equipos ya fabricados existentes en el mercado que presenten especificaciones y funcionalidades similares a las del dispositivo a implementar. Esto proporcionará información sobre el tamaño del mercado, las necesidades y preferencias de los consumidores, la competencia existente y el precio óptimo del producto, todo ello relacionado con una hipotética salida al mercado del dispositivo.

#### ❖ ALLROUND Finder 2.0 de PAJ GPS



Figura 18. ALLROUND Finder 2.0 de PAJ GPS (extraído de [32])

Se trata de un dispositivo de seguimiento GPS que se utiliza para rastrear la ubicación de objetos, personas o mascotas. Permite seguir su ubicación en tiempo real, consultar la distancia y los trayectos realizados durante los últimos 365 días y la configuración de diversas alertas (alarma cuando el rastreador GPS se mueve, alerta cuando el rastreador GPS excede una cierta velocidad, etc.) No obstante, esto es posible gracias a que se conecta al móvil a través de una tarjeta SIM y utiliza su red para que el usuario pueda rastrear la ubicación del dispositivo en tiempo real mediante una aplicación de móvil específica. Cabe destacar que para poder utilizar esta aplicación hay que pagar una suscripción mensual de 6.99€ mensuales. [32]

#### Especificaciones técnicas:

- Precisión GPS: menos de 10 metros
- Batería interna: 3,7V / 5.000 mAh Litio-Ion
- Autonomía (duración de la batería): hasta 20 días de funcionamiento continuo
- RED: GSM 4 bandas
- Módulo GPS: Quectel MC25
- Resistencia al agua: sumergible hasta IP64
- Temperatura de funcionamiento: de -20 °C a 75 °C
- Humedad: 10% a 85%
- Dimensiones: 10,6 x 6,3 x 2.3 cm
- Peso: 166 gramos



- Precio (incl. IVA): 29,99€ (+ 6,99€ mensuales)

❖ **Tracker GPS OLIVER** de Oliver



Figura 19. Tracker GPS de Oliver (extraído de [33])

OLIVER es un mini GPS que, con ciencia deportiva y a través de una app, brinda información clave para medir rendimiento y prevenir lesiones. Es un dispositivo orientado sobre todo a fútbol pero puede extender su ámbito de uso a otros deportes. Entre las métricas que OLIVER recoge, analiza e interpreta se encuentran: métricas atléticas (tiempos de actividad, distancia recorrida, velocidad máxima, número de sprints, etc.), métricas de fútbol (mapa de calor e interacciones con el balón, fuerza de chute, etc.) y métricas de salud (p.ej.: monitoreo de cargas para medir el riesgo de lesión). Resulta interesante señalar que a partir del primer año de uso se debe pagar una suscripción mensual de 9,99€. [33]

Especificaciones técnicas:

- Compatible con iOS y Android
- Autonomía (duración de la batería): 6 horas
- Sensores: GPS 25 Hz y acelerómetro IMU 50 Hz
- Memoria: Flash Storage. Permite recoger datos de sesiones de hasta 4 horas.
- Conectividad: Bluetooth
- Dimensiones: 4 x 3 cm
- Peso: 15 gramos
- Precio: 149,99€ (+9,99€ mensuales a partir del segundo año de uso)



❖ Amazfit GTR 3 Pro de Amazfit



Figura 20. Amazfit GTR 3 Pro (extraído de [34])

Se trata de un dispositivo que combina las capacidades de un reloj inteligente con un avanzado sistema de seguimiento GPS. Este dispositivo no solo mide la distancia y velocidad durante las carreras, sino que también ofrece funcionalidades adicionales como el monitoreo del sueño, detección de estrés y diversas aplicaciones inteligentes. Su precio refleja la integración de múltiples tecnologías avanzadas y su diseño elegante y robusto. [34]

Especificaciones técnicas:

- Batería interna tipo Litio-ion
- Dimensiones: 4,6 x 4,6 x 1.07 cm
- Peso: 57 gramos
- Precio: 149.90€

❖ Reloj GPS Forerunner 165 de Garmin



Figura 21. Reloj Garmin Forerunner 165 de Garmin (extraído de [35])



Garmin es una de las principales marcas en el mundo del deporte que ofrecen dispositivos con GPS incorporado, principalmente relojes. En este caso se ha elegido el modelo más barato de todos los relojes GPS que ofrece este fabricante, porque se considera que es el que más se puede llegar a ajustar al localizador GPS de este trabajo. [35]

#### Especificaciones técnicas:

- Compatible con iOS y Android
- Autonomía (duración de la batería): hasta 11 días en el mejor de los casos. Hasta 17 horas si se hace uso del modo GNSS (recibe señales de distintos sistemas de navegación por satélite).
- Sensores que incorpora: GPS, monitor de frecuencia cardíaca, altímetro barométrico, brújula, acelerómetro, termómetro y sensor de luz ambiental
- Memoria: 4 GB
- Conectividad: Bluetooth y ANT+
- Resistencia al agua: sumergible hasta IP64
- Temperatura de funcionamiento: de -20 °C a 75 °C
- Humedad: 10% a 85%
- Dimensiones: 4,3 cm x 4,3 cm x 1,16 cm
- Peso: 39 gramos
- Precio (incl. IVA): 279,99€

A parte de las ya mencionadas, incluye un abrumador número de utilidades adicionales, como control de sueño, detección de estrés, funciones de seguimiento y seguridad o funciones inteligentes como previsión del tiempo.

Resumiendo, cada uno de estos productos destaca en diferentes áreas y ofrece un conjunto de características avanzadas que los convierten en opciones atractivas para aquellos usuarios que buscan soluciones de seguimiento GPS y monitorización de actividades deportivas. Su elevado precio se justifica por la integración de tecnologías punteras, investigación y desarrollo extensivos, además de disponer de capacidad de producción a gran escala.

Dadas las capacidades de los dispositivos analizados, se puede concluir que el localizador GPS desarrollado en este trabajo ofrece una propuesta única y diferenciada en el mercado. Aunque las prestaciones de los dispositivos comerciales puedan superar a las del prototipo, este constituye una solución asequible y eficaz



para aquellos usuarios que no requieran de características más complejas y costosas.

De este modo, con el prototipo a implementar se pretende demostrar la viabilidad técnica y funcional de un localizador GPS básico para el registro de rutas y actividades al aire libre, sirviendo como una base sólida para futuras mejoras y desarrollos.

### 3.3. ELEMENTOS QUE FORMAN EL DISPOSITIVO

Una vez concebido qué tipo de dispositivo se quiere implementar, que requisitos de diseño debe cumplir y después de haber examinado opciones similares presentes en el mercado, en este apartado se pretende describir cada uno de los elementos que conformarán el dispositivo final. Ésta descripción incluirá sus características y especificaciones técnicas, así como la justificación de su elección para dar solución al diseño por delante de otras alternativas.

Además, se ilustrará el esquema final de funcionamiento del localizador GPS, junto con la explicación de su modo de operación.

#### 3.3.1. MÓDULO GPS GY-GPS6MV2

Dada la naturaleza de este Trabajo de Fin de Grado, el primer elemento a seleccionar debe ser el sensor GPS responsable de capturar los datos asociados a la ubicación del usuario. Después de sopesar distintas alternativas disponibles en el mercado, tales como los receptores GPS SparqEE GPSv1.0, Quectel L80-MR9 o el Beitian BN880Q, se optó por utilizar el módulo GY-GPS6MV2 integrado con el chip NEO-M8N. Esta decisión se justifica por su superior precisión en la recogida de datos, mayor eficiencia energética y mejor relación calidad/precio en comparación con las opciones mencionadas.

Cabe destacar que también se consideraron versiones anteriores de la misma familia GY-GPSMV2, integrando los chips NEO-M6N y NEO-M7N. Sin embargo, debido a que su precio era muy similar al del NEO-M8N pero con características inferiores, no justificaban el ahorro. [36]

En definitiva, el GY-GPS6MV2 es un dispositivo que integra un receptor GPS, en este caso el circuito integrado NEO-M8N, y una antena en un único módulo. En la *Figura 22*, se pueden apreciar los pines de los que se compone:

- ❖ **VCC:** alimentación del dispositivo a 3,3V o 5V
- ❖ **GND:** conexión a masa del dispositivo
- ❖ **RX:** línea para la recepción de datos por parte del Arduino Nano Every, mediante el protocolo UART (ver apartado 2.3)
- ❖ **TX:** línea para la transmisión de datos al Arduino Nano Every, mediante el protocolo UART (ver apartado 2.3)



Figura 22. Módulo GPS GY-GPS6MV2 (extraído de [37])

Por su parte, el circuito integrado NEO-M8N (Figura 23) pertenece al fabricante Ublox y ofrece una variedad de características avanzadas, entre las que destacan el soporte para el sistema global de navegación por satélite GNSS, alta sensibilidad, baja potencia de consumo y capacidad de obtener una señal de posicionamiento precisa incluso en entornos desafiantes, como en áreas urbanas. [38]



Figura 23. Chip GPS NEO-M8N (extraído de [39])

Las especificaciones técnicas que ofrece el circuito integrado NEO-M8N se muestran en la *Tabla 1*:





Tabla 1. Especificaciones técnicas del chip GPS NEO-M8N (elaboración propia)

Especificaciones técnicas	
Modelo	NEO-M8N
Categoría	Standard Precision
Global Navigation Satellite System (GNSS)	GPS/QZSS
	GLONASS
	Galileo
	BeiDou
Alimentación	2.7V - 3.6 V
Interfaces	UART
	USB
	SPI
	DDC (I <sup>2</sup> Compliant)
Funciones	Programable (flash)
	Almacenamiento de datos
Ámbito de aplicación	Profesional

El módulo GY-GPS6MV2 es ampliamente utilizado en aplicaciones como sistemas de navegación para vehículos, drones, sistemas de seguimiento de objetos, dispositivos de localización personal, entre otros. Además, cuenta con el interfaz UART, cuyo funcionamiento ya se explicó anteriormente. Esto permite su integración con microcontroladores y sistemas embebidos, como en este caso, por lo que es una opción perfecta para las necesidades de este proyecto. [39]

#### 3.3.2. ARDUINO NANO EVERY

A la hora de elegir la placa de Arduino que iba a ser la responsable de procesar los datos recibidos por el receptor GPS, se presentaba un abanico de posibilidades inmenso. Es bien conocido que Arduino tiene en su catálogo varias familias de dispositivos, cada una de la cuáles está formada, a su vez, por numerosos modelos.

Por esta razón, para tomar la decisión se han tenido en cuenta los requisitos de diseño ya mencionados en el apartado 3.1. La familia de Arduino que mejor que se ajustaba a estos, sobre todo en aspectos de tamaño y eficiencia, fue la nombrada como familia “Nano”.

En particular, el modelo seleccionado es la placa Arduino Nano Every (Figura 24). Esta se trata de una versión mejorada de la clásica Arduino Nano, diseñada para proyectos más avanzados y con mayores requerimientos de memoria y capacidad de procesamiento.



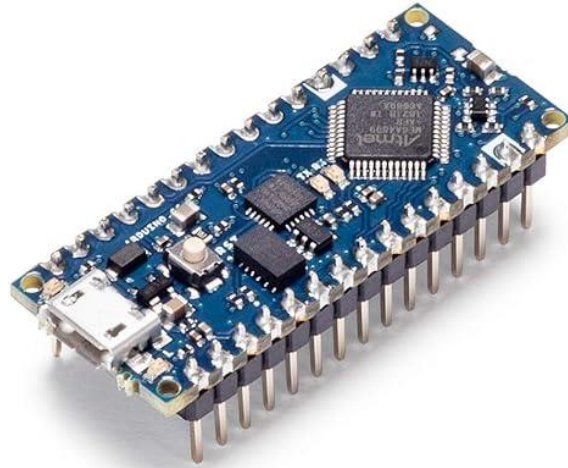


Figura 24. Placa Arduino Nano Every (extraído de [40])

Las bondades de este circuito integrado que han decantado su elección se describen a continuación:

1. Tamaño compacto y portabilidad: tiene unas dimensiones de 45 mm x 18 mm, haciéndola extremadamente compacta y adecuada para proyectos portátiles. Como ya se comentó en el apartado 3.1, un tamaño lo más pequeño posible es característica inherente a la creación de un buen localizador GPS. Esto permite que el dispositivo sea fácilmente transportable y discreto, ideal para la realización de actividades al aire libre como caminar, correr o montar en bicicleta.
2. Eficiencia energética: el microcontrolador ATmega4809 que lleva incorporado presenta una elevada eficiencia energética, lo que es fundamental para un equipo portátil que funcionará con batería portátil, como es este caso. Esto garantiza una mayor duración de la batería, permitiendo al usuario registrar rutas durante períodos prolongados sin necesidad de recargas frecuentes.
3. Interfaz UART: la presencia de una interfaz UART (ver apartado 2.3) es crucial para este proyecto, ya que el módulo GPS utilizado (Figura 22) envía los datos que capta de los satélites a través de UART. De esta forma, el Arduino Nano Every puede recibir estos datos en tiempo real, procesarlos y convertirlos en información útil y legible para el usuario, como coordenadas de latitud, longitud, altitud, fecha y hora.
4. Interfaz SPI: este interfaz (ver apartado 2.4) permite la conexión de una tarjeta de memoria microSD a la placa para el almacenamiento de los datos. De hecho, la alta velocidad de comunicación del SPI (en comparación con I2C o SMBus) asegura que los datos de las rutas se puedan escribir rápidamente en la tarjeta, garantizando un registro eficiente y sin pérdidas. Esto es vital para almacenar la información relativa a rutas y certificar su integridad.



5. Flexibilidad y escalabilidad: con 22 pines digitales I/O y 8 entradas analógicas, el Arduino Nano Every, es capaz de soportar tanto el módulo GPS como el lector de tarjetas microSD holgadamente. De hecho, ofrece la posibilidad de agregar en un futuro sensores y/o periféricos adicionales (p.ej.: termómetros, sensores de humedad, etc.) que aumenten las funcionalidades del dispositivo y permita extender su ámbito de aplicación.
6. Precio: se trata de la placa de Arduino perteneciente a la familia Nano más barata con un precio en la tienda oficial de 12,50€ (sin IVA). [40]

En la *Tabla 2* se ofrece un resumen de los elementos y funcionalidades que presenta esta placa.

Tabla 2. Especificaciones técnicas del Arduino Nano Every (elaboración propia)

Especificaciones técnicas		
Modelo	Nano Every	
Microcontrolador	ATMega4809	
Memoria	48 KB Flash	
	6 KB SRAM	
	256 bytes EEPROM	
Arquitectura	AVR	
Alimentación	Voltaje de funcionamiento	5V
	Voltaje de entrada nominal	7-21V
	Corriente DC por cada pin I/O	15 mA
Pines	Nº de pines de entrada/salida digitales	22
	Nº de pines de entrada analógicos	8
	Pines PWM	5
Interfaces de comunicación	UART	
	I2C	
	SPI	
Velocidad de reloj	16 MHz	
Tamaño	18 x 45 mm	
Peso	5 g	

El número y tipo de pines de los que dispone esta placa se muestran en la *Figura 25*.

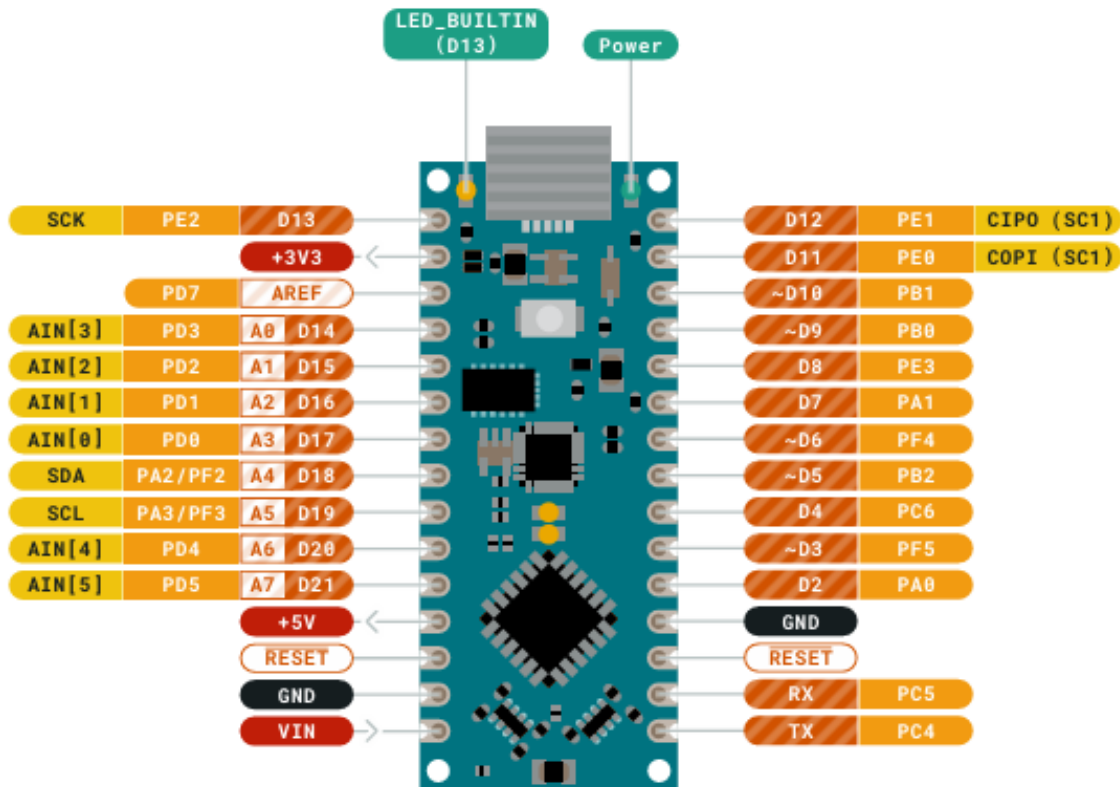


Figura 25. Pinout del Arduino Nano Every (extraído de [40])

Los pines que se van a utilizar y su conexión con el resto de periféricos se explicará en 3.5.

### 3.3.3. MÓDULO LECTOR TARJETAS MICROSD

Como ya se explicó con anterioridad, se pretende que la información procesada por el microcontrolador se almacene en un lugar de memoria para su posterior visualización.

De nuevo, las opciones de almacenamiento eran varias, desde utilizar la memoria EEPROM o la memoria Flash externa ambas integradas en el propio Arduino Nano, hasta dispositivo de almacenamiento USB, pasando por algún método de almacenamiento en la nube.

Dado que el modelo de Arduino escogido (*Figura 24*) no dispone de acceso a internet, la opción de almacenar los datos en la nube quedó descartada.

Por otro lado, el tamaño de un USB no se adecúa a las necesidades del proyecto, por lo que también se desecha esta posibilidad.

Por último, utilizar las memorias internas del Arduino Nano (EEPROM o Flash), podría suponer una alternativa ideal pues no supondría añadir ningún periférico extra, reduciendo tamaño y coste. Además, los datos que se almacenen en cualquiera de estas memorias son persistentes, lo que significa que se mantienen incluso cuando el dispositivo está apagado, que es justo lo que se necesita para esta aplicación. De



esta forma, el usuario realiza la ruta, los datos se almacenan en la memoria y una vez finalizada, el dispositivo se apaga y éstos se extraen para proceder a su tratamiento y visualización.

No obstante, si se realizan cálculos acerca de la cantidad de información a almacenar se obtiene que la capacidad de estas memorias es insuficiente.

Primero de todo, hay que considerar el número y tipo de datos que se pretenden almacenar. Estos se muestran en la *Tabla 3*:

Tabla 3. Información a almacenar (elaboración propia)

Información a almacenar				
Dato	Tipo	Tamaño		
		Bytes	Bits	
Coordenadas	Latitud	float	4	32
	Longitud	float	4	32
	Altitud (m)	float	4	32
Parámetros	Distancia (km)	float	4	32
	Distancia total (km)	float	4	32
	Tiempo (s)	uint16_t	2	16
	Tiempo total (s)	uint16_t	2	16
	Velocidad (km/h)	float	4	32
	Ritmo (min/km)	float	4	32
	Pendiente (%)	float	4	32
	Pendiente Absoluta (%)	float	4	32
Fecha	Día	uint8_t	1	8
	Mes	uint8_t	1	8
	Año	uint16_t	2	16
Hora	Hora	uint8_t	1	8
	Minuto	uint8_t	1	8
	Segundo	uint8_t	1	8
			<b>47</b>	<b>376</b>

Segundo, teniendo en cuenta que las memorias EEPROM y Flash del Arduino Nano Every son de 256 B y 48 KB, respectivamente, (según la *Tabla 2*), se realizan los siguientes cálculos (ver (E-1), (E-2), (E-3), (E-4)) para conocer durante cuánto tiempo se podría almacenar información:



**Memoria EEPROM:**

$$\begin{aligned} \text{N}^\circ \text{ líneas de información} &= 256 \text{ bytes} \cdot \frac{1 \text{ línea de información}}{47 \text{ bytes}} \\ &= 5,45 \text{ líneas de información} \approx 5 \text{ líneas de información} \end{aligned} \quad (\text{E-1})$$

Suponiendo que se almacena 1 línea de información completa cada 10 segundos:

$$\begin{aligned} \text{Tiempo total} &= 5 \text{ líneas de información} \cdot \frac{10 \text{ segundos}}{1 \text{ línea de información}} \\ &= 50 \text{ s} \end{aligned} \quad (\text{E-2})$$

**Memoria Flash:**

$$\begin{aligned} \text{N}^\circ \text{ líneas de información} &= 48.000 \text{ bytes} \cdot \frac{1 \text{ línea de información}}{47 \text{ bytes}} \\ &= 1021,28 \text{ líneas de información} \\ &\approx 1021 \text{ líneas de información} \end{aligned} \quad (\text{E-3})$$

Suponiendo que se almacena 1 línea de información completa cada 10 segundos:

$$\begin{aligned} \text{Tiempo total} &= 1021 \text{ líneas de información} \cdot \frac{10 \text{ segundos}}{1 \text{ línea de información}} \\ &= 10210 \text{ s} \approx 170 \text{ min} < 3 \text{ horas} \end{aligned} \quad (\text{E-4})$$

Por lo tanto, la opción de utilizar la memoria EEPROM del Arduino Nano queda totalmente descartada, ya que sólo se podría almacenar información durante menos de un minuto.

Por su parte, la memoria Flash ofrece claramente mejores prestaciones en ese sentido que la EEPROM, albergando capacidad de almacenamiento para casi 3 horas. Aun así, se considera insuficiente puesto que limita mucho el rango de aplicación del localizador GPS. Por ejemplo, para una persona que dese realizar un recorrido largo que implique una mañana entera de registro de datos, utilizar la memoria Flash del Arduino Nano como célula de memoria no sería una opción válida.

En definitiva, suponiendo que cómo mínimo, se quiere ofrecer 8 horas de almacenamiento de datos y que se almacena 1 línea de información completa cada 10 segundos, la capacidad de la memoria deberá de ser de al menos la calculada a través de las ecuaciones (E-5) y (E-6):

$$\begin{aligned} 8 \text{ h} &= 480 \text{ min} = 28800 \text{ seg} \cdot \frac{1 \text{ línea de información}}{10 \text{ seg}} \\ &= 2880 \text{ líneas de información} \end{aligned} \quad (\text{E-5})$$

$$2880 \text{ líneas de información} * \frac{47 \text{ bytes}}{1 \text{ línea de información}} = 135360 \text{ bytes} \quad (\text{E-6})$$
$$= 135,36 \text{ KB} = 0,00013536 \text{ GB}$$

Por esta razón, finalmente se ha optado por utilizar una tarjeta microSD de 8 GB (Figura 26), que, para esta aplicación, proporciona una capacidad de memoria prácticamente ilimitada. Adicionalmente, esta alternativa ofrece la posibilidad de ampliar la capacidad fácilmente, existiendo tarjetas microSD de 32 GB, 64 GB, etc.



Figura 26. Tarjeta de memoria microSD (extraído de [41])

Cabe destacar, que para utilizar la tarjeta microSD es necesario un módulo lector de tarjetas microSD (Figura 27). Este se comunica con la placa de Arduino a través del interfaz SPI (ver apartado 2.4) y, a través de las bibliotecas *SPI.h* y *SD.h*, de Arduino IDE, permite abrir, leer, escribir, crear y eliminar archivos en la tarjeta.



Figura 27. Módulo lector de tarjetas microSD (extraído de [41])

Los pines de los que dispone el módulo lector de tarjetas microSD se muestra en la Figura 27:

- ❖ **VCC:** alimentación del dispositivo a 3,3V o 5V
- ❖ **GND:** conexión a masa del dispositivo
- ❖ **MOSI, MISO, CS y CLK:** ver apartado 2.4.1.



#### 3.3.4. BATERÍA LIPO 3.7 V / 1000 mAh

Para la alimentación del dispositivo, se ha optado por utilizar una batería de polímero de litio (o batería LiPo) al ser un tipo de batería recargable caracterizada por su reducido tamaño, buena capacidad y alta seguridad, al soportar altos picos de corriente. Todo esto hace que esta opción se presente como una fuente de alimentación muy eficiente, siendo ideal para las necesidades de este proyecto.

En concreto, el modelo utilizado es el 603050, que funciona a 3,7V y cuya capacidad es de 1000 mAh (*Figura 28*):



*Figura 28. Batería LiPo modelo 603050 (extraído de [42])*

#### 3.3.5. CARGADOR LIPO

Después de la elección de la alimentación, se presentan dos vicisitudes. Por un lado, el hecho de alimentar la placa Arduino con 5V, hace insuficientes los 3,7V que ofrece la batería, siendo necesario un convertidor DC-DC que eleve dicha tensión.

Por otra parte, ya se ha comentado anteriormente que una de las ventajas de utilizar una batería LiPo residía en que esta era recargable. Sin embargo, se trata de un tipo de batería especial que requiere de un cargador específico. [42], [43]

Este debe proporcionar una corriente constante durante el ciclo de carga pero también debe ajustarla en función del voltaje de la celda que detecta durante el proceso, tal y como se muestra en la siguiente gráfica (*Figura 29*):

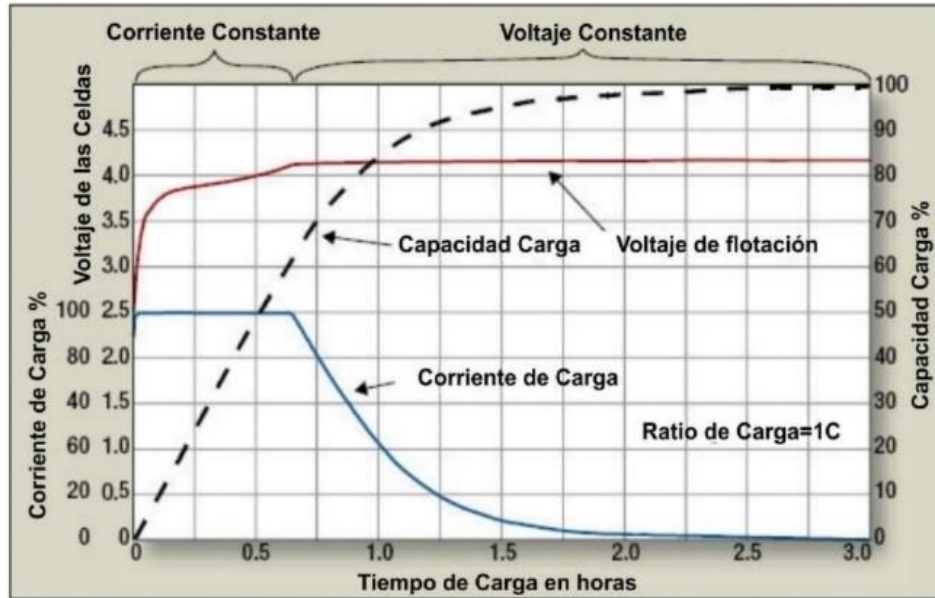


Figura 29. Curva de carga de una batería LiPo (extraído de [44])

Como solución a estos “inconvenientes”, se optó por utilizar un elemento que fuese un convertidor DC-DC elevador y cargador de baterías LiPo simultáneamente.

En concreto, el cargador LiPo elegido es el modelo PowerBoost 500C del fabricante Adafruit. Este se trata de un módulo que se conecta siempre entre la batería LiPo y la placa Arduino y cuyo funcionamiento se describe a continuación:

- ❖ Cuando la batería se encuentra cargada, el módulo alimenta la placa Arduino desde la batería, elevando el voltaje de 3,7V a 5V.

A modo de apunte, resulta interesante resaltar que en realidad eleva hasta 5.2V, dejando un poco de “margen” para cables largos, momentos puntuales de demanda alta, o la adición de algún pequeño componente a la salida como un diodo.

- ❖ Cuando la batería se descarga, existe la posibilidad de conectar el módulo a un cargador Micro-USB (enchufado a la red eléctrica), a partir del cual se alimenta la placa Arduino a la vez que se carga la batería, de modo similar a cómo se carga la batería del móvil.

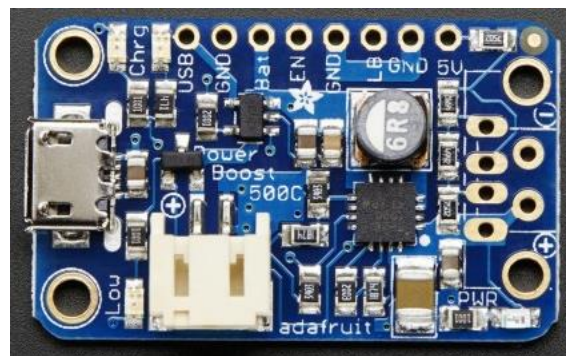


Figura 30. Cargador LiPo PowerBoost 500C (extraído de [45])



Para controlar el encendido y apagado del dispositivo se va a utilizar un interruptor de tipo *Rocker* (ver *Figura 31*), cuyos terminales irán soldados a los pines EN (enable) y GND (ground) del cargador LiPo [45]



*Figura 31. Interruptor tipo Rocker (extraído de [46])*

Una vez seleccionados los elementos que van a constituir el localizador GPS, a continuación se procede a explicar cómo se reparten los consumos de corriente según la configuración propuesta.

Antes que nada, es de vital importancia saber cuánta corriente necesitan para funcionar tanto el Arduino Nano Every como los módulos GPS y lector de tarjetas microSD. Después de realizar mediciones en el laboratorio y de buscar información entre sus hojas de características, se han estimado los siguientes consumos de corriente (ver *Tabla 4*):

*Tabla 4. Reparto de consumos estimados (elaboración propia)*

Componente	Consumo de corriente (mA)	
	Reposo	Carga máxima
Arduino Nano Every	15	50
Módulo GPS	3	21
Lector microSD	10	100
	28	171

Como se puede observar, se han diferenciado dos situaciones: en reposo y en situación de carga máxima. La primera se refiere a cuando el dispositivo no está realizando operaciones intensivas y sólo mantiene las funciones básicas, mientras que la segunda se da cuando todos los componentes están operando a plena capacidad. Esto es cuando el módulo GPS está funcionando en modo continuo, actualizando su posición constantemente, la placa Arduino está procesando datos, ejecutando cálculos, gestionando la comunicación con sus periféricos asociados y



cuando se está escribiendo información en un archivo de texto de la tarjeta microSD, mediante el módulo lector.

Dado que normalmente el dispositivo va a estar en situación de carga máxima y además es el caso menos favorable, se va a considerar que el consumo total de corriente del dispositivo es de 171 mA, aproximadamente.

Por su parte, el cargador LiPo utilizado (*Figura 30*), puede proporcionar 500 mA continuos y picos de hasta 1 A, mientras que la batería puede llegar a suministrar hasta 2 A de corriente de descarga máxima (acorde con sus hojas de características), por lo que la alimentación elegida sí es capaz de cumplir con la demanda más alta del dispositivo. Ahora bien, es fundamental saber durante cuánto tiempo, ya que esto determinará la autonomía del localizador GPS.

La corriente que suministra el cargador LiPo depende de la demanda de la carga conectada a su salida hasta un máximo de 500 mA continuos. Si la carga requiere menos de 500 mA, el cargador proporcionará únicamente la corriente necesaria para satisfacer esa demanda. En este caso, sucede justamente eso, ya que la demanda total de corriente es de 171 mA. Sin embargo, hay que tener en cuenta que la eficiencia del cargador es del 90%, según su hoja de características, por lo que, la corriente que suministra el cargador Powerboost 500C desde la batería LiPo para alimentar el Arduino y sus periféricos es de 190 mA, tal y como se muestra en la ecuación (E-7),

$$I_{entrada} = \frac{I_{salida}}{Eficiencia} = \frac{171 \text{ mA}}{0,9} = 190 \text{ mA} \quad (E-7)$$

Donde:

**$I_{entrada}$ :** corriente suministrada por el cargador LiPo.

**$I_{salida}$ :** corriente que le llega a la placa Arduino.

Teniendo en cuenta que la batería utilizada (*Figura 28*) posee una capacidad de 1000 mAh, la duración de la batería se obtiene a partir de la ecuación (E-8):

$$Duración \text{ batería} = \frac{Capacidad}{Corriente \text{ demandada}} = \frac{1000 \text{ mAh}}{190 \text{ mA}} \approx 5,26 \text{ horas} \quad (E-8)$$

Esta estimación es válida bajo el supuesto de que la eficiencia del sistema se mantiene y el consumo de corriente es constante. Esto último no siempre va a ser así, ya que habrá momentos en los que algunos o todos los componentes se encuentren en estado de reposo y el consumo total de corriente será menor. En otras palabras, la duración de 5,26 horas se da para el peor de los casos en el que todos los componentes operan en carga máxima de manera continua, por lo que generalmente la batería durará más.



Por lo tanto, se puede concluir que el tipo de alimentación escogida se adecúa perfectamente a las necesidades del proyecto.

### 3.4. MODO DE OPERACIÓN DEL DISPOSITIVO

A continuación, se pretende explicar cómo funciona el localizador GPS y qué cómo se pretende procesar, almacenar y visualizar la información registrada.

#### 3.4.1. OBTENCIÓN DE DATOS EN FORMATO NMEA

El primer punto de operación del dispositivo es que el chip NEO-M8N (*Figura 23*) del módulo GPS GY-GPS6MV2 (*Figura 22*) se conecte a la red satelital GNSS y empiece a recibir los datos asociados a la ubicación del usuario en tiempo real. Como se comentó en 2.1.5, el receptor GPS devuelve líneas de datos en formato NMEA, por lo que es necesario procesarlas y convertirlas a un formato comprensible.

