Universidad Complutense de Madrid

Facultad de Informática

Reconstrucción de trayectorias en la especialidad deportiva de lanzamiento de martillo a partir de datos de acelerómetros

Curso Académico 2018/2019

Trabajo fin de grado
Grado en Ingeniería de Computadores

Autor:
Carlos Raspeño Priego

Directores de proyecto:
Luis Llana Díaz
Alberto Núñez Covarrubias

Resumen

En el mundo del deporte los resultados que obtienen los atletas son determinantes. La existencia de sensores que recogen todo tipo de información resulta muy importantes a la hora de analizar este tipo de datos con el fin de mejorar el rendimiento de los deportistas.

La adopción de sistemas como las placas Raspberry Pi facilitan el acceso a la tecnología. Esto, junto a la gran comunidad existente en torno a los proyectos que se realizan con estos sistemas, dan como resultado que la gran mayoría de la documentación y ayuda se encuentre de manera online, permitiendo que desarrolladores de todo distintos lugares intercambien sus conocimientos.

En este proyecto se ha utilizado una placa Rasberry Pi Zero y una unidad de medición inercial para recoger los datos de un atleta de lanzamiento de martillo, enviarlos - a través de la conectividad Bluetooth - a otro dispositivo y poder realizar el análisis exhaustivo de dichos datos. De esta manera, sería posible analizar los datos del atleta y usarlos para mejorar su técnica y ejecución, así como otros aspectos que puedan mejorar su rendimiento y convertirlo en un mejor atleta.

Palabras clave

- Raspberry Pi
- Bluetooth
- Unidad de medición inercial
- Multiplataforma
- Python
- Cliente-Servidor

Abstract

In sports, the results obtained by athletes are decisive. The existence of sensors that collect all kind of information is very important for analyzing this type of data in order to improve the performance of athletes.

The adoption of systems such as Raspberry Pi boards makes easier the access to technology. This, together with the large community around the projects that are carried out with these systems, makes the majority of documentation and help is found online, allowing developers from all different places to share their knowledge.

In this project, a Rasberry Pi Zero board and an inertial measurement unit have been used to collect data from a hammer throw athlete, send them - via Bluetooth connectivity - to another device and make an exhaustive analysis of these data. In this way, it would be possible to analyze the athlete's data and use them to improve his technique and execution, as well as other things that can improve his performance and make him a better athlete.

Keywords

- Raspberry Pi
- Bluetooth
- Inertial measurement unit
- Multi platform
- Python
- Client-Server

Índice general

Capítulo 1

Introducción

Los avances tecnológicos en los últimos años han hecho posible la reducción del tamaño de los componentes. Como consecuencia de esto, la reducción del tamaño de los dispositivos ha dado como resultado la aparición de los *wearable*, dispositivos electrónicos que se incorporan en alguna parte de nuestro cuerpo interactuando de forma continua con el usuario, así como con otros dispositivos con la finalidad de realizar alguna función concreta, tales como relojes inteligentes, pulseras de actividad y gafas de realidad aumentada.

Proyectos como el de la placa Raspberry Pi permiten, de una manera asequible, disponer de las capacidades de un ordenador en unas dimensiones reducidas. Su principal objetivo era estimular el aprendizaje de la informática en las escuelas, pero su bajo coste, junto con su versatilidad han hecho que sea uno de los de los componentes más usados en la comunidad "maker". Éste término se fundamenta en la frase "hacértelo tú mismo", un concepto que ha sido capaz de generar una gran cantidad de proyectos simplemente con una conexión a internet para acceder a la información, tiempo, foros en los que se produce un intercambio de conocimiento, y recursos de muy bajo coste como las placas Raspberry Pi o Arduino.

Actualmente existen multitud de libros acerca del mundo de la Raspberry. Sin embargo, debido a la naturaleza cambiante de este tipo de tecnología y a los nuevos dispositivos que se pueden utilizar en estas placas, la mayor parte de la documentación útil se encuentra de manera online gracias a la gran comunidad que existe en el desarrollo de proyectos con este tipo de hardware.

Debido a ello, para realizar este proyecto se ha pretendido unir estos dos mundos, usando una RaspberryPi Zero y una unidad de medición inercial con el fin de crear, desde cero, un dispositivo capaz de registrar datos de un atleta y analizarlos con el fin de usarlos en el beneficio de éste.

1.1. Antecedentes

En la actualidad los *wearables* deportivos, como por ejemplo, las pulseras de actividad y smartwatches con sensores deportivos, entre otros, se han convertido en uno de los dispositivos más demandados, ya sea para hacer deporte, únicamente para controlar la actividad que realizamos a diario o simplemente como complemento.

Este tipo de dispositivos cuentan con sensores que permiten llevar un registro, de una manera general, del ejercicio que hacemos en nuestro día a día, así como diferentes parámetros como la frecuencia cardíaca, control de sueño o contador de calorías. Estos datos suelen ser transferidos mediante la conectividad *bluetooth* a un *smartphone* el cual se encarga mostrar nuestro registro de entrenamiento, así como de enviar los datos a los servidores de la empresa fabricante del *wearable* en cuestión para así tener estos datos en la nube, o procesarlos para su visualizado en la aplicación.

Actualmente este tipo de dispositivos son usados por todo aquel que desee llevar un registro de su actividad con el fin de mejorar su condición física. Se pueden alcanzar las metas y objetivos. Gracias a sus beneficios son un producto de mucha demanda en estos tiempos.

1.2. Objetivos

El objetivo de este proyecto es el desarrollo de un programa que sea capaz de controlar a distancia el dispositivo usado por el atleta, recopilando las mediciones llevadas a cabo por dicho dispositivo en el desarrollo de la actividad deportiva, así como la creación del elemento que el atleta llevará puesto, y que alojará los distintos componentes electrónicos necesarios para realizar dicha tarea.

Para ello, los objetivos específicos para su desarrollo son los siguientes:

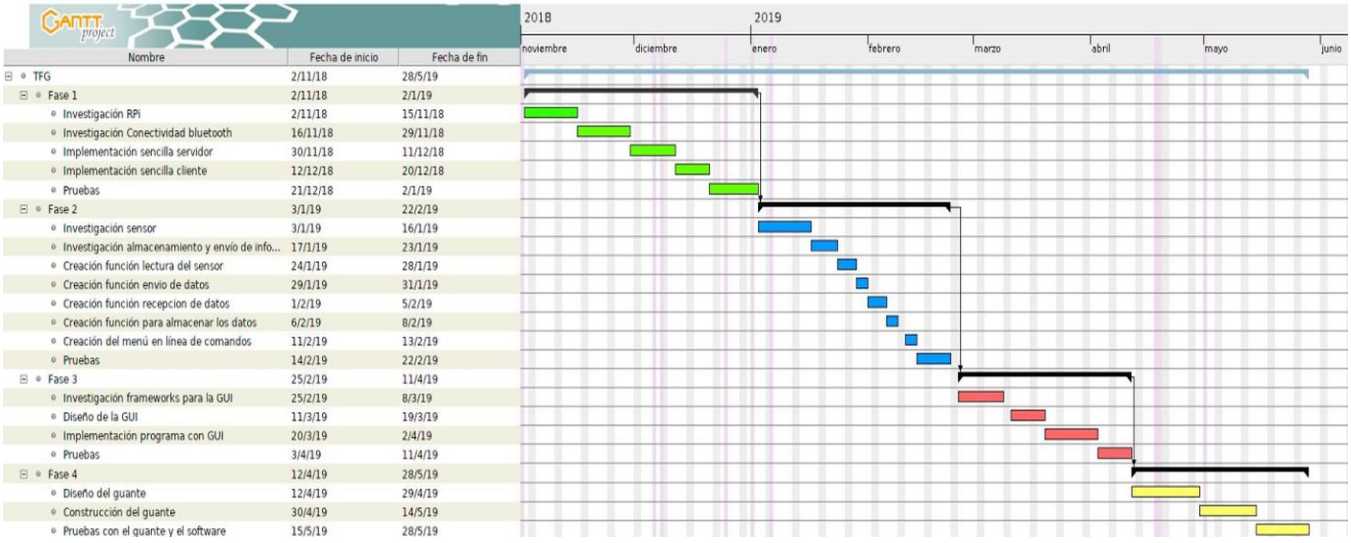
- Creación de un programa multiplataforma que pueda ser usado bajo diferentes sistemas operativos.
- Creación de un sistema de comunicación bajo la conectividad bluetooth.
- Implementación de un programa capaz de ejecutarse bajo linea de comandos con el fin de poder usarlo en sistemas livianos o que carezcan de interfaz gráfica.
- Implementación de un programa con una interfaz gráfica que resulte sencilla de usar.
- Permitir el almacenamiento persistente de la información extraída de los sensores para su envío y análisis posterior.

- Creación de un dispositivo capaz de contener a la Raspberry Pi y el sensor permitiendo al atleta realizar los lanzamientos sin problema llevando dicho dispositivo puesto.