#### 3.4.2. PROCESADO Y CONVERSIÓN DE DATOS

Para realizar esta tarea, se utiliza el Arduino Nano Every junto con el entorno de desarrollo Arduino IDE y las librerías pertinentes. De este modo, se ha llevado a cabo un pequeño código en el que, entre otras cosas, se procesa y convierte la información recibida del receptor GPS (*Figura 32*):

```
// Inclusión de librerías a utilizar
#include <SoftwareSerial.h> // Librería para utilizar "SoftwareSerial"
#include <TinyGPSPlus.h>    // Librería para utilizar del módulo GPS

// Declaración de constantes
static const uint32_t SERIAL_VB = 115200; // Velocidad de transmisión
(baudios/segundo) para la comunicación serial
static const uint32_t GPS_VB = 9600; // Velocidad de transmisión
(baudios/segundo) del módulo GPS
static const int RX_PIN = 4, TX_PIN = 3; // Pines del Arduino Nano
asignados para la recepción y transmisión de datos vía UART

// Declaración de variables globales
uint8_t hora = 0, minutos = 0, segundos = 0, dia = 0, mes = 0;
uint16_t anio = 0;
uint32_t tiempo_actual= 0;
float latitud = 0, longitud = 0, altitud = 0;

/ Declaración de objetos
TinyGPSPlus mi_GPS; // Se declara el objeto "mi_GPS" de la librería
TinyGPSPlus
SoftwareSerial Serial_GPS(RX_PIN, TX_PIN); // Se crea el puerto serie
"Serial_GPS" Se asignan los pines de recepción (D4 - RX_PIN) y
transmisión (D3 - TX_PIN) del Arduino Nano
```



```
/ Función para inicializar el dispositivo
void inicializar_dispositivo(){
    Serial.begin(SERIAL_VB); // Inicio del puerto serie
    Serial_GPS.begin(GPS_VB); // Inicio del puerto serie del módulo GPS
    while(!Serial); // Espera hasta que la comunicación serial esté lista
}

// Función para comprobar si el GPS está enviando datos y si existe
conexión con la placa Arduino
void comprobar_GPS(){
    while (Serial_GPS.available() > 0) { // Se comprueba continuamente si
hay datos disponibles para ser leídos desde el puerto serial del GPS
        if (mi_GPS.encode(Serial_GPS.read())) { // Si hay datos
disponibles, el programa lee un byte del puerto serial del GPS y luego lo
decodifica
            }
        }
    }

    if (millis() > 10000 && mi_GPS.charsProcessed() < 10) {
        Serial.println(F("ERROR: No se ha detectado módulo GPS. Compruebe
cableado."));
        while (true); // Si no se detecta el módulo GPS, el programa se
detiene aquí
    }
}

// Función para actualizar la información extraída del GPS y los sensores
y almacenarla en variables
void actualizar_info(){
    if (mi_GPS.location.isValid()) {
        latitud = redondear_dec(mi_GPS.location.lat(), 6);
        longitud = redondear_dec(mi_GPS.location.lng(), 6);
    }
    if (mi_GPS.altitude.isValid()) {
        altitud = mi_GPS.altitude.meters();
    }
    if (mi_GPS.time.isValid()) {
        hora = (mi_GPS.time.hour());
        minutos = mi_GPS.time.minute();
        segundos = mi_GPS.time.second();
    }
    if (mi_GPS.date.isValid()) {
        dia = mi_GPS.date.day();
        mes = mi_GPS.date.month();
        anio = mi_GPS.date.year();
    }
}
```



```
}  
}  
// Función de inicialización del programa  
void setup() {  
  inicializar_dipositivo();  
}  
  
// Función principal del programa  
void loop() {  
  comprobar_GPS();  
  actualizar_info();  
}
```

Figura 32. Código para el procesado y conversión de datos recibidos del módulo GPS en Arduino IDE (elaboración propia)

Este fragmento de código pertenece al sketch de Arduino *transformarDatos.ino* mostrado en el ANEXO V. PROGRAMAS DESARROLLADOS.

### 3.4.3. CÁLCULO DE PARÁMETROS DE RUTAS

Después de transformar los datos en bruto recibidos del receptor GPS y almacenarlos en variables, es posible calcular distintos parámetros asociados a las rutas realizadas como la distancia, la velocidad, el ritmo, o la pendiente a partir de los valores de latitud, longitud, altitud, fecha y hora. A continuación, se explica cómo.

#### Distancia entre dos puntos conocidas su latitud y longitud. Fórmula de Haversine

Uno de los parámetros de las rutas realizadas por el usuario que se pretende almacenar es la distancia recorrida. El problema surge a raíz de que, como ya se ha comentado anteriormente, el receptor GPS sólo envía la latitud, longitud, hora y fecha. Por esta razón, se buscó una manera de, a partir de estos datos, calcular la distancia.

En principio, calcular la distancia sobre un plano entre dos puntos debería ser relativamente sencillo. No obstante, en este caso los puntos se ubican sobre una superficie curva (la esfera terrestre), por lo que éste cálculo no es trivial.

Es aquí donde entra en juego la fórmula del semiverseno, más conocida como Fórmula de Haversine (E-9).

$$d = 2 \cdot R \cdot \arcsen \left( \sqrt{\sen^2 \left( \frac{\phi_2 - \phi_1}{2} \right) + \cos(\phi_1) \cdot \cos(\phi_2) \cdot \sen^2 \left( \frac{\lambda_2 - \lambda_1}{2} \right)} \right) \quad (\text{E-9})$$



Donde:

**d**: distancia ortodrómica o arco del círculo máximo que une dos puntos de una esfera en kilómetros En otras palabras, camino más corto que existe entre dos puntos ubicados en una esfera.

**R**: radio de la esfera en kilómetros

$\phi_1$ : latitud del punto 1 en grados

$\phi_2$ : latitud del punto 2 en grados

$\lambda_1$ : longitud del punto 1 en grados

$\lambda_2$ : longitud del punto 2 en grados

Esta constituye una ecuación fundamental para la navegación astronómica, ya que permite calcular la distancia más corta entre dos puntos en la superficie de una esfera, como la Tierra, usando sus coordenadas de latitud y longitud (*Figura 33*). A esta distancia también se le conoce con el nombre de distancia ortodrómica. [47]

Resulta interesante añadir que la fórmula de Haversine es un caso especial de una fórmula más general de trigonometría esférica, la ley de los semiversenos, que relaciona los lados y ángulos de los “triángulos esféricos”. [48]

A nivel anecdótico, la función semiverseno es una función trigonométrica que se define con la ecuación (E-10):

$$\text{semiversen}(\theta) = \frac{1 - \cos(\theta)}{2} = \text{sen}^2\left(\frac{\theta}{2}\right) \quad (\text{E-10})$$

De esta forma, y, utilizando (E-9) en el contexto de este trabajo, es posible calcular la distancia que va recorriendo el usuario a lo largo de su ruta. Basta con aplicar esta ecuación (E-9) e ir calculando recurrentemente la distancia entre ubicaciones sucesivas. Es decir, el módulo GPS envía la ubicación (coordenadas de latitud y longitud) del usuario a la placa de Arduino cada 10 segundos, y ésta calcula la distancia que hay entre la ubicación actual y la anterior.





Figura 33. Distancia ortodrómica entre dos puntos situados sobre una esfera (elaboración propia)

### Velocidad

Otro de los parámetros a registrar asociados a la ruta realizada por el usuario es la velocidad que éste ha alcanzado a lo largo del recorrido. Conociendo esto, será posible identificar la velocidad máxima y mínima, así como calcular la velocidad media de la sesión.

Para obtener la velocidad del usuario se presentan dos opciones. Por una parte, el propio módulo GPS GY-GPS6MV2 proporciona información acerca de su velocidad de manera continua. Sin embargo, se trata de información muy poco precisa, puesto que se hicieron pruebas con el módulo GPS estático y devolvía velocidades de entre 0 y 5 kilómetros por hora. Por esta razón, finalmente se escogerá la segunda opción. Esta consiste en, a partir de la distancia calculada entre dos puntos y el tiempo transcurrido, calcular la velocidad que ha llevado el usuario entre ubicaciones sucesivas. Para ello, se utilizará la ecuación básica de cinemática (E-11):



$$v = \frac{d}{t} \quad (E-11)$$

Donde:

**v:** velocidad en kilómetros por hora (km/h)

**d:** distancia en kilómetros (km)

**t:** tiempo en horas (h)

### Ritmo

El ritmo se refiere al tiempo que tarda una persona en completar una unidad de distancia, generalmente expresado en minutos por kilómetro (min/km). En otras palabras, es el tiempo necesario para cubrir una distancia específica. Por ejemplo, un ritmo de 5 min/km significa que se tarda 5 minutos en recorrer un kilómetro.

Es una medida comúnmente utilizada en actividades como correr, caminar o montar en bicicleta, ya que ayuda a los deportistas a evaluar su progreso y a alcanzar metas personales, como completar una maratón en un tiempo determinado.

Para calcular el ritmo de una ruta se divide el tiempo total empleado en minutos por la distancia total recorrida en kilómetros (E-12).

$$\text{Ritmo} = \frac{\text{Tiempo total}}{\text{Distancia total}} \quad (E-12)$$

Por ejemplo, si se recorren 12 kilómetros en 72 minutos, el ritmo sería de 6 min/km.

### Pendiente

Otro de los parámetros que se ha decidido que resulta interesante obtener del recorrido realizado por el usuario es el de la pendiente.

La pendiente constituye una medida de la inclinación del terreno en relación con un punto de referencia. Esta generalmente se expresa como una relación entre la diferencia de altitud y la distancia horizontal recorrida.

En el contexto de un recorrido o una ruta, la pendiente positiva indica que se está subiendo, es decir, ascendiendo hacia terreno más alto. Por otro lado, la pendiente negativa indica que un descenso, esto es, un movimiento hacia terreno más bajo.

Esto es importante en actividades como el senderismo o el ciclismo, donde comprender la pendiente del terreno puede ayudar a planificar rutas, estimar el esfuerzo requerido y entender mejor las características del terreno.

Por ejemplo, si se está realizando un recorrido en bicicleta y se aprecia una pendiente positiva, se trata de un ascenso y es posible que sea necesario ajustar la velocidad o esfuerzo para mantener el ritmo. Del mismo modo, una pendiente negativa





indicaría que se está descendiendo y es posible aumentar la velocidad con seguridad.

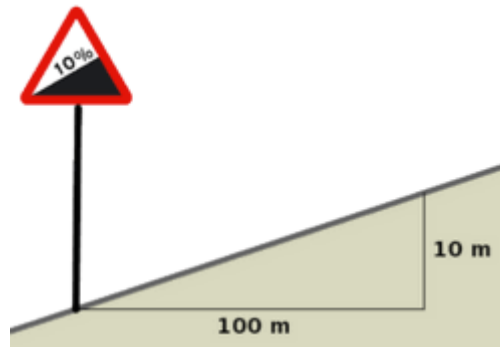


Figura 34. Pendiente entre dos puntos (extraído de [49])

En este caso, para calcular la pendiente entre un punto y el anterior se utiliza la ecuación (E-13):

$$\text{Pendiente (\%)} = \frac{h_{\text{punto2}} - h_{\text{punto1}}}{d} \cdot 100 \quad (\text{E-13})$$

Donde:

$h_{\text{punto2}}$ : es la altura del punto actual en metros

$h_{\text{punto1}}$ : es la altura del punto anterior en metros

$d$ : es la distancia horizontal que hay entre ambos puntos en metros

En resumen, la noción de pendiente positiva y negativa es relevante y útil al hablar de un recorrido o una ruta realizada, ya que proporciona información importante sobre la topografía y la dificultad del terreno.

Consecuentemente, en este caso se pretende conocer la pendiente del recorrido, calculando continuamente la pendiente que existe entre un punto y el anterior. De este modo, se efectúa la diferencia de altitud y se divide entre la distancia que separa ambos puntos. El resultado se multiplica por cien, obteniendo finalmente un valor de la pendiente en tanto por ciento.

#### Calorías quemadas

Por último, me gustaría mencionar que durante el desarrollo de este Trabajo de Fin de Grado, consideré la posibilidad de implementar un algoritmo para calcular las calorías quemadas durante las actividades deportivas realizadas, basándome en datos como el peso, la altura del usuario, la distancia recorrida y la velocidad. Sin embargo, tras investigar más en profundidad y revisar documentación técnica sobre dispositivos GPS comerciales, llegué a la conclusión de que esta aproximación sería poco fiable sin la incorporación de un sensor de frecuencia cardíaca.



La frecuencia cardíaca, es un indicador directo de la intensidad del esfuerzo físico. Sin datos de frecuencia cardíaca, cualquier estimación de calorías quemadas es propensa a ser inexacta porque no tiene en cuenta cómo responde el cuerpo del usuario a la actividad física. La frecuencia cardíaca refleja el nivel de esfuerzo y la energía consumida por el organismo en tiempo real, proporcionando una medida mucho más precisa del gasto calórico.

Por lo tanto, aquellos algoritmos que no contengan datos de frecuencia cardíaca pueden subestimar o sobrestimar significativamente el número de calorías quemadas, especialmente en actividades de intensidad variable o en individuos con diferentes niveles de condición física. Por ejemplo, dos personas con el mismo peso y altura pueden quemar un número muy diferente de calorías realizando la misma actividad, dependiendo de su nivel de esfuerzo y su ritmo cardíaco.

En resumen, la decisión de no implementar un algoritmo de cálculo de calorías quemadas en el dispositivo se basa en el hecho de que, sin un sensor de frecuencia cardíaca, las estimaciones serían muy poco fiables. El cálculo del gasto calórico real es muy complejo y está influenciado por una variedad de factores que un modelo matemático simplificado, como el que se tenía en mente, no es capaz de capturar completamente. Esta limitación técnica justifica la exclusión de dicha funcionalidad en el diseño del actual proyecto. [50]

El código empleado para calcular los parámetros mencionados pertenece al sketch de Arduino *transformarDatos.ino* mostrado en el ANEXO V. PROGRAMAS DESARROLLADOS.

#### 3.4.4. ALMACENAMIENTO DE LA INFORMACIÓN

Tal y como se explicó en 3.3.3, la célula de memoria que se va a utilizar para recoger los parámetros de las rutas es una tarjeta microSD de 8 GB. En Arduino IDE, existen librerías que permiten la comunicación entre el Arduino Nano y la microSD a partir del interfaz de comunicación SPI (2.4), de forma que se puedan leer y escribir datos en ella. En este caso, sólo es necesario escribir información en la tarjeta. Para ello, se crea un fichero de texto cada vez que se enciende el dispositivo y se escriben ahí los parámetros calculados previamente. El nombre de este fichero es "DATOS\_N.txt", siendo N el índice del fichero. Así, en la tarjeta habrá almacenados tantos ficheros de texto como veces se haya encendido el dispositivo. Por lo tanto, este sistema está pensado para encender el localizador GPS sólo cuando se vaya a iniciar una ruta y se apague sólo cuando ésta finalice.

Más adelante, en el apartado 7.2.5, se explicará esto con mayor profundidad.

Por otra parte, cabe destacar que los parámetros de ruta calculados por el microcontrolador se almacenan en la microSD cada 10 segundos. Este intervalo de tiempo se seleccionó para equilibrar la necesidad de capturar datos relevantes con la eficiencia de almacenamiento. Guardar datos continuamente no sería eficiente, ya



que aumentaría innecesariamente la cantidad de datos almacenados sin aportar beneficios adicionales en términos de precisión o utilidad de la información. Al registrar los datos cada 10 segundos, se asegura una representación detallada de la ruta recorrida, manteniendo un uso eficiente del espacio de almacenamiento y prolongando la vida útil de la tarjeta microSD.

El código utilizado para escribir datos en la tarjeta microSD pertenece al sketch de Arduino mostrado en el ANEXO V. PROGRAMAS DESARROLLADOS.

Con el fin de proporcionar una representación gráfica de cómo funciona este programa, en la *Figura 35* se ilustra su diagrama de bloques correspondiente. Como se puede observar, este programa engloba los ya vistos procedimientos de procesado y conversión de datos, el cálculo de los parámetros de rutas y el almacenamiento de la información.

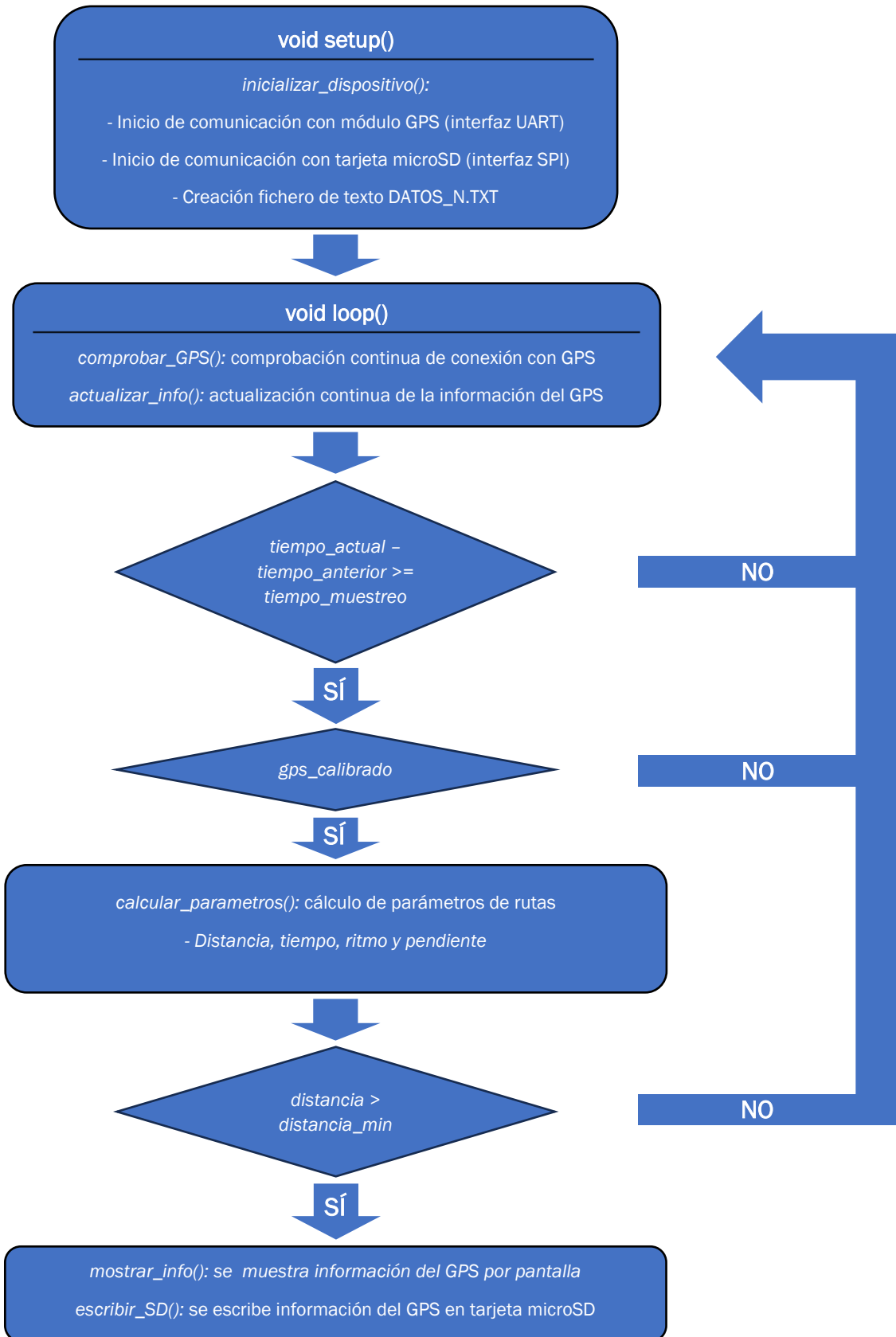


Figura 35. Diagrama de bloques del sketch de Arduino `transformarDatos.ino` (elaboración propia)



### 3.4.5. UNIFICACIÓN DE LA INFORMACIÓN

Según lo que se ha explicado en el apartado anterior, por cada ruta que realice el usuario, se crea un nuevo fichero de texto en la tarjeta microSD que contiene los parámetros de la ruta.

A pesar de que la herramienta de visualización de datos utilizada también permite importar archivos “.txt”, se ha optado por primero transformar los ficheros de texto almacenados en la microSD a Excel. La principal razón es que Excel permite una edición manual más sencilla de los datos, facilitando el manejo de los datos en caso de necesitar realizar ajustes rápidos o correcciones.

Para realizar esta transformación de fichero de texto a fichero Excel, se ha creado un pequeño fragmento de código utilizando la librería *Pandas* de Python, el cual se muestra en el ANEXO V. PROGRAMAS DESARROLLADOS.

El hecho de utilizar un lenguaje de programación tan potente como Python, permite también combinar todos los ficheros de texto en un mismo archivo Excel. Además, aprovechando esta poderosa herramienta, se ha decidido añadir dos columnas nuevas, nombradas como *tipoRuta* y *rutaID*, cuya utilidad se explicará en los apartados 6.1 y 6.2. De esta manera, el resultado de ejecutar *unificarDatos.py* será un archivo Excel de nombre *historicoDATOS.xlsx*, donde aparecerán unificadas todas las rutas realizadas. Cabe destacar, que este proceso también se pensó en llevar a cabo mediante la programación en Arduino IDE. Sin embargo, dado que resultaba una tarea mucho más complicada, se decidió a aprovechar las capacidades de Python para hacerlo posible.

Por último, con el objetivo de que el usuario final pueda utilizar estas funcionalidades del dispositivo independientemente de sus conocimientos en Excel y Python, se ha creado un archivo ejecutable llamado *unificarDatos.exe*. De este modo, bastará con que el usuario ubique este programa en la misma carpeta donde vaya situando los ficheros de texto extraídos de la tarjeta de memoria microSD y lo ejecute cada vez que almacene un archivo asociado a una ruta nueva.

### 3.4.6. VISUALIZACIÓN DE LA INFORMACIÓN

El último eslabón de esta cadena de funcionamiento del localizador GPS es la visualización de la información registrada, de manera que el usuario pueda analizar cómo han sido las rutas que ha llevado cabo.

A la hora de decidir qué herramienta se iba a utilizar para este fin, surgieron varias posibilidades. La opción más evidente era desarrollar una aplicación móvil específica para este caso. Sin embargo, después de una breve investigación, se llegó a la conclusión de que el tiempo de aprendizaje y ejecución de una aplicación así superaba enormemente los plazos y propósito de este trabajo.



Descartada esa opción, se optó por utilizar una herramienta de visualización de datos ya existente, tales como el propio Microsoft Excel, Tableau, Google Charts, Zoho Analytics, etc. [51]

De entre todas estas, finalmente se decidió utilizar Power BI para visualizar los parámetros y estadísticas de las rutas realizadas por el usuario.

Power BI es una plataforma integral de servicios de software, aplicaciones y conectores que trabajan conjuntamente para transformar datos de diversas fuentes en información coherente, interactiva y visualmente atractiva. Estos datos pueden proceder desde una simple hoja de cálculo de Excel hasta complejos orígenes de datos híbridos, tanto locales como basados en la nube. Power BI facilita la conexión a diversas fuentes de datos, la visualización de datos relevantes y el descubrimiento de información crítica, permitiendo compartir esta información de manera eficaz con usuarios específicos o con una audiencia más amplia. [52]

En el contexto de este proyecto, se aprovechará la capacidad de Power BI para manejar grandes volúmenes de datos y generar visualizaciones interactivas a través de informes creados por y para el usuario. En particular, se hará uso de Power BI Desktop, una aplicación de escritorio para Windows que permite la creación y gestión de informes y plantillas de informe. Un ejemplo de este tipo de informes, también conocidos como *dashboard*, se muestra en la *Figura 36*:

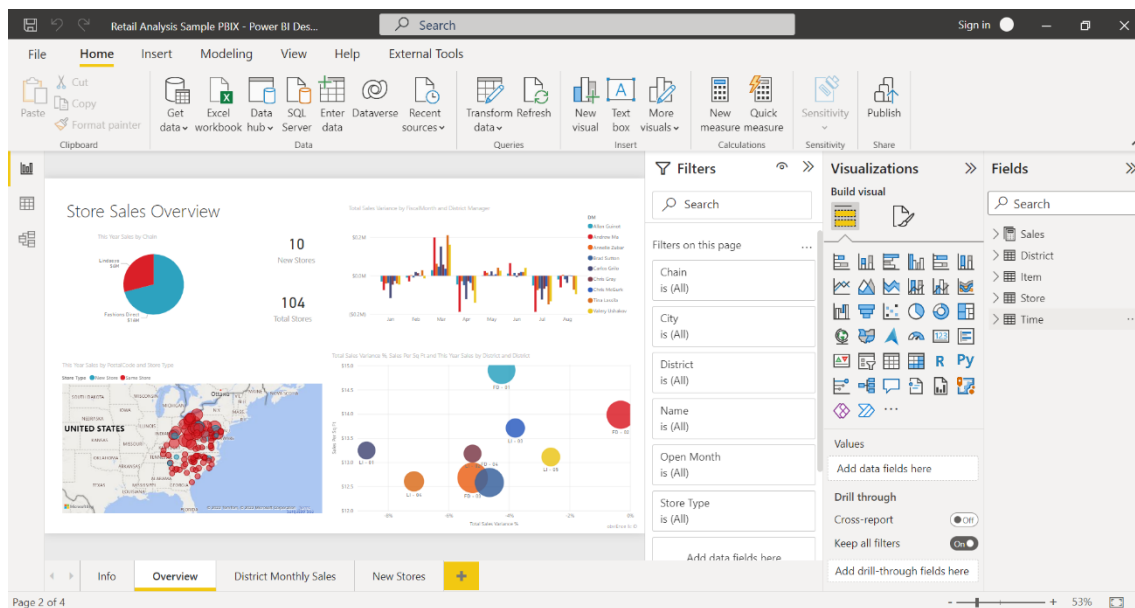


Figura 36. Ejemplo de informe en Power BI (extraído de [52])

Entre las razones que sustentan esta elección se incluyen la disponibilidad de una versión gratuita, la no necesidad de tener conocimientos en lenguajes de programación como SQL para combinar y filtrar datos y la presencia de una interfaz de usuario intuitiva que permite una curva de aprendizaje corta. Por otra parte, dada la naturaleza de este proyecto, era fundamental encontrar una aplicación que



integrarse mapas en los que poder representar el recorrido realizado por el usuario, utilidad que también ofrece Power BI. Además, existen numerosos vídeos y contenidos educativos que facilitan la integración de nuevos usuarios a Power BI.

Como inconveniente más notable, la versión gratuita de Power BI tiene una limitación en el manejo de datos, permitiendo operar con hasta 2 GB de datos a la vez, lo cual no es adecuado para grandes conjuntos de datos. Sin embargo, para los fines de este proyecto, dicha capacidad es más que suficiente.

En el apartado 6, se describirá cómo ha sido el proceso de creación de un informe en Power BI y cómo se ha aplicado a este proyecto.

Por último, a modo de resumen y para ofrecer una descripción más gráfica, se muestra la *Figura 37*:

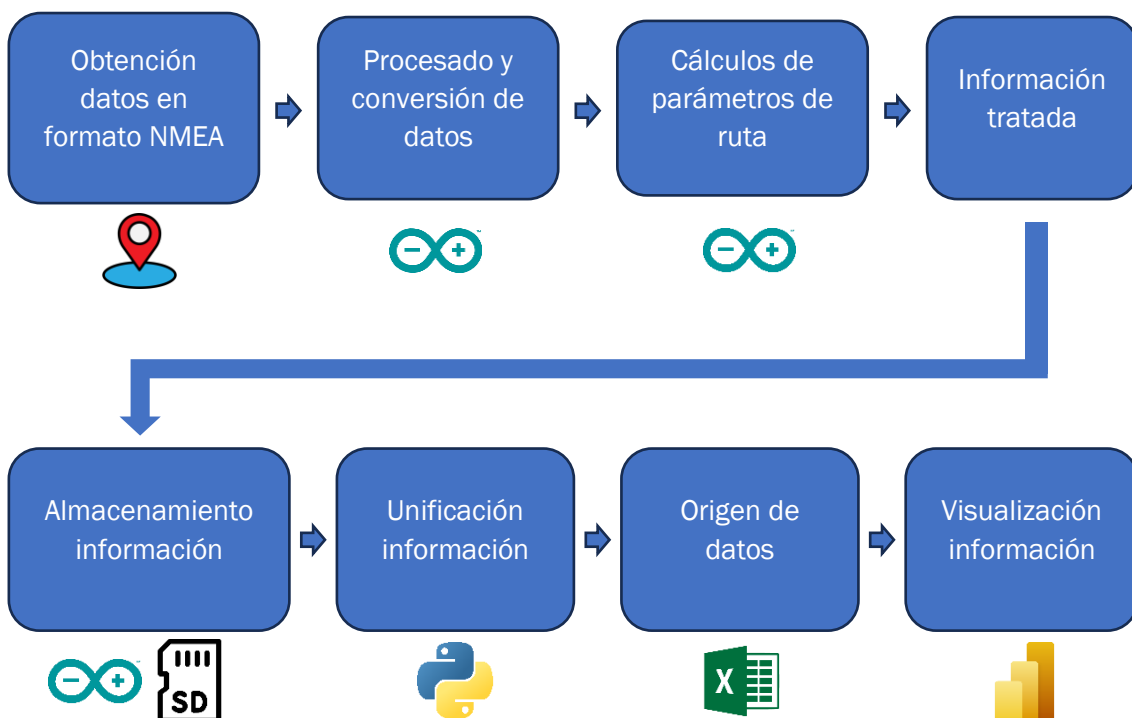


Figura 37. Esquema final de funcionamiento del localizador GPS (elaboración propia)

Como se puede apreciar, este esquema complementa la idea inicial del proyecto mostrada en la *Figura 1*.

### 3.5. PROTOTIPADO INICIAL

Una vez elegidos los componentes que van a constituir el localizador GPS y esbozado cuál va a ser su modo de operación, se ha realizado su montaje (*Figura 38*) en una protoboard con el fin de comprobar que todo funcionaba correctamente.



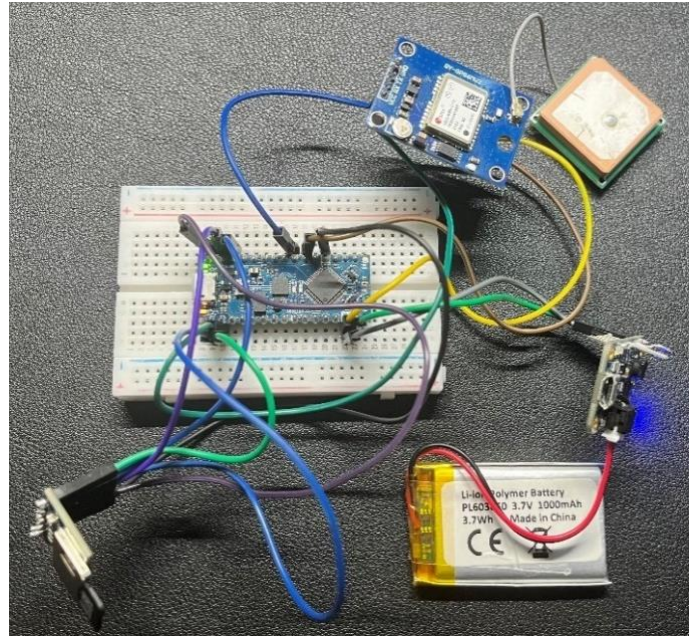


Figura 38. Prototipado inicial realizado (elaboración propia)

Como la *Figura 38* no es muy descriptiva en términos de cómo están los componentes conectados entre sí, a continuación se muestra el esquema de montaje inicial utilizando la herramienta Fritzing (*Figura 39*):

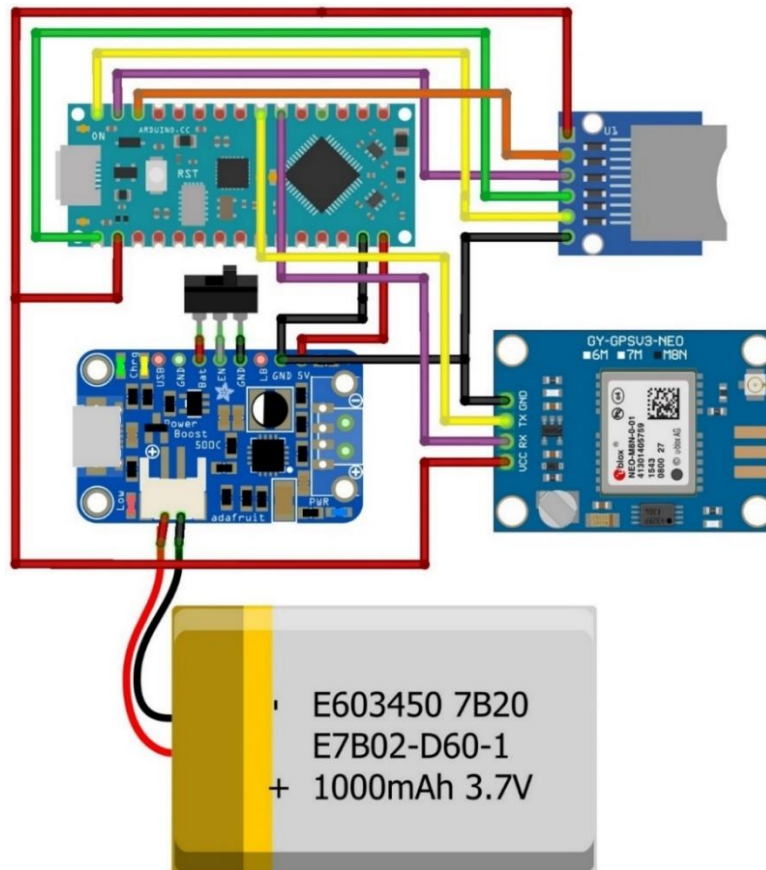


Figura 39. Esquema de montaje inicial en Fritzing (elaboración propia)





Se puede apreciar que el conexionado entre elementos se ha llevado a cabo de la siguiente manera (*Tabla 5, Tabla 6 y Tabla 7*):

*Tabla 5. Conexión de pines entre Arduino Nano y módulo GPS (elaboración propia)*

Arduino Nano Every	Módulo GPS GY-GPS6MV2
3V3	VCC
GND	GND
D4 (RXPin)	TX
D3 (TXPin)	RX

*Tabla 6. Conexión de pines entre Arduino Nano y lector de tarjetas microSD (elaboración propia)*

Arduino Nano Every	Módulo Lector microSD
3V3	VCC
GND	GND
D13	CLK
D12	MISO
D11	MOSI
D10	CS

*Tabla 7. Conexión de pines entre Arduino Nano y cargador LiPo (elaboración propia)*

Arduino Nano Every	Cargador LiPo Powerboost
VIN	5V
GND	GND

Como se observa en la *Tabla 5*, las conexiones entre la placa de Arduino y el módulo GPS van cruzadas, esto es, el pin TX del módulo GPS al RXPin (asignado al pin digital D4) del Arduino Nano y el pin RX del GPS a TXPin (asignado al pin digital D3) del Arduino Nano. Esto se debe a que ambos elementos intercambian información utilizando el interfaz de comunicación UART (ver apartado 2.3). De esta forma, el módulo GPS envía la información por su pin TX y el Arduino Nano recibe dicha información por el pin RXPin (o D4), y viceversa.