1.3. Plan de trabajo

El trabajo se ha estructurado en las siguientes fases con el fin de cumplir los objetivos anteriormente mencionados:

- **Fase 1**: Estudio de la conectividad y diferentes maneras de implementación e implementación sencilla de un cliente-servidor.
- **Fase 2**: Estudio de la librería que controla el sensor y diferentes maneras de envío-recepción y almacenamiento de información. Implementación de la funciones que realizan la lectura de los datos del sensor, así como de las funciones necesarias para el envío y recepción de estos datos.
- **Fase 3**: Investigación de frameworks para el desarrollo de la interfaz gráfica y creación de la misma.
- **Fase 4**: Diseño y construcción del guante que incorporará la placa y el sensor.



Nombre	Fecha de inicio	Fecha de fin
TFG	2/11/18	28/5/19
Fase 1	2/11/18	2/1/19
Investigación RPi	2/11/18	15/11/18
Investigación Conectividad bluetooth	16/11/18	29/11/18
Implementación sencilla servidor	30/11/18	11/12/18
Implementación sencilla cliente	12/12/18	20/12/18
Pruebas	21/12/18	2/1/19
Fase 2	3/1/19	22/2/19
Investigación sensor	3/1/19	16/1/19
Investigación almacenamiento y envío de info...	17/1/19	23/1/19
Creación función lectura del sensor	24/1/19	28/1/19
Creación función envio de datos	29/1/19	31/1/19
Creación función recepcion de datos	1/2/19	5/2/19
Creación función para almacenar los datos	6/2/19	8/2/19
Creación del menú en línea de comandos	11/2/19	13/2/19
Pruebas	14/2/19	22/2/19
Fase 3	25/2/19	11/4/19
Investigación frameworks para la GUI	25/2/19	8/3/19
Diseño de la GUI	11/3/19	19/3/19
Implementación programa con GUI	20/3/19	2/4/19
Pruebas	3/4/19	11/4/19
Fase 4	12/4/19	28/5/19
Diseño del guante	12/4/19	29/4/19
Construcción del guante	30/4/19	14/5/19
Pruebas con el guante y el software	15/5/19	28/5/19

Figura 1.1: Gráfico Gantt

Capítulo 2

Introduction

Technological advances in last years have made possible to reduce the size of components. As a consequence of this, the reduction in the size of the devices has resulted in the appearance of the *wearables*, electronic devices to wear in some part of our body interacting continuously with the user, as well as with other devices in order to perform a specific function, such as smartwatches, smartbands and augmented reality glasses.

Projects like the Raspberry Pi board allow, in an affordable way, to have the capacities of a computer in small dimensions. Its main objective was to stimulate computer learning in schools, but its low cost and versatility have made it one of the most widely used components in the 'maker' community. This term is based on the phrase "do it yourself", a concept that has been able to generate a large number of projects simply with an internet connection to access information, time, forums to find and exchange of knowledge, and very low cost resources such as Raspberry Pi or Arduino boards.

Nowadays there are many books about the world of the Raspberry. However, due to the changing nature of this type of technology and the new devices that can be used on these boards, most of the useful documentation is found online thanks to the large community that exists in the development of projects with this kind of hardware.

Because of this, to make this project we want to unite these two worlds, using a RaspberryPi Zero and an inertial measurement unit in order to create, from scratch, a device capable of recording data of an athlete and analyze them for using them for the benefit of the athlete.

2.1. Background

In our days, sports *wearables*, such as smartbands and smartwatches with sports sensors, have become one of the most demanded devices, for doing sports, just to control the activity we do daily or simply as a complement.

This kind of devices have sensors that allow to keep a record of our daily exercise, as well as different parameters such as heart rate, sleep control or calorie counter. These data are usually transferred via *bluetooth* connectivity to a *smartphone* which is responsible of show our training record, as well as sending the data to the servers of the company manufacturer of the *wearable* in order to have these data in the cloud, or process this data for displaying in the application.

At present this kind of devices is used by anyone who wants to keep track of their activity in order to improve their physical condition. Goals and objectives can be achieved. Thanks to its benefits they are a product of great demand.

2.2. Objetives

The goal of this project is the development of a software with the ability of remotely controlling the device used by the athlete, recording the measurements of the device in the sporting activity, as well as the creation of the element the athlete will wear, this device has all the electronic components necessary to perform this task.

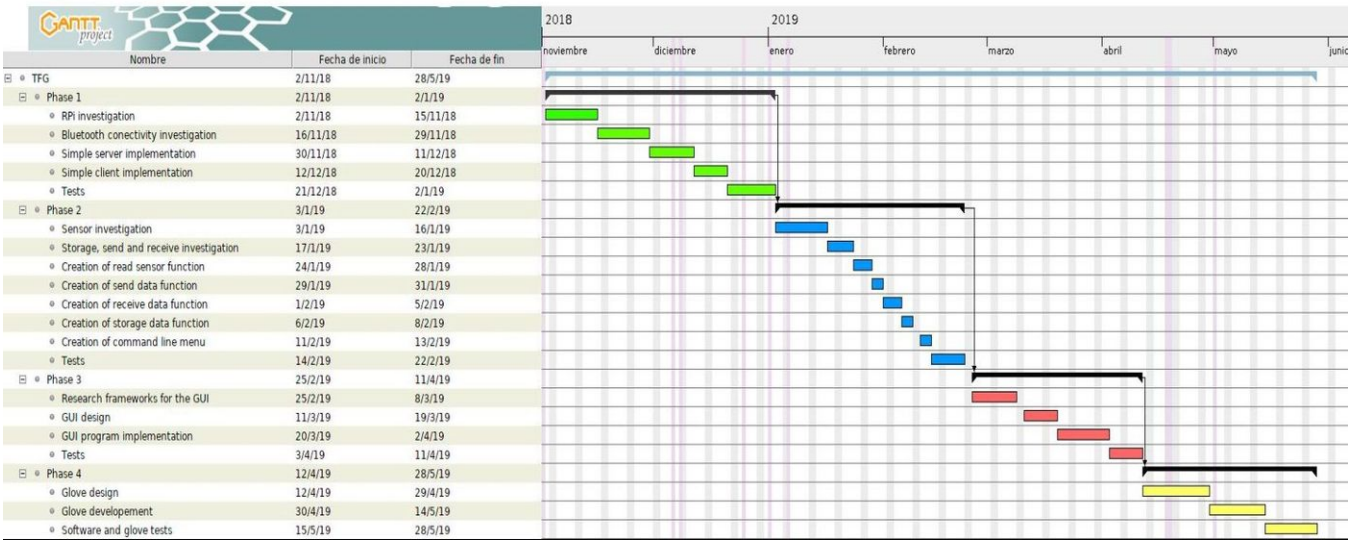
The specific objectives for its development are the following:

- Creation of a multiplatform software that can be used under different operating systems.
- Creation of a communication system under bluetooth connectivity.
- Implementation of a software capable of running under command line in order to be able to use it in light systems or systems without a graphical user interface.
- Implementation of a software with a friendly graphical user interface.
- Allow the persistent storage of the information extracted from the sensors for sending and analysis.
- Creation of a device able to contain inside the Raspberry Pi and the sensor allowing the athlete to make hammer throws with no problems for wearing the device.

2.3. Workplan

The work has been structured in the following phases in order to reach the objectives above:

- **Phase 1**: Study of connectivity and different ways of implementation and simple implementation of a client-server software.
- **Phase 2**: Study of the library that controls the sensor and different ways of sending-receiving and storing information. Implementation of the functions that carry out the reading of the sensor data, as well as the necessary functions for sending and receiving this data.
- **Phase 3**: Investigation of frameworks for the development of the graphical user interface and its creation.
- **Phase 4**: Design and construction of the glove that will incorporate the board and the sensor.



Nombre	Fecha de inicio	Fecha de fin
TFG	2/11/18	28/5/19
Phase 1	2/11/18	2/1/19
RPi investigation	2/11/18	15/11/18
Bluetooth conectivity investigation	16/11/18	29/11/18
Simple server implementation	30/11/18	11/12/18
Simple client implementation	12/12/18	20/12/18
Tests	21/12/18	2/1/19
Phase 2	3/1/19	22/2/19
Sensor investigation	3/1/19	16/1/19
Storage, send and receive investigation	17/1/19	23/1/19
Creation of read sensor function	24/1/19	28/1/19
Creation of send data function	29/1/19	31/1/19
Creation of receive data function	1/2/19	5/2/19
Creation of storage data function	6/2/19	8/2/19
Creation of command line menu	11/2/19	13/2/19
Tests	14/2/19	22/2/19
Phase 3	25/2/19	11/4/19
Research frameworks for the GUI	25/2/19	8/3/19
GUI design	11/3/19	19/3/19
GUI program implementation	20/3/19	2/4/19
Tests	3/4/19	11/4/19
Phase 4	12/4/19	28/5/19
Glove design	12/4/19	29/4/19
Glove developement	30/4/19	14/5/19
Software and glove tests	15/5/19	28/5/19

Figura 2.1: Gantt chart

Capítulo 3

Dispositivos

El hardware empleado para realizar el proyecto consiste en una placa Raspberry Pi zero, conectada a una unidad de medición inercial o IMU.

En un principio se hizo uso del sensor MPU-9250 como IMU, sin embargo, tras implementarlo y realizar las primeras pruebas, fue sustituido por el sensor BNO055, el cual es capaz de tomar mediciones que resultan mas útiles a la hora de analizar los datos y realizar los correspondientes cálculos matemáticos necesarios para su análisis.

Para dotar de portabilidad al conjunto, se ha empleado una batería de polímero de litio o LiPo, acompañada de un circuito de carga TP4056. La batería con el circuito de carga acoplado, se conecta a la placa a través de una fuente de alimentación Lipo Shim, que permite fácilmente la conexión de la batería a través del conector de tipo JST, además de otras funciones explicadas en este apartado.

En este capítulo se presentará con detalle el hardware usado para realizar el proyecto.

3.1. Raspberry Pi

La Raspberry Pi[19] es un ordenador de bajo coste y tamaño reducido que fue desarrollado en el Reino Unido por la Fundación Raspberry Pi, con el objetivo de estimular la enseñanza de informática en las escuelas y fue puesto a la venta por primera vez en en año 2012.

Actualmente existen tres generaciones de placas Raspberry Pi: Pi 1, Pi 2 y Pi 3, y en general ha existido un Modelo A y un Modelo B de la mayoría de las generaciones. El Modelo A es una variante más barata y tiende a tener RAM y puertos reducidos como USB y Ethernet.

Actualmente se encuentra en la tercera versión del hardware llamada Raspberry Pi 3 y cuenta con diferentes versiones con distintas características:

- **Raspberry pi 3B:** Incorpora un procesaror Quad-Core de la compañía Broadcom que funciona a 1.20GHz y cuenta con 1 GB de RAM. Su mayor novedad con respecto a los anteriores modelos fue la inclusión de Wi-Fi y Bluetooth (4.1 Low Energy) sin necesidad de adaptadores.
- **Raspberry pi 3B+:** Se trata de una actualizacion del modelo anterior y entre sus mejoras cuenta con un nuevo procesador y mejor conectividad,su procesador pasa de tener 1.2Ghz a tener 1.4Ghz e incorpora doble banda a 2,4GHz y 5GHz de conectividad, y su nuevo puerto Ethernet se triplica, pasa de 100 Mbits/s en el modelo anterior a 300 Mbits/s en el nuevo modelo, también cuenta con Bluetooth 4.2 (Low Energy).
- **Raspberry pi 3A+:** Los modelos A+ presentan menores prestaciones a un menor precio por lo que estan destinados a proyectos mas pequeños. Cuenta con 256 MB RAM, un solo puerto USB y sin puerto de conexión de red por cable (RJ-45), sin embargo, sigue manteniendo el mismo procesador que el modelo B+.

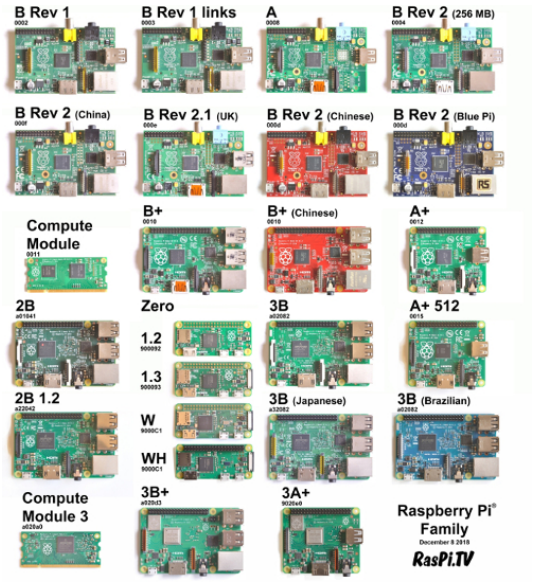


Figura 3.1: Diferentes versiones de Raspberry Pi a lo largo de los años. Fuente: https://opensource.com/resources/raspberry-pi

Uno de los principales atractivos de este tipo de placas es la inclusión de los GPIO. El GPIO[17] (General Purpose Input Output) es un sistema de entrada y salida de propósito general, y esta formado por una serie conexiones que se pueden usar como entradas o salidas. Estos pines están incluidos en todos los modelos de Raspberry Pi.

La primera versión de Raspberry Pi cuenta con 26 pines GPIO mientras que a partir de la segunda versión, el número de pines aumentó a 40 (figura 2.2). Sin embargo, ya que los 26 primeros pines mantienen su función original, la compatibilidad es total.

Cuando se utilizan los pines de GPIO hay que poner especial atención para no dañar la propia placa. Los pines de GPIO pueden generar y consumir tensiones compatibles con los circuitos de 3.3V por lo que conectar componentes de 5V puede provocar daños irreversibles en la placa.

La intensidad de corriente que sale de los pines del GPIO es de unos 3mA por cada pin, con la que se pueden encender diodos led, pero poco más.

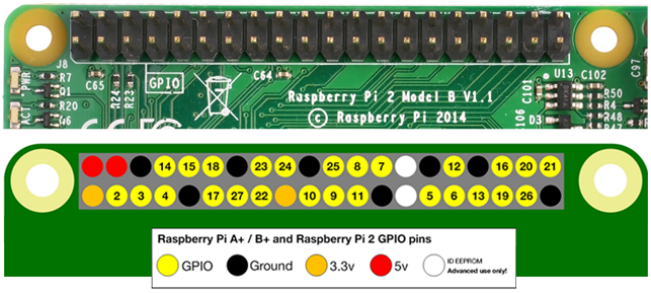


Figura 3.2: Pines GPIO para Raspberry Pi 2 Modelo B. Fuente: www.programoergosum.com/cursos-online/raspberry-pi/238-control-de-gpio-con-python-en-raspberry-pi/que-es-gpio

Además del modelo principal, también se creó el modelo Zero (el cual se utilizará para la realización de este proyecto, figura 2.3). Este modelo se caracteriza por ser mucho más pequeño y menos potente que la versión principal. Sin embargo, presenta un gasto energético mucho menor y resulta más asequible por lo que esta destinado a proyectos en los que el tamaño y consumos sean importantes pero no se requiera de mucha potencia.

La Raspberry Pi Zero tiene un microprocesador Broadcom BCM2835, que funciona a 1GHz con un solo núcleo, posee 512MB de RAM, y comparte la gráfica VideoCore IV. Debido a su tamaño cuenta con un puerto MiniHDMI, dos MicroUSB (uno de alimentación y otro de datos). Posee salida RCA, pero en vez de por clavija son solo dos conectores integrados en la placa y usa MicroSD como sistema de almacenamiento.

Existe una actualizacion de este modelo llamada Raspberry Pi Zero W, cuya única novedad con respecto a su antecesora es la inclusión de Wi-Fi y Bluetooth

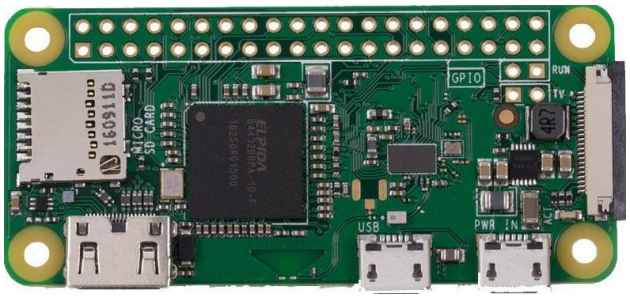


Figura 3.3: Raspberry Pi Zero W. Fuente:
https://www.raspberrypi.org/products/raspberry-pi-zero-w/

3.2. Sensor BNO055

El BNO055 es un sensor que incorpora un acelerómetro, magnetómetro y giroscopio en una placa junto con un procesador basado en ARM Cortex-M0 de alta velocidad para procesar todos los datos recogidos sensor.

El BNO055 puede sacar los siguientes datos a partir de los sensores:

- **Orientación Absoluta (Vector de Euler, 100Hz):** Tres ejes de orientación basados en una esfera de 360º.
- **Vector de Velocidad Angular (100Hz):** Tres ejes de velocidad de rotación en rad/seg.
- **Vector de Aceleración (100Hz):** Tres ejes de aceleración (gravedad + movimiento lineal) en m/s^2.
- **Vector de Campo Magnético (20Hz)**: Tres ejes de sensor de campo magnético en micro Tesla (uT).
- **Vector Lineal de Aceleración (100Hz):** Tres ejes de aceleración lineal (aceleración menos gravedad) en m/s^2.
- **Vector de Gravedad (100Hz):** Tres ejes de aceleración gravitacional (menos cualquier movimiento) en m/s^2.
- **Temperatura (1Hz):** Temperatura ambiente en grados celsius.

En cuanto a los pines de conexión, el BNO055 consta de los siguientes.