### 3.6. COMPROBACIÓN INICIAL DE FUNCIONAMIENTO

Utilizando el prototipado inicial mostrado en la *Figura 39*, se han llevado a cabo las primeras pruebas de funcionamiento del dispositivo. En ellas, se ha verificado que el módulo GPS recibiese y enviase los datos relativos a su posición, el Arduino Nano Every procesase dichos datos e hiciera los cálculos pertinentes y que la información se escribiese en la tarjeta microSD. Dicho de otra manera, que todos los elementos se comportasen según lo esperado. Uno de los primeros ficheros que se han obtenido como resultado de todo este proceso se muestra en la *Figura 40*:



DATOS\_1: Bloc de notas

Latitud	Longitud	Distancia (km)		Distancia total (km)		Tiempo (s)		Tiempo total (s)		Velocidad (km/h)		Ritmo (min/km)	Altitud (m)
Pendiente(%)	Pendiente	Absoluta(%)	Hora	Fecha									
41.64068	-4.757974	0	0	0	0	0	19.79	719.9	0.00	0.00	12:00:00	12/03/2024	
41.641209	-4.757792	0.061	0.061	322	322	0.68	19.79	719.59	-0.51	0.51	12:05:22	12/03/2024	
41.641338	-4.75629	0.126	0.187	301	623	2.24	19.79	718.86	-0.58	0.58	12:10:23	12/03/2024	
41.642348	-4.755775	0.1206	0.3076	322	945	3.44	19.79	715.46	-2.82	2.82	12:15:45	12/03/2024	
41.642845	-4.752621	0.2688	0.5764	299	1244	6.94	19.79	699.74	-5.85	5.85	12:20:44	12/03/2024	
41.642588	-4.750282	0.1971	0.7735	290	1534	9.6	19.79	687.66	-6.13	6.13	12:25:34	12/03/2024	
41.641209	-4.744617	0.4967	1.2702	278	1812	16.45	19.79	680.47	-1.45	1.45	12:30:12	12/03/2024	
41.640872	-4.742772	0.1584	1.4286	302	2114	17.03	19.79	685.57	3.22	3.22	12:35:14	12/03/2024	
41.642701	-4.741119	0.2461	1.6747	329	2443	18.32	19.79	686.88	0.53	0.53	12:40:43	12/03/2024	
41.64359	-4.739789	0.1489	1.8235	282	2725	23.28	19.79	688.67	1.20	1.20	12:45:25	12/03/2024	
41.644369	-4.737635	0.1995	2.023	303	3028	24.04	19.79	689.22	0.28	0.28	12:50:28	12/03/2024	
41.643671	-4.735919	0.1629	2.1859	321	3349	24.51	19.79	689.39	0.10	0.10	12:55:49	12/03/2024	
41.643667	-4.735954	0.003	2.1889	273	3622	28.86	19.79	689.4	0.33	0.33	13:00:22	12/03/2024	
41.645098	-4.734015	0.2272	2.416	297	3919	29.29	19.79	688.51	-0.39	0.39	13:05:19	12/03/2024	
41.648301	-4.729779	0.5024	2.9185	318	4237	33.04	19.79	686.96	-0.31	0.31	13:10:37	12/03/2024	
41.649509	-4.732831	0.2879	3.2064	279	4516	41.37	19.79	687.61	0.23	0.23	13:15:16	12/03/2024	
41.650727	-4.738711	0.5086	3.715	339	4855	39.45	19.79	687.89	0.06	0.06	13:20:55	12/03/2024	
41.648306	-4.740299	0.3008	4.0158	303	5158	47.71	19.79	686.37	-0.51	0.51	13:25:58	12/03/2024	
41.645837	-4.741801	0.3026	4.3184	301	5459	51.65	19.79	687.76	0.46	0.46	13:30:59	12/03/2024	
41.642978	-4.743946	0.3657	4.6841	246	5705	68.55	19.79	688.98	0.33	0.33	13:35:05	12/03/2024	
41.641647	-4.744899	0.1684	4.8525	301	6006	58.04	19.79	704.61	9.28	9.28	13:40:06	12/03/2024	
41.642946	-4.748225	0.3129	5.1653	302	6308	61.57	19.79	711.65	2.25	2.25	13:45:08	12/03/2024	
41.643411	-4.75213	0.3297	5.495	348	6656	56.85	19.79	717.58	1.80	1.80	13:50:56	12/03/2024	
41.643026	-4.756121	0.3355	5.8305	251	6907	83.63	19.79	721.08	1.04	1.04	13:55:07	12/03/2024	
41.641278	-4.757816	0.2408	6.0714	304	7211	71.9	19.79	720.75	-0.14	0.14	14:00:11	12/03/2024	

Figura 40. Ejemplo de fichero resultado DATOS.txt (elaboración propia)

Cabe mencionar que inicialmente el Arduino se había programado para que cada vez que se encendiese el localizador GPS, se crease un nuevo fichero DATOS.txt y se borrara el anterior. No obstante, esto obligaba al usuario a tener que guardar dicho fichero en otro sitio distinto a la tarjeta cada vez que terminase un recorrido, lo que suponía una limitación muy grande. En consecuencia, finalmente se ha optado por crear un nuevo fichero DATOS\_N.txt cada vez que se encienda el dispositivo, como se mencionó en los apartados 3.4.4 y 3.4.5. De esta manera, el usuario puede realizar múltiples rutas antes de extraer la microSD y guardar la información en su ordenador.

### 3.6.1. LIMITACIONES EN LA PRECISIÓN DEL RECEPTOR GPS NEO-M8N

A pesar de que durante esta comprobación inicial no se detectaron errores críticos que invalidaran los cálculos teóricos realizados previamente, sí que se han identificado ciertas limitaciones, las cuales se detallan a continuación.

#### Limitación del módulo GPS en espacios cerrados

Se han realizado pruebas tanto en entornos interiores como exteriores, observándose una notable diferencia en la rapidez y precisión del sensor GPS entre ambos escenarios. En interiores, el módulo tardaba varios minutos en calibrarse y comenzar la transmisión de datos, mientras que al aire libre, este tiempo se reducía a menos de un minuto, y la precisión de los datos era significativamente superior.

Particularmente, cuando se ha probado en una habitación cercana a la calle con ventanas, el módulo aún mostraba dificultades para calibrarse rápidamente. En una habitación completamente interior, sin contacto directo con el exterior, el módulo GPS no ha sido capaz de establecer comunicación con ninguna red satelital. Por lo tanto, el uso de este dispositivo se ve limitado a actividades al aire libre, tal y como se definió al inicio del proyecto. [53]



Además, cabe destacar que, para asegurar que solo se almacenan datos en la microSD después de que el sensor GPS se haya calibrado adecuadamente, se ha implementado en el código de Arduino una condición que obliga al microcontrolador a realizar cálculos y almacenar los resultados de estos única y exclusivamente cuando el dispositivo esté calibrado, es decir, cuando los valores de latitud, longitud y altitud recibidos sean válidos (ver ANEXO V. PROGRAMAS DESARROLLADOS).

#### Limitación en la precisión del módulo GPS

Por otro lado, se ha observado que el error en la precisión del GPS al aire libre es del orden de unos pocos metros. Este problema es especialmente evidente cuando el dispositivo está estacionario, ya que aun así detecta movimientos falsos y la posición del usuario varía entre 0 y 5 metros aproximadamente. Como resultado, se registran datos de velocidad y distancia distintos de cero, incluso cuando el usuario está inmóvil. [53]

Para mitigar este problema, se han realizado ajustes en el código de Arduino, como establecer una distancia mínima de movimiento por debajo de la cual se considera que el usuario no se ha desplazado. Esta distancia se ha fijado en 5 metros, de manera que solo se considera que hay movimiento cuando el desplazamiento del usuario supera los 5 metros. Teniendo en cuenta que la posición del usuario se actualiza cada 10 segundos, este ajuste garantiza que solo se registre movimiento cuando realmente se ha producido un desplazamiento significativo. (ver ANEXO V. PROGRAMAS DESARROLLADOS).

#### Limitación en la medición de altitud

Finalmente, durante las primeras pruebas del dispositivo, uno de los aspectos que ha requerido un análisis detallado ha sido la precisión en la medición de la altitud utilizando el chip GPS NEO-M8N. Se ha podido constatar que este chip, aunque eficaz en la obtención de coordenadas horizontales, presenta serias limitaciones en la medición de la altitud.

En numerosas pruebas realizadas, se ha observado que los valores de altitud registrados por el NEO-M8N variaban significativamente incluso cuando el dispositivo permanecía inmóvil en un mismo lugar. Esta fluctuación ha sido aún más pronunciada cuando ha habido desplazamientos de unos pocos metros, resultando en variaciones de altitud que estaban claramente alejadas de la realidad. Esta inconsistencia en los datos de altitud plantea un desafío considerable, especialmente en aplicaciones donde la precisión es crítica. Investigando más a fondo y revisando documentación técnica sobre el funcionamiento de los GPS, se han llegado a conocer las razones detrás de estas limitaciones.

Como ya se explicó en el apartado 2.1.2 la tecnología GPS utiliza señales de múltiples satélites para determinar la posición del receptor en tres dimensiones: latitud,



longitud y altitud. Sin embargo, la medición de la altitud es intrínsecamente más complicada que la de las coordenadas horizontales.

La precisión en la determinación de la altitud depende de varios factores, como la geometría de los satélites y las variaciones atmosféricas. La geometría de los satélites es menos favorable para la medición de la altitud porque los satélites visibles suelen estar distribuidos alrededor del receptor, con pocos satélites directamente arriba o abajo. Esta distribución provoca que los errores en la medición de la altitud se amplifiquen en comparación con los errores en la latitud y longitud.

Además, las señales GPS deben atravesar la atmósfera, lo que introduce retrasos debido a la ionosfera y la troposfera. Estos retrasos son más difíciles de corregir con precisión en la dimensión vertical, contribuyendo a errores significativos en la altitud medida. Las correcciones estándar aplicadas a las mediciones horizontales no son tan efectivas para las mediciones verticales, lo que resulta en la variabilidad observada.

Para abordar esta limitación, es común el uso de altímetros barométricos o barómetros, que ofrecen una precisión mucho mayor en la medición de la altitud. Estos dispositivos miden la presión atmosférica, la cual varía con la altitud, permitiendo calcular esta con una mayor exactitud. A diferencia de los GPS, los altímetros barométricos no dependen de la geometría de los satélites ni de las variaciones atmosféricas en la misma medida, proporcionando datos de altitud más estables y confiables.

En conclusión, las pruebas realizadas y la documentación técnica revisada confirman que la medición de la altitud con el chip GPS NEO-M8N está sujeta a fluctuaciones significativas, limitando su precisión y fiabilidad. Este hallazgo subraya la necesidad de utilizar tecnologías complementarias, como altímetros barométricos, para aplicaciones donde la precisión en la altitud es crucial. Integrar un altímetro barométrico en futuros diseños del dispositivo podría mejorar considerablemente la exactitud de las mediciones de altitud, haciendo el dispositivo más robusto y fiable. [54]

### 3.7. PRIMERA APROXIMACIÓN DEL COSTE DEL EQUIPO

A continuación, se muestra la lista de materiales inicial, donde se incluyen tanto sus precios unitarios como para una producción alta.



Tabla 8. Lista de componentes inicial (elaboración propia)

LISTA DE COMPONENTES							
Componente						Coste por unidad (sin IVA)	
Nombre	Tipo	Modelo	Nº de referencia	Fabricante	Unidades	1 ud.	Producción alta (+100 uds.)
Placa Arduino	Nano	Every	ABX00028	Arduino	1	10.95 €	10.10 €
Módulo GPS con antena	GY-GPS6MV 2	NEO-M8N	-	U-blox	1	7.99 €	7.03 €
Cargador LiPo	Power Boost	500C	PRO-0109	Adafruit	1	15.95 €	15.95 €
Batería LiPo	3.7V/1000mAh	6030650	BAT-0004	-	1	5.95 €	5.95 €
Módulo Lector microSD	-	-	-	TZT	1	0.77 €	0.77 €
Tarjeta de memoria	microSD	8 GB	SP008GBSTH BU1V10SP	Silicon Power	1	3.37 €	3.37 €
Interruptor	Rocker	SR	SRB22A2FBB NN	ZF	1	1.55 €	1.35 €
<b>TOTAL</b>						<b>46.53 €</b>	<b>44.52 €</b>





# CAPÍTULO 4: DISEÑO DEL DISPOSITIVO





## 4. DISEÑO DEL DISPOSITIVO

Una vez descritos todos los componentes que van a constituir el localizador GPS, su modo de operación y habiendo verificado que todo funciona correctamente, se procede a diseñar el dispositivo completo, con el objetivo de obtener un equipo funcional que cumpla con los requisitos de diseño marcados en el apartado 3.1.

Este diseño consta de dos partes principales: la creación de la placa de circuito impreso (PCB) y el diseño de la carcasa que envuelve la PCB. Para respectivos diseños se han utilizado los programas Autodesk EAGLE y Fusion 360.

Esta elección se fundamenta en varias razones. Primero, como alumno de la Universidad de Valladolid, puedo acceder a las licencias “premium” de estos programas. Segundo, considero que ambos programas son relativamente intuitivos, lo que proporciona una experiencia agradable y accesible para usuarios “novatos”, permitiendo la realización de proyectos no excesivamente complejos. Finalmente, al ser parte del mismo ecosistema de software (Autodesk), existe compatibilidad entre ambos programas. Esto ha permitido realizar el diseño de la PCB en EAGLE, para más tarde exportar su modelo 3D a Fusion 360 y poder diseñar la carcasa. Esta integración ha facilitado enormemente la tarea, ya que el modelo 3D permite realizar las mediciones necesarias de manera sencilla.

### 4.1. REALIZACIÓN DEL DISEÑO DE LA PLACA DE CIRCUITO IMPRESO

Antes de explicar el diseño de la PCB, es importante mencionar que se consideraron otras posibilidades para hacer el dispositivo más compacto. La opción más relevante fue utilizar una placa de expansión para Arduino, conocida como Arduino Shield, adaptada para una placa Arduino Nano (*Figura 41*).

Este accesorio está diseñado para aumentar la funcionalidad de las placas Arduino, proporcionando una forma conveniente de conectar múltiples componentes y periféricos adicionales, lo que facilita la creación de prototipos complejos.



Figura 41. Placa de expansión para Arduino Nano o Arduino Nano Shield (extraído de [55])

Finalmente, se desechó esta alternativa ya que no se ajustaba exactamente a las necesidades del dispositivo: aumentaba considerablemente el tamaño del dispositivo en comparación con una PCB, no todos los componentes cabían en la propia placa y obligaba a utilizar cables, reduciendo la compacidad del equipo. En consecuencia, se decidió diseñar una placa de circuito impreso para acomodar los componentes.

A continuación, en la *Figura 42*, se muestra un detalle del esquema eléctrico a partir del cual se ha llevado a cabo el diseño de la placa, cuyo plano completo se muestra en el ANEXO I. PLANOS:

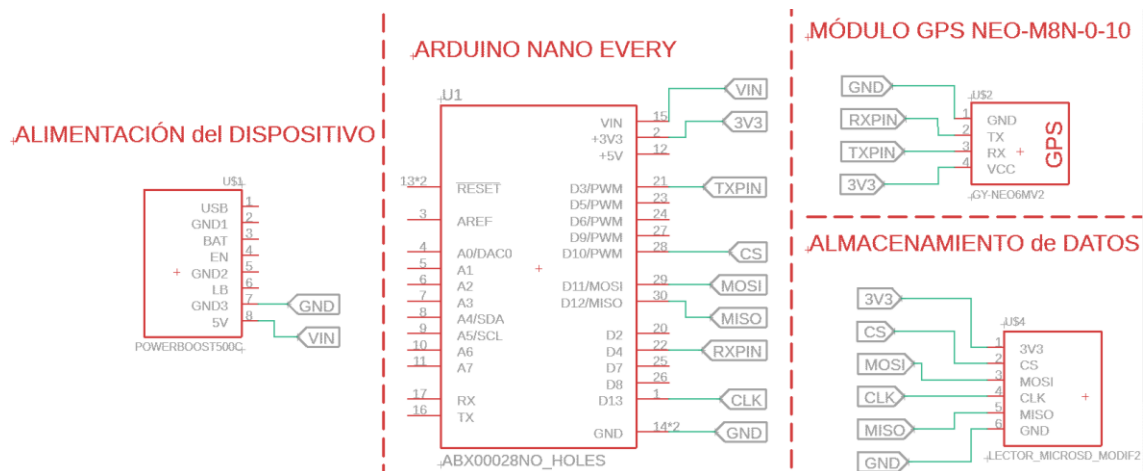


Figura 42. Detalle del esquema eléctrico del dispositivo (elaboración propia)

El objetivo principal del diseño de la PCB era reducir el tamaño lo máximo posible, una condición esencial para conseguir un localizador GPS fácil de llevar durante actividades deportivas al aire libre. Además, era fundamental situar los conectores micro-USB lo más cerca del borde de la placa. De este modo, el dispositivo puede conectarse a la red eléctrica en caso de que la batería esté baja o a un ordenador para manipular la placa Arduino (cargando un programa de prueba o modificando el existente) sin que ningún componente obstaculice su conexión con el exterior. Lo mismo se aplicó al módulo lector de microSD, el cual se ubicó próximo al límite exterior de la placa para facilitar la introducción y extracción de la tarjeta de memoria..

Teniendo en cuenta estas consideraciones, el emplazamiento de los componentes en la placa de circuito impreso se muestra en la *Figura 43*:

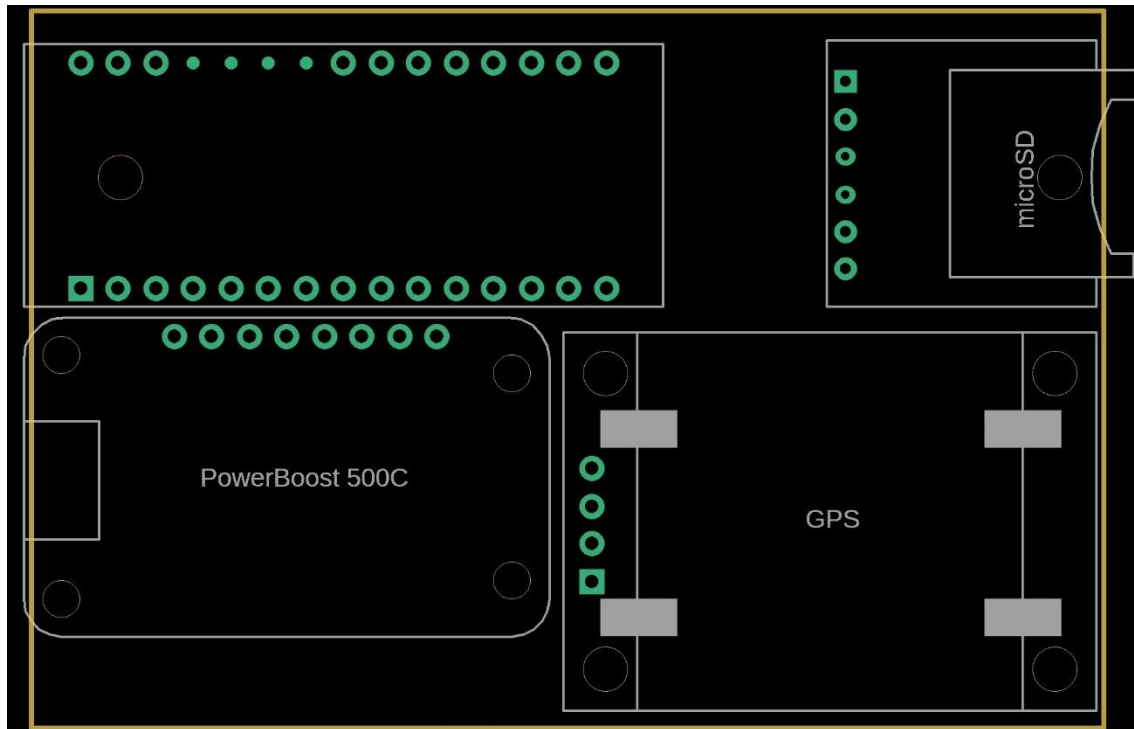


Figura 43. Emplazamiento de los componentes en la PCB (elaboración propia)

Además de obtener una placa pequeña y ubicar los conectores cerca del borde, este emplazamiento también pretende reducir al máximo el número y la longitud de las pistas necesarias en la capa de soldadura, siempre teniendo en cuenta el conexionado de los componentes. En definitiva, el tamaño final de la placa es de 73 x 49,70 mm.

Para finalizar el diseño, es necesario realizar las conexiones entre los componentes, es decir, diseñar las pistas de circuito impreso. Previamente, se debe determinar el grosor de estas teniendo en cuenta las corrientes que circulan por cada nodo y la separación entre ellas en función de sus diferencias de potencial, según la normativa correspondiente. Con el fin de reducir al mínimo el número de pistas en la cara de componentes, el rutado se ha realizado en una sola cara (cara de soldadura.)

Los cálculos de los espesores de las pistas y su separación entre ellas se describen a continuación.

#### 4.1.1. ANCHURA MÍNIMA DE LAS PISTAS:

Para calcular la anchura mínima de las pistas se utilizará como regla de diseño el cálculo analítico de la anchura de las pistas que viene dado por las ecuaciones (E-14) y (E-15):

$$W = \frac{A}{e} \quad (\text{E-14})$$

$$A = \left[ \frac{I}{(0.048 \cdot \Delta T^{0.44})} \right]^{0.725} \quad (\text{E-15})$$



Donde:

**W:** anchura de pista en mils.

**A:** área de la sección de la pista en mils

**e:** espesor de la pista en mils

**ΔT:** variación de temperatura en grados centígrados

En este caso, el espesor de la pista es de 35 μm (equivalente a 1,38 mils).

Por otro lado, dado que el dispositivo final está pensado para funcionar en rangos de temperaturas diversos, se ha considerado una variación de temperatura de 80 °C.

El último parámetro necesario para calcular el ancho de las pistas es el valor máximo de intensidad de corriente que va a circular por ellas. En este sentido, la zona más problemática del circuito se localiza en la conexión entre el cargador y la placa Arduino, donde, como se vio en el apartado 3.3.5, circularán como máximo aproximadamente 190 mA.

Por tanto, sustituyendo en las ecuaciones (E-14) y (E-15) se obtiene que el ancho mínimo de pistas será:

$$W = \frac{\left( \frac{0.190A}{0.048 \cdot (80^{\circ}\text{C})^{0.44}} \right)^{\frac{1}{0.725}}}{1.38 \text{ mils}}$$

$$W = 0.34 \text{ mils} \equiv 0,0086 \text{ mm}$$

Es decir, se obtiene que la mínima anchura de las pistas del circuito debe ser de, aproximadamente, 0,009 mm.

Al haber tenido en cuenta la sección crítica del circuito, se considera que la anchura mínima para el resto de pistas del circuito es menor a la calculada.

No obstante, debido a un problema con la insoladora y por recomendaciones del personal del laboratorio se ha utilizado una anchura final de pistas de 32 mils (0,8128 mm).

#### 4.1.2. SEPARACIÓN DE LAS PISTAS

Por otro lado, para calcular la separación entre pistas, se tendrá en cuenta la norma IPC 2221 (revisión B, en vigor desde 2012). Más concretamente, la tabla 6-1 extraída de esta norma, en la cual aparecen valores que indican la separación mínima o “clearance” entre conductores en función de la tensión entre dos conductores:



Table 6-1 Electrical Conductor Spacing

Voltage Between Conductors (DC or AC Peaks)	Minimum Spacing						
	Bare Board				Assembly		
	B1	B2	B3	B4	A5	A6	A7
0-15	0.05 mm [0.00197 in]	0.1 mm [0.0039 in]	0.1 mm [0.0039 in]	0.05 mm [0.00197 in]	0.13 mm [0.00512 in]	0.13 mm [0.00512 in]	0.13 mm [0.00512 in]
16-30	0.05 mm [0.00197 in]	0.1 mm [0.0039 in]	0.1 mm [0.0039 in]	0.05 mm [0.00197 in]	0.13 mm [0.00512 in]	0.25 mm [0.00984 in]	0.13 mm [0.00512 in]
31-50	0.1 mm [0.0039 in]	0.6 mm [0.024 in]	0.6 mm [0.024 in]	0.13 mm [0.00512 in]	0.13 mm [0.00512 in]	0.4 mm [0.016 in]	0.13 mm [0.00512 in]
51-100	0.1 mm [0.0039 in]	0.6 mm [0.024 in]	1.5 mm [0.0591 in]	0.13 mm [0.00512 in]	0.13 mm [0.00512 in]	0.5 mm [0.020 in]	0.13 mm [0.00512 in]
101-150	0.2 mm [0.0079 in]	0.6 mm [0.024 in]	3.2 mm [0.126 in]	0.4 mm [0.016 in]	0.4 mm [0.016 in]	0.8 mm [0.031 in]	0.4 mm [0.016 in]
151-170	0.2 mm [0.0079 in]	1.25 mm [0.0492 in]	3.2 mm [0.126 in]	0.4 mm [0.016 in]	0.4 mm [0.016 in]	0.8 mm [0.031 in]	0.4 mm [0.016 in]
171-250	0.2 mm [0.0079 in]	1.25 mm [0.0492 in]	6.4 mm [0.252 in]	0.4 mm [0.016 in]	0.4 mm [0.016 in]	0.8 mm [0.031 in]	0.4 mm [0.016 in]
251-300	0.2 mm [0.0079 in]	1.25 mm [0.0492 in]	12.5 mm [0.4921 in]	0.4 mm [0.016 in]	0.4 mm [0.016 in]	0.8 mm [0.031 in]	0.8 mm [0.031 in]
301-500	0.25 mm [0.00984 in]	2.5 mm [0.0984 in]	12.5 mm [0.4921 in]	0.8 mm [0.031 in]	0.8 mm [0.031 in]	1.5 mm [0.0591 in]	0.8 mm [0.031 in]
> 500 See para. 6.3 for calc.	0.0025 mm /volt	0.005 mm /volt	0.025 mm /volt	0.00305 mm /volt	0.00305 mm /volt	0.00305 mm /volt	0.00305 mm /volt

- B1 - Internal Conductors
- B2 - External Conductors, uncoated, sea level to 3050 m [10,007 feet]
- B3 - External Conductors, uncoated, over 3050 m [10,007 feet]
- B4 - External Conductors, with permanent polymer coating (any elevation)
- A5 - External Conductors, with conformal coating over assembly (any elevation)
- A6 - External Component lead/termination, uncoated, sea level to 3050 m [10,007 feet]
- A7 - External Component lead termination, with conformal coating (any elevation)

Figura 44. Requisitos de espacio de conductores según norma 2221B (extraído de [56])

Este caso se enmarca en la situación B2, al tratarse de una placa de circuito impreso (Bare Board), sin recubrimiento, situada por encima del nivel del mar y por debajo de los 3050 m. Además, el voltaje máximo entre conductores viene dado entre los pines VIN y GND del cargador LiPo, que proporciona una tensión de salida de aproximadamente 5.2V. Por lo tanto, la separación mínima entre pistas será de 0.1 mm (3,937 mils). Por último, resulta interesante recordar que la placa utilizada es de tecnología a una capa, por lo que todas las pistas se situarán en la capa de soldadura.

De esta forma y teniendo en cuenta todas las especificaciones ya mencionadas, el rutado final de la placa queda de la siguiente manera (Figura 45):



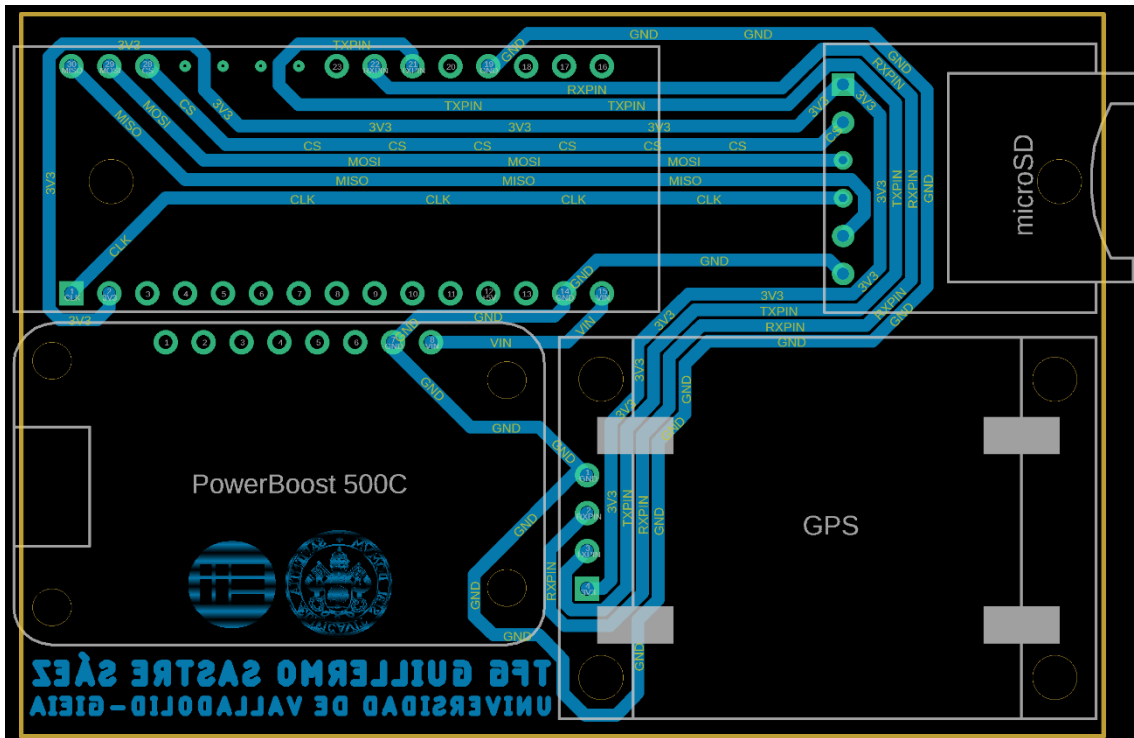


Figura 45. Rutado de pistas (elaboración propia)

## 4.2. GENERACIÓN DE LA DOCUMENTACIÓN PARA LA FABRICACIÓN DE LA PCB

### 4.2.1. FOTOLITOS

❖ Cara de componentes (Top):

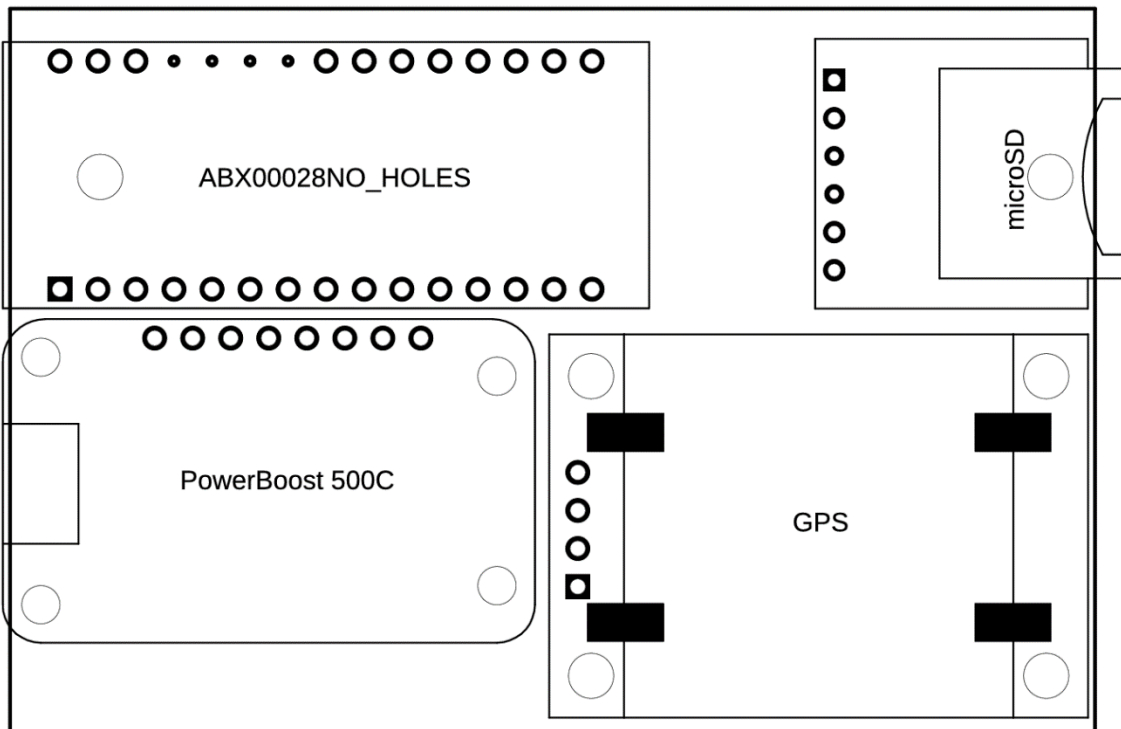


Figura 46. Fotolito cara de componentes (elaboración propia)

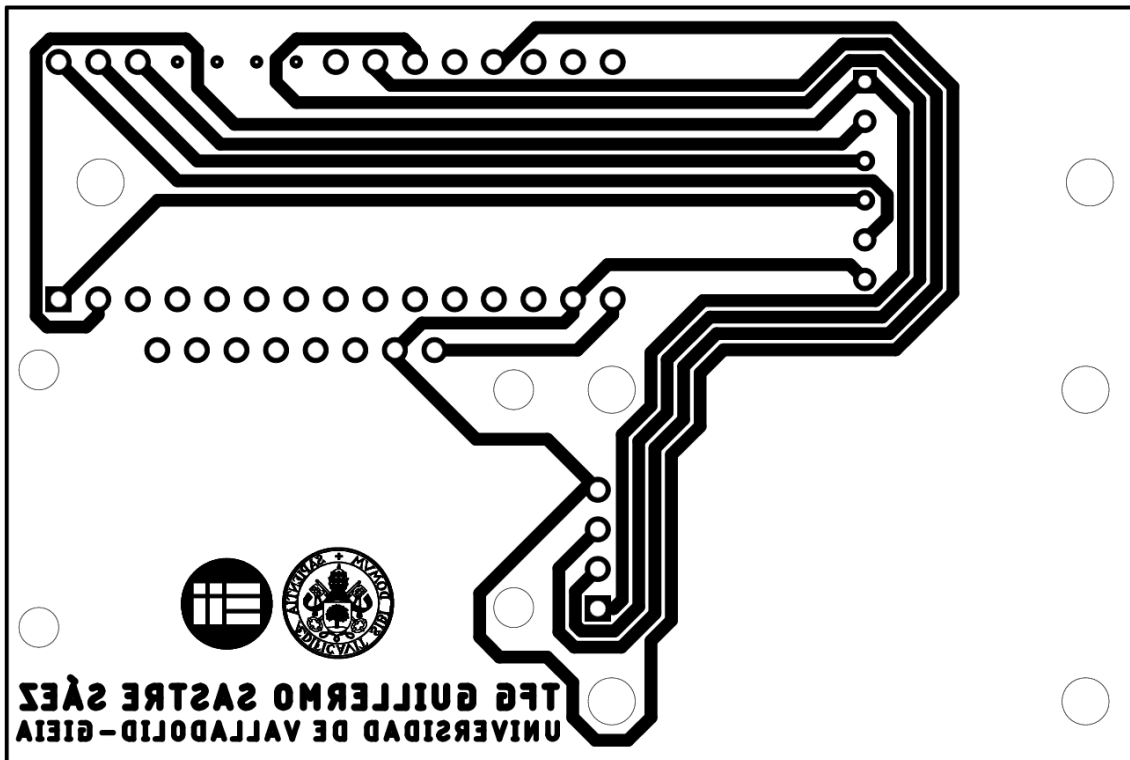
❖ Cara de soldadura (Bottom):

Figura 47. Fotolito cara de soldadura (elaboración propia)

Los fotolitos del circuito impreso son esenciales para la fabricación de una PCB porque actúan como máscaras que permiten transferir el diseño del circuito al material de la placa. Este proceso asegura que las trazas de cobre y otros elementos del circuito sean creados con precisión, lo cual es crucial para el correcto funcionamiento y fiabilidad del dispositivo electrónico final.