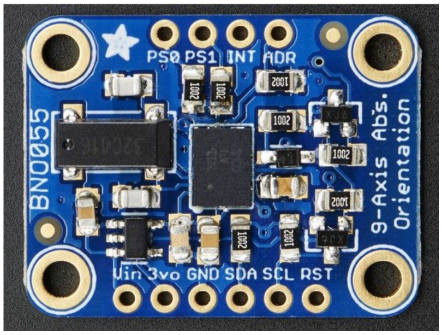


Figura 3.4: Vista del sensor BNO055 y sus pines. Fuente: https://learn.adafruit.com/adafruit-bno055-absolute-orientation-sensor/overview

Pines de alimentación:

- **VIN:** Entada de alimentación 3.3-5.0V.
- **3VO:** Salida desde el regulador de voltaje lineal incorporado, puede tomar hasta 50 mA según sea necesario.
- **GND:** El pin GND para alimentación y lógica.

Pines para la conexión I2C:

- **SCL:** Pin de reloj I2C, se conecta a la línea de reloj I2C del microcontrolador. Este pin se puede usar con lógica de 3V o 5V, y tiene un pullup de 10K.
- **SDA:** Pin de datos I2C, se conecta a la linea de datos I2C del microcontrolador. Este pin se puede utilizar con lógica de 3V o 5V, y tiene un pullup de 10K.

Otros pines:

- **RST:** Pin de reinicio de hardware. Poner la tensión de este pin en bajo y luego en alto provoca un reinicio en el sensor.
- **INT:** El pin de salida de interrupción de HW, que se puede configurar para generar una señal de interrupción cuando se producen ciertos eventos como el movimiento del acelerómetro, etc. (Actualmente no se admite en la biblioteca Adafruit, pero el chip y el HW pueden generar esta señal). El nivel de voltaje es 3V

- **ADR:** Poner la tensión de este pin en alto para cambiar la dirección I2C predeterminada para el BNO055 si necesita conectar dos circuitos integrados en el mismo bus I2C. La dirección por defecto es 0x28. Si este pin está conectado a 3V, la dirección será 0x29
- **PSO** y **PS1:** Estos pines pueden usarse para cambiar el modo del dispositivo (también puede hacer HID-I2C y UART) y también se proporcionan en caso de que Bosch proporcione una actualización de firmware en algún momento para el MCU ARM Cortex en el interior del sensor. Normalmente se deben dejar desconectados.

3.3. Sensor MPU-9250

El sensor MPU-9250[3] es una unidad de medición inercial o IMU (del inglés inertial measurement unit) fabricado por Invensense que incorpora un acelerómetro, un magnetómetro y un giroscopio.

Este sensor permite la comunicación tanto por SPI (serial peripheral interface) como por I2C (Inter-Integrated Circuit) y sus componentes principales cuentan con las siguientes características:

Giroscopio:

- Giroscopio con ejes X, Y y Z de salida digital con un rango programable por el usuario de +- 250, +- 500, +- 1000, y +- 2000 grados/ seg y conversores analógicos digitales (ADC) de 16bits.
- Low-pass filter programable digitalmente.
- Corriente de operación de 3.2mA.
- Modo sleep de 8μA.
- Factor de escala de sensibilidad calibrada de fábrica.
- Test de Autoevaluación.

Acelerómetro:

- Triple eje de salida digital con un rango de escala programable de +- 2g, +- 4g, +- 8gand +- 16gand y conversores analógicos digitales (ADC) de 16bits.
- Corriente de operación de 450μA.
- Corriente del modo baja potencia: 8,4μA a 0.98Hz, 19,8μA a 31.25Hz.
- Modo sleep de 8μA.
- Interrupciones programables por el usuario.
- Interrupción Wake-on-motion (reactivación por movimiento) para el funcionamiento de baja potencia del procesador de aplicaciones.
- Test de Autoevaluación.

Magnetómetro:

- Sensor magnético de silicio de efecto Hall con 3 ejes y concentrador magnético.
- Amplio rango de medición dinámico y alta resolución con menor consumo de corriente.
- Resolución de datos de salida de 14 bits (0,6μT / LSB).
- Rango de medición de +- 4800μT.
- Corriente de operación del magnetómetro de 280μA a una tasa de repetición de 8Hz.
- Función de Autoevaluación con fuente magnética interna para confirmar el funcionamiento del sensor magnético.

El motor interno Digital Motion Processing (DMP) admite funciones avanzadas de procesamiento de movimiento y baja potencia, como el reconocimiento de gestos mediante interrupciones programables y la funcionalidad de podómetro de baja potencia permite que el procesador host entre en modo sleep mientras el DMP mantiene el conteo de pasos.

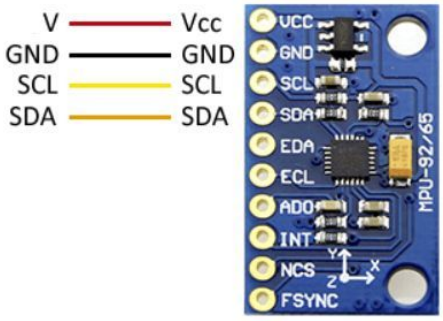


Figura 3.5: Sensor MPU-9250 y pines usados en el proyecto. Fuente: https://www.luisllamas.es/usar-arduino-con-los-imu-de-9dof-mpu-9150-y-mpu-9250/

3.4. TP4056 charger

El TP4056[4] es un cargador lineal completo de corriente constante / voltaje constante para baterías de iones de litio de una celda. Su paquete SOP y su bajo número de componentes externos hacen que el TP4056 sea ideal para aplicaciones portátiles.

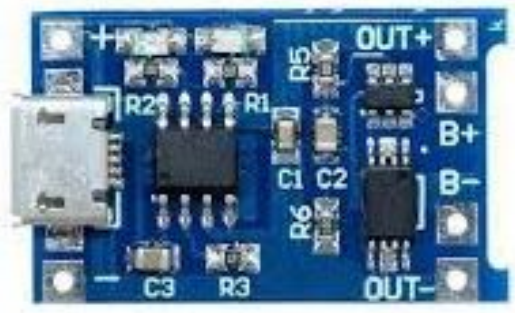


Figura 3.6: Vista del TP4056. Fuente: https://www.daraz.pk/products/tp4056-37v-li-ion-charger-module-1a-with-battery-protection-i111146658-s1262278023.html

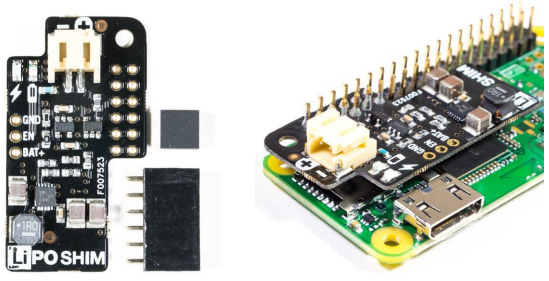
3.5. Fuente de alimentación Lipo Shim

Su principal objetivo es ser la fuente de alimentación para Raspberry Pi. Este componente[13] no permite la carga por lo que se hace necesario un cargador como el TP4056 anteriormente descrito si se utiliza una batería.

El convertidor de refuerzo TPS61232 de Texas Instruments ofrece hasta un 96 por ciento de eficiencia. La placa incluye indicadores LED de encendido y batería baja. Durante el apagado (debido a la baja tensión o la selección externa), la corriente de reposo es solo de 15 uA.

Especificaciones:

- PCB de 0.8 mm de grosor
- Perfil lo más bajo posible
- Conector JST de 2 polos, ideal para la mayoría de baterías LiPo/LiIon
- LEDs indicadores de alimentación y bajo nivel de batería
- Proporciona corriente continua de 1.5A
- Alerta de nivel bajo de batería a 3.4V
- Apagado automático a 3.0V para proteger la batería
- Pines VBAT+, GND y EN accesibles
- Consumo de corriente en reposo de 15uA



(a) Lipo Shim (b) Lipo Shim montado en el GPIO de la RPI

Figura 3.7: Lipo Shim. Fuente: https://thepihut.com/products/lipo-shim

Capítulo 4

Desarrollo

En este capítulo se describirá el proceso de desarrollo del proyecto, así como los recursos usados.

Para el desarrollo del proyecto se han creado 3 programas distintos. Uno es el servidor, otro el cliente con GUI, y el otro es un cliente sin GUI que funciona sobre la terminal, para poder utilizar en entornos sin interfaz gráfica.

- El servidor (ubicado en la Raspberry Pi) será el encargado de recoger los datos del sensor, así como el envío de estos hacia el cliente.
- El cliente será el programa que manejará el usuario y se encargará de realizar la conexión con el servidor (que permanece a la espera de una conexión), de la recepción de los datos, etc.

Debido a que en una primera versión se usó el sensor MPU-9250 y más tarde se cambió por el BNO055 (ya que este es da mas datos útiles para el proyecto), se creó un servidor, un cliente con GUI y un cliente sin GUI para cada uno de los sensores usados.

4.1. Python

El lenguaje de programación elegido para realizar este proyecto es Python[15] (en concreto Python 3) debido a su portabilidad y capacidad de ejecución en distintas plataformas.

Python es un lenguaje de programación interpretado, interactivo y orientado a objetos. Incorpora módulos, excepciones, tipificación dinámica, tipos de datos dinámicos de muy alto nivel y clases. Python combina un poder notable con una sintaxis muy clara. Tiene

interfaces para muchas bibliotecas y llamadas de sistema, así como para varios sistemas de ventanas y es extensible en C o C ++. También se puede usar como un lenguaje de extensión para aplicaciones que necesitan una interfaz programable. Finalmente, Python es portátil: se ejecuta en muchas variantes de Unix, en Mac y en Windows 2000 y posteriores.

La Python Software Foundation[16] (PSF) es una organización independiente sin ánimo de lucro que posee los derechos de autor de las versiones 2.1 y posteriores de Python. La misión del PSF es avanzar en la tecnología de código abierto relacionada con el lenguaje de programación Python y dar a conocer el uso de Python. La página de inicio de PSF se encuentra en https://www.python.org/psf/.

La versión 3 de Python es incompatible con la versión 2 del mismo. El lenguaje es casi el mismo, pero muchos detalles, especialmente cómo funcionan los objetos incorporados, como los diccionarios y las cadenas, han cambiado considerablemente, y finalmente se han eliminado muchas características obsoletas. Además, la biblioteca estándar se ha reorganizado en algunos lugares destacados.

Si se desea reescribir el código de la versión 2 para hacerla compatible con la versión 3 se lanzó una herramienta llamada 2to3 que facilita esta tarea.

Debido a que Python 2 todavía tiene soporte oficial (en concreto hasta 2020) y que muchas librerías no eran totalmente compatibles con python 3, hasta ahora muchos desarrolladores han seguido utilizándola por encima de la versión 3, sin embargo cada vez más se recomienda utilizar la versión 3 y portar el código para hacerlo compatible ya que esta versión se considera el presente y el futuro de este lenguaje.

En la pagina https://www.python.org/downloads/ se pueden descargar las versiones actuales y pasadas para los diferentes sistemas operativos, en concreto:

- Windows (a partir de la versión 3.5 y superiores no es compatible con Windows XP y versiones anteriores)
- Linux/UNIX (Python viene integrado por defecto en la muchas de las distribuciones mas populares de Linux)
- Mac OS X

También se proporcionan paquetes para otros sistemas:

- AIX
- IBM i
- iOS

- OS/390 and z/OS
- Solaris
- VMS
- HP-UX
- Paquetes alternativos de Python para Linux

Figura 4.1: Logo de Python. Fuente:
https://www.python.org/community/logos

Se recomienda a los proyectos y compañías que usan Python que incorporen el logotipo de Python en sus sitios web, folletos, empaques y otros lugares para indicar el uso o la implementación en Python. El logotipo formado por las "dos serpientes"solo, sin la marca de texto que lo acompaña está permitido en los mismos términos que el logotipo combinado.

Python posee una licencia de código abierto, denominada Python Software Foundation License (o PSFL), que es compatible con la Licencia pública general de GNU a partir de la versión 2.1.1, e incompatible en ciertas versiones anteriores, y cumple con los requisitos OSI para ser declarada licencia de software libre. La Python Software Foundation License no es copyleft y permite la modificación de su código fuente, por lo que todas las licencias de Python, permiten distribuir versiones modificadas sin necesidad de hacer el código open-source.

En la pagina web oficial https://docs.python.org/3/license.html es posible consultar todos los detalles de esta licencia.

4.2. Sistema operativo utilizado

El sistema operativo que se ha usado en la Raspberry Pi se trata de Raspbian Lite. Esta versión de Raspbian carece de interfaz gráfica y es más ligera que la versión estándar. Raspbian es una distribución del sistema operativo GNU/Linux basada en Debian diseñada para ser usada en esta placa.

Para conseguir que nuestro servidor se ejecute automáticamente al encender la Raspberry Pi, se ha creado un servicio con *Systemd* cuyo código es el siguiente y que se guardará en un archivo en la ruta `/etc/systemd/system/` con el nombre pi.service:

```
[Unit]
Description=Script de inicio TFG
After=network.target

[Service]
ExecStart=/home/pi/tfg/piscript.sh
Type=oneshot
RemainAfterExit=true

[Install]
WantedBy=default.target
```

Donde:

- **Description:** Contiene información que es mostrada en el log *systemd* cuando se ejecuta `systemctl status pi.service`.
- **After:** indica el servicio tras el que correrá nuestro script, en este caso correrá después de la inicialización de la red (**networking.target**).
- **ExecStart:** Especificar la ruta completa al script que va a ser ejecutado por el servicio.
- **Type:** Indica el tipo de servicio, en este caso con **oneshot** se ejecutan los archivos y se espera a su finalización.
- **RemainAfterExit:** Indica que tras la ejecución del script el servicio se marca como activo.
- **WantedBy:** Especifica dentro de qué target de systemd debe ser instalado, en este caso se ha dejado a **default**.

Este servicio ejecuta un script en *bash* ubicado en la ruta `/home/pi/tfg/` llamado piscript.sh:

```
#!/bin/bash

sudo hciconfig hci0 piscan

cd /home/pi/tfg

sudo python3 rfcomm-server.py
```

El funcionamiento de este script es bastante sencillo, por un lado la instrucción `sudo hciconfig hci0 piscan` activa el modo *discoverable* del bluetooth de la Raspberry haciéndola visible para otros dispositivos mientras que `sudo python3 rfcomm-server.py` ejecuta nuestro servidor.

Una vez esta todo configurado se deberá reiniciar el sistema o bien ejecutar este comando para que *Systemd* reconozca el servicio:

```
systemctl daemon-reload
```

Para iniciar el servicio bastara con ejecutar el siguiente comando en una terminal:

```
systemctl start pi.service
```

Si queremos parar la ejecución del servicio:

```
systemctl stop pi.service
```

Para que se inicie automáticamente cuando se enciende el sistema:

```
systemctl enable pi.service
```

Para eliminarlo del inicio automatico:

```
systemctl disable pi.service
```

4.3. Conectividad

Para conectar el cliente y el servidor se ha usado la conectividad bluetooth.

Bluetooth[2] es una especificación industrial para Redes Inalámbricas de Área Personal (WPAN) de corto alcance creado por Bluetooth Special Interest Group, Inc. que posibilita la transmisión de voz y datos entre diferentes dispositivos mediante un enlace por radiofrecuencia en la banda ISM de los 2.4 GHz con un rango óptimo de 10 m.

Debido a que uno de los tipos de wifi[20] opera en el mismo rango que el bluetooth (2.4 GHz) es posible que si se esta haciendo uso de esta conexión en la Raspberry Pi o en el dispositivo cliente se produzcan interferencias que dificulten la conexión o el intercambio de datos.

El estándar de intercambio de datos inalámbrico Bluetooth utiliza una variedad de protocolos[14]. Los protocolos centrales están definidos por la organización comercial Bluetooth SIG. Se han adoptado protocolos adicionales de otros organismos de normalización.

Bluetooth está definido como un protocolo de arquitectura de capa que está formado por unos protocolos centrales, protocolos de reemplazo de cable, protocolos de control de telefonía, y protocolos adoptados.

La pila de protocolo Bluetooth se divide en dos partes: el "controller stack" o "pila de controlador" que contiene la interfaz de radio bluetooth cuyo funcionamiento es critico con respecto al tiempo, y el "host stack" o "pila de host" que trata con datos de alto nivel. La pila del controlador generalmente se implementa en una placa, la cual tamben contiene la radio Bluetooth y un microprocesador. La pila de host se implementa generalmente como parte del sistema operativo, o como un paquete instalable.

Como mínimo, toda pila de protocolos de Bluetooth debe tener los siguientes protocolos: LMP(controller stack), L2CAP(host stack) y SDP(host stack). Además, los dispositivos que se comunican por Bluetooth pueden usar casi siempre los protocolos HCI(controller stack) y RFCOMM(host stack).

En este proyecto para comunicar el cliente con el servidor se ha utilizado el protocolo RFCOMM asi como el protocolo SDP para establecer la conexión con el servicio deseado.

Toda la información referente a este apartado, se puede encontrar de manera más extensa en las referencias 2, 20 y 14 de la bibliografía.

4.3.1. LMP

El protocolo de control de enlace (Link Management Protocol, LMP) se usa para el establecimiento y control del enlace de radio entre dos dispositivos. Está implementado en el controlador.

4.3.2. L2Cap

L2CAP es un protocolo que define una serie de puertos que se pueden utilizar para mandar y recibir datos mediante las capas inferiores de bluetooth, es decir, define una forma en la que varias aplicaciones pueden enviar datos y recibir datos utilizando las ondas de radio de bluetooth.

Las funciones de L2CAP incluyen:

- Multiplexación de datos entre diferentes protocolos de capa superior.
- Segmentación y reensamblaje de paquetes.
- Proporcionar administración de transmisión unidireccional de datos de multidifusión a un grupo de otros dispositivos Bluetooth.
- Gestión de calidad de servicio (QoS) para protocolos de capa superior.