#### 4.2.2. FICHEROS DE FABRICACIÓN

Por otra parte, los ficheros generados a partir del diseño y utilizando la utilidad CAM Processor de EAGLE se describen y enumeran a continuación:

- ❖ Ficheros Gerber (.gbr): contienen la información gráfica de cada capa de la PCB y son utilizados por las máquinas de fabricación para crear las diferentes capas del circuito impreso.
- ❖ Fichero de taladrado (.xln): contiene las coordenadas y tamaños de los agujeros que deben ser perforados en la PCB. Estos agujeros pueden ser para vías, pads de componentes o montaje. El fichero incluye información sobre el diámetro de cada agujero y su posición en el plano X-Y de la PCB.
- ❖ Lista de materiales o BOM (.txt): contiene todos los componentes necesarios para ensamblar la PCB, incluyendo detalles como la referencia del componente, su descripción y la cantidad existente de cada uno.



En este caso, como no existe la figura de un fabricante externo, sólo se ha hecho uso del fichero de taladrado para crear los agujeros necesarios de manera automática utilizando la máquina CNC en el laboratorio.

### 4.3. DISEÑO DE LA CARCASA

El uso de una carcasa que proteja la PCB y sus componentes es indispensable para un dispositivo de estas características, pensado para utilizarse en espacios exteriores. A continuación, se enumeran algunos de los motivos que justifican el empleo de una carcasa:

1. Protección contra elementos ambientales (agua, polvo, etc.): al utilizarse en actividades deportivas al aire libre, el dispositivo debe ser resistente al agua y al polvo. Una carcasa bien diseñada puede proporcionar esta protección, asegurando que el dispositivo funcione correctamente en diversas condiciones meteorológicas.
2. Robustez y durabilidad: durante actividades deportivas, es común que el dispositivo esté sujeto a golpes, caídas y vibraciones. La carcasa protege la PCB y otros componentes internos contra estos impactos, aumentando la durabilidad del dispositivo.
3. Portabilidad y ergonomía: un diseño personalizado de la carcasa permite optimizar el tamaño y la forma del dispositivo, haciéndolo más cómodo de llevar durante actividades deportivas.
4. Estética y funcionalidad: una carcasa diseñada específicamente puede mejorar la apariencia del dispositivo, haciéndolo más atractivo para los usuarios. Además, puede incorporar características funcionales adicionales, como clips para sujeción o puntos de anclaje.

En este caso, el diseño de la carcasa se realizó a partir de la placa de circuito impreso, de manera que se ajustase a ésta, y no viceversa.

Para ello, se tuvieron en cuenta los siguientes aspectos:

- ❖ Creación de orificios correspondientes a los puertos de salida del dispositivo. Estos incluyen el conector micro-USB del Arduino Nano Every, el conector micro-USB del cargador LiPo, la entrada al módulo lector de tarjetas microSD y el interruptor de encendido/apagado del dispositivo.
- ❖ Espacio para contener la batería LiPo.
- ❖ División de la carcasa en dos mitades: parte superior y parte inferior, permitiendo crear un mecanismo de apertura y cierre. Este último consistirá en 4 tornillos situados en las esquinas de la carcasa.
- ❖ Creación de pequeños huecos en la base de la carcasa para la inclusión de apoyos de plástico. Estos tienen la doble función de “unir” la PCB a la carcasa a la vez que sujetan sus componentes.





Inicialmente, se consideró comprar una caja estándar con espacio suficiente para albergar la PCB y la batería LiPo. Sin embargo, esta opción apenas dejaba lugar para la personalización y dificultaba el cumplimiento de los requisitos mencionados anteriormente. En consecuencia, se optó por diseñar una carcasa desde cero utilizando el programa Fusion 360. Adicionalmente, y como ya se mencionó al inicio del apartado 4, la posibilidad de importar el modelo 3D de la PCB (Figura 48) diseñada facilitó enormemente la creación de la envolvente.

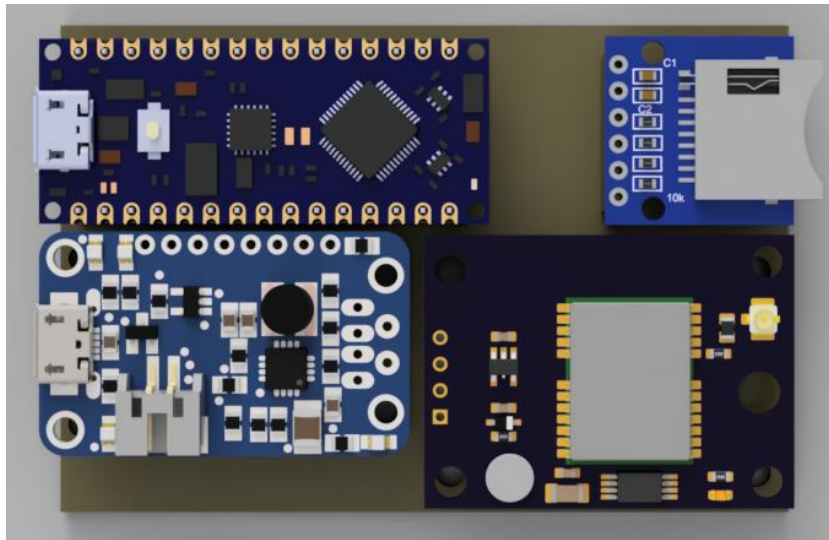


Figura 48. Modelo 3D de la PCB diseñada (elaboración propia)

De esta manera, las dimensiones finales de la carcasa diseñada son 89 x 82,50 x 27 mm (largo x ancho x alto) y su aspecto se muestra en la Figura 49:



Figura 49. Modelo 3D de la carcasa final (elaboración propia)





# CAPÍTULO 5: FABRICACIÓN Y AJUSTE DEL DISPOSITIVO





## 5. FABRICACIÓN Y AJUSTE DEL DISPOSITIVO

Después de haber diseñado la placa de circuito impreso y la carcasa, se procedió a su fabricación. Cabe destacar que ambas se han llevado a cabo en las instalaciones de la Escuela de Ingenierías Industriales de Valladolid. La PCB se fabricó en el laboratorio de la asignatura “Métodos y Herramientas de Diseño Electrónico”, mientras que la carcasa se produjo utilizando una de las máquinas de impresión 3D de las que dispone la Escuela.

### 5.1. FABRICACIÓN DE LA PLACA DE CIRCUITO IMPRESO

A pesar de existir la posibilidad de enviar a fabricar la placa de circuito impreso diseñada a una empresa especializada en ello, se consideró que siendo un Trabajo de Fin de Grado, ser partícipe de la fabricación de la PCB, suponía una excelente oportunidad para repasar y profundizar los conocimientos adquiridos en otras asignaturas del grado universitario.

De esta manera, la fabricación de la PCB se realizó en el laboratorio siguiendo todas las normas de seguridad requeridas y de acuerdo con el siguiente orden:

#### Taladrado de la PCB.

Como ya se mencionó con anterioridad, se utilizó la máquina CNC del laboratorio y el fichero de taladrado generado con EAGLE. Una vez taladrada la placa, se comprobó que las patillas de los pines hembra, sobre los que van montados los componentes, entraban por los agujeros sin forzar.

#### Insolado de la PCB de una cara

Se imprimió el fotolito correspondiente a la cara de soldadura (*Figura 47*) en papel vegetal y se pegó a la PCB en contacto con la fotorresina, después de haber retirado el papel protector adherido a la placa de circuito impreso. La placa con el fotolito se colocó en la insoladora debajo del cristal, con la cara a insolar hacia arriba. Tras asegurarse de que el vacío se hacía correctamente bajo el cristal de la insoladora, se esperó un tiempo de insolado de tres minutos.

#### Revelado de las pistas del circuito

Una vez realizado el insolado, se retiró el fotolito, se introdujo la placa en una cubeta con 50 ml de revelador positivo para placas de circuito impreso. Se agitó ligeramente hasta que las pistas del circuito impreso se apreciaron claramente.

#### Atacado químico de las pistas del circuito

Para el grabado del cobre, se introdujo la placa en una cubeta con el agente atacador compuesto por una mezcla de 20 ml de agua, 20 ml de ácido clorhídrico al 35% y 20 ml de agua oxigenada. Se agitó ligeramente hasta que se apreció burbujeo en la mezcla, momento en el cual se retiró la placa. Se lavó la placa con abundante agua para detener el proceso de grabado y, usando papel y alcohol, se eliminó totalmente



la capa de fotorresina que cubría las pistas. Todo el proceso de grabado se realizó en una campana extractora.

#### Comprobación de continuidades y cortocircuitos

A continuación, se comprobó la continuidad de todas la pistas y el aislamiento entre ellas de forma visual y con el polímetro, de acuerdo con el esquema del circuito (Figura 42).

Inicialmente las pistas correspondientes a la alimentación de 3,3V y el pin RX del receptor GPS aparecieron cortocircuitadas. Esto se solventó rompiendo el punto de conexión mecánicamente.

#### Montaje y soldadura de los componentes en la PCB

Primero se cortó la placa de acuerdo con sus dimensiones. Como ya se comentó anteriormente, en lugar de soldar directamente los componentes a la placa, se decidió montarlos sobre pines hembra soldados a la placa. De esta forma, se pueden manipular con facilidad para hacer pruebas o para futuras aplicaciones.

Finalmente, la placa de circuito impreso fabricada se muestra en las siguientes imágenes (Figura 50, Figura 51 y Figura 52):

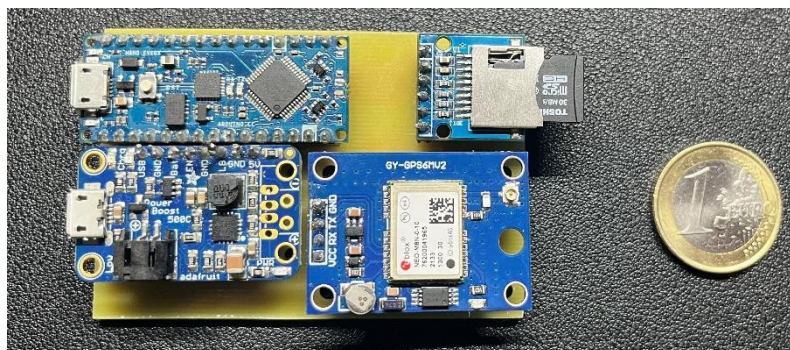


Figura 50. PCB fabricada con componentes (cara superior) (elaboración propia)

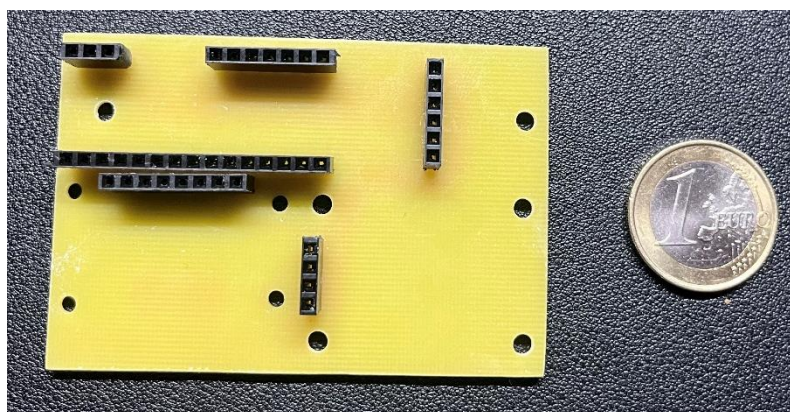


Figura 51. PCB fabricada sin componentes (cara superior) (elaboración propia)



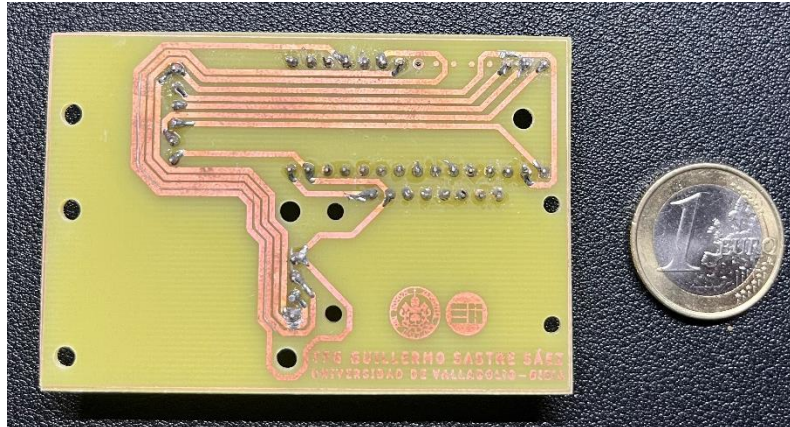


Figura 52. PCB fabricada (cara inferior) (elaboración propia)

### 5.2. FABRICACIÓN Y MONTAJE DE LA CARCASA

Tal y como se mencionó anteriormente, para fabricar la carcasa se utilizó una de las máquinas de impresión 3D situadas en la Escuela, empleando PLA (ácido poliláctico) como material. Esta alternativa se ha considerado la más adecuada para corregir posibles errores y ajustar el diseño de la carcasa al equipo. Además, era la opción más barata.

No obstante, cabe destacar que este método de fabricación no sería adecuado para una producción del dispositivo a gran escala. Es un proceso lento (varias horas para fabricar una única carcasa), poco exacto (aquellas partes que requieren de un mayor detalle no quedan exactamente igual al diseño) y el material utilizado no ofrece las mejores prestaciones en términos de robustez y durabilidad.

Por lo tanto, en una hipotética salida al mercado del producto, sería obligatorio considerar otros métodos de fabricación que ofrezcan mayor eficiencia y calidad. La transición a técnicas como el moldeo por inyección, el moldeo por compresión, la fundición a presión o la fabricación en serie mediante CNC, permitiría la fabricación eficiente y rentable de carcasas para el dispositivo en un escenario de producción comercial. En ese supuesto caso, la elección final del método de fabricación dependería de diversos factores, incluyendo el volumen de producción, los requisitos de precisión y el presupuesto disponible.

En cualquier caso, la carcasa fabricada montada sobre el dispositivo final se muestra en la *Figura 53*:



Figura 53. Vista superior de la carcasa final (elaboración propia)

### 5.3. AJUSTE Y VERIFICACIÓN DEL EQUIPO

Una vez montada la PCB sobre la carcasa, se han soldado los terminales del interruptor de tipo Rocker a los pines EN y GND del cargador LiPo para controlar el encendido y apagado del dispositivo, como ya se mencionó en el apartado 3.3.5. Además, la batería LiPo y la antena del módulo GPS se han pegado con silicona a la base de la carcasa para una mayor compacidad del prototipo.

En la *Figura 54* se exhibe el aspecto final del montaje de los componentes sobre la carcasa:

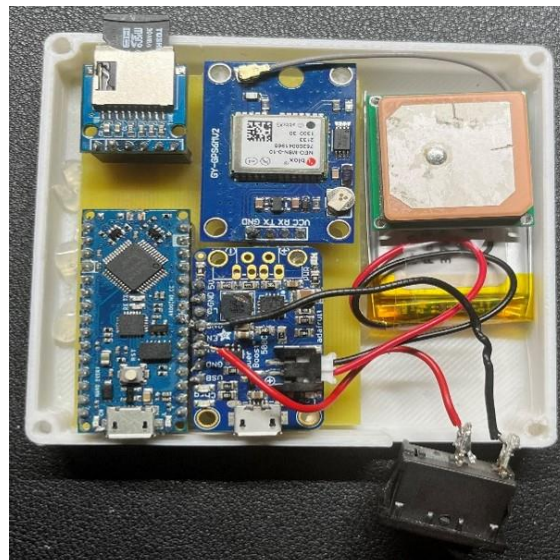


Figura 54. Componentes del localizador GPS montados sobre la carcasa final (elaboración propia)

Para finalizar, se ha verificado que el prototipo funciona correctamente y, a continuación, se proporcionan detalles acerca de los códigos LED de diagnóstico del dispositivo (*Tabla 9*):





Tabla 9. Códigos LED de diagnóstico del dispositivo (elaboración propia)

Códigos LED		
Estado localizador GPS	Comportamiento del LED	Componente
Dispositivo encendido	Verde permanente	Arduino Nano Every
Batería $\geq$ 20%	Azul permanente	Cargador LiPo
Batería $<$ 20% (necesidad de recarga)	Rojo permanente	Cargador LiPo
Batería recargándose	Naranja permanente	Cargador LiPo
Batería cargada al 100 %	Verde permanente	Cargador LiPo
Receptor GPS calibrado y enviando datos	Azul parpadeando	Módulo GPS

### 5.4. CORRECCIÓN DE ERRORES

Tanto para la PCB como para la carcasa fue necesario diseñar y fabricar más de un prototipo, ya que las primeras versiones presentaban ciertos errores críticos que debían ser corregidos.

A continuación, se detallan estos errores y las soluciones implementadas.

#### 5.4.1. ERRORES PCB

##### Separación de conectores micro-USB al borde de la placa

La primera versión de la PCB tenía una separación excesiva entre los conectores micro-USB del Arduino y del cargador LiPo y el borde de la placa. Esto dificultaba la conexión de cables cuando la placa estaba montada en la carcasa. Para solventar esto, se redujo la separación entre los conectores y el borde de la placa en la versión final.

##### Accesibilidad del lector de microSD

El orificio para el lector de microSD permitía la inserción de la tarjeta, pero resultaba muy difícil extraerla. Como solución, se ajustó el diseño para mejorar el acceso a la tarjeta microSD.

##### Reducción del tamaño de la PCB

Además de los ajustes mencionados, se aprovechó la oportunidad para reducir el tamaño de la placa de circuito impreso, obteniendo una nueva versión más compacta y funcional.

A continuación, se muestran imágenes de aquella primera versión de la PCB (*Figura 55*, *Figura 56* y *Figura 57*) donde se puede apreciar mejor los defectos ya mencionados:

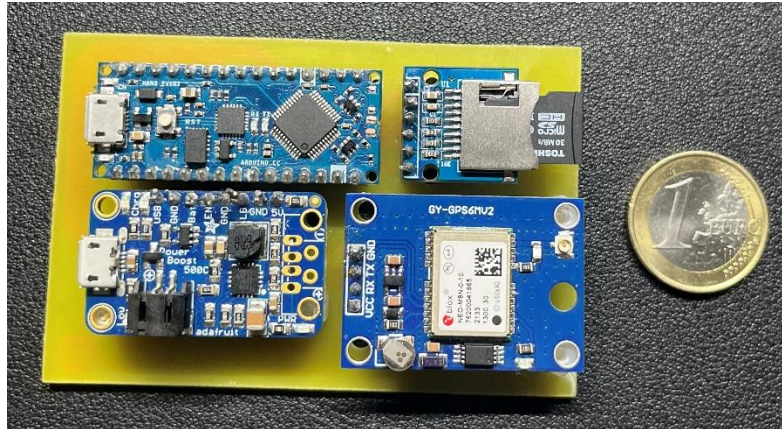


Figura 55. Vista superior. Primera versión de la PCB con componentes (elaboración propia)

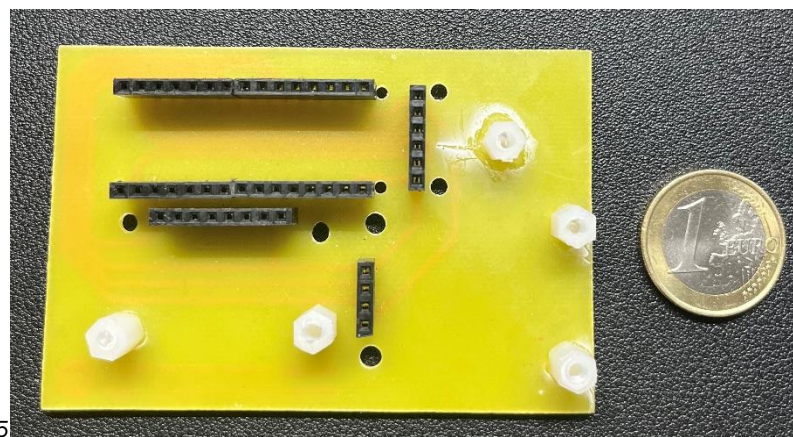


Figura 56. Vista superior. Primera versión de la PCB sin componentes (elaboración propia)

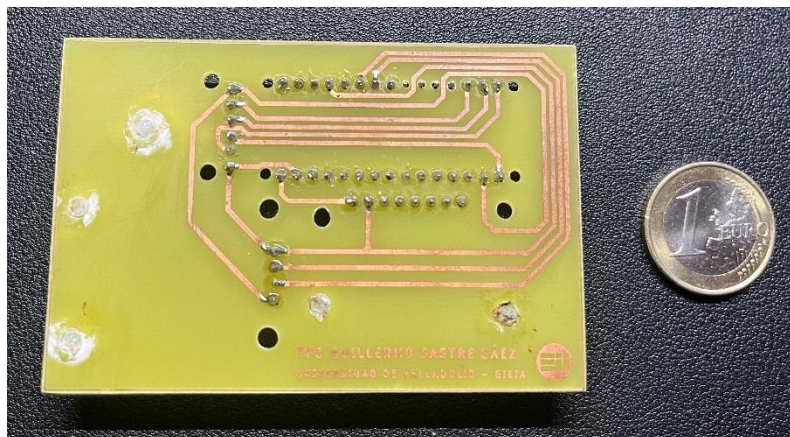


Figura 57. Vista inferior. Primera versión de la PCB (elaboración propia)

#### 5.4.2. ERRORES CARCASA

Por su parte, el primer diseño de la carcasa también llevaba asociados numerosos errores que se corrigieron en versiones posteriores.

##### Mecanismo de apertura y cierre

Primero, el mecanismo de cierre de la carcasa se había decidido que iban dos pestañas situadas en los extremos que iban a ir encajadas a presión en sus



respectivos agujeros. No obstante, la realidad fue que el tamaño de estas pestañas era muy pequeño y el material utilizado demasiado frágil, por lo que aquellas se rompieron con facilidad después de unos pocos intentos de apertura y cierre de la caja. Para solucionar este problema, se intentó utilizar unas piezas de madera junto con un tirafondos para tratar de cerrar la carcasa. Esto no terminó de funcionar según se esperaba ya que no permitía el cierre completo del dispositivo.

En definitiva, se rediseñó el mecanismo de cierre utilizando cuatro tornillos, uno en cada esquina de la carcasa, para mayor robustez y fiabilidad.

### Altura de los orificios de los conectores micro-USB

Debido a un error de cálculo, la altura de los orificios de los conectores micro-USB había quedado 1,5 mm por debajo de la posición correcta, haciéndolos inutilizables. Para ello, se ajustó 1,5 mm la altura estos orificios.

### Problema con tarjeta microSD

La tarjeta microSD sobresalía mucho del módulo lector, aumentando el riesgo de daño. Para solventarlo, se extendió la pared lateral de la carcasa de manera que la tarjeta SD no sobresaliera tanto.

### Soportes para la PCB

Inicialmente se había optado por crear unos soportes de PLA en la base de la carcasa que sirviesen para “unir” la placa con la carcasa a la vez que sujetaban sus componentes. De nuevo, el problema fue que estos soportes tenían un diámetro muy pequeño (inferior a 3 mm) por lo que era muy probable que se rompieran con el uso del dispositivo. Esto se corrigió utilizando apoyos de plástico adheridos a la carcasa, de modo que ofrecieran una mayor robustez.

### Orificio para el interruptor de encendido y apagado

Por último, el orificio donde destinado al interruptor de encendido y apagado (*Figura 31*) era demasiado pequeño. Por esta razón, se amplió el orificio en la versión posterior para permitir el encaje adecuado del interruptor.

Con el objetivo de proporcionar una descripción más gráfica de estos defectos, se muestran las siguientes imágenes de la primera versión de la carcasa (*Figura 58*, *Figura 59* y *Figura 60*):





Figura 58. Vista superior exterior. Primera versión de la carcasa sin componentes (izquierda) y con componentes (derecha) (elaboración propia)

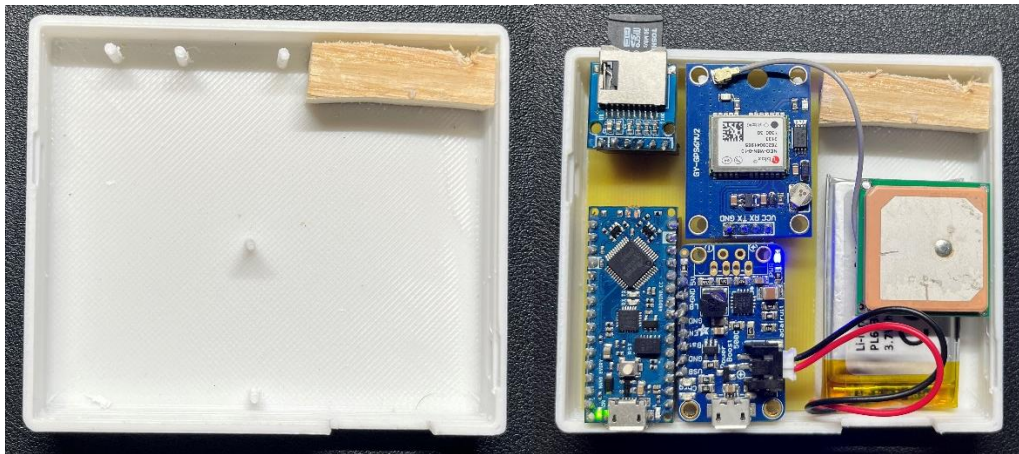


Figura 59. Vista superior interior.-Primera versión de la carcasa sin componentes (izquierda) y con componentes (derecha) (elaboración propia)

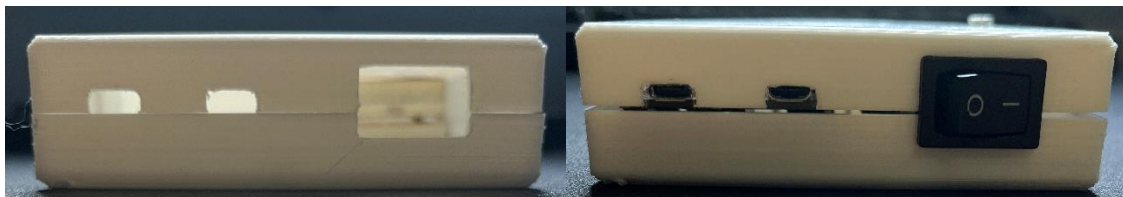


Figura 60. Vista lateral. Primera versión de la carcasa sin componentes (izquierda) y con componentes (derecha) (elaboración propia)



# CAPÍTULO 6: TRATAMIENTO Y VISUALIZACIÓN DE DATOS





## 6. TRATAMIENTO Y VISUALIZACIÓN DE DATOS

En este apartado se describe el tratamiento de los datos almacenados en la tarjeta microSD y se argumenta por qué este proceso es importante de cara a visualizar la información obtenida de las rutas realizadas.

### 6.1. TRATAMIENTO DE LOS DATOS OBTENIDOS

Tal y como se expuso en el apartado 3.4.5, después de almacenar la información correspondiente a cada ruta realizada en un fichero de texto distinto, el siguiente paso consiste en unificarla, utilizando para ello un pequeño programa desarrollado en Python con la librería *Pandas* de nombre *unificarDatos.py*. Según se explicó en ese apartado, el resultado de dicho proceso es un único archivo Excel de nombre *historicoDATOS.xlsx*.

A parte de la unificación, también se comentó que se iba a emplear dicho programa para añadir dos columnas nuevas de nombre *tipoRuta* y *rutaID*.

#### Adición de columna *tipoRuta*

Anteriormente, ya se comentó que el localizador GPS estaba pensado para ser utilizado en distintas actividades deportivas al aire libre, tales como correr, pasear o montar en bicicleta, pero también para registrar datos ligados a recorridos en vehículos como una moto o un coche. Por esta razón, se ha considerado útil añadir una columna que indique el tipo de ruta que se ha realizado. De este modo, cada vez que el usuario ejecute el programa *unificarDatos.py*, y, si hay rutas nuevas, aparecerá un cuadro de diálogo en el cual se pregunte al usuario por el tipo de actividad que ha realizado, pudiendo elegir entre las opciones de “Andar”, “Correr”, “Ciclismo” u “Otras”. En esta última se incluyen aquellas rutas que se han realizado utilizando un vehículo a motor como un coche, una moto, etc.

#### Adición de columna *rutaID*

Al combinar todas las rutas en un mismo archivo, esta columna permite diferenciar unas de otras, algo muy útil a la hora de visualizar los parámetros y estadísticas asociados a estas. Es decir, al añadir esta columna se pretende dar un identificador a cada ruta. Este identificador vendrá dado por el número de ruta, de manera que, por ejemplo, todas las filas de datos que pertenezcan a la sexta ruta tendrán asociadas una columna llamada *rutaID* con el número 6.

#### 6.1.1. CREACIÓN DE ARCHIVO EJECUTABLE

Además, como también se mencionó en el apartado 3.4.5 y con la finalidad de facilitar este proceso al propietario del dispositivo, se ha convertido el archivo *unificarDatos.py* en un archivo ejecutable *unificarDatos.exe*.

A continuación, se describe este procedimiento de unificación de los archivos de texto utilizando *unificarDatos.exe* (*Figura 61* y *Figura 62*):



Nombre	Fecha de modificación	Tipo	Tamaño
build	12/06/2024 13:36	Carpeta de archivos	
copiasSeguridad	10/06/2024 18:47	Carpeta de archivos	
dist	12/06/2024 13:37	Carpeta de archivos	
pruebas	12/06/2024 12:53	Carpeta de archivos	
DATOS_1	10/06/2024 17:46	Documento de texto	3 KB
DATOS_2	10/06/2024 17:46	Documento de texto	2 KB
DATOS_3	10/06/2024 17:46	Documento de texto	45 KB
DATOS_4	10/06/2024 17:45	Documento de texto	11 KB
DATOS_5	10/06/2024 17:45	Documento de texto	3 KB
DATOS_6	10/06/2024 17:44	Documento de texto	19 KB
historicoDATOS	12/06/2024 13:11	Hoja de cálculo de M...	143 KB
unificarDatos	12/06/2024 13:37	Aplicación	34.843 KB
unificarDatos	12/06/2024 13:24	Archivo de origen Py...	6 KB
unificarDatos.spec	12/06/2024 13:36	Archivo SPEC	1 KB

Figura 61. Carpeta con `unificarDatos.py` y 6 ficheros de texto a unificar (elaboración propia)

```
-----  
Bienvenido al programa: 'Unificación de Rutas'  
-----  
Este programa te permite convertir y unir los ficheros de texto (.TXT) de nombre 'DATOS_.TXT' que contienen información  
acerca de las rutas realizadas  
El resultado será un archivo Excel (.xlsx) de nombre 'historicoDatos.xlsx'  
  
Se encontraron 6 ficheros 'DATOS_.TXT' disponibles para convertir y unir.  
¿Deseas convertir y unir estos archivos? (S/N): S  
Para el archivo DATOS_1.TXT, ¿qué tipo de actividad ha realizado? (Andar/Correr/Ciclismo/Otras): Andar  
Para el archivo DATOS_2.TXT, ¿qué tipo de actividad ha realizado? (Andar/Correr/Ciclismo/Otras): Correr  
Para el archivo DATOS_3.TXT, ¿qué tipo de actividad ha realizado? (Andar/Correr/Ciclismo/Otras): Correr  
Para el archivo DATOS_4.TXT, ¿qué tipo de actividad ha realizado? (Andar/Correr/Ciclismo/Otras): Ciclismo  
Para el archivo DATOS_5.TXT, ¿qué tipo de actividad ha realizado? (Andar/Correr/Ciclismo/Otras): Otras  
Para el archivo DATOS_6.TXT, ¿qué tipo de actividad ha realizado? (Andar/Correr/Ciclismo/Otras): Andar  
Operación realizada exitosamente.  
Pulsa Enter para salir  
-
```

Figura 62. Programa 'Unificación de Rutas' (elaboración propia)

Se aprecia que el primer paso consiste en situar en una misma carpeta todos los ficheros de texto `DATOS_N.txt` que se pretenden unificar junto con el archivo ejecutable `unificarDatos.exe` (Figura 61). A continuación, y de manera intuitiva, será necesario introducir por pantalla las respuestas correspondientes al cuadro de diálogo que aparece (Figura 62).