En su modo básico, L2CAP proporciona a los paquetes una carga útil que se puede configurar hasta 64 kB, y con una MTU por defecto de 672 bytes.

En los modos de Retransmisión y control de flujo, L2CAP puede configurarse para datos asíncronos o para un canal de datos fiables mediante la retransmisión y la comprobación de CRC. La fiabilidad en cualquiera de estos modos es opcional y / o adicionalmente garantizada por la capa inferior Bluetooth BDR / EDR.

El apéndice 1 de la especificación de Bluetooth añade dos modos adicionales a L2CAP. Estos nuevos modos dejan obsoletos los anteriores modos de retransmisión y control de flujo:

- **Modo de retransmisión mejorado (Enhanced Retransmission Mode, ERTM):**Este modo es una versión mejorada del modo original de retransmisión. Proporciona un canal L2CAP confiable.
- **Modo streaming (Streaming Mode, SM):** Es un modo muy simple, sin retransmisión ni control de flujo. Proporciona un canal L2CAP no confiable.

4.3.3. SDP

Service discovery protocol (SDP) se utiliza para permitir que los dispositivos descubran qué servicios son compatibles entre sí y qué parámetros usar para conectarse a ellos.

Cada servicio se identifica mediante un identificador único universal (UUID), con servicios oficiales (perfiles Bluetooth) asignados a un UUID de forma corta (16 bits en lugar de los 128 completos).

En la pila de protocolos, SDP está vinculado a L2CAP.

4.3.4. RFCOMM

El protocolo Bluetooth RFCOMM es un conjunto simple de protocolos de transporte, hecho sobre el protocolo L2CAP, que proporciona puertos serie RS-232 emulados (hasta sesenta conexiones simultáneas a un dispositivo Bluetooth a la vez). El protocolo se basa en la norma ETSI TS 07.10.
A veces se le conoce como emulación de puerto serie. El perfil del puerto serie de Bluetooth se basa en este protocolo.

RFCOMM proporciona un flujo de datos simple y confiable para el usuario, similar a TCP. Muchas aplicaciones Bluetooth utilizan RFCOMM debido a su amplio soporte y API disponible públicamente en la mayoría de los sistemas operativos. Además, las aplicaciones que usan un puerto en serie para comunicarse se pueden portar rápidamente para usar RFCOMM.

En la pila de protocolos, RFCOMM está vinculado a L2CAP.

4.4. Pybluez

PyBluez[8] es un modulo de Python que permite que el código acceda a los recursos Bluetooth de la máquina host. El uso de este modulo facilita la programación para comunicarse a través de Bluetooth usando Python. Los sockets de Python pueden ser usados para la comunicación por Bluetooth (a partir de Python 3.3). Para una aplicación simple, el código es casi idéntico frente a usar PyBluez, sin embargo, para algunas tareas, como el descubrimiento de dispositivos y los anuncios de servicios Bluetooth, no parece posible realizarlos utilizando sockets de Python.

Se ha hecho uso principalmente de las siguientes funciones, tal y como se explica en la API de Pybluez.

En la parte del servidor, la función:

```
advertise_service(sock, name, service_id='', service_classes=[],
   profiles=[], provider='', description='')
```

Anuncia un servicio con el servidor SDP local. "**Sock**" debe ser un socket de escucha enlazado, "nombre" debe ser el nombre del servicio y "service_id" (si se especifica) debe ser una cadena de la siguiente forma "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXXXX", donde cada 'X' es un dígito hexadecimal.

"**service_classes**" es una lista de clases de servicio a las que pertenece este servicio. Cada servicio de clase es un UUID de 16 bits en la forma "XXXX", donde cada 'X' es un dígito hexadecimal, o un UUID de 128 bits en la forma "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX". Hay algunas constantes para los servicios estándar, como por ejemplo, SERIAL_PORT_CLASS que es igual a "1101".
Algunas constantes de clase son:

SERIAL_PORT_CLASS	LAN_ACCESS_CLASS	DIALUP_NET_CLASS
HEADSET_CLASS	CORDLESS_TELEPHONY_CLASS	AUDIO_SOURCE_CLASS
AUDIO_SINK_CLASS	PANU_CLASS	NAP_CLASS
GN_CLASS		

"**profiles**" es una lista de perfiles de servicio que cumple este servicio. Cada perfil es una tupla con (uuid, versión). La mayoría de los perfiles estándar utilizan clases estándar como UUID. PyBluez ofrece una lista de perfiles estándar, por ejemplo SERIAL_PORT_PROFILE. Todos los perfiles estándar tienen el mismo nombre que las clases, excepto que el sufijo _CLASS se reemplaza por _PROFILE.

"**provider**" es una cadena de texto que especifica el proveedor del servicio.

"**description**" es una cadena de texto que describe el servicio.

De esta manera, para crear un servicio y anunciarlo mediante SDP para que un cliente pueda conectarse buscando este servicio en concreto y aceptar la conexion del socket de dicho cliente se ha creado con un código como el siguiente:

```
server_sock=BluetoothSocket( RFCOMM )
server_sock.bind(("",PORT_ANY))
server_sock.listen(1)

port = server_sock.getsockname()[1]

```

```
uuid = #uuid que deseemos para este servicio concreto

advertise_service( server_sock, """nombre del servicio""",
                   service_id = uuid,
                   service_classes = [ uuid, SERIAL_PORT_CLASS ],
                   profiles = [ SERIAL_PORT_PROFILE ]
                    )


client_sock, client_info = server_sock.accept()
```

Tras la conexión, se ha creado un bucle que recibe las distintas ordenes del cliente y las procesa.
Para cerrar la conexion con el socket cliente se usará:

```
client_sock.close()
```

Y para cerrar el socket del servidor y terminar la ejecución se usará:

```
server_sock.close()
```

En cuanto al cliente, la función:

```
find_service(name=None, uuid=None, address=None)
```

Busca servicios SDP que coincidan con los criterios especificados y devuelve los resultados de la búsqueda. Si no se especifican criterios, devuelve una lista de todos los servicios cercanos detectados.
Si se especifica más de uno, entonces los resultados de búsqueda coincidirán con los criterios especificados. Si se especifica uuid, debe ser un UUID de 16 bits en la forma "XXX", donde cada 'X' es un dígito hexadecimal, o como un UUID de 128 bits en la forma "XXXXXXXX-XXXX-XXXX-XXXX- XXXXXXXXXXXX".
Un caso especial de dirección es "localhost", que buscará servicios en la máquina local.

Los resultados de la búsqueda serán una lista de diccionarios. Cada diccionario representa una coincidencia de búsqueda y tendrá los siguientes pares clave / valor:

- **Host:** La dirección Bluetooth del dispositivo que anuncia el servicio.
- **Nombre:** El nombre del servicio que se anuncia.
- **Descripción**: Una descripción del servicio que se anuncia.

- **Proveedor**: El nombre de la persona u organización que presta el servicio.
- **Protocolo**: Ya sea 'RFCOMM', 'L2CAP'. Ninguno si el protocolo no estaba especificado, o 'DESCONOCIDO' si el protocolo fue especificado pero poco reconocido.
- **Puerto**: El L2CAP PSM # si el protocolo es 'L2CAP', el RFCOMM canal # si el protocolo es 'RFCOMM', o Ninguno si no fue especificado.
- **Clases de servicio**: Una lista de ID de clase de servicio (cadenas UUID), posiblemente vacío.
- **Perfiles**: Una lista de perfiles - (UUID, versión) , posiblemente vacío.
- **Service-id**: El ID de servicio. Ninguno si no fue establecido. Ver la especificación de Bluetooth para la diferencia entre Service ID yService Class ID List.

De esta manera, para crear un cliente que busque un servicio determinado con un uuid concreto se ha creado con un codigo como el siguiente:

```
addr = None

if len(sys.argv) < 2:
    print("Buscando el servicio en los dispositivos cercanos")
else:
    addr = sys.argv[1]
    print("Buscando el servicio en %s" % addr)

# Busqueda del servicio
uuid = #uuid del servicio que se quiere buscar
service_matches = find_service(uuid=uuid, address=addr)

if len(service_matches) == 0:
    print("No se pudo encontrar el servicio =(")
    sys.exit(0)

first_match = service_matches[0]
port = first_match["port"]
name = first_match["name"]
host = first_match["host"]

print("Conectando a \"%s\" en %s" % (name, host))

# Creacion del socket cliente
sock = BluetoothSocket(RFCOMM)
sock.connect((host, port))
```

Con dicho código, si se especifica una dirección mac Bluetooth en la que esta el servidor, se buscara el uuid en dicha dirección, si no, la dirección (addr en el código) permanecerá a "None" y se buscará en todos los dispositivos cercanos, devolviendo las coincidencias. Hay que tener en cuenta que si se encuentran varios servidores con el mismo uuid, solo se tomara como válido el primero que se encuentre.