Por otra parte en la Figura 63, se observa que si se vuelve a ejecutar `unificarDatos.exe` y no hay ninguna ruta nueva, aparecerá un mensaje en pantalla indicando que no hay ficheros `DATOS_N.TXT` disponibles para unificar.

```
-----  
Bienvenido al programa: 'Unificación de Rutas'  
-----  
Este programa te permite convertir y unir los ficheros de texto (.TXT) de nombre 'DATOS_.TXT' que contienen información  
acerca de las rutas realizadas  
El resultado será un archivo Excel (.xlsx) de nombre 'historicoDatos.xlsx'  
  
No hay ficheros 'DATOS_.TXT' disponibles para convertir y unir.  
Pulsa Enter para salir
```

Figura 63. Programa 'Unificación de Rutas' sin rutas nuevas (elaboración propia)





Sin embargo, si se añade una ruta nueva (fichero *DATOS\_7.TXT*), el programa sí que detecta que hay un fichero nuevo disponible para unificar y repite el mismo procedimiento pero sólo para este. (*Figura 64* y *Figura 65*)

Nombre	Fecha de modificación	Tipo	Tamaño
build	12/06/2024 13:36	Carpeta de archivos	
copiasSeguridad	10/06/2024 18:47	Carpeta de archivos	
dist	12/06/2024 13:37	Carpeta de archivos	
pruebas	12/06/2024 12:53	Carpeta de archivos	
DATOS_1	10/06/2024 17:46	Documento de texto	3 KB
DATOS_2	10/06/2024 17:46	Documento de texto	2 KB
DATOS_3	10/06/2024 17:46	Documento de texto	45 KB
DATOS_4	10/06/2024 17:45	Documento de texto	11 KB
DATOS_5	10/06/2024 17:45	Documento de texto	3 KB
DATOS_6	10/06/2024 17:44	Documento de texto	19 KB
DATOS_7	10/06/2024 17:46	Documento de texto	3 KB
historicoDATOS	12/06/2024 13:53	Hoja de cálculo de M...	73 KB
unificarDatos	12/06/2024 13:37	Aplicación	34.843 KB
unificarDatos	12/06/2024 13:24	Archivo de origen Py...	6 KB
unificarDatos.spec	12/06/2024 13:36	Archivo SPEC	1 KB

Figura 64. Carpeta con *unificarDatos.py* y un nuevo fichero de texto a unificar (*elaboración propia*)

```

-----
Bienvenido al programa: 'Unificación de Rutas'
-----
Este programa te permite convertir y unir los ficheros de texto (.TXT) que contienen información
acerca de las rutas realizadas
El resultado será un archivo Excel (.xlsx) de nombre 'historicoDatos.xlsx'

Se encontraron 1 ficheros 'DATOS_.TXT' disponibles para convertir y unir.
¿Deseas convertir y unir estos archivos? (S/N): S
Para el archivo DATOS_7.TXT, ¿qué tipo de actividad ha realizado? (Andar/Correr/Ciclismo/Otras): Otras
Operación realizada exitosamente.
Pulsa Enter para salir

```

Figura 65. Programa 'Unificación de Rutas' con una nueva ruta (*elaboración propia*)

Es importante añadir que para que el programa asigne un índice a cada ruta, los ficheros de texto deben estar almacenados empezando por *DATOS\_1.TXT* y de manera consecutiva (*DATOS\_1.TXT*, *DATOS\_2.TXT*, *DATOS\_3.TXT*, etc.)

De este modo, el aspecto final del archivo Excel resultado (*historicoDATOS.xlsx*) se muestra en la *Figura 66*:

Latitud	Longitud	stancia (km)	ancia total	Tiempo (s)	mpo total	bcidad (km)	mo (min/km)	Altitud (m)	endiente(%)	nte Absol	Hora	Fecha	tipoRuta	rutaID
41.64121	-4.75779	0.061	0.061	322	322	0.68	19.79	719.59	-0.51	0.51	12:05:22	12/03/2024	Andar	1
41.64064	-4.76056	0.0018	0.0052	10	40	0.65	109.43	707.8	-127.78	127.78	18:25:12	17/04/2024	Correr	2
41.64075	-4.76055	0.0057	0.0273	10	60	2.07	32.66	732.8	-89.47	89.47	12:40:24	23/04/2024	Correr	3
41.63773	-4.75516	0.0597	1.0289	10	690	21.48	7.18	755.9	-3.52	3.52	17:59:07	30/04/2024	Ciclismo	4
41.64069	-4.76042	0.0058	0.3876	10	371	2.08	17	730.1	41.38	41.38	20:55:19	07/05/2024	Otras	5
41.64085	-4.76077	0.0026	0.2537	9	557	1.02	28.58	736.7	-31.33	31	12:15:42	23/05/2024	Andar	6
41.64128	-4.75782	0.2408	6.0714	304	7211	71.9	19.79	720.75	-0.14	0.14	14:00:11	30/05/2024	Otras	7

Figura 66. Ejemplo de *historicoDATOS.xlsx* (*elaboración propia*)

En esta *Figura 66* sólo se muestra una fila de cada ruta a modo de ejemplo, dado que el archivo Excel completo es mucho más extenso.



## 6.2. VISUALIZACIÓN DE LOS DATOS TRATADOS

Como ya se ha mencionado anteriormente, se emplea la aplicación Power BI de Microsoft como herramienta principal para la visualización y análisis de los datos recopilados durante las rutas realizadas por el portador del localizador GPS. Esta permite crear informes interactivos donde poder analizar los diversos parámetros de las rutas, así como calcular estadísticas a partir de ellos.

Un aspecto fundamental de Power BI radica en que para crear este tipo de informes, es necesario conectarse a un origen de datos que sirva como fuente primaria de información. Como es de suponer, en este caso el origen de datos es el archivo Excel *historicoDATOS.xlsx*, cuyo procedimiento de obtención ya se ha explicado en el apartado 6.1.

De esta manera, a partir del contenido de este archivo (observar *Figura 66*), se creará un informe de 3 páginas

### Página 1: Parámetros, gráficas y estadísticas

En la página 1, aparecerán los parámetros correspondientes a la ruta realizada:

- ❖ Hora de comienzo del recorrido
- ❖ Hora de finalización del recorrido
- ❖ Tiempo total empleado en formato horas:minutos:segundos
- ❖ Distancia total recorrida en kilómetros
- ❖ Velocidad máxima y mínima alcanzada en kilómetros por hora
- ❖ Altitud máxima y mínima alcanzada en metros
- ❖ Pendiente máxima y mínima del recorrido en tanto por ciento

La representación de estos parámetros en gráficas:

- ❖ Perfil de distancia: en él se representa la distancia recorrida en función de la hora y en función del tiempo empleado en realizar el recorrido.
- ❖ Perfil de velocidad: en él se representa la velocidad alcanzada en función de la hora, en función del tiempo empleado y en función de la distancia recorrida.
- ❖ Perfil de altitud: en él se representa la altitud del recorrido en función de la hora, en función del tiempo empleado y en función de la distancia recorrida.
- ❖ Perfil de pendiente: en él se representa la pendiente del recorrido en función de la hora, en función del tiempo y en función de la distancia recorrida.

Y, además, aprovechando algunas de las utilidades ofrecidas por Power BI, esta primera hoja también contiene las siguientes estadísticas, calculadas a partir de los parámetros mencionados anteriormente:

- ❖ Ritmo medio en minutos por kilómetro
- ❖ Velocidad media en kilómetros por hora



- ❖ Altitud media en metros
- ❖ Pendiente media en tanto por ciento

En la *Tabla 10*, se muestran todos estos datos:

*Tabla 10. Datos asociados a las rutas realizadas que aparecen en el informe de Power BI (elaboración propia)*

Parámetros	Gráficas	Estadísticas
Fecha	Perfil de distancia	Ritmo medio (min/km)
Tipo de actividad	Perfil de velocidad	Velocidad media (km/h)
Hora de inicio	Perfil de altitud	Altitud media (m)
Hora de fin	Perfil de pendiente	Pendiente media (%)
Tiempo total		
Distancia total (km)		
Velocidad máxima y mínima (km/h)		
Altitud máxima y mínima		
Pendiente máxima y mínima (%)		

Asimismo, en la *Figura 67*, se puede observar un ejemplo de esta primera página del informe. Se trata de un informe dinámico donde el usuario puede seleccionar el dato que quiere analizar, el número de ruta, cambiar entre gráficas, etc.



*Figura 67. Página 1 del informe - Parámetros, estadísticas y gráficas (elaboración propia)*

Resulta interesante destacar que gracias a la visualización de los datos relativos a las rutas realizadas, se puede apreciar con claridad la falta de exactitud del sensor GPS NEO-M8N utilizado en la medida de la altitud, un problema al que ya se hizo referencia en el apartado 3.6.1. Este error en la medición afecta principalmente a los perfiles de altitud y pendiente dando como resultado valores muy dispares que no se corresponden con la realidad (*Figura 68*).



Figura 68. Perfil de altitud (arriba) y pendiente (abajo) obtenido en una de las rutas realizadas (elaboración propia)

Con el fin de dotar de validez a este hecho, se ha llevado a cabo una comparación entre 20 valores de altitud obtenidos a partir del localizador GPS desarrollado y sus correspondientes reales. Para conocer los valores reales se ha utilizado el sistema de información geográfica Google Earth. El resultado se muestra en la *Figura 69*.



Figura 69. Comparativa entre el perfil de altitud obtenido utilizando el localizador GPS (arriba) y utilizando valores reales (abajo) (elaboración propia)

Se observa que las discrepancias entre ambas mediciones de altitud son muy notables. En especial, llama la atención la enorme fluctuación en la altitud de la primera gráfica, variando en un rango de aproximadamente 100 metros (entre 712 m y 808 m). Esto no ocurre en la segunda gráfica, donde el rango de altitud se sitúa entre los 717 m y 722 m, es decir, 5 metros de diferencia, como es lógico al tratarse de una situación real.

### Página 2: Trazado del recorrido

Esta segunda página tiene como objeto visual principal un mapa interactivo donde aparece dibujado el trazado de la ruta realizada (ver Figura 70).





Figura 70. Página 2 del informe – Trazado del recorrido (elaboración propia)

Las flechas que se observan corresponden a los instantes de tiempo en los que el localizador GPS ha registrado la ubicación del usuario. Manteniendo el ratón del ordenador encima de una de estas flechas, aparece información relativa a su instante de tiempo correspondiente, según se aprecia en la *Figura 71*.

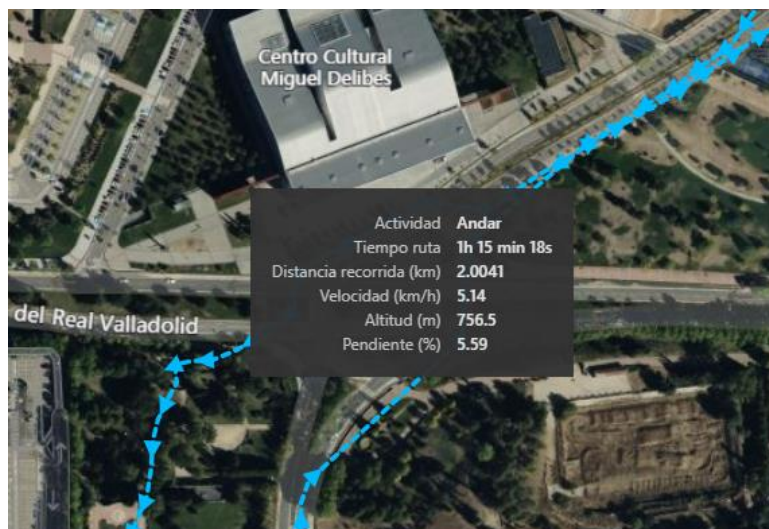


Figura 71. Detalle trazado del recorrido (elaboración propia)

### Página 3: Histórico de rutas

En la tercera y última página del informe aparece información relativa al conjunto de rutas, como es el caso del número de rutas realizadas, la distancia total recorrida, el tiempo total empleado, el número de rutas por actividad, la distancia por actividad y el tiempo por actividad.

La *Figura 72* muestra un ejemplo de esta página:



Figura 72. Página 3 del informe - Histórico de rutas (elaboración propia)

Resulta interesante añadir que ha sido posible realizar este “histórico” de las rutas realizadas gracias a que cada ruta está identificada con un tipo de actividad y un índice. De ahí surge la importancia de la adición de las columnas tipoRuta y rutaID al archivo *historicoDATOS.xlsx*, a la que ya se hacía referencia en el apartado 3.4.5.

### 6.2.1. CREACIÓN DE PLANTILLA DE INFORME

Para finalizar y de cara a facilitar la visualización y el análisis de estos datos al usuario, se ha creado una plantilla a partir de este informe nombrada como *plantillaInformeRutas.pbix* (ver Figura 73).

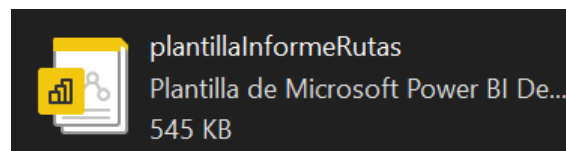


Figura 73. Plantilla para informe de rutas (elaboración propia)

De esta forma, el procedimiento para que un propietario cualquiera del localizador GPS pueda crear su propio informe en Power BI deberá proceder según se describe a continuación:

1. Abrir el fichero *plantillaInformeRutas.pbix*
2. Seleccionar la ruta del archivo que va a constituir el origen de datos del informe. Para ello, bastará con introducir su ruta de directorios en el cuadro de diálogo que aparecerá al abrir el fichero *plantillaInformeRutas.pbix*. (Figura 74)

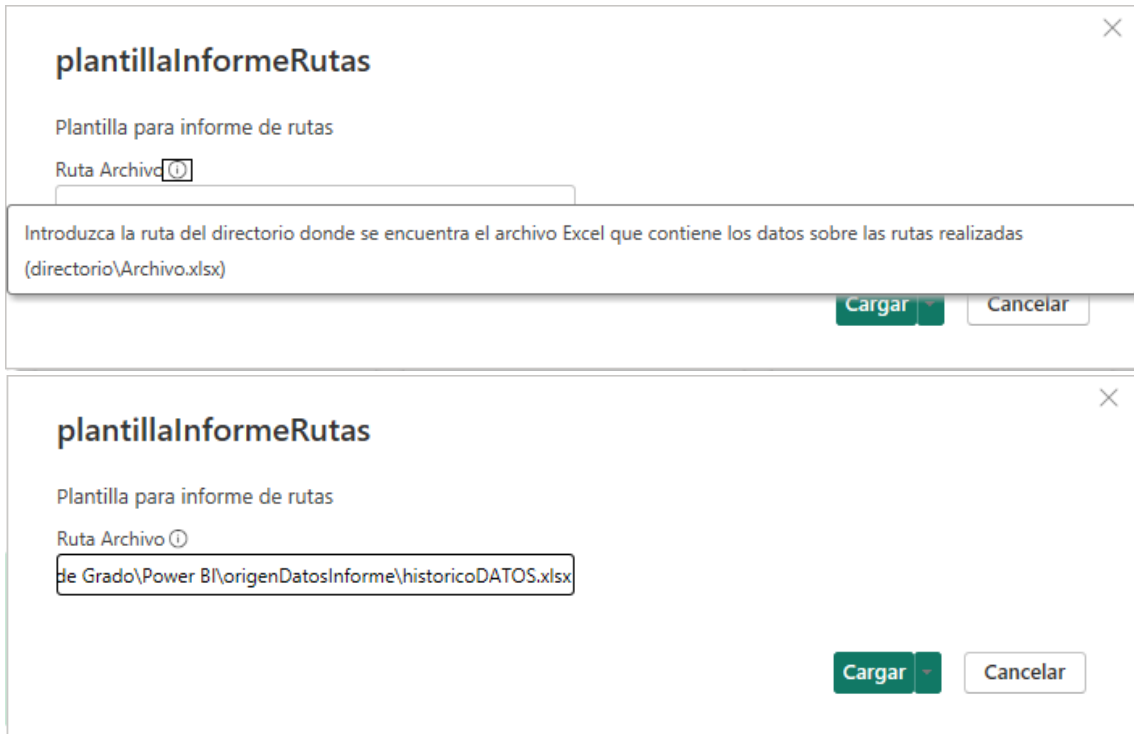


Figura 74. Cuadro de diálogo inicial al crear un informe de rutas (elaboración propia)

3. Pulsar el botón *Cargar*. Automáticamente se creará el informe en base a las rutas contenidas en el archivo origen de datos seleccionado en el paso anterior (ver *Figura 75*).



Figura 75. Ejemplo de informe creado a partir de la plantilla (elaboración propia)





# CAPÍTULO 7: COMERCIALIZACIÓN DEL EQUIPO





## 7. COMERCIALIZACIÓN DEL EQUIPO

En este apartado se exponen aquellos aspectos que habría que tener en cuenta de cara a una hipotética comercialización del dispositivo GPS desarrollado. Aquí se detallan desde el coste total del equipo, hasta las especificaciones técnicas, funcionalidades, aplicaciones y bondades del producto. Además, se proporciona un manual de usuario que describe detalladamente el funcionamiento del dispositivo y se presenta un catálogo comercial cuyo objetivo es hacerlo más atractivo.

### 7.1. COSTE TOTAL DEL EQUIPO

Uno de los elementos esenciales en una supuesta salida al mercado de este producto, es realizar un presupuesto total del dispositivo implementado, en el que se desglosen el coste de todos sus componentes, lo que ayudará a estimar cuál sería su precio de venta al público (PVP).

Partiendo de la aproximación realizada en el apartado 3.2, se ha añadido el coste de los elementos necesarios para fabricar la PCB y la carcasa del dispositivo, dando como resultado la *Tabla 11*:

Tabla 11. Coste total del localizador GPS desarrollado (elaboración propia)

LOCALIZADOR GPS							
Componente						Coste por unidad (sin IVA)	
Nombre	Tipo	Modelo	Nº de referencia	Fabricante	Unidades	1 ud.	Producción alta (+100 uds.)
Placa Arduino	Nano	Every	ABX00028	Arduino	1	10.95 €	10.10 €
Módulo GPS con antena	GY-GPS6MV2	NEO-M8N	-	U-blox	1	7.99 €	7.03 €
Cargador LiPo	Power Boost	500C	PRO-0109	Adafruit	1	15.95 €	15.95 €
Batería LiPo	3.7V/1000mAh	6030650	BAT-0004	-	1	5.95 €	5.95 €
Módulo Lector microSD	-	-	-	TZT	1	0.77 €	0.77 €
Tarjeta de memoria	microSD	8 GB	SP008GBS THBU1V10 SP	Silicon Power	1	3.37 €	3.37 €
Interruptor	Rocker	SR	SRB22A2F BBNN	ZF	1	1.55 €	1.35 €
PCB	-	-	-	JLPCPCB	1	0.40 €	0.29 €
Conector para PCB	Hembra	-	M20-7822046	Harwin	48	0.45 €	0.38 €
Separador de Nylon	Macho-Hembra	M3x12mm	D01497	DURATOOL	5	0.15 €	0.11 €
Filamento para impresión 3D	PLA	Color blanco	-	ANYCUBIC	1	1.00 €	1.00 €
Tornillos Allen	Avellanado de acero	M3x20 mm	M3 20 PRSTMC Z100	TR Fastenings	4	0.03 €	0.02 €
<b>TOTAL (sin IVA)</b>						70.59 €	64.58 €
<b>TOTAL (con IVA)</b>						85.41 €	78.14 €



Por lo tanto, con el fin de obtener un margen de beneficio del 10% por cada unidad, se va a establecer un PVP de 94,99€.

### 7.2. CATÁLOGO COMERCIAL DEL EQUIPO

Con el fin de hacer atractivo el producto de forma que se facilite su teórica salida al mercado, a continuación se exponen, desde un punto de vista comercial, las especificaciones técnicas del dispositivo, sus aplicaciones y bondades y un manual de usuario con las instrucciones de funcionamiento.

#### 7.2.1. ESPECIFICACIONES TÉCNICAS

- ❖ Autonomía de hasta 6 horas. Se empleará una batería LiPo modelo 603050 de 3.7V/1000 mAh (*Figura 28*) junto con un cargador LiPo modelo PowerBoost 500C (*Figura 30*)
- ❖ Interruptor ON/OFF de tipo *Rocker* para encendido y apagado del dispositivo (*Figura 31*).
- ❖ Utiliza un módulo GY-GPS6MV2 (*Figura 22*) con receptor GPS NEO-M8N (*Figura 23*) de Ublox ofreciendo una recepción concurrente de hasta 3 sistemas GNSS.
- ❖ Utiliza una placa Arduino Nano Every (*Figura 24*), la cual lleva incorporado el microcontrolador ATmega4809 para procesar y convertir los datos captados por el módulo GPS y realizar cálculos a partir de ellos.
- ❖ Utiliza un módulo lector de tarjetas microSD (*Figura 27*) junto con una tarjeta de memoria microSD de 8 GB (*Figura 26*), para almacenar los datos asociados a los recorridos. Existe la posibilidad ampliar la memoria del dispositivo sustituyendo la microSD por una de mayor capacidad en caso de necesidad.
- ❖ Carcasa personalizada con entradas a puerto micro-USB de la placa Arduino, puerto micro-USB del cargador y puerto de entrada/salida de tarjeta de memoria microSD.
- ❖ Dimensiones: 89 x 82 x 27 mm (largo x ancho x alto)
- ❖ Peso: 120 g
- ❖ Precio: 94,99 €

#### 7.2.2. FUNCIONALIDADES

El dispositivo es capaz de actuar como *datalogger*, registrando y almacenando una serie de datos críticos ligados a las rutas realizadas por el usuario. Entre estos destacan la distancia recorrida, que permite al usuario conocer la longitud total de la ruta; la velocidad máxima, mínima y media alcanzada durante la actividad; el ritmo medio del recorrido, muy útil para prácticas deportivas como la carrera o el ciclismo; la altitud, esencial para terrenos montañosos y la pendiente del terreno, lo cual ayuda a comprender la dificultad de la ruta.



Además, gracias a la integración con Power BI, los datos recopilados pueden ser visualizados y analizados de forma clara e intuitiva a través de informes interactivos. Estos ofrecen al usuario la posibilidad de llevar a cabo comparaciones entre rutas distintas, consultar el trazado del recorrido y disponer de un registro histórico de todas las rutas realizadas, proporcionando una visión completa del rendimiento a lo largo del tiempo.

### 7.2.3. APLICACIONES

La principal aplicación de este dispositivo GPS es el registro detallado de parámetros asociados a rutas de distinto tipo, desde actividades deportivas como senderismo, carrera o ciclismo, hasta trayectos en vehículos motorizados como coches o motos.

Inherente a esta, surgen otras secundarias como el análisis y mejora del rendimiento deportivo, para los amantes del ejercicio físico; la monitorización de vehículos, facilitando el análisis de trayectos y el control de distancias recorridas y la exploración de nuevos lugares, ideal para aquellas personas que desean documentar sus recorridos y compartir sus experiencias.

### 7.2.4. BONDADES

El uso combinado de la placa Arduino Nano Every y el módulo GY-GPS6MV2, junto con un diseño compacto y eficiente ofrece las siguientes ventajas:

- ❖ Precisión en el registro de datos: el sensor GPS NEO-M8N incorporado asegura una alta precisión en la localización y captura de datos.
- ❖ Facilidad de uso: el dispositivo es fácil de poner en marcha, con un modo de operación muy intuitivo y una interfaz sencilla para la visualización de datos.
- ❖ Versatilidad: puede ser utilizado en una amplia gama de aplicaciones (ver 7.2.3).
- ❖ Portabilidad: gracias a su diseño compacto y ligero, el dispositivo es fácil de transportar en una mochila o funda o incluso en un bolsillo, lo que lo hace ideal para actividades al aire libre donde la comodidad y la movilidad son esenciales.
- ❖ Precio económico: con un PVP de 94,99 €, el dispositivo constituye una solución accesible para aquellos que buscan un sistema de monitoreo GPS eficiente y preciso sin incurrir en inversiones elevadas.

### 7.2.5. MANUAL DE USUARIO

#### Instrucciones de funcionamiento

**1. Introduzca la tarjeta microSD:** es importante que antes de encender el dispositivo, la tarjeta microSD esté colocada en el módulo lector. Si el dispositivo se enciende antes de colocar la tarjeta los datos no se almacenarán.

**2. Encienda el localizador GPS:** el dispositivo comenzará a recolectar datos. Además, se recomienda que tras poner el dispositivo en marcha por primera vez, se realicen



movimientos en un espacio abierto (por ejemplo, en un paseo libre o en un viaje corto en coche). Esto podrá acelerar el proceso de conexión a las redes satelitales. Una buena conexión a dichas redes será necesaria para que el dispositivo envíe y registre datos de manera precisa. El receptor GPS estará calibrado cuando parpadee una luz azul.

**3. Coloque el localizador GPS en una funda y realice la ruta:** sitúe el dispositivo dentro de un bolsillo o una funda para facilitar su transporte y póngase en marcha.

**4. Extraiga la tarjeta microSD y conéctela a su ordenador:** una vez finalizada la ruta, se habrán almacenado un fichero de texto *DATOS\_1.TXT*, que contiene los datos ligados a esa ruta.

**5. Ejecute el fichero *unificarDatos.exe*:** después de almacenar el fichero de texto en la misma carpeta que *unificarDatos.exe*, bastará con ejecutar este último para obtener el archivo Excel *historicoDATOS.xlsx*.

**6. Abra la plantilla de informe de Power BI:** a continuación, abra el fichero *plantillaInformeRutas.pbit* e introduzca la ruta del archivo *historicoDATOS.xlsx* para sincronizar los datos. ¡Listo! Ahora ya puede ver las métricas de su(s) recorrido(s).

### El producto y su uso

- ❖ Recargar la batería: para cargar el localizador GPS, conecte la salida del cargador LiPo a una toma de corriente utilizando un cable de carga micro-USB y un adaptador de corriente compatible.
- ❖ Cargar un programa a Arduino: para realizar pruebas de funcionamiento o modificar el sketch cargado por defecto, conecta la salida de la placa Arduino Nano Every a un ordenador utilizando un cable de carga micro-USB.
- ❖ Apertura y cierre de la carcasa: para abrir y cerrar la carcasa, bastará con extraer los 4 tornillos situados en las esquinas y volver a introducirlos en su posición origen.

Por último, con el fin de sintetizar toda esta información acerca del dispositivo, se ha elaborado un catálogo comercial, el cual se exhibe en la *Figura 76*:

# LOCALIZADOR GPS

## CATÁLOGO COMERCIAL

### DESCRIPCIÓN DEL PRODUCTO

Este localizador GPS compacto y portátil es capaz de actuar como datalogger, registrando y almacenando una serie de datos críticos ligados a las rutas realizadas por el usuario. Entre estos destacan la distancia recorrida, que permite al usuario conocer la longitud total de la ruta; la velocidad máxima, mínima y media alcanzada durante la actividad; el ritmo medio del recorrido, muy útil para prácticas deportivas como la carrera o el ciclismo; la altitud, esencial para terrenos montañosos y la pendiente del terreno, lo cual ayuda a comprender la dificultad de la ruta.

El dispositivo destaca por su robustez y portabilidad, lo que lo hace ideal para un uso prolongado en diversas condiciones ambientales. Su diseño ergonómico asegura una fácil manipulación y transporte, mientras que su batería de larga duración garantiza que no te quedes sin datos importantes durante tus actividades más extensas.

Además, gracias a la integración con Power BI, los datos recopilados pueden ser visualizados y analizados de forma clara e intuitiva a través de informes interactivos. Estos ofrecen al usuario la posibilidad de llevar a cabo comparaciones entre rutas distintas, consultar el trazado del recorrido y disponer de un registro histórico de todas las rutas realizadas, proporcionando una visión completa del rendimiento a lo largo del tiempo.



### ESPECIFICACIONES TÉCNICAS

- Precisión GPS: menos de 10 metros
- Batería interna: 3.7V / 1.000 mAh Litio-Ion
- Autonomía: hasta 6 horas
- Arduino Nano Every con microcontrolador ATmega4809
- Módulo GPS: GY-GPS6MV2 con chip NEO-M8N de Ublox
- Red GNSS: recepción concurrente de hasta 3 sistemas
- Cargador LiPo PowerBoost 500C de Adafruit
- Lector de tarjetas microSD
- Memoria: 8 GB (ampliable)
- Interruptor ON/OFF tipo Rocker
- Dimensiones PCB: 73 x 49,7 mm
- Dimensiones carcasa: 89 x 82 x 27 mm
- Peso: 120 g
- Temperatura de funcionamiento: de -20 °C a 60 °C
- Precio: 94,99€

### APLICACIONES

- Registro detallado de parámetros de rutas
- Análisis y mejora del rendimiento deportivo
- Visualización de trayectos y control de distancias recorridas

### VENTAJAS

- Bajo coste
- Versatilidad
- Facilidad de uso
- Portabilidad







# CAPÍTULO 8: CONCLUSIONES





## 8. CONCLUSIONES

En este apartado se exponen las conclusiones derivadas de la realización del proyecto, integrando tanto los logros obtenidos como las dificultades enfrentadas durante el proceso.

Primero de todo, todos los objetivos planteados en el apartado 1.3 han sido alcanzados satisfactoriamente. El objetivo principal era diseñar y fabricar un prototipo de localizador GPS funcional y portátil que permitiera al usuario registrar rutas o recorridos para su posterior análisis y visualización en un ordenador. Tras meses de trabajo, se ha logrado materializar este prototipo. Para ello, ha sido necesario desarrollar varios componentes clave:

Por una parte, se ha diseñado y fabricado una placa de circuito impreso (PCB) donde se han montado los componentes que forman el dispositivo. El uso de una PCB ha permitido reducir significativamente el tamaño del localizador GPS, resultando en un aparato cómodo de llevar durante actividades deportivas. Adicionalmente, se ha construido una carcasa personalizada que no solo protege el dispositivo, aumentando su robustez y durabilidad, sino que también mejora su apariencia, haciéndolo más atractivo de cara a su comercialización.

En cuanto al software, se ha creado un sketch en Arduino IDE para el procesamiento y conversión de los datos recibidos del sensor GPS. Este programa también calcula los parámetros relativos a los recorridos realizados y guarda esta información en ficheros de texto ubicados en una tarjeta de memoria microSD. Además, se ha desarrollado un script en Python, utilizando la librería Pandas, que unifica los ficheros de texto almacenados en un único archivo Excel, sirviendo como origen de datos para el informe final. Para facilitar su manejo, este script se ha convertido en un archivo ejecutable. Por último, también se ha diseñado una plantilla de informe en Power BI, que permite al usuario crear informes basados en las rutas realizadas, analizando distintas métricas, gráficos y estadísticas. Esta plantilla cuenta con una interfaz atractiva y usable, utilizando objetos visuales sencillos que permiten una interpretación rápida y eficaz de los datos.

Para conseguir todo esto, ha sido esencial la adquisición de conocimientos en diversas aplicaciones y tecnologías, como Arduino IDE, EAGLE, Fusion 360, Python (especialmente la librería Pandas) y Power BI. No obstante, el desarrollo del proyecto no ha estado exento de desafíos significativos que han sido necesario enfrentar durante las distintas fases del proyecto.

Una de las primeras dificultades que encontré fue la incertidumbre inicial sobre los métodos exactos para lograr el dispositivo deseado. Aunque había una visión clara del objetivo final, desconocía cuál iba a ser el método para procesar la información del sensor GPS, dónde almacenarla y qué herramientas utilizar para la visualización de datos. Esta situación fue especialmente abrumadora al principio del proyecto, debido a la gran cantidad de posibles enfoques y soluciones. Con el tiempo, y a través



de decisiones bien fundamentadas, se clarificó el camino a seguir, lo que ha permitido ir avanzando de manera sistemática y efectiva.

Sin embargo, en mi opinión el mayor obstáculo ha sido la materialización de un prototipo funcional. Y es que, a pesar de que los diseños y cálculos teóricos han proporcionado una base sólida, la implementación práctica a menudo revelaba discrepancias que han requerido de ajustes y modificaciones. En este caso, a medida que se avanzaba, se iban observando diferencias entre los resultados prácticos y las expectativas teóricas, lo que ha obligado a recalibrar el diseño en varias ocasiones, lo que conlleva ciertas desviaciones de tiempo respecto a la planificación inicial. Tanto es así, que se han fabricado varias versiones tanto de la PCB como de la carcasa ya que las primeras presentaban errores críticos que comprometían la operatividad del dispositivo. Cada iteración ha implicado la realización de un nuevo diseño, junto con su correspondiente fabricación y verificación, lo que ha demandado una considerable cantidad de tiempo. A pesar de todo, este proceso iterativo ha sido esencial para lograr un dispositivo funcional y eficiente. La identificación y corrección de errores en las primeras versiones ha permitido desarrollar un localizador GPS más compacto, robusto y fácil de utilizar.

En conclusión, el desarrollo de este Trabajo de Fin de Grado ha sido un proceso complejo y enriquecedor, que ha conllevado la puesta en práctica de numerosas competencias técnicas y generales. A pesar de las dificultades encontradas, se han logrado todos los objetivos planteados, resultando en un dispositivo innovador y funcional que combina hardware y software de manera efectiva. Este proyecto no solo ha permitido aplicar y consolidar conocimientos adquiridos a lo largo del grado universitario, sino que también ha contribuido al desarrollo personal y profesional, fomentando la creatividad, la innovación y la capacidad de resolver problemas de manera autónoma.

### **8.1. IMPACTO DEL PROYECTO EN LOS OBJETIVOS DE DESARROLLO SOSTENIBLE**

Alineado con el plan de acción específico, integral, transversal y participativo para la implementación de la Agenda 2030 y el cumplimiento de los Objetivos de Desarrollo Sostenible en la Universidad de Valladolid, en este apartado se realiza una labor de reflexión en la relevancia de este Trabajo de Fin de Grado sobre diferentes dimensiones de los ODS.

Los Objetivos de Desarrollo Sostenible (ODS) son un conjunto de 17 objetivos globales adoptados por todos los Estados miembros de las Naciones Unidas en 2015 como parte de la Agenda 2030 para el Desarrollo Sostenible (*Figura 77*). Estos objetivos buscan abordar desafíos globales como la pobreza, la desigualdad, el cambio climático, la degradación ambiental, la paz y la justicia. Los ODS están interconectados y abarcan una amplia gama de temas que son fundamentales para el bienestar humano y la sostenibilidad del planeta. [57]



Figura 77. Objetivos de Desarrollo Sostenible (extraído de [57])

Este Trabajo de Fin de Grado puede enmarcarse en el contexto de varios ODS como, por ejemplo, el ODS 9: Industria, Innovación e Infraestructura, al combinar hardware y software de manera innovadora y contribuir al desarrollo de infraestructuras tecnológicas más avanzadas y accesibles o el ODS 12: Producción y Consumo Responsable, al tratarse de un prototipo diseñado con eficiencia energética y uso responsable de recursos, promoviendo prácticas de producción más sostenibles y con menor impacto ambiental. Además, también está alineado con los ODS 7: Energía Asequible y No Contaminante y ODS 11: Ciudades y Comunidades Sostenibles ya que este proyecto fomenta e incentiva actividades como andar, correr o montar en bicicleta, lo que contribuye al uso de modos de transporte no contaminantes y la recreación en entornos naturales, reduciendo la huella de carbono y mejorando la calidad de vida en las comunidades.