Tras la conexión, se ha creado un bucle (al igual que en el servidor) con el menu y las distintas opciones que tiene el programa y cuyas órenes se mandarán al servidor para procesarlas. En el caso de la versión con GUI, ese bucle no esta presente ya que se ha usado un framework para la creación de dicho menu con interfaz grafica.

En este caso para terminar solo sera necesario cerrar el socket propio del cliente:

```
1 sock.close()
```

La API completa de Pybluez se encuentra en el directorio "docs" del repositorio oficial, así como los requisitos o dependencias de cada plataforma para su instalación https://github.com/pybluez/pybluez.

Para la instalación se utilizará pip (tambien se encuentran los binarios para Windows en PyPI) con la siguiente instrucción:

```
pip install pybluez
```

En caso de que pip no funcione sera necesario descargar el repositorio de Github (o realizar un git clone sobre dicho repositorio) y ejecutar la siguiente orden dentro de la ruta de dicha descarga:

```
python setup.py install
```

En los testeos realizados, sobre *GNU/Linux* ha sido necesaria la descarga del repositorio y el método anteriormente descrito para su instalación.

En el caso de *Windows*, a pesar de cumplir con las dependencias descritas en las instrucciones de instalación en su *Github*, ninguno de los métodos parecía funcionar para instalar Pybluez, sin embargo se encontró una manera modificando el archivo *setup.py* ya que no éste reconocía de manera correcta la ruta del Windows SDK. Para ello se cambio la siguiente linea de codigo:

```
1 candidate_paths.append(r'Microsoft SDKs\Windows\v10.0A')  # Visual
     Studio 14
```

Por la siguiente:

```
candidate_paths.append(r'C:/Program Files (x86)/Windows Kits/10')  #
    Visual Studio 14
```

Tras lo cual, al ejecutar `python setup.py install` Pybluez se instaló de manera correcta.

4.5. Kivy

Para la creacion de la GUI, se ha usado el framework Kivy. Tal y como se especifica en su pagina web[12] y Github[7], Kivy es un framework Python de código abierto y multiplataforma para el desarrollo de aplicaciones que hacen uso de interfaces de usuario innovadoras y multitáctiles. El objetivo es permitir un diseño de interacción rápido y fácil y una creación rápida de prototipos al tiempo que hace que el código sea reutilizable e implementable.

Kivy está escrito en Python y Cython, basado en OpenGL ES 2, es compatible con varios dispositivos de entrada y tiene una extensa biblioteca de widgets. Con el mismo código base, puede funcionar en Windows, macOS, Linux, Android e iOS. Todos los widgets de Kivy están construidos con soporte multitáctil.

Kivy tiene licencia MIT, está desarrollada activamente por una gran comunidad y cuenta con el respaldo de muchos proyectos administrados por la Organización Kivy.

Figura 4.2: Logo de Kivy. Fuente:
https://kivy-designer.readthedocs.io/en/latest/

Una de las peculiaridades de este framework es que permite separar la parte del diseño visual de la interfaz, de la interacción y la lógica de la aplicación mediante el lenguaje KV. De esta manera se puede programar los widgets y su comportamiento en python, y especificar, por ejemplo, la disposición de los botones, transiciones y otras componentes visuales de la interfaz de usuario en Kv.

En el proyecto el código de la GUI esta dividido en 3 archivos:

- **rfcomm_client.py**: Contiene las funciones necesarias para la conexión, desconexión, y recepción del archivo con la información de los sensores mediante bluetooth.
- **main.py**: Contiene las clases (o *widgets*), que componen cada una de las vistas del programa y su lógica, ventanas con errores etc.
- **app.kv**: Contiene la parte gráfica de las clases de *main.py*, es decir, en este archivo se encuentra el diseño visual de cada ventana.

Por ejemplo, una de las clases implementadas es la clase menú que muestra el menu principal. En el archivo *main.py* esta definida de la siguiente manera:

```
class Menu(Screen):

    def csv(self):
        sock.send("Genera")

    def recibir(self):
        sock.send("Envia")
        ret = sock.recv(1024)
        recibido = ret.decode('utf-8')
        if recibido == "Error":
            noFile()
        else:
            sock.send("Sync")
            recibir(sock)
            fileRec()
```

En ella están definidas únicamente las funciones **csv** y **recibir**. La función **csv** únicamente se encarga de enviar una orden a el servidor para que empiece a recopilar información del sensor, y la función **recibir** envía dicha opción a el servidor, y si no ocurre ningún error, llama a la función recibir de *rfcomm_client.py* para proceder a la recepción del archivo de datos. En caso de haber algún error, se lanzara una ventana de tipo *popup* que mostrará dicho error.

En el archivo *app.kv* la definición gráfica de dicha función es la siguiente:

```
<Menu>:
    GridLayout:
        cols:1
        Label:
            text: "Menu Principal"
```

```
            font_size: (root.width**2 + root.height**2) / 14**4

        BoxLayout:
            orientation: 'vertical'
            Button:

                text: "Recibir datos en tiempo real"
                on_release:
                    app.root.current = "TR"
                    root.manager.transition.direction = "left"

            Button:

                text: "Generar archivo de datos en el dispositivo"
                on_release:
                    app.root.current = "csv"
                    root.manager.transition.direction = "left"
                    root.csv()

            Button:

                text: "Recibir archivo de datos"
                on_release:
                    root.recibir()

            Button:

                text: "Calibrar sensores"
                on_release:
                    app.root.current = "cal"
                    root.manager.transition.direction = "left"

            Button:

                text: "Salir"
                on_release:
                    app.stop()
```

En dicho código se definen el texto que indica en que ventana se encuentra ("Menu principal en este caso"), los distintos botones que generan acciones o transiciones a las distintas ventanas así como la disposición de estos en el panel.

También es posible definir ventanas *popup* (como la anteriormente mencionada) que se mostraran al encontrarse un error, o para mostrar información relevante como por ejemplo en el siguiente caso:

```
def fileRec():
    pop = Popup(title='Recibido',
                content=Label(text='Archivo recibido'),
                size_hint=(None, None), size=(400, 400))
    pop.open()
```

Esta ventana forma parte del proyecto y se muestra cuando se ha recibido el archivo de datos correctamente. Estas ventanas no forman parte del archivo *app.kv* sino que están definidas en el archivo *main.py* a continuación la definición de las diferentes clases.

De esta manera se han creado todas las ventanas que componen la GUI. Seria posible definir la parte visual directamente en Python en *main.py* prescindiendo de *app.kv*, sin embargo de esta manera resulta mas claro y fácil de desarrollar y mantener.

Debido a que *main.py* es el archivo Python que se ejecutará, es necesario especificar el archivo .kv que se utilizara, además de incluir las diferentes clases en el *ScreenManager*. Esto se realiza al final del archivo tras la definición de todas las clases y ventanas necesarias:

```
#Archivo kv asociado a las vistas
kv = Builder.load_file("app.kv")

sm = ScreenManager()
#Clases definidas
sm.add_widget(MainWindow(name='Main'))
sm.add_widget(Automatico(name='Auto'))
sm.add_widget(Manual(name='Manual'))
sm.add_widget(Menu(name='Menu'))
sm.add_widget(TiempoReal(name='TR'))
sm.add_widget(Csv(name='csv'))
sm.add_widget(Calibrate(name='cal'))


class Gui(App):
    def build(self):
        return sm


if __name__ == '__main__':
    #Creacion del socket
    sock = BluetoothSocket(RFCOMM)
    Gui().run()
    #Desconexion al salir del programa
    disconnect(sock)
```

Debido a que Kivy funciona con su propio bucle, cualquier bucle interno que definamos estará dentro del bucle principal de Kivy y por lo tanto provocará que la interfaz gráfica se bloquee a la espera de la finalización de dicho bucle. Por ello, para cualquier función que necesite de un bucle (como por ejemplo, en la recepción de datos en tiempo real) sera necesario el uso de threads para evitar dicho bloqueo.

En este proyecto se ha testeado el funcionamiento de la GUI construida con este framework en *GNU/Linux* y *Windows 10*.

Para la instalación en *GNU/Linux*, es necesario añadir primero los repositorios. Se observó que si se añadía el repositorio de la rama estable, la versión instalada de Kivy daba problemas con el código escrito en Python 3, por ello para solucionar este problema, se optó por añadir el repositorio de la rama *nightly*, el cual instalaba una versión mas alta de Kivy, la cual no presentaba ningún problema.

Rama estable:

```
sudo add-apt-repository ppa:kivy-team/kivy
```

Rama nightly usada en el proyecto:

```
sudo add-apt-repository ppa:kivy-team/kivy-daily
```

Una vez añadidos los repositorios para instalar Kivy habria que ejecutar uno de los siguientes comandos en funcion de la version de Python deseada (en el caso de este proyecto la version 3):

Python 2:

```
sudo apt-get install python-kivy
```

Python 3 usado en el proyecto:

```
sudo apt-get install python3-kivy
```

En el caso de este proyecto no fue necesario nada mas, pero en su pagina oficial "https://kivy.org/doc/stable/installation/installation-linux.html" se dan instrucciones sobre posibles errores o dependencias necesarias en caso de no cumplirlas.