No obstante, de todos los ODS, considero que el dispositivo desarrollado en este Trabajo de Fin de Grado tiene una relevancia especialmente significativa en el ODS 3: Salud y Bienestar, por los motivos que se expondrán más adelante. Este ODS busca garantizar una vida sana y promover el bienestar para todas las personas en todas las edades. Para ello, incluye metas como la reducción de la mortalidad materna e infantil, la lucha contra enfermedades transmisibles y no transmisibles, el acceso a servicios de salud esenciales y de calidad, y la promoción de la salud mental y el bienestar. La consecución del ODS 3 es fundamental para el desarrollo sostenible, ya que una población saludable es esencial para el progreso económico y social. [58]

En mi opinión, el localizador GPS desarrollado tiene el potencial para contribuir en varios aspectos de este ODS:

En primer lugar, el dispositivo, puede ser utilizado para el seguimiento y monitoreo de la actividad física de los usuarios. La actividad física regular es esencial para mantener una buena salud y prevenir enfermedades crónicas como la obesidad, la diabetes y las enfermedades cardiovasculares. Al proporcionar datos precisos sobre



## Capítulo 8: Conclusiones



la distancia recorrida, la velocidad y la ubicación, el dispositivo puede motivar a las personas a mantenerse activas y llevar un estilo de vida saludable.

Asimismo, la promoción de actividades al aire libre como el senderismo, la carrera o el ciclismo no solo benefician la salud física, sino también la salud mental. Está demostrado que la práctica habitual de este tipo de deportes puede reducir los niveles de estrés, ansiedad y depresión, mejorando el estado de ánimo y la calidad de vida.

Finalmente, los datos recopilados por el dispositivo pueden ser valiosos para la investigación en salud y bienestar. Investigadores y profesionales de la salud pueden utilizar estos datos para analizar patrones de actividad física, estudiar el impacto del ejercicio en la salud y desarrollar programas de intervención personalizados. Esto puede conducir a mejoras en la atención sanitaria y la promoción de hábitos saludables a nivel comunitario.





# CAPÍTULO 9: LÍNEAS FUTURAS





## 9. LÍNEAS FUTURAS

En este último apartado se proponen una serie de aspectos a mejorar que se consideran esenciales para dar un salto de calidad al dispositivo desarrollado y que serían interesantes de implementar en el futuro:

- ❖ **Sustitución de la placa de Arduino por un microcontrolador que disponga de conectividad Wi-Fi y/o Bluetooth:** implementar un microcontrolador con conectividad Wi-Fi y/o Bluetooth permitiría almacenar los datos en la nube, eliminando la necesidad de una tarjeta de memoria y su correspondiente módulo lector. Esto aumentaría la eficiencia del dispositivo en términos de consumo, tamaño y peso. Además, permitiría rastrear la posición del usuario en tiempo real y enviar avisos o alarmas cuando se cumplan ciertos hitos, como haber recorrido una distancia predeterminada o haber alcanzado un límite de tiempo.
- ❖ **Adición de nuevos sensores:** las posibilidades a la hora de añadir sensores que proporcionen funcionalidades extra al dispositivo son infinitas. Sin embargo, se han seleccionado las siguientes por ser las que más valor aportarían:
  - Sensor de frecuencia cardíaca: si bien es cierto que se podría haber realizado una estimación de las calorías quemadas por el usuario durante la actividad física, la realidad es que sin datos de frecuencia cardíaca dicha estimación es muy inexacta, como ya se explicó en el al final del apartado 3.4.3. Por lo tanto, para poder implementar un algoritmo de cálculo de calorías que ofrezca unos valores cercanos a la realidad, es imprescindible añadir (como mínimo) un sensor que permita conocer la frecuencia cardíaca del individuo durante la consecución de las rutas.
  - Unidad de medición inercial (IMU): compuesta por 3 acelerómetros, 3 giroscopios y 3 magnetómetros, una IMU mide e informa sobre la aceleración, la orientación, las velocidades angulares y otras fuerzas gravitatorias. [59]  
  
Su inclusión permitiría ampliar el rango de actividades a monitorizar. Desde flexiones hasta sentadillas, permitiría al usuario registrar datos relativos a una variedad de ejercicios deportivos más allá de las rutas de senderismo, ciclismo, etc.
  - Barómetro: tal y como se mencionó en el apartado 3.6.1., las medidas de altitud del chip GPS NEO-M8N no son muy exactas, proporcionando valores muy distintos a la realidad. Por esta razón, se considera esencial encontrar otra opción para medir la altitud. Un altímetro barométrico o barómetro se presenta como una de las formas más fiables de medir la altitud.



- ❖ **Mejorar el módulo GPS:** es recomendable realizar un estudio aún más exhaustivo del mercado para encontrar un módulo GPS que ofrezca una mayor precisión y rapidez en la localización, sin comprometer el consumo energético ni los costes.
- ❖ **Reducción de tamaño:** a pesar de conseguir un aparato relativamente pequeño y cómodo, un nuevo diseño podría reducir aún más sus dimensiones. Esto implicaría prescindir de módulos completos y utilizar únicamente los componentes electrónicos necesarios. Por ejemplo, de la placa Arduino Nano Every solo se utilizan 9 de los 30 pines disponibles, por lo que se está infrutilizando. La optimización de cada módulo por separado, junto con la ya mencionada eliminación del módulo lector de tarjetas y la microSD (almacenamiento en la nube), permitiría obtener un dispositivo mucho más compacto.
- ❖ **Mejorar la autonomía del dispositivo:** sustituir la batería LiPo de 1000 mAh por otra de mayor capacidad, permitiría aumentar el tiempo de funcionamiento del dispositivo, lo que aumentaría su autonomía.
- ❖ **Realizar una carcasa más profesional:** utilizar materiales de mayor calidad en la carcasa brindaría mayor robustez, durabilidad y protección contra condiciones climatológicas adversas. Esto es crucial para mejorar la longevidad y fiabilidad del dispositivo en diversos entornos.
- ❖ **Crear una aplicación propia para la visualización de datos:** desarrollar una aplicación personalizada eliminaría la dependencia de Power BI, permitiendo una mayor personalización de las métricas y estadísticas mostradas al usuario. Una aplicación propia podría ser diseñada para ordenadores, tabletas y móviles, simplificando el proceso de visualización y permitiendo la integración de datos personales como edad, peso y altura, entre otros.
- ❖ **Proceso de fabricación a gran escala:** establecer un proceso de fabricación tanto de la placa de circuito impreso como de la carcasa que permita una producción a gran escala reduciría los costes por unidad. Esto implicaría optimizar los procesos de diseño y ensamblaje, así como seleccionar materiales y proveedores adecuados para mantener los estándares de calidad.
- ❖ **Estrategia de marketing:** desarrollar una estrategia de marketing eficaz sería esencial para fomentar la venta del dispositivo al público general. Esto incluiría estudios de mercado para identificar el público objetivo, campañas publicitarias en medios digitales y tradicionales, colaboraciones con expertos del sector, y una presencia activa en redes sociales. Además, se podrían realizar demostraciones en eventos y ferias tecnológicas para mostrar las capacidades del dispositivo y atraer potenciales compradores.



# CAPÍTULO 10: BIBLIOGRAFÍA





## 10. BIBLIOGRAFÍA

- [1] «GPS.gov: El Sistema de Posicionamiento Global». Accedido: 20 de mayo de 2024. [En línea]. Disponible en: <https://www.gps.gov/systems/gps/spanish.php>
- [2] «COMO, CUANDO Y DONDE: RECEPTOR GPS | Leon Aventura». Accedido: 19 de junio de 2024. [En línea]. Disponible en: <https://www.leonaventura.com/receptor-gps/>
- [3] «Bienvenidos a GPS.gov». Accedido: 20 de mayo de 2024. [En línea]. Disponible en: <https://www.gps.gov/spanish.php>
- [4] «¿Cómo funcionan los dispositivos GPS? Trilateración vs Triangulación | El blog de franz». Accedido: 17 de junio de 2024. [En línea]. Disponible en: <https://acolita.com/como-funcionan-los-dispositivos-gps-trilateracion-vs-triangulacion/>
- [5] «GPS, How does it work? | ICT #12 - YouTube». Accedido: 17 de junio de 2024. [En línea]. Disponible en: [https://www.youtube.com/watch?v=8eTII19\\_57g&t=2s](https://www.youtube.com/watch?v=8eTII19_57g&t=2s)
- [6] «How GPS Works Today - YouTube». Accedido: 17 de junio de 2024. [En línea]. Disponible en: [https://www.youtube.com/watch?v=wCcARVbL\\_Dk](https://www.youtube.com/watch?v=wCcARVbL_Dk)
- [7] «La física que hace funcionar al sistema GPS», *WikiVersus*, Accedido: 17 de junio de 2024. [En línea]. Disponible en: <https://wikiversus.com/deportes-y-aire-libre/reloj-gps-pulsometro/como-funciona-el-gps/>
- [8] «GPS - Wikipedia, la enciclopedia libre». Accedido: 17 de junio de 2024. [En línea]. Disponible en: <https://es.wikipedia.org/wiki/GPS>
- [9] «GPS.gov: Aplicaciones». Accedido: 20 de mayo de 2024. [En línea]. Disponible en: <https://www.gps.gov/applications/spanish.php>
- [10] «GPS.gov: Recreación». Accedido: 17 de junio de 2024. [En línea]. Disponible en: <https://www.gps.gov/applications/recreation/spanish.php>
- [11] «¿Qué es el sistema GPS? | Garmin». Accedido: 19 de junio de 2024. [En línea]. Disponible en: <https://www.garmin.com/es-ES/aboutgps/>
- [12] «Latitude and longitude | Definition, Examples, Diagrams, & Facts | Britannica». Accedido: 20 de mayo de 2024. [En línea]. Disponible en: <https://www.britannica.com/science/latitude>
- [13] «Coordenadas geográficas - Wikipedia, la enciclopedia libre». Accedido: 17 de junio de 2024. [En línea]. Disponible en: [https://es.wikipedia.org/wiki/Coordenadas\\_geogr%C3%A1ficas](https://es.wikipedia.org/wiki/Coordenadas_geogr%C3%A1ficas)
- [14] «Altitud, Altura y Elevación». Accedido: 20 de mayo de 2024. [En línea]. Disponible en: <https://sicami.com/article/informacion-tecnica-y-curiosidades/altitud-altura-elevacion>





- [15] «□ u-center GNSS evaluation software for Windows User guide», 2022, Accedido: 15 de junio de 2024. [En línea]. Disponible en: [www.u-blox.com](http://www.u-blox.com)
- [16] «Tramas de GPS: Pasos a tomar en cuenta para interpretarlas». Accedido: 22 de mayo de 2024. [En línea]. Disponible en: <https://www.deltatracking.com/2020/09/01/pasos-a-tomar-en-cuenta-para-interpretar-tramas-de-gps/>
- [17] «Suma de verificación - Wikipedia, la enciclopedia libre». Accedido: 15 de junio de 2024. [En línea]. Disponible en: [https://es.wikipedia.org/wiki/Suma\\_de\\_verificaci%C3%B3n](https://es.wikipedia.org/wiki/Suma_de_verificaci%C3%B3n)
- [18] «What is Arduino? | Arduino». Accedido: 19 de mayo de 2024. [En línea]. Disponible en: <https://www.arduino.cc/en/Guide/Introduction>
- [19] «Arduino Forum». Accedido: 19 de marzo de 2024. [En línea]. Disponible en: <https://forum.arduino.cc/>
- [20] «Arduino Sketches | Arduino Documentation». Accedido: 19 de mayo de 2024. [En línea]. Disponible en: <https://docs.arduino.cc/learn/programming/sketches/>
- [21] «Qué es UART | Rohde & Schwarz». Accedido: 19 de marzo de 2024. [En línea]. Disponible en: [https://www.rohde-schwarz.com/es/productos/test-y-medida/essentials-test-equipment/digital-oscilloscopes/que-es-uart\\_254524.html](https://www.rohde-schwarz.com/es/productos/test-y-medida/essentials-test-equipment/digital-oscilloscopes/que-es-uart_254524.html)
- [22] «▷ Comunicación Serial con Arduino - [marzo, 2024 ]». Accedido: 19 de marzo de 2024. [En línea]. Disponible en: <https://controlautomaticoeducacion.com/arduino/comunicacion-serial-con-arduino/>
- [23] «Puerto Serial - protocolo y su teoría - HeTPro-Tutoriales». Accedido: 19 de marzo de 2024. [En línea]. Disponible en: <https://hetpro-store.com/TUTORIALES/puerto-serial/>
- [24] «Arduino Serial, ejemplos y funcionamiento - HeTPro-Tutoriales». Accedido: 19 de marzo de 2024. [En línea]. Disponible en: <https://hetpro-store.com/TUTORIALES/arduino-serial/>
- [25] «Cómo configurar la comunicación UART en Arduino | Arduino.cl - Compra tu Arduino en Línea». Accedido: 17 de junio de 2024. [En línea]. Disponible en: <https://arduino.cl/como-configurar-la-comunicacion-uart-en-arduino/>
- [26] «UART - El receptor-transmisor asíncrono universal | PiCockpit». Accedido: 17 de junio de 2024. [En línea]. Disponible en: [https://picockpit.com/raspberry-pi/es/uart-el-receptor-asincrono-universal-transmisor/#UART\\_Applications](https://picockpit.com/raspberry-pi/es/uart-el-receptor-asincrono-universal-transmisor/#UART_Applications)
- [27] P. Dhaker, «Introduction to SPI Interface», *Analog Dialogue*, 2018, Accedido: 17 de junio de 2024. [En línea]. Disponible en:



<https://www.analog.com/en/analog-dialogue/articles/introduction-to-spi-interface.html>

- [28] «Serial Peripheral Interface - Wikipedia, la enciclopedia libre». Accedido: 21 de mayo de 2024. [En línea]. Disponible en: [https://es.wikipedia.org/wiki/Serial\\_Peripheral\\_Interface](https://es.wikipedia.org/wiki/Serial_Peripheral_Interface)
- [29] «Arduino desde cero en Español - Capítulo 37 - Lector tarjeta MicroSD bus SPI (y data-logger DHT11) - YouTube». Accedido: 19 de marzo de 2024. [En línea]. Disponible en: <https://www.youtube.com/watch?v=nNDqRpfEy40&t=1465s>
- [30] «Tech Confronts Its Use of the Labels “Master” and “Slave”», *Wired*, Accedido: 17 de junio de 2024. [En línea]. Disponible en: <https://www.wired.com/story/tech-confronts-use-labels-master-slave/>
- [31] «Serial Peripheral Interface (SPI) - SparkFun Learn». Accedido: 17 de junio de 2024. [En línea]. Disponible en: <https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all>
- [32] «ALLROUND Finder 2.0 GPS Tracker - 30 Días de Garantía». Accedido: 19 de junio de 2024. [En línea]. Disponible en: [https://www.paj-gps.es/producto/allround-finder-2-0/?gad\\_source=1&gclid=Cj0KQCQjw0ruyBhDuARIsANSZ3wpm7wGg80UUf4QyFLNRxYNShcjFG5qXbuS\\_I9VZuPXi6RLtFy1ML7QaAtk1EALw\\_wcB](https://www.paj-gps.es/producto/allround-finder-2-0/?gad_source=1&gclid=Cj0KQCQjw0ruyBhDuARIsANSZ3wpm7wGg80UUf4QyFLNRxYNShcjFG5qXbuS_I9VZuPXi6RLtFy1ML7QaAtk1EALw_wcB)
- [33] «Dispositivo GPS para jugadores de Fútbol | OLIVER». Accedido: 19 de junio de 2024. [En línea]. Disponible en: <https://shop.oliversports.ai/>
- [34] «Amazfit GTR 3 Pro Smartwatch Pantalla AMOLED de 1.45" Frecuencia Cardíaca Sueño Estrés Monitorización de SpO2 Reloj Deportivo con 150 Modos Deportivos GPS Llamadas Bluetooth Control de Música Alexa : Amazon.es: Electrónica». Accedido: 19 de junio de 2024. [En línea]. Disponible en: [https://www.amazon.es/dp/B09FXSF448?ref=cm\\_sw\\_r\\_cp\\_ud\\_dp\\_WY6KX0Z3A8QMM7YQVVJM&ref\\_=cm\\_sw\\_r\\_cp\\_ud\\_dp\\_WY6KX0Z3A8QMM7YQVVJM&social\\_share=cm\\_sw\\_r\\_cp\\_ud\\_dp\\_WY6KX0Z3A8QMM7YQVVJM](https://www.amazon.es/dp/B09FXSF448?ref=cm_sw_r_cp_ud_dp_WY6KX0Z3A8QMM7YQVVJM&ref_=cm_sw_r_cp_ud_dp_WY6KX0Z3A8QMM7YQVVJM&social_share=cm_sw_r_cp_ud_dp_WY6KX0Z3A8QMM7YQVVJM)
- [35] «Forerunner® 165 | Reloj de running». Accedido: 19 de junio de 2024. [En línea]. Disponible en: <https://www.garmin.com/es-ES/p/1055469#overview>
- [36] «GPS module showdown do-over with full precision and two more modules (Neo-6M and BN880Q) - YouTube». Accedido: 15 de junio de 2024. [En línea]. Disponible en: <https://www.youtube.com/watch?v=abMoRJ-1ynY>
- [37] «Tutorial Módulo GPS con Arduino». Accedido: 19 de junio de 2024. [En línea]. Disponible en: [https://naylampmechatronics.com/blog/18\\_tutorial-modulo-gps-con-arduino.html](https://naylampmechatronics.com/blog/18_tutorial-modulo-gps-con-arduino.html)
- [38] S. Moss, «NEO-M8 u-blox M8 concurrent GNSS modules Title NEO-M8 Subtitle u-blox M8 concurrent GNSS modules Document type Data sheet Document



- number This document applies to the following products: Product name Type number Firmware version PCN/IN reference». [En línea]. Disponible en: [www.u-blox.com](http://www.u-blox.com)
- [39] «NEO-M8 series | u-blox». Accedido: 19 de junio de 2024. [En línea]. Disponible en: <https://www.u-blox.com/en/product/neo-m8-series>
- [40] «Arduino® Nano Every».
- [41] «Tutorial Arduino y memoria SD y micro SD.» Accedido: 19 de junio de 2024. [En línea]. Disponible en: [https://naylampmechatronics.com/blog/38\\_tutorial-arduino-y-memoria-sd-y-micro-sd.html](https://naylampmechatronics.com/blog/38_tutorial-arduino-y-memoria-sd-y-micro-sd.html)
- [42] «Bateria Lipo 1000mAh, 3.7V modelo 603050 BricoGeek 603050 | BricoGeek.com». Accedido: 19 de junio de 2024. [En línea]. Disponible en: [https://tienda.bricogeek.com/baterias-lipo/135-bateria-lipo-1000mah-603050-37v.html?gad\\_source=1&gclid=CjwKCAjwg8qzBhAoEiwAWagLrO4tAYC3xNfP3gQ8AgmYzIH4Pa6ml\\_7ykorLW4lkeHPmFRoz6710hoCsVwQAvD\\_BwE](https://tienda.bricogeek.com/baterias-lipo/135-bateria-lipo-1000mah-603050-37v.html?gad_source=1&gclid=CjwKCAjwg8qzBhAoEiwAWagLrO4tAYC3xNfP3gQ8AgmYzIH4Pa6ml_7ykorLW4lkeHPmFRoz6710hoCsVwQAvD_BwE)
- [43] «Todo lo que debes saber sobre las Baterías de Litio (LiPo) | Introducción | BricoGeek Lab». Accedido: 2 de junio de 2024. [En línea]. Disponible en: <https://lab.bricogeek.com/tutorial/todo-lo-que-debes-saber-sobre-las-baterias-de-litio-lipo>
- [44] «Cargadores de Baterías de litio... Que son? - Fácil Electro Baterías, Electrónica, Tecnología Electrónica». Accedido: 2 de junio de 2024. [En línea]. Disponible en: <https://www.facilelectro.es/cargadores-de-baterias-de-litio-que-son/>
- [45] «Adafruit PowerBoost 500 + Charger Created by lady ada». [En línea]. Disponible en: <https://learn.adafruit.com/adafruit-powerboost-500-plus-charger>
- [46] «Interruptor de balancín, SRB22A2FBBNN, Contacto SPST, On-Off, 10 A, No, Negro | RS». Accedido: 19 de junio de 2024. [En línea]. Disponible en: <https://es.rs-online.com/web/p/interruptores-de-balancin/4406979?gb=s>
- [47] «Fórmula del semiverseno - Wikipedia, la enciclopedia libre». Accedido: 31 de mayo de 2024. [En línea]. Disponible en: [https://es.wikipedia.org/wiki/F%C3%B3rmula\\_del\\_semiverseno](https://es.wikipedia.org/wiki/F%C3%B3rmula_del_semiverseno)
- [48] J. de M. y Ríos y J. de M. y Ríos, *Memoria sobre algunos metodos nuevos de calcular la longitud por las distancias lunares y explicaciones prácticas de una teoría para la solución de otros problemas de navegación*. Imp. Real, 1795. Accedido: 31 de mayo de 2024. [En línea]. Disponible en: <https://books.google.cat/books?id=O30t00qIX2AC&printsec=frontcover&dq=Memoria+Sobre+Algunos+Metodos+Nuevos+De+Calcular+La+Longitud&hl=ca#v=onepage&q&f=false>



- [49] «Cálculo de la Pendiente». Accedido: 19 de junio de 2024. [En línea]. Disponible en: <https://www.altimetricas.net/articulos/4ComoPendiente.asp>
- [50] «Cómo funciona el cálculo de calorías en relojes GPS y pulseras de actividad - Correr una Maratón - Review de Garmin, Polar, Suunto, COROS...» Accedido: 11 de junio de 2024. [En línea]. Disponible en: [https://www.correrunamaraton.com/calculo-calorias-relojes-gps-pulseras-actividad/?utm\\_content=cmp-true](https://www.correrunamaraton.com/calculo-calorias-relojes-gps-pulseras-actividad/?utm_content=cmp-true)
- [51] «Tu guía de las principales herramientas de visualización de datos en 2024». Accedido: 4 de junio de 2024. [En línea]. Disponible en: <https://kinsta.com/es/blog/herramientas-de-visualizacion-de-datos/>
- [52] «¿Qué es Power BI? - Power BI | Microsoft Learn». Accedido: 20 de mayo de 2024. [En línea]. Disponible en: <https://learn.microsoft.com/es-es/power-bi/fundamentals/power-bi-overview>
- [53] «#22 Comparison of precision between Neo-M8N and Neo-6M Modules - YouTube». Accedido: 17 de junio de 2024. [En línea]. Disponible en: <https://www.youtube.com/watch?v=FmdG66kTfsI>
- [54] «Altímetro barométrico. Todo lo que debes saber sobre la altitud en los GPS». Accedido: 11 de junio de 2024. [En línea]. Disponible en: <https://www.correrunamaraton.com/gps-como-funciona-altimetro-barometrico/>
- [55] «Nano shield v3 para Arduino BricoGeek | BricoGeek.com». Accedido: 19 de junio de 2024. [En línea]. Disponible en: <https://tienda.bricogeek.com/shields-arduino/1910-nano-shield-v3-para-arduino.html>
- [56] «Calculadora de IPC-2221 en el diseño de alta tensión | Altium». Accedido: 19 de junio de 2024. [En línea]. Disponible en: <https://resources.altium.com/es/p/using-an-ipc-2221-calculator-for-high-voltage-design>
- [57] «Objetivos y metas de desarrollo sostenible - Desarrollo Sostenible». Accedido: 11 de junio de 2024. [En línea]. Disponible en: <https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/>
- [58] «Salud - Desarrollo Sostenible». Accedido: 17 de junio de 2024. [En línea]. Disponible en: <https://www.un.org/sustainabledevelopment/es/health/>
- [59] «IMU - Unidad de medición inercial - SBG Systems». Accedido: 11 de junio de 2024. [En línea]. Disponible en: <https://www.sbg-systems.com/es/unidad-de-medicion-inercial-sensor-imu/>





Anexos



# ANEXOS

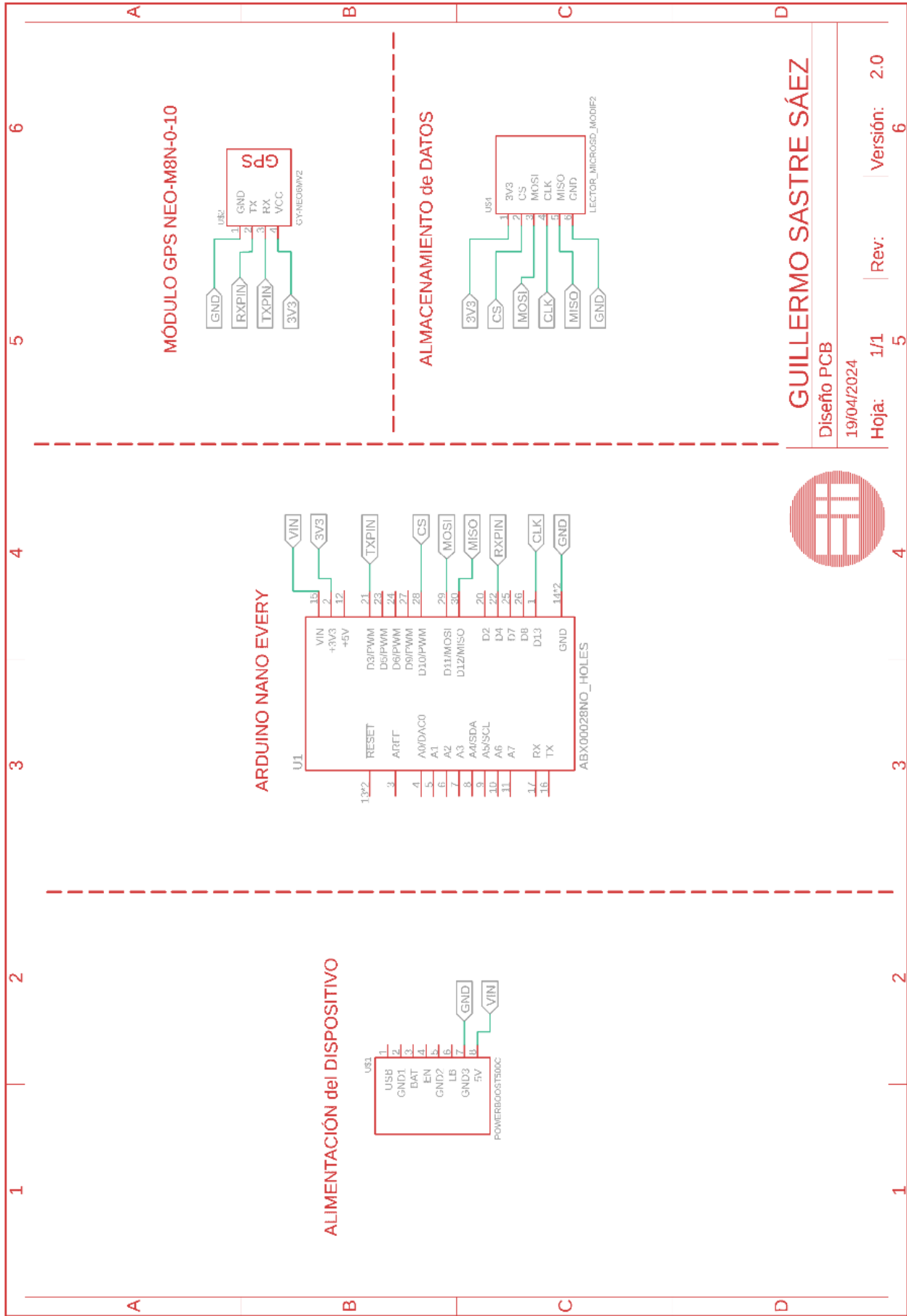






## ANEXO I. PLANOS

1. Esquema eléctrico
2. Planos PCB
  - Cara de componentes
  - Cara de soldadura
  - Cara de serigrafía superior
3. Carcasa



GUILLERMO SASTRE SÁEZ

Diseño PCB

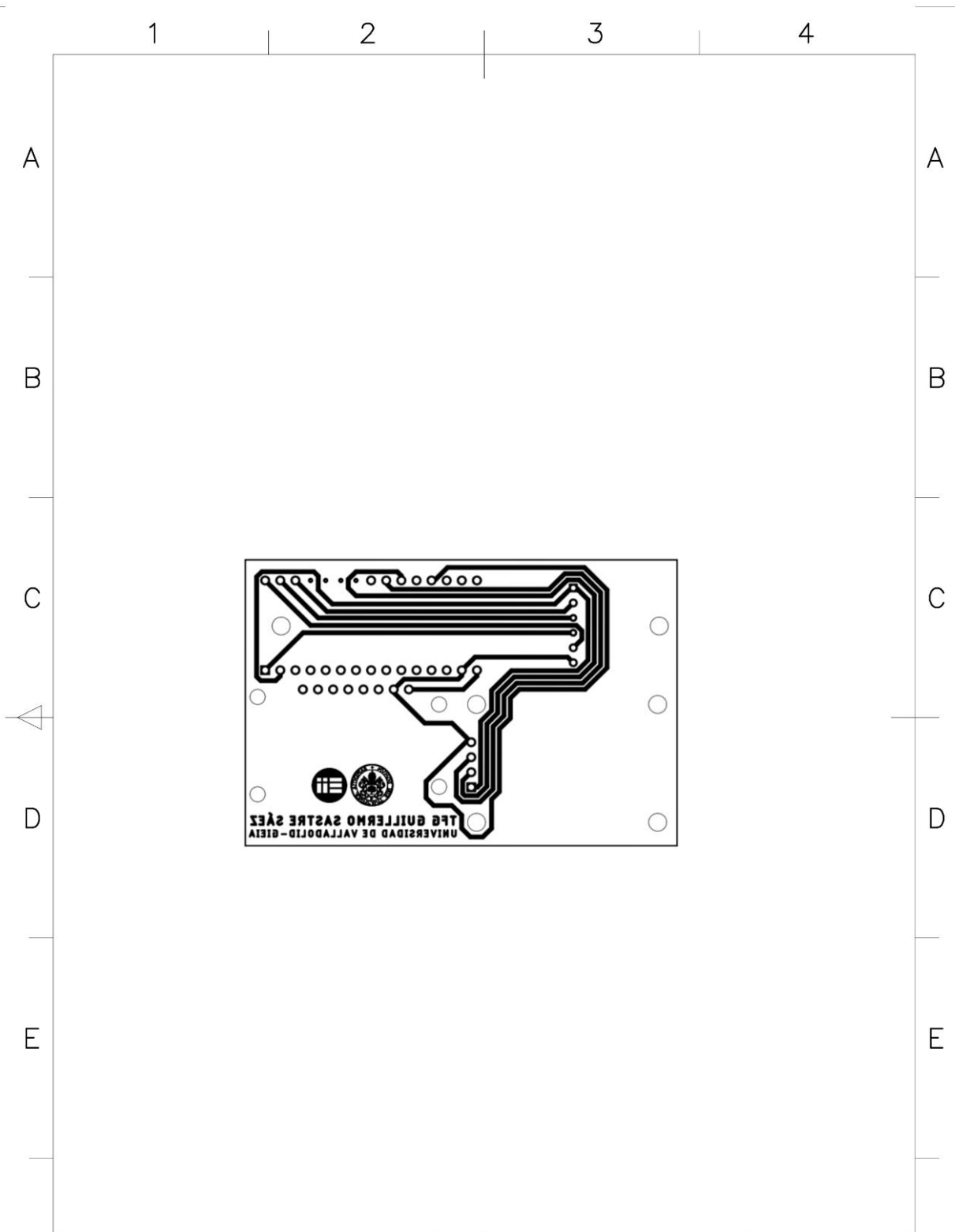
19/04/2024

Hoja: 1/1

Rev: 1/1

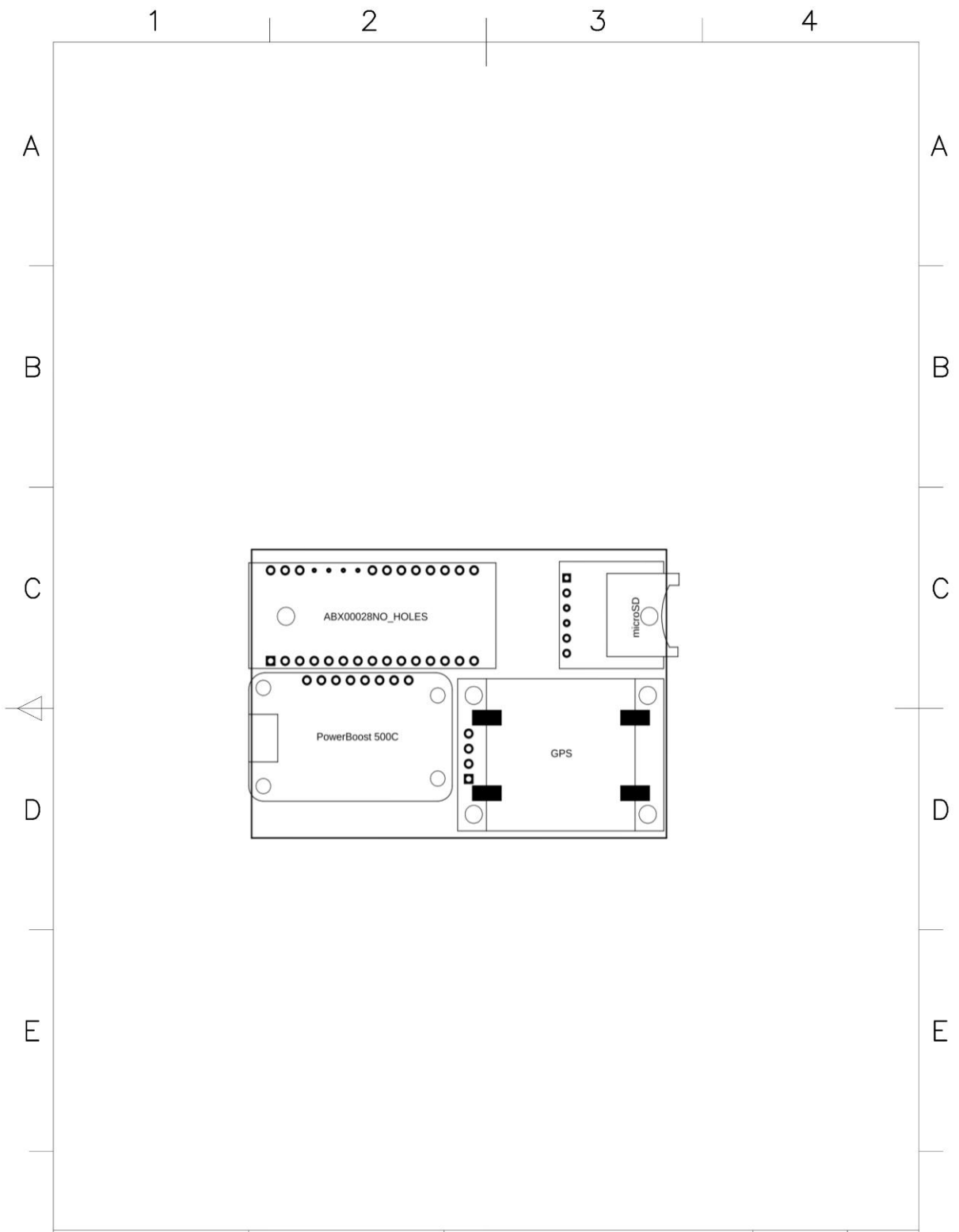
Versión: 2.0



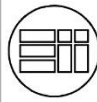


Dibujado por – Fecha G. Sastre 20/04/24	Aprobado por – Fecha G. Sastre 20/04/24	Trabajo de Fin de Grado	Fecha 20/04/24	Escala 1:1
--	--	-------------------------	-------------------	---------------

<b>UNIVERSIDAD DE VALLADOLID</b> ESCUELA DE INGENIERIAS INDUSTRIALES DPTO. TECNOLOGIA ELECTRONICA	<b>GUILLERMO SASTRE SAEZ</b>	
	REFERENCIA DE LA PCB CAPA: SOLDADURA	Rev. <b>1</b>



Dibujado por – Fecha G. Sastre 20/04/24	Aprobado por – Fecha G. Sastre 20/04/24	Trabajo de Fin de Grado	Fecha 20/04/24	Escala 1:1
--	--	-------------------------	-------------------	---------------

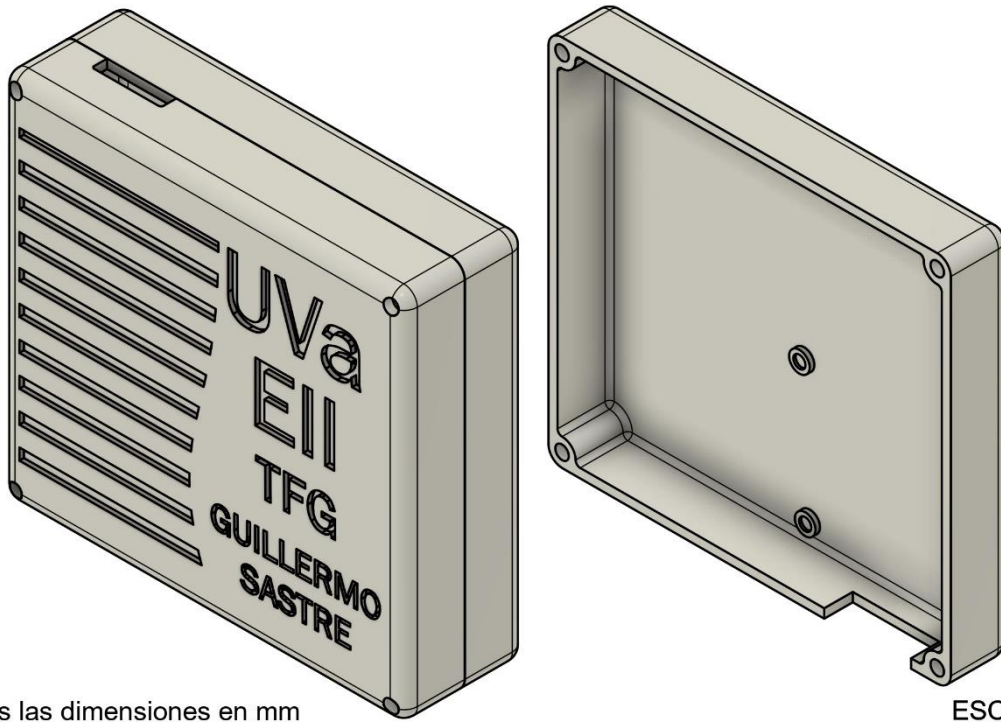
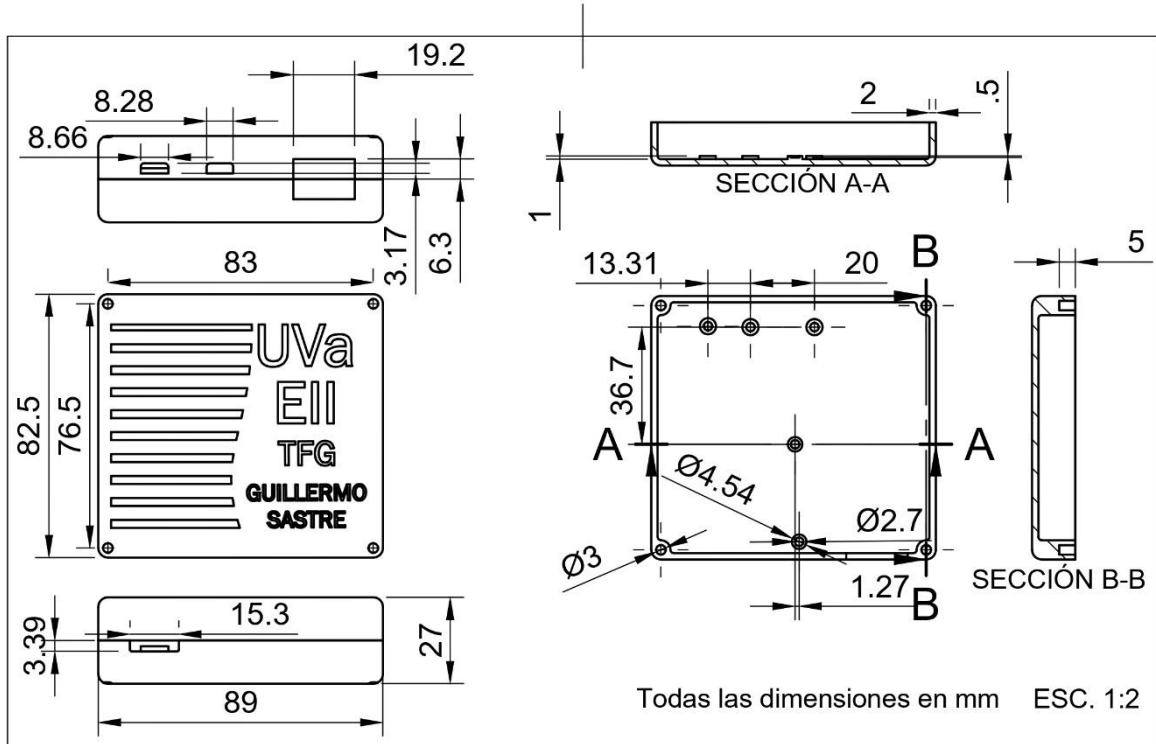


**UNIVERSIDAD DE VALLADOLID**  
**ESCUELA DE INGENIERIAS INDUSTRIALES**  
**DPTO. TECNOLOGIA ELECTRONICA**

**GUILLERMO SASTRE SAEZ**

REFERENCIA DE LA PCB CAPA: SERIGRAFIA	Rev. 1	Plano: 3 de 3
--	-----------	------------------

1  4



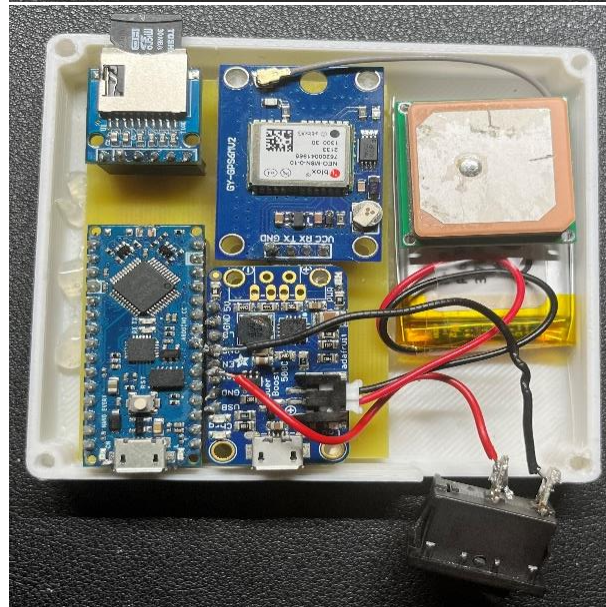
Dept. Tec. Elec.	Technical reference	Created by Guillermo Sastre Sáez 8/06/2024	Approved by Guillermo Sastre Sáez 08/06/2024
	Document type TFG Guillermo Sastre Sáez	Document status	
	Title Carcasa	DWG No.	
	Rev.	Date of issue 08/06/2024	Sheet 1/1



## ANEXO II. FOTOGRAFÍAS

1. Vista superior del prototipo construido
2. Vista superior sin parte de arriba del prototipo construido
3. Vista frontal del prototipo construido
4. Vista posterior del prototipo construido







### ANEXO III. HOJAS DE CARACTERÍSTICAS

1. Hoja de características de placa Arduino Nano Every (ABX00028) de Arduino.
2. Hoja de características de cargador LiPo PowerBoost 500C de Adafruit.
3. Hoja de características de batería LiPo 1000 mAh / 3.7 V (603050).
4. Hoja de características de módulo GPS GY-GPS6MV2 NEO-M8N-0-10.
5. Hoja de características de módulo lector de tarjetas microSD.

## 1 The Board

As all Nano form factor boards, Nano Every does not have a battery charger but can be powered through USB or headers.

**NOTE:** Arduino Nano Every is 5V compatible so it is a drop in replacement for the standard Nano board

## 2 Ratings

### 2.1 Recommended Operating Conditions

Symbol	Description	Min	Max
	Conservative thermal limits for the whole board:	-40 °C ( 40 °F)	85°C ( 185 °F)

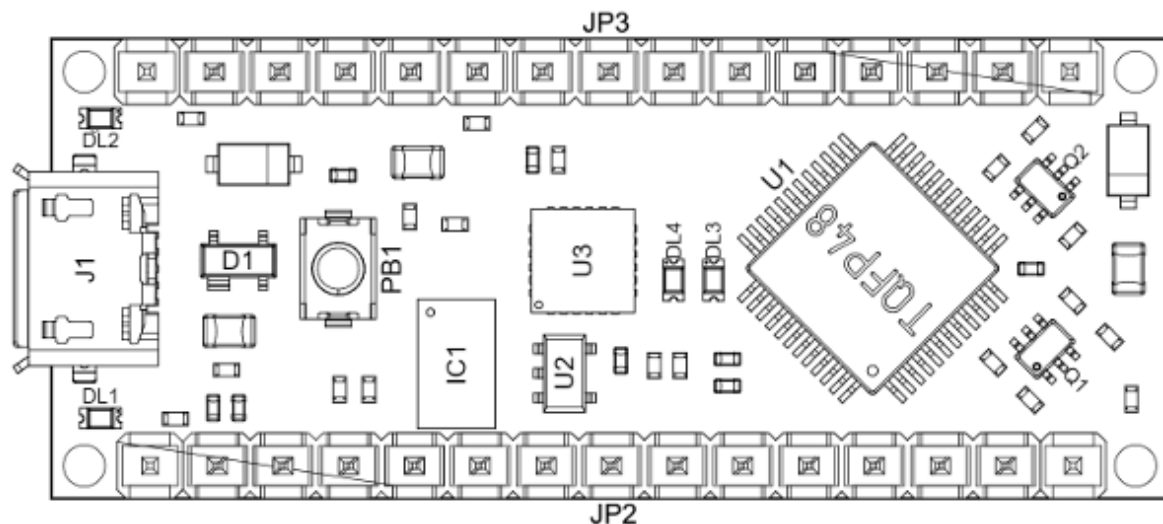
### 2.2 Power Consumption

Symbol	Description	Min	Typ	Max	Unit
PBL	Power consumption with busy loop		TBC		mW
PLP	Power consumption in low power mode		TBC		mW
PMAX	Maximum Power Consumption		TBC		mW

## 3 Functional Overview

### 3.1 Board Topology

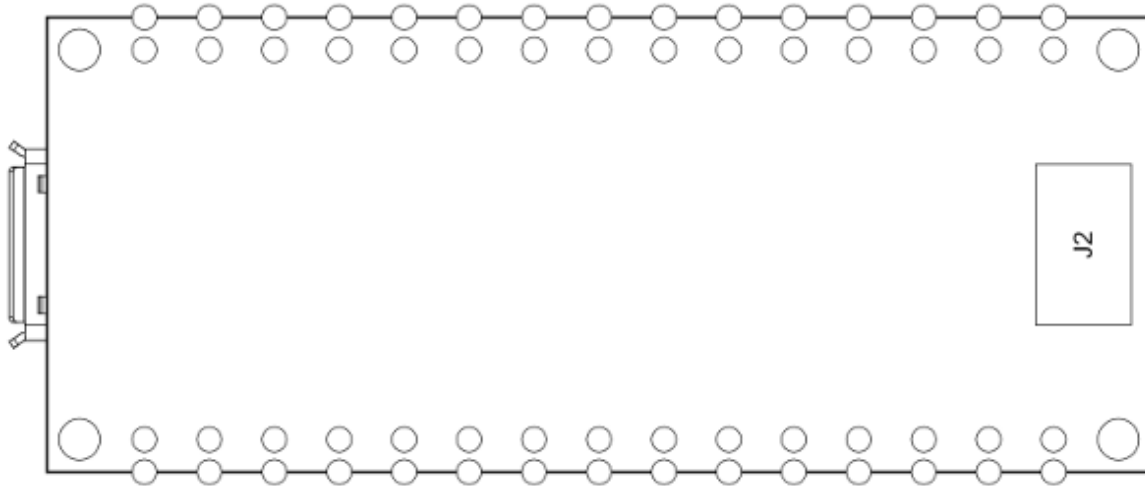
Top:



Board topology top

Ref.	Description	Ref.	Description
U1	ATMEGA4809-A.6 IC Module	D1	PRTR5VOU2X Diodes
U2	AP2112k-3.3TRG1 Diodes	PB1	T-1185AP1C-160G-GTR Push button
U3	ATSAM-D11 Chip	IC1	MPM3610AGQV-P Module
J1	Micro USB Connector		

Bottom:



Board topology bot

Ref.	Description	Ref.	Description
J2	Bridge?		

### 3.2 Processor

The Main Processor is an AVR running at up to 20MHz. Most of its pins are connected to the external headers, however some are reserved for internal communication with the USB Bridge coprocessor.

Communication with SAMD11D14A happens through a serial port and a single wire programming through the following pins:

ATMega4809 Pin	ATMega4809 Acronym	SAMD11 Pin	SAMD11 Acronym	Description
9	PB05	15	PA22	SAMD11 TX □ ATMega4809 RX
8	PB04	16	PA23	ATMega4809 TX □ SAMD11 RX
41	UPDI	12	PA15	UPDI RX
11	PA14	UPDI TX		

### 3.3 USB Bridge

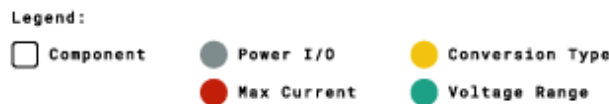
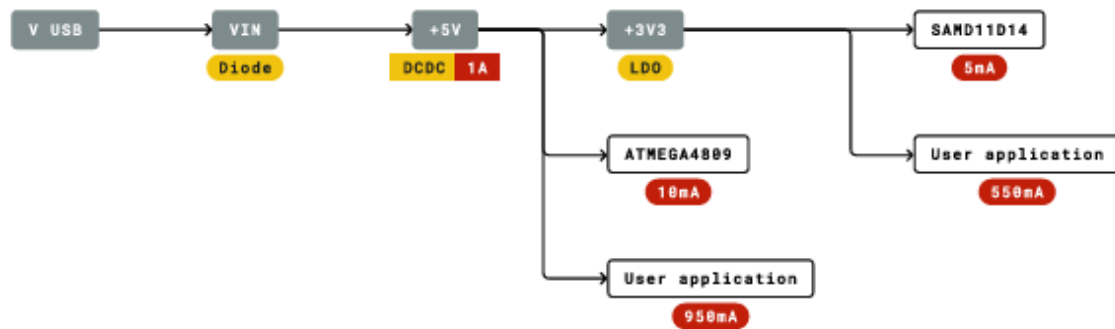
The SAMD11D14A processor is shipped with a firmware that implements USB to serial bridge and handles ATMega4809 firmware upgrade through the UPDI interface.

Firmware also has a bootloader that allows reprogramming to implement other USB classes, expanding the possibilities of classic Nano boards that are limited to serial bridge.

**NOTE:** SAMD11D14A pins are 3.3V only and are connected to ATMega4809 through level shifters. Although it is possible to wire its pins to the external world care must be taken as they are NOT 5V tolerant

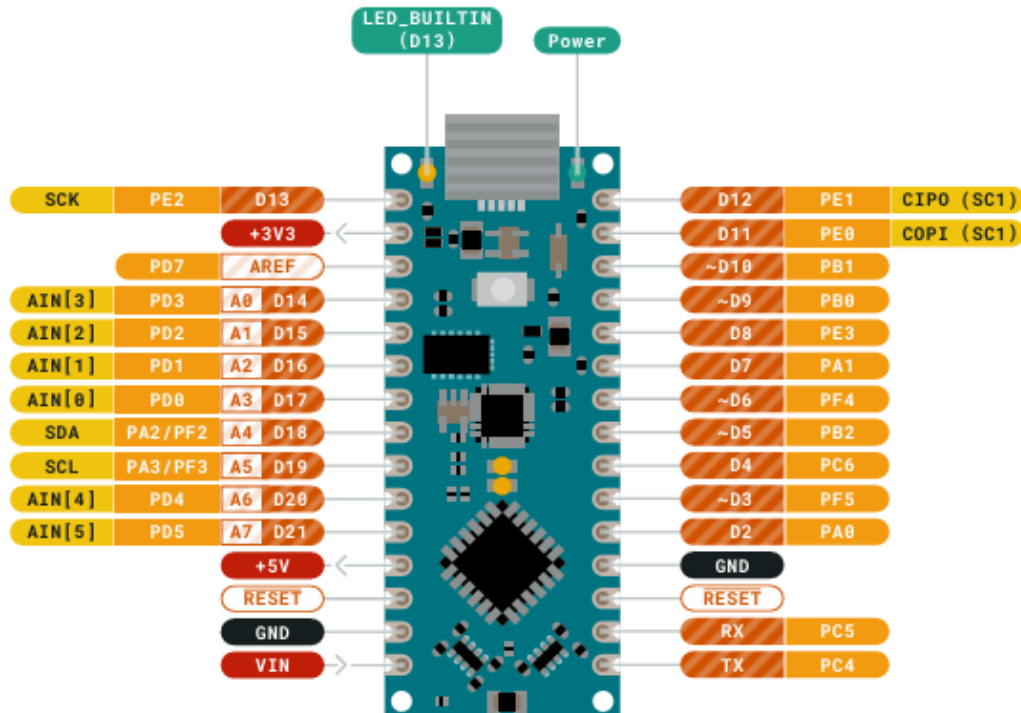
### 3.4 Power Tree

The board can be powered via USB connector,  $V_{IN}$  or  $V_{USB}$  pins on headers.



Power tree

**NOTE:** Since  $V_{USB}$  feeds  $V_{IN}$  via a Schottky diode and a DC-DC regulator specified minimum input voltage is 4.5V the minimum supply voltage from USB has to be increased to a voltage in the range between 4.8V to 4.96V depending on the current being drawn.



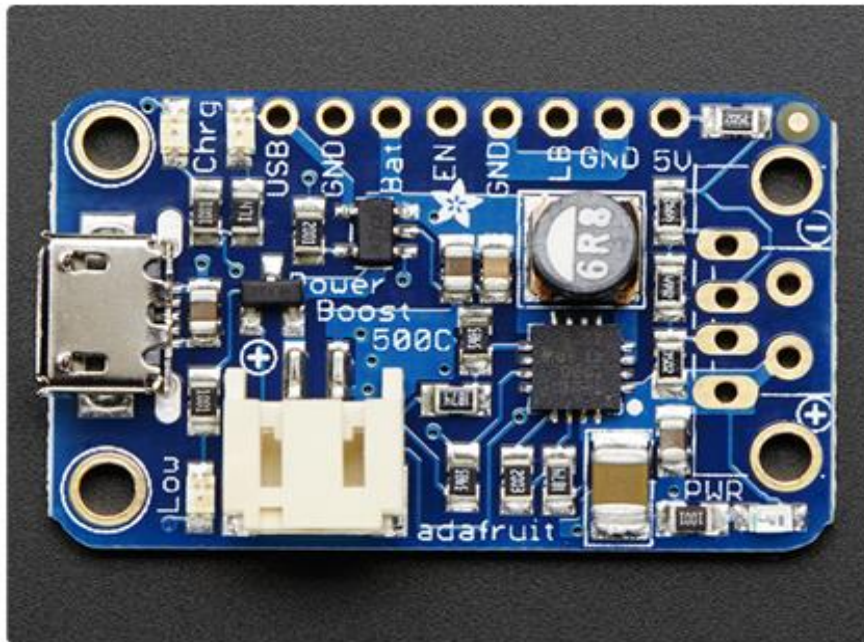


## Overview



PowerBoost 500C is the perfect power supply for your portable project! With a built-in battery charger circuit, you'll be able to keep your project running even while recharging the battery! This little DC/DC boost converter module can be powered by any 3.7V Lilon/LiPoly battery, and convert the battery output to 5.2V DC for running your 5V projects.

Like our popular [5V 1A USB wall adapter \(http://adafru.it/501\)](http://adafru.it/501), we tweaked the output to be 5.2V instead of a straight-up 5.0V so that there's a little bit of 'headroom' for long cables, high draw, the addition of a diode on the output if you wish, etc. The 5.2V is safe for all 5V-powered electronics like Arduino, Raspberry Pi, or Beagle Bone while preventing icky brown-outs during high current draw because of USB cable resistance.



The PowerBoost 500C has at the heart a [TPS61090 boost converter from TI \(https://adafru.it/duQ\)](https://adafru.it/duQ). This boost converter chip has some really nice extras such as low battery detection, 2A internal switch, synchronous conversion, excellent efficiency, and 700KHz high-frequency operation. Check out these specs!

- Synchronous operation means you can disconnect the output completely by connecting the **EN**able pin to ground. This will completely turn off the output
- 2A internal switch (~2.5A peak limiting) means you can get **500mA+** from a 3.7V LiPoly/Lilon battery. **We had no problem drawing 1000mA**, just make sure your battery can handle it!
- Low battery indicator LED lights up red when the voltage dips below 3.2V, optimized for LiPo/Lilon battery usage
- Onboard 500mA charge-rate 'iOS' data resistors. Solder in the USB connector and you can plug in any iPhone or iPod for 500mA charge rate. Not suggested for large iPads.
- Full breakout for battery in, control pins and power out
- 90%+ operating efficiency in most cases (see datasheet for efficiency graphs), and low quiescent current: 5mA when enabled and power LED is on, 20uA when disabled (power and low batt LED are off)

<https://adafru.it/dzw>

[adafruit\\_products\\_1944iso\\_USB\\_ORIG.jpg \(https://adafru.it/iQf\)](https://adafru.it/iQf)

To make this even more useful, we stuck a MicroLipo charger on the other side. The charger circuitry is powered from a microUSB jack, and will recharge any 3.7V/4.2V Lilon or LiPoly battery at 500mA max rate. There's two LEDs for monitoring the charge

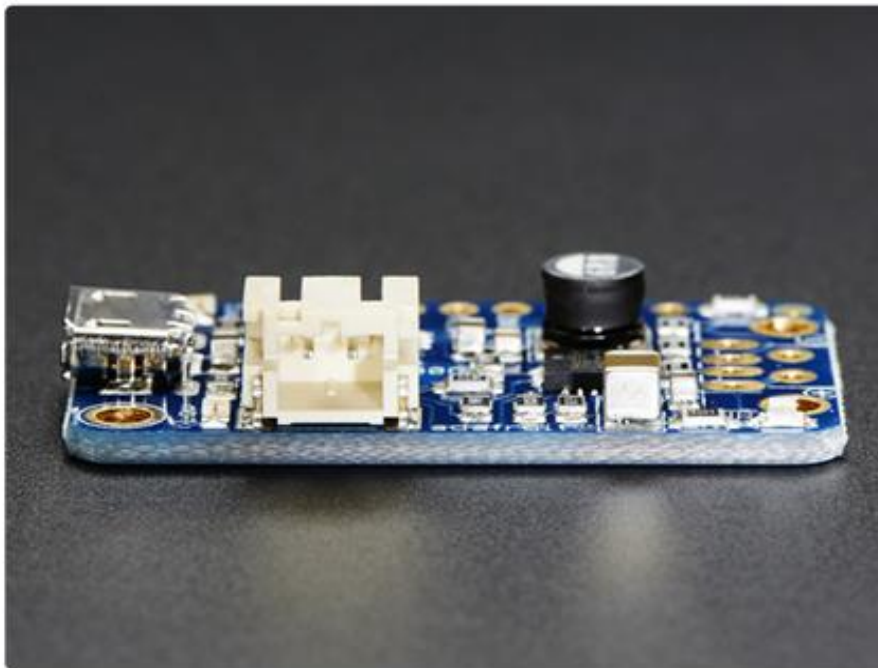




rate, a yellow one tells you its working, a green one lights up when its done. You can charge and boost at the same time no problem, without any interruption on the output so its fine for use as a "UPS" (un-interruptable power supply). Just be aware that the charge rate is 500mA max, so if you're drawing more than ~300mA continuously from the 5V output side, the battery will slowly drain since the charge rate is less than the dis-charge rate.

[adafruit\\_products\\_1944quarter\\_ORIG\\_298.jpg \(https://adafru.it/iQA\)](https://adafru.it/iQA)

Great for powering your robot, Arduino project, single-board-computer such as Raspberry Pi or BeagleBone! Each order comes with one fully assembled and tested PCB and a loose USB A jack. If you are powering your project from USB, solder the USB A jack in (a 3-minute soldering task). [If you would like to use a terminal block, pick up a 3.5mm 2pin block here \(http://adafru.it/724\)](http://adafru.it/724) and solder to the output spot where the USB jack would go. Or dont solder anything in for a more compact power pack.

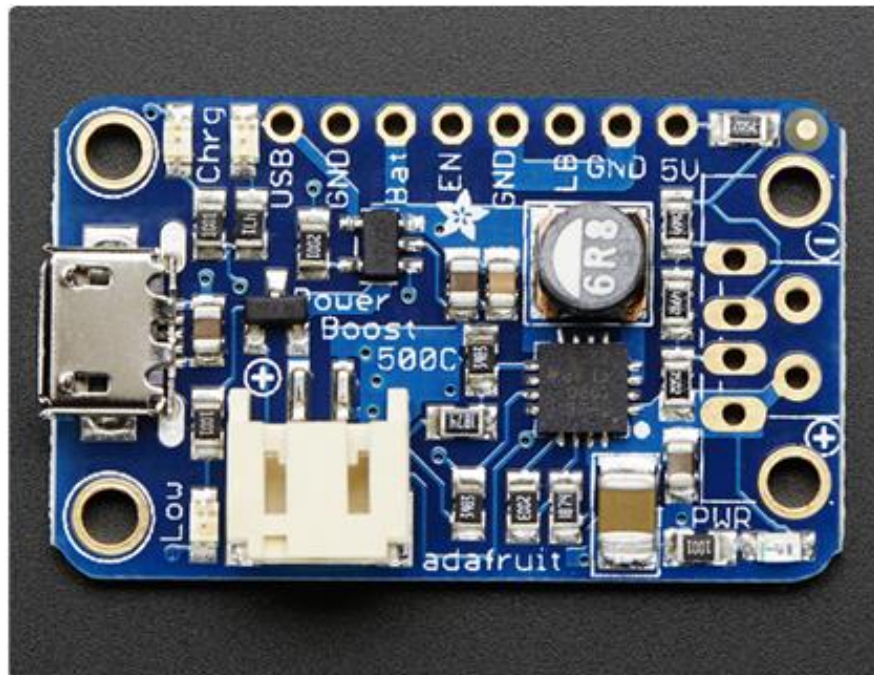


If you're trying to figure out how much current your project is using, check out the [CHARGER DOCTOR! \(http://adafru.it/1852\)](http://adafru.it/1852)

---

## Pinouts

For many people, the PowerBoost 500C can be used with just the microUSB charge input, battery plug and power outputs. However, we have a couple handy breakouts so lets get started!



## Power Pins

There's three power voltages, the USB input for charging the battery (4.75-5.25V whatever is coming out of the USB port), the battery itself (3-4.2V) and the output (5-5.2V)

- **USB** - this is the micro USB 5V power pin. It's the pin that is used to charge the battery, NOT the output power! You can use this if you want to grab power from the microUSB port when it is plugged in
- **GND** - this is the power ground. This boost converter is not 'isolated' - the ground input is the same as the ground output
- **BAT** - this is the battery input, connected directly to the JST connector. For most Lithium batteries, this will range from 3.0V when near-dead to 4.2V when fully-charged. Higher voltages will let you draw more current and in general, are more efficient. Try to keep the wires going to this pin nice and short - 3" or less is best!
- **5V** - this is the boosted output. When the board is running, the voltage will be 5.2V approximately. It may dip down to 5V as the current draw starts to go up (over 500mA). When the board is disabled, this output is 'floating' but you should still try not to apply a voltage to it while the board is disabled. There's a green LED connected to this pin which will let you know when there's power output



Using a bench-top supply or long wires can add too much inductance to the input of the boost converter and destroy it! We really do recommend using Lipoly batteries with short wires

## Control Pins

There's two 'control' pins.

- **EN** - this is the 'enable' pin. By default it is pulled 'high' to **VBAT**. To turn off the booster, connect this pin to ground. The switch can be as small as you like, it is just a signal. Contrast this to an inline power switch which would have to be able to handle up to 2A of current! When the chip is disabled the output is completely disconnected from the input.
- **LBO** - not a leveraged buy out! this is the **Low Battery Output**. By default it is pulled high to **BAT** but when the charger detects a low voltage (under 3.2V) the pin will drop down to 0V. You can use this to signal when its time to shut down or alert the user that the battery is low. There is also a red LED connected to this pin.

The LBO output is pulled up to  $V_{in}$ , and is not suitable for direct connection to 3.3V logic GPIO!

## LEDs

There are **four** onboard LEDs.

- The **Blue** LED sits next to the USB connector socket, and indicates the 5V output power state.
- The **Red** LED is next to the battery JST port and indicates when the battery voltage is below 3.2VDC (Low Battery Output)
- The **Yellow** LED is next to the microUSB connector and indicates when the battery is being charged
- The **Green** LED is also next to the microUSB connector and indicates when the battery is done charging (all full)





<b>UNIONFORTUNE PRODUCT SPECIFICATION</b>	<b>Doc. No.</b>	2006.3.16
	<b>Edition No.</b>	2.0
	<b>Sheet</b>	1/5

**1 Scope**

This product specification describes UNIONFORTUNE polymer lithium-ion battery. Please using the test methods that recommend in this specification. If you have any opinions or advices about the test items and methods, please contact us. Please read the cautions recommended in the specifications first, take the credibility measure of the cell's using.

**2 Product Type, Model and Dimension**

2.1 Type: Polymer lithium-ion battery

2.2 Model: 063450

2.3 Cell Dimension(Max, Thickness×Width×Length[mm<sup>3</sup>]): 6×34×50

Pack Dimension(Max, Thickness×Width×Length[mm<sup>3</sup>]): None

**3 Specification**

Item	Specifications	Remark
Nominal Capacity	<u>1000 mAh</u>	0.2C <sub>5</sub> A discharge
Nominal Voltage	3.7V	Average Voltage at 0.2C <sub>5</sub> A discharge
Charge Current	Standard: 0.2 C <sub>5</sub> A; Max: 1 C <sub>5</sub> A	Working temperature: 0~40℃
Charge cut-off Voltage	4.20±0.03V	
Standard Discharge Current	0.2C <sub>5</sub> A	Working temperature: 20~60℃
Max Discharge Current	2.0C <sub>5</sub> A	Working temperature: 0~60℃
Discharge cut-off Voltage	2.75 V	
Cell Voltage	3.7-3.9 V	When leave factory
Impedance	≤ <u>300 mΩ</u>	AC 1KHz after 50% charge
Weight	Approx: <u>20g</u>	
Storage temperature	≤1 month	-20~45℃
	≤3 month	0~30℃
	≤6 month	20±5℃
Storage humidity	65±20% RH	Best 20±5℃ for long-time storage

**4 General Performance**

**Definition of Standard charging method** □ At 20±5℃ charging the cell initially with constant current 0.2C<sub>5</sub>A till voltage 4.2V, then with constant voltage 4.2V till current declines to 0.05C<sub>5</sub>A.

Item	Test Methods	Performance	
4.1	0.2C Capacity	After standard charging, laying the battery 0.5h, then discharging at 0.2C <sub>5</sub> A to voltage 2.75V, recording the discharging time.	≥ 300min
4.2	1C Capacity	After standard charging, laying the battery 0.5h, then discharging at 1C <sub>5</sub> A to voltage 2.75V, recording the discharging time.	≥ 51min
4.3	Cycle Life	Constant current 1C <sub>5</sub> A charge to 4.2V, then constant voltage charge to current declines to 0.05C <sub>5</sub> A, stay 5min □ constant current 1C <sub>5</sub> A discharge to 2.75V □ stay 5min. Repeat above steps till continuously discharging time less than 36min.	≥ 300times
4.4	Capability of keeping electricity	20±5℃ After standard charging, laying the battery 28days, discharging at 0.2C <sub>5</sub> A to voltage 2.75V, recording the discharging time.	≥ 240min



Product summary

# NEO-M8 series

## Versatile u-blox M8 GNSS modules

**Versatile GNSS modules in different variants for easy manufacturing**

- Concurrent reception of up to 3 GNSS (GPS, Galileo, GLONASS, BeiDou)
- Industry leading -167 dBm navigation sensitivity
- Security and integrity protection
- Supports all satellite augmentation systems
- Advanced jamming and spoofing detection
- Product variants to meet performance and cost requirements
- Backward compatible with NEO-7 and NEO-6 families



12.2 × 16.0 × 2.4 mm



**Product description**

The NEO-M8 series of concurrent GNSS modules is built on the high-performing u-blox M8 GNSS engine in the industry-proven NEO form factor.

The NEO-M8 modules utilize concurrent reception of up to three GNSS systems (GPS/Galileo together with BeiDou or GLONASS), recognize multiple constellations simultaneously and provide outstanding positioning accuracy in scenarios where urban canyon or weak signals are involved. For even better and faster positioning improvement, the NEO-M8 series supports augmentation of QZSS, GAGAN and IMES together with WAAS, EGNOS, and MSAS. The NEO-M8 series also supports message integrity protection, geofencing, and spoofing detection with configurable interface settings to easily fit to customer applications.

The NEO-M8M is optimized for cost-sensitive applications, while NEO-M8N and NEO-M8Q provide the best performance. The future-proof NEO-M8N and NEO-M8J include an internal flash that allows future firmware updates. This makes NEO-M8N and NEO-M8J perfectly suited to industrial and automotive applications.

The I2C-compliant DDC interface provides connectivity and enables synergies with most u-blox cellular modules. For RF optimization, NEO-M8J, NEO-M8N, and NEO-M8Q feature an additional front-end LNA for easier antenna integration and a front-end SAW filter for increased jamming immunity.

u-blox M8 modules use GNSS chips qualified according to AEC-Q100, are manufactured in ISO/TS 16949 certified sites, and are fully tested on a system level. Qualification tests are performed as stipulated in the ISO16750 standard: "Road vehicles – Environmental conditions and testing for electrical and electronic equipment".

	NEO-M8J	NEO-M8M	NEO-M8N	NEO-M8Q
<b>Grade</b>				
Automotive				
Professional	•	•	•	•
Standard				
<b>GNSS</b>				
GPS / QZSS	•	•	•	•
GLONASS	•	•	•	•
Galileo	•	•	•	•
BeiDou	•	•	•	•
Number of concurrent GNSS	3	3	3	3
<b>Interfaces</b>				
UART	1	1	1	1
USB	1	1	1	1
SPI	1	1	1	1
DDC (I2C compliant)	1	1	1	1
<b>Features</b>				
Programmable (Flash)	•		•	
Data logging	•		•	
Additional SAW	•		•	•
Additional LNA	•		•	•
RTC crystal	•	•	•	•
Oscillator	C	C	T	T
Timepulse	1	1	1	1
<b>Power supply</b>				
1.65 V – 3.6 V		•		
2.7 V – 3.6 V	•		•	•

C = Crystal      T = TCXO



## NEO-M8 series

### Features

Receiver type	72-channel u-blox M8 engine GPS/QZSS L1 C/A, GLONASS L10F BeiDou B1I, Galileo E1B/C SBAS L1 C/A: WAAS, EGNOS, MSAS, GAGAN	
Nav. update rate <sup>1</sup>	Single GNSS: up to 18 Hz 2 concurrent GNSS: up to 10 Hz	
Position accuracy	2.5 m CEP	
Acquisition <sup>2</sup>	NEO-M8N/Q	NEO-M8M/J
Cold starts:	26 s	26 s
Aided starts:	2 s	3 s
Hot starts:	1 s	1 s
Sensitivity <sup>2</sup>		
Tracking & Nav.:	-167 dBm	-164 dBm
Cold starts:	-148 dBm	-148 dBm
Hot starts:	-157 dBm	-157 dBm
Assistance GNSS	AssistNow Online AssistNow Offline (up to 35 days) AssistNow Autonomous (up to 6 days) OMA SUPL & 3GPP compliant	
Oscillator	TCXO (NEO-M8N/Q) Crystal (NEO-M8M/J)	
RTC crystal	Built-in	
Anti jamming	Active CW detection and removal. Extra onboard SAW band pass filter (NEO-M8N/Q/J)	
Memory	ROM (NEO-M8M/Q) or flash (NEO-M8N/J)	
Supported antennas	Active and passive	
Raw data	Code phase output	
Odometer	Integrated in navigation filter	
Geofencing	Up to 4 circular areas GPIO for waking up external CPU	
Spoofing detection	Built-in	
Signal integrity	Signature feature with SHA 256	
Data-logger <sup>3</sup>	For position, velocity, time, odometer data	

<sup>1</sup> NEO-M8M/Q

<sup>2</sup> For default mode: GPS/SBAS/QZSS+GLONASS

<sup>3</sup> NEO-M8J and NEO-M8N

### Electrical data

Power supply	1.65 V to 3.6 V (NEO-M8M) 2.7 V to 3.6 V (NEO-M8N/Q/J)
Power	21 mA at 3.0 V (Continuous)
Consumption <sup>4</sup>	5.3 mA at 3.0 V Power Save mode (1 Hz)
Backup Supply	1.4 V to 3.6 V

<sup>4</sup> NEO-M8M in default mode: GPS/SBAS/QZSS+GLONASS

### Package

24 pin LCC (Leadless Chip Carrier): 12.2 x 16.0 x 2.4 mm, 1.6 g

### Environmental data, quality & reliability

Operating temp.	-40 °C to +85 °C
Storage temp.	-40 °C to +85 °C (NEO-M8N/Q/J) -40 °C to +105 °C (NEO-M8M)
RoHS compliant (lead-free)	
Qualification according to ISO 16750	
Manufactured and fully tested in ISO/TS 16949 certified production sites	
Uses u-blox M8 chips qualified according to AEC-Q100	

### Interfaces

Serial interfaces	1 UART 1 USB V2.0 full speed 12 Mbit/s 1 SPI (optional) 1 DDC (I2C compliant)
Digital I/O	Configurable timepulse 1 EXTINT input for Wakeup
Timepulse	Configurable: 0.25 Hz to 10 MHz
Protocols	NMEA, UBX binary, RTCM

### Support products

u-blox M8 Evaluation Kits:

Easy-to-use kits to get familiar with u-blox M8 positioning technology, evaluate functionality, and visualize GNSS performance.

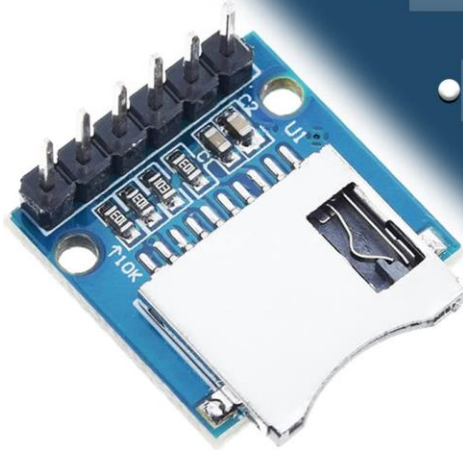
EVK-M8N	u-blox M8 GNSS Evaluation Kit, with TCXO, supports NEO-M8N/Q
EVK-M8C	u-blox M8 GNSS Evaluation Kit, with crystal, supports NEO-M8M/J

### Product variants

NEO-M8J	u-blox M8 concurrent GNSS LCC module, crystal, flash, SAW, LNA
NEO-M8M	u-blox M8 concurrent GNSS LCC module, crystal, ROM
NEO-M8N	u-blox M8 concurrent GNSS LCC module, TCXO, flash, SAW, LNA
NEO-M8Q	u-blox M8 concurrent GNSS LCC module, TCXO, ROM, SAW, LNA

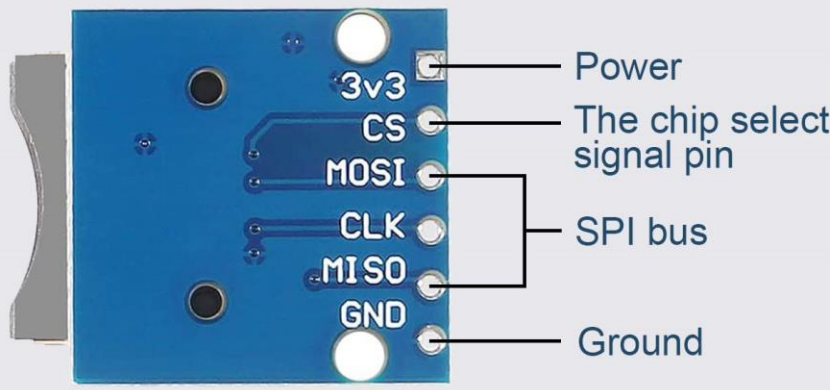
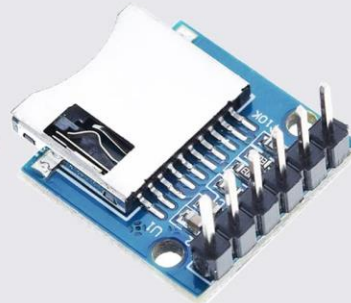
## SPECIFICATION

- Communication Interface: Standard SPI
- Interface Voltage Level: 3.3V or 5V
- Voltage: 4.5V~5.5V DC
- Current: 0.2~200mA



## INTERFACE

Micro SD TF  
Storage Board





## ADVANTAGE

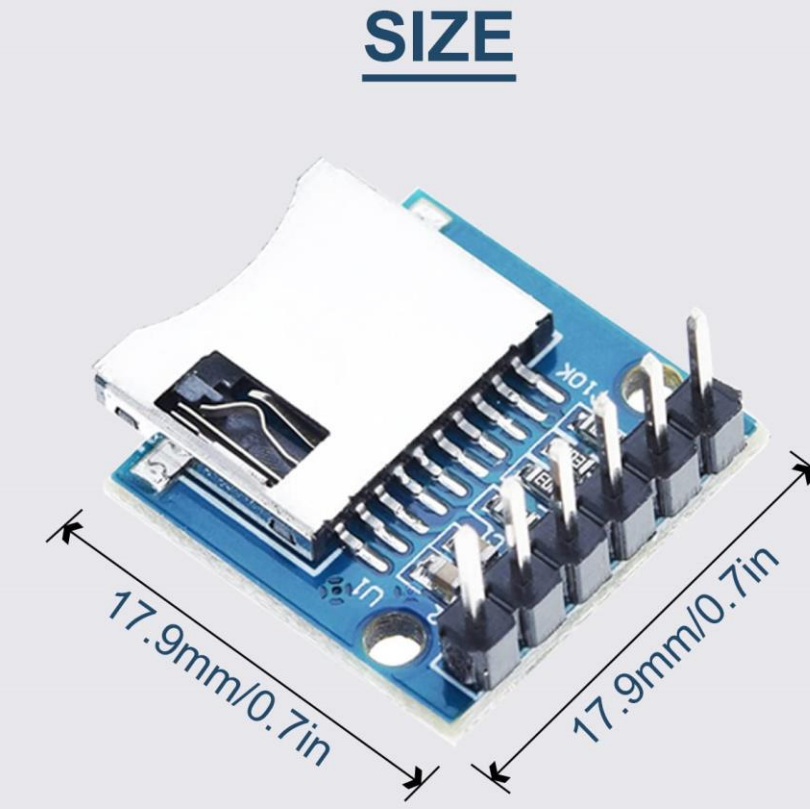
Small Volume

Easy To Use



Many other SD breakouts are much bigger than they need to be, and they're not only small and convenient, but easy to use at the same time. and works as expected in both SPI and SD 1-bit modes

## SIZE

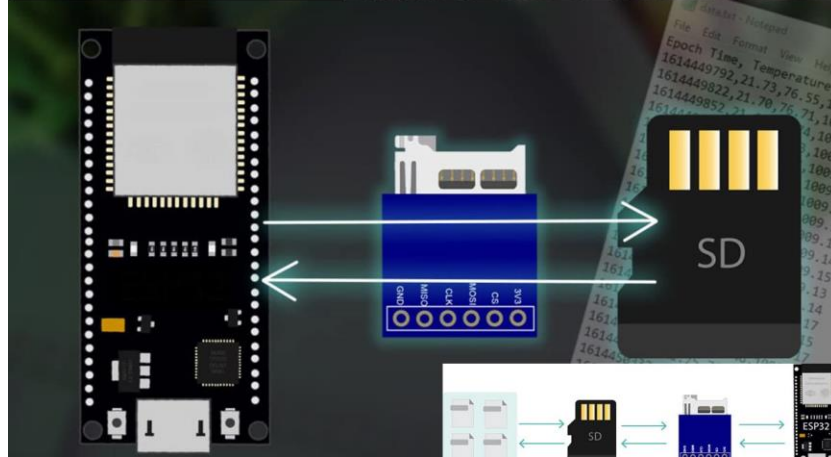


17.9mm/0.7in

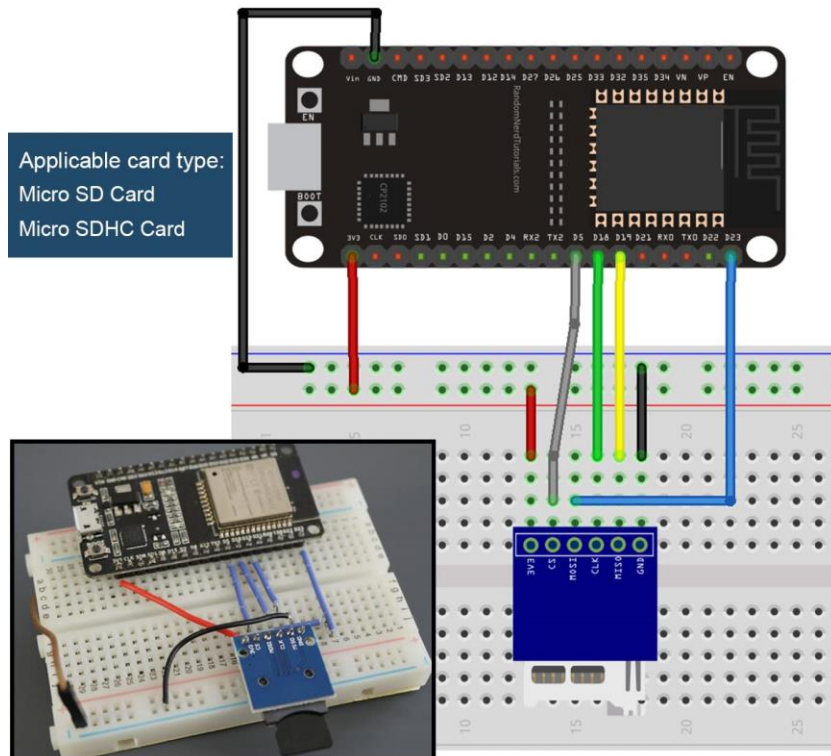
17.9mm/0.7in



Card read and write



## APPLICATION







## ANEXO V. PROGRAMAS DESARROLLADOS

1. Sketch de Arduino: *transformarDatos.ino*
2. Script de Python: *unificarDatos.py*



```
//Programa 'transformarDatos.ino'

// Inclusión de librerías a utilizar
#include <SoftwareSerial.h>           // Librería para utilizar
"SoftwareSerial"
#include <TinyGPSPlus.h>             // Librería para utilizar
del módulo GPS
#include <Math.h>                    // Librería para realizar
operaciones matemáticas
#include <SPI.h>                     // Librería para
comunicación SPI
#include <SD.h>                      // Librería para utilizar
la tarjeta microSD

// Declaración de constantes
static const uint32_t SERIAL_VB = 115200; // Velocidad de
transmisión (baudios/segundo) para la comunicación serial
static const uint32_t GPS_VB = 9600;     // Velocidad de
transmisión (baudios/segundo) del módulo GPS
static const int RX_PIN = 4, TX_PIN = 3; // Pines del Arduino Nano
asignados para la recepción y transmisión de datos vía UART
static const float RADIO_TIERRA = 6392.121; // Radio de la Tierra en
km. Necesario para calcular la distancia entre un punto y otro

// Declaración de variables globales
uint8_t hora = 0, hora_inicio = 0, hora_ESP = 0, minutos = 0,
minutos_inicio = 0, segundos = 0, segundos_inicio = 0, dia = 0, mes =
0;
uint16_t tiempo = 0, tiempo_total = 0, anio = 0;
uint32_t tiempo_actual = 0, tiempo_anterior = 0, tiempo_muestreo =
10000; // Período de muestreo
float latitud = 0.0, latitud_inicio = 0.0, longitud = 0.0,
longitud_inicio = 0.0, altitud = 0.0, altitud_inicio = 0.0, distancia
= 0.0,
    distancia_total = 0.0, velocidad = 0.0, ritmo = 0.0, pendiente =
0.0, pendiente_abs = 0.0,
float distancia_min = 0.005; // Distancia mínima que
debe haber entre un punto y otro.
int indice = 1;
String fichero;
bool gps_calibrado = false; // Indicador de calibración del GPS

// Declaración de objetos
TinyGPSPlus mi_GPS; // Se declara el objeto
"mi_GPS" de la librería TinyGPSPlus
```



```
SoftwareSerial Serial_GPS(RX_PIN, TX_PIN); // Se crea el puerto
serie "Serial_GPS". Se asignan los pines de recepción (D4 - RX_PIN) y
transmisión (D3 - TX_PIN) del Arduino Nano
File mi_fichero; // Se declara el objeto
"mi_fichero" de la librería File

// Función para inicializar el dispositivo
void inicializar_dispositivo(){

    Serial.begin(SERIAL_VB); // Inicio del puerto serie (UART)
    Serial_GPS.begin(GPS_VB); // Inicio del puerto serie del módulo
GPS (UART)
    while(!Serial); // Espera hasta que la comunicación
serial esté lista

    if (!SD.begin(10)) { // Inicio de comunicación SPI con
tarjeta microSD indicando el pin asignado para chip select (CS)
        Serial.println(F("ERROR: No se pudo inicializar la tarjeta"));
        while (true); // Si falla la inicialización, el
programa se detiene aquí
    }

    Serial.println("Latitud\tLongitud\tDistancia (km)\tDistancia total
(km)\tTiempo (s)\tTiempo total (s)\tVelocidad (km/h)\tRitmo
(min/km)\tAltitud (m)\tPendiente (%)\tPendiente Absoluta
(%)\tHora\tFecha");

    // Incrementa el índice del fichero si ya existe
    while (SD.exists("DATOS_" + String(indice) + ".TXT")) {
        indice++;
    }
    Serial.println(indice);
    fichero = "DATOS_" + String(indice) + ".TXT";

    // Crea un nuevo fichero si no existe
    if (!SD.exists(fichero)){
        mi_fichero = SD.open(fichero, FILE_WRITE);
        mi_fichero.println("Latitud\tLongitud\tDistancia (km)\tDistancia
total (km)\tTiempo (s)\tTiempo total (s)\tVelocidad (km/h)\tRitmo
(min/km)\tAltitud (m)\tPendiente (%)\tPendiente Absoluta
(%)\tHora\tFecha");
        mi_fichero.close();
    }
}
```



```
// Función para comprobar si el GPS está enviando datos y si existe
conexión con la placa Arduino
void comprobar_GPS(){

    while (Serial_GPS.available() > 0) {           // Comprueba
continuaamente si hay datos disponibles para ser leídos desde el puerto
serial del GPS
        if (mi_GPS.encode(Serial_GPS.read())) {   // Si hay datos
disponibles, el programa lee un byte del puerto serial del GPS y luego
lo decodifica
        }
    }

    if (millis() > 10000 && mi_GPS.charsProcessed() < 10) {
        Serial.println(F("ERROR: No se ha detectado módulo GPS. Compruebe
cableado."));
        while (true);                             // Si no se detecta el
módulo GPS, el programa se detiene aquí
    }
}

// Función para actualizar la información extraída del receptor GPS y
almacenarla en variables
void actualizar_info(){

    if (mi_GPS.location.isValid()) {
        latitud = redondear_dec(mi_GPS.location.lat(), 6);
        longitud = redondear_dec(mi_GPS.location.lng(), 6);
    }
    if (mi_GPS.altitude.isValid()) {
        altitud = mi_GPS.altitude.meters();
    }
    if (mi_GPS.time.isValid()) {
        hora = mi_GPS.time.hour();
        minutos = mi_GPS.time.minute();
        segundos = mi_GPS.time.second();
    }
    if (mi_GPS.date.isValid()) {
        dia = mi_GPS.date.day();
        mes = mi_GPS.date.month();
        anio = mi_GPS.date.year();
    }

    // Comprobar si el GPS está calibrado
    if (latitud == 0.0 && longitud == 0.0 && altitud == 0.0) {
        gps_calibrado = false;
    }
}
```





```
} else {
    gps_calibrado = true;
}

}

// Función para realizar los cálculos de distancia, tiempo entre
puntos y velocidad
void calcular_parametros(){

    if (gps_calibrado){
        // Cálculo de la distancia entre puntos (km)
        distancia = distancia_geografica(latitud_inicio, longitud_inicio,
latitud, longitud, RADIO_TIERRA);
        if(distancia > distancia_min){
            distancia_total += distancia;
        }
        latitud_inicio = latitud;
        longitud_inicio = longitud;

        // Cálculo del tiempo entre puntos (seg)
        tiempo = tiempo_entre_puntos(hora_inicio, minutos_inicio,
segundos_inicio, hora, minutos, segundos);
        tiempo_total += tiempo;
        hora_inicio = hora;
        minutos_inicio = minutos;
        segundos_inicio = segundos;

        // Cálculo de velocidad (km/h)
        if (distancia > distancia_min){
            velocidad = (distancia / tiempo) * 3600;
        } else{
            velocidad = 0.0;
        }

        // Cálculo del ritmo (min/km)
        if (distancia_total > 0.0){
            ritmo = (tiempo_total / 60.0) / (distancia_total);
        } else {
            ritmo = 0.0;
        }

        // Cálculo de la pendiente (%)
        if (distancia > distancia_min){
            pendiente = (altitud - altitud_inicio) / (distancia * 100) *
100; // Altura y distancia en metros. Se multiplica por 100 para
obtener resultado en tanto por ciento
```



```
    } else{
        pendiente = 0.0;
    }
    pendiente_abs = abs(pendiente);
    altitud_inicio = altitud;
}
}

// Función para calcular la distancia geográfica en kilómetros entre
// dos puntos dadas sus coordenadas de latitud y longitud (Fórmula de
// Haversine)
float distancia_geografica(float latitud_inicio, float
longitud_inicio, float latitud_fin, float longitud_fin, float
radio_Tierra) {

    float rad = M_PI /
180; // Conversión a
radianes
    float distancia_geografica = 0.0;

    if (latitud_inicio != 0.0 || longitud_inicio !=
0.0){ // Condición para "estabilizar" el primer
instante
        float diferencia_latitud = latitud_fin - latitud_inicio;
        float diferencia_longitud = longitud_fin - longitud_inicio;

        // Fórmula de Haversine
        float aux = pow(sin(diferencia_latitud * rad / 2), 2) +
cos(latitud_inicio * rad) * cos(latitud_fin * rad) *
pow(sin(diferencia_longitud * rad / 2), 2);
        distancia_geografica = 2 * radio_Tierra * asin(sqrt(aux));
    } else {
        distancia_geografica = 0.0;
    }

    return distancia_geografica;
}

// Función para calcular el tiempo transcurrido en segundos entre dos
// instantes de tiempo
float tiempo_entre_puntos(uint8_t hora_inicio, uint8_t minutos_inicio,
uint8_t segundos_inicio, uint8_t hora_fin, uint8_t minutos_fin,
uint8_t segundos_fin) {

    float tiempo_entre_puntos = 0;
```



```
if (hora_inicio != 0 || minutos_inicio != 0 || segundos_inicio !=
0){ // Condición para "estabilizar" el primer instante
float horas_transcurridas = hora_fin - hora_inicio;
float minutos_transcurridos = minutos_fin - minutos_inicio;
float segundos_transcurridos = segundos_fin - segundos_inicio;
tiempo_entre_puntos = (horas_transcurridas * 3600) +
(minutos_transcurridos * 60) + segundos_transcurridos;
} else {
tiempo_entre_puntos = 0;
}

return tiempo_entre_puntos;
}

// Función para redondear a las cifras decimales deseadas
float redondear_dec(float numero, int n_decimales){

float factor = pow (10, n_decimales);
return round (numero * factor) / factor; // La función "round()"
redondea un número decimal al entero más cercano. Si el decimal es
mayor o igual que la mitad, se redondea hacia arriba, si el decimal es
menor que la mitad se redondea hacia abajo.

}

// Función para mostrar la información por pantalla
void mostrar_info(){

Serial.print(latitud, 6);

// Se imprime "Longitud" por pantalla
Serial.print("\t");
Serial.print(longitud, 6);

// Se imprime "Distancia" y "Distancia total" por pantalla
Serial.print("\t");
Serial.print(distancia, 4);
Serial.print("\t");
Serial.print(distancia_total, 4);

// Se imprime "Tiempo entre puntos" y "Tiempo total" por pantalla
Serial.print("\t");
Serial.print(tiempo);
Serial.print("\t");
Serial.print(tiempo_total);

//Se imprime "Velocidad" por pantalla
```



```
Serial.print("\t");
Serial.print(velocidad, 2);

//Se imprime "Ritmo" por pantalla
Serial.print("\t");
Serial.print(ritmo, 2);

// Se imprime "Altitud" por pantalla
Serial.print("\t");
Serial.print(altitud, 2);

// Se imprime "Pendiente" por pantalla
Serial.print("\t");
Serial.print(pendiente, 2);

// Se imprime "Pendiente Absoluta" por pantalla
Serial.print("\t");
Serial.print(pendiente_abs, 2);

// Se imprime "Hora(horas:minutos:segundos)" por pantalla
Serial.print("\t");
hora_ESP = (hora + 2) % 24;
if (hora_ESP < 10) Serial.print(F("0"));
Serial.print(hora_ESP);
Serial.print(F(":"));
if (minutos < 10) Serial.print(F("0"));
Serial.print(minutos);
Serial.print(F(":"));
if (segundos < 10) Serial.print(F("0"));
Serial.print(segundos);

// Se imprime "Fecha(día/mes/año)" por pantalla
Serial.print("\t");
if (dia < 10) Serial.print(F("0"));
Serial.print(dia);
Serial.print(F("/"));
if (mes < 10) Serial.print(F("0"));
Serial.print(mes);
Serial.print(F("/"));
Serial.print(anio);

Serial.println();
}

// Función para escribir la información en fichero de texto de de la
microSD
void escribir_SD(){
```



```
// Abrimos fichero para escribir la información extraída del GPS
mi_fichero = SD.open(fichero, FILE_WRITE);

// Se escribe "Latitud" en fichero
mi_fichero.print(latitud, 6);

// Se escribe "Longitud" en fichero
mi_fichero.print("\t");
mi_fichero.print(longitud, 6);

// Se imprime "Distancia" y "Distancia total" en fichero
mi_fichero.print("\t");
mi_fichero.print(distancia, 4);
mi_fichero.print("\t");
mi_fichero.print(distancia_total, 4);

// Se escribe "Tiempo entre puntos" y "Tiempo total" en fichero
mi_fichero.print("\t");
mi_fichero.print(tiempo);
mi_fichero.print("\t");
mi_fichero.print(tiempo_total);

//Se escribe "Velocidad" en fichero
mi_fichero.print("\t");
mi_fichero.print(velocidad, 2);

//Se escribe "Ritmo" en fichero
mi_fichero.print("\t");
mi_fichero.print(ritmo, 2);

// Se imprime "Altitud" en fichero
mi_fichero.print("\t");
mi_fichero.print(altitud, 2);

// Se imprime "Pendiente" en fichero
mi_fichero.print("\t");
mi_fichero.print(pendiente, 2);

// Se escribe "Pendiente Absoluta" en fichero
mi_fichero.print("\t");
mi_fichero.print(pendiente_abs, 2);

// Se escribe "Hora(horas:minutos:segundos)" en fichero
mi_fichero.print("\t");
hora_ESP = (hora + 2) % 24;
if (hora_ESP < 10) mi_fichero.print(F("0"));
```



```
mi_fichero.print(hora_ESP);
mi_fichero.print(F(":"));
if (minutos < 10) mi_fichero.print(F("0"));
mi_fichero.print(minutos);
mi_fichero.print(F(":"));
if (segundos < 10) mi_fichero.print(F("0"));
mi_fichero.print(segundos);

// Se escribe "Fecha(día/mes/año)" en fichero
mi_fichero.print("\t");
if (dia < 10) mi_fichero.print(F("0"));
mi_fichero.print(dia);
mi_fichero.print(F("/"));
if (mes < 10) mi_fichero.print(F("0"));
mi_fichero.print(mes);
mi_fichero.print(F("/"));
mi_fichero.print(anio);

mi_fichero.println();

mi_fichero.close();
}

// Función de inicialización del programa
void setup() {

  inicializar_dispositivo();
}

// Función principal del programa
void loop() {

  tiempo_actual = millis();           // Obtener el tiempo actual en
  milisegundos

  comprobar_GPS();
  actualizar_info();

  // Se comprueba si ha transcurrido el intervalo de tiempo deseado
  (tiempo_muestreo) desde la última ejecución
  if ((tiempo_actual - tiempo_anterior) >= tiempo_muestreo){
    calcular_parametros();
    tiempo_anterior = tiempo_actual;
    if (distancia > distancia_min){
```



```
    mostrar_info(); // Muestra por pantalla la información extraída
del GPS
    escribir_SD(); // Escribe en fichero la información extraída
del GPS
}
}
}
```

```
# Programa 'unificarDatos.py'

import pandas as pd
import os

def mensaje_inicial():
    # Imprime un mensaje de bienvenida y una descripción del programa
    print("\n")
    print("-----")
    print("Bienvenido al programa: 'Unificación de Rutas'")
    print("-----")
    print("Este programa te permite convertir y unir los ficheros de
texto (.TXT) de nombre 'DATOS.TXT' que contienen información acerca
de las rutas realizadas")
    print("El resultado será un archivo Excel (.xlsx) de nombre
'historicoDatos.xlsx'")
    print("\n")

def obtener_rutas_existente():
    # Verifica si existe un archivo Excel anterior con rutas
unificadas
    if os.path.exists('historicoDATOS.xlsx'):
        df_existente = pd.read_excel('historicoDATOS.xlsx')
        # Obtiene una lista de los ID de rutas ya existentes
        rutas_existente = df_existente['rutaID'].unique()
    else:
        # Si no existe el archivo, retorna una lista vacía
        rutas_existente = []
    return rutas_existente

def obtener_archivos_a_procesar(rutas_existente):
    # Lista todos los archivos en la carpeta actual
    archivos_en_carpeta = os.listdir()
    # Filtra los archivos que comienzan con "DATOS_" y terminan con
".TXT"
    # y que su ID no esté en rutas_existente
    archivos_a_procesar = [archivo for archivo in archivos_en_carpeta
if archivo.startswith("DATOS_") and archivo.endswith(".TXT") and
int(archivo.split('_')[1].split('.')[0]) not in rutas_existente]
```





```
return archivos_a_procesar

def preguntar_convertir():
    # Pregunta al usuario si desea continuar con la conversión y
    # unificación
    respuesta = input("¿Deseas convertir y unir estos archivos? (S/N):
    ").upper()
    if respuesta == 'N':
        # Si la respuesta es 'N', el programa termina
        print("Pulsa Enter para salir")
        input()
        exit()
    elif respuesta != 'S':
        # Si la respuesta no es válida, se vuelve a preguntar
        print("Respuesta inválida. Por favor, ingresa 'S' para Sí o
        'N' para No.")
        preguntar_convertir()

def convertir_y_unir(archivos, rutas_existente):
    opciones_tipo_ruta = ["Andar", "Correr", "Ciclismo", "Otras"]
    dfs = []
    ruta_id_counter = 1 # Inicializa el contador de ID de rutas en 1
    for archivo in archivos:
        # Lee cada archivo TXT y lo convierte en un DataFrame
        df = pd.read_csv(archivo, delimiter='\t')

        # Filtra filas inválidas según criterios específicos
        if 'Latitud' in df.columns and 'Longitud' in df.columns:
            df = df[~((df['Latitud'] == 0) & (df['Longitud'] == 0))]
        if 'Fecha' in df.columns:
            df = df[df['Fecha'] != '00/00/0']
        if 'Hora' in df.columns:
            df = df[df['Hora'] != '02:00:00']

        if not df.empty:
            # Pide al usuario que ingrese el tipo de actividad para el
            # archivo actual
            tipo_ruta = input(f"Para el archivo {archivo}, ¿qué tipo
            de actividad ha realizado? (Andar/Correr/Ciclismo/Otras):
            ").capitalize()
            while tipo_ruta not in opciones_tipo_ruta:
                print("Opción inválida. Por favor, ingrese una de las
                siguientes opciones: Andar, Correr, Ciclismo, Otras")
                tipo_ruta = input(f"Para el archivo {archivo}, ¿qué
                tipo de actividad ha realizado? (Andar/Correr/Ciclismo/Otras):
                ").capitalize()
            # Añade las columnas tipoRuta y rutaID al DataFrame
```



```
df['tipoRuta'] = tipo_ruta
df['rutaID'] = ruta_id_counter
ruta_id_counter += 1
dfs.append(df)
else:
    # Muestra un mensaje si el archivo está vacío o contiene
datos no válidos
    print(f"El fichero {archivo} contiene datos no válidos o
está vacío, por lo que no se unificará.")

if dfs:
    # Si hay DataFrames válidos, se concatenan en uno solo
    resultado = pd.concat(dfs, ignore_index=True)
    if os.path.exists('historicoDATOS.xlsx'):
        # Si existe un archivo Excel anterior, se concatena con el
nuevo resultado
        df_existente = pd.read_excel('historicoDATOS.xlsx')
        resultado = pd.concat([df_existente, resultado],
ignore_index=True)
    try:
        # Guarda el DataFrame resultante en un archivo Excel
        resultado.to_excel('historicoDATOS.xlsx', index=False)
        print("Operación realizada exitosamente.")
    except Exception as e:
        # Maneja errores al guardar el archivo
        print(f"Error: algo salió mal. {e}")
else:
    # Si no hay datos válidos, muestra un mensaje
    print("No se encontraron datos válidos para unir.")
print("Pulsa Enter para salir")
input()

def main():
    # Ejecuta las funciones en orden para el flujo del programa
    mensaje_inicial()
    rutas_existente = obtener_rutas_existente()
    archivos = obtener_archivos_a_procesar(rutas_existente)
    if not archivos:
        # Si no hay archivos nuevos, muestra un mensaje y termina
        print("No hay ficheros 'DATOS_.TXT' disponibles para convertir
y unir.")
        print("Pulsa Enter para salir")
        input()
        exit()
    else:
        # Muestra la cantidad de archivos encontrados y pregunta si
continuar
```



```
    print(f"Se encontraron {len(archivos)} ficheros 'DATOS_.TXT'  
disponibles para convertir y unir.")  
    preguntar_convertir()  
    convertir_y_unir(archivos, rutas_existente)  
  
if __name__ == "__main__":  
    main()
```