En cuanto a la versión de *Windows 10*, no hubo ningún problema con la instalación y siguiendo las instrucciones oficiales en "https://kivy.org/doc/stable/installation/installation-windows.html" se instaló correctamen-

te.

Para ello se instalaron las dependencias necesarias de la siguiente manera:

```
python -m pip install docutils pygments pypiwin32 kivy.deps.sdl2 kivy.
   deps.glew
python -m pip install kivy.deps.gstreamer
```

Una vez cubiertas las dependencias, se instaló Kivy:

```
python -m pip install kivy
```

4.6. Sensores

Para la lectura de los sensores se han utilizado las librerías RTIMU[6] para el sensor MPU-9250 y la librería Adafruit_Python_BNO055[5] para el sensor BNO055.
En ambos casos, las funciones necesarias para la lectura de los sensores se han implementado dentro del archivo rfcomm-server.py.

Para realizar la lectura con RTIMU se ha hecho de la siguiente manera:

- Inicialización del sensor:

```
1      s = RTIMU.Settings("RTIMULib")
2      imu = RTIMU.RTIMU(s)
3      if (not imu.IMUInit()):
4        sys.exit(1)
5      imu.setSlerpPower(0.02)
6      imu.setGyroEnable(True)
7      imu.setAccelEnable(True)
8      imu.setCompassEnable(True)
9      poll_interval = imu.IMUGetPollInterval()
```

- Lectura del sensor:

```
1  if imu.IMURead():
2              data = imu.getIMUData()
```

```
            time.sleep(poll_interval / 1000.0)
            print(data['timestamp'])
            print(data['accel'])
            print(data['compass'])
            print(data['gyro'])
            print(data['fusionQPose'])
```

En cuanto al sensor BNO055 el esquema de montaje[10] ha sido el siguiente:

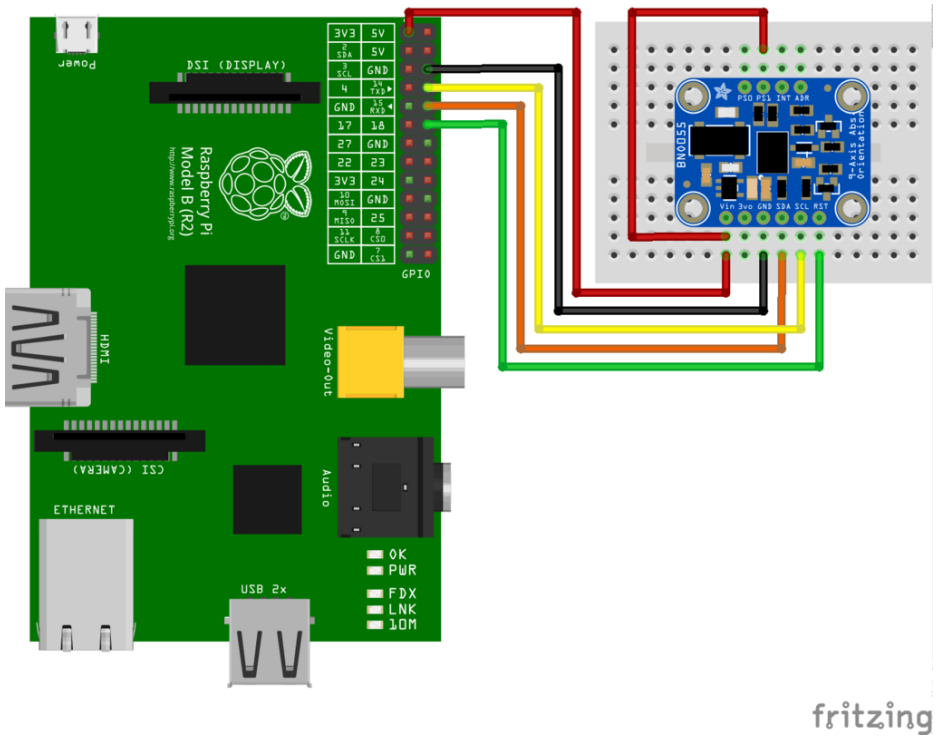


Figura 4.3: conexion del BNO055 con la placa. Fuemte: https://learn.adafruit.com/bno055-absolute-orientation-sensor-with-raspberry-pi-and-beaglebone-black/hardware

- Conectar BNO055 Vin a Raspberry Pi 3.3V power.
- Conectar BNO055 GND a Raspbery Pi ground.
- Conectar BNO055 SDA (now UART TX) a Raspberry Pi RXD pin.
- Conectar BNO055 SCL (now UART RX) a Raspberry Pi TXD pin.
- Conectar BNO055 PS1 a BNO055 Vin / Raspberry Pi 3.3V power .

Tras conectar la placa se han de ejecutar los siguientes comandos para instalar las dependencias necesarias[18]:

```
sudo apt-get update
sudo apt-get install -y build-essential python-dev python-smbus python-
   pip git
```

Una vez instaladas las dependencias habrá que bajarse el directorio de Github o bien hacer un "git clone https://github.com/adafruit/Adafruit_Python_BNO055.git".
Una vez hecho esto, para instalar la librería habrá que ejecutar la siguiente instrucción sobre el directorio del proyecto de github obtenido:

```
sudo python setup.py install
```

Tras esto el sensor ya esta listo para empezar a funcionar y obtener datos. Para ello, tras importar la biblioteca a el proyecto habra que especificar la manera en la que se ha conectado:

```
1 bno = BNO055.BNO055(serial_port='/dev/serial0', rst=17)
2 if not bno.begin():
3         raise RuntimeError('Failed to initialize BNO055! Is the sensor
             connected?')
```

Debido a que la conexión I2c resulta problemática en Raspberry Pi a causa de un bug presente en la placa con los dispositivos que usan "I2C clock stretching"[11], se ha conectado el sensor en modo Uart y el pin de reset en el 17 (por defecto en los ejemplos de la librería y en el esquema de la figura anterior, el reset viene especificado en el 18, de no cambiarlo el programa no detectará el sensor y saldrá dando un error).

El Broadcomm BCM2835, que se utiliza en la Raspberry Pi, tiene un error grave en su implementación de I2C[1], que puede evitar la comunicación I2C con algunos dispositivos y conducir a la corrupción de datos (tanto en la dirección de lectura como de escritura). Por lo que se recomienda no usar dispositivos I2C que usen clock stretching directamente con la Raspberry Pi o con cualquier otro dispositivo basado en Broadcomm BCM2835.

La especificación I2C permite a los esclavos alargar el ciclo de reloj (mantener SCL bajo durante una comunicación para ralentizar la comunicación y hacer que el maestro espere hasta que finalicen). Después de que un esclavo I2C libere el reloj, el maestro debe continuar el reloj SCL.

El error en la Raspberry Pi (o en realidad de su procesador ARM Broadcomm BCM2835) es que su reloj I2C solo se "enmascara" cuando el esclavo pone el SCL en bajo, y la Raspberry Pi no garantiza que el siguiente ciclo de reloj tenga la longitud completa. Esto puede llevar

a “picos” de SCL no válidos, que son demasiado cortos para que el esclavo los reconozca, por lo que el maestro y el esclavo pueden "no estar sincronizados". Además, la Raspberry Pi parece leer el valor de SDA, mientras que el esclavo sigue bajando el SCL, por lo que incluso los tramos muy cortos hacen que la Raspberry Pi lea datos incorrectos.

Toda esta información acerca del bug I2C de la Raspberry Pi, se puede encontrar de manera mucho mas detallada en la referencia numero 1 de la bibliografía.

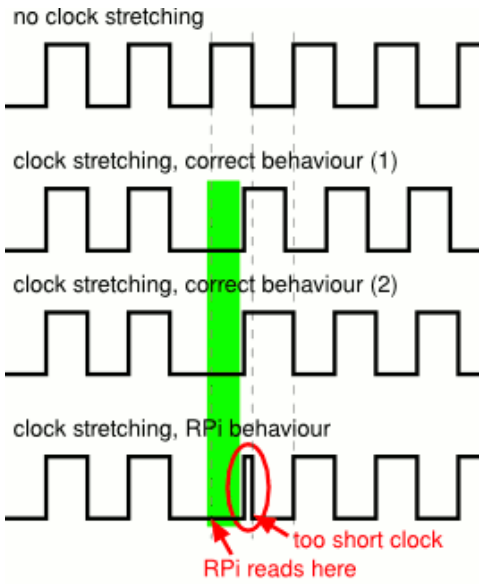


Figura 4.4: clock stretching. Fuente:
https://www.advamation.com/knowhow/raspberrypi/rpi-i2c-bug.html

Una vez conectada en modo Uart y especificada la conexión y su pin de reset, la lectura resulta sencilla:

```
        #Euler heading, roll, pith (grados)
        print(bno.read_euler())
        # quaternion:
        print(bno.read_quaternion())
        # Magnetometro (micro-Teslas):
        print(bno.read_magnetometer())
        # Gyroscope (grados por segundo):
        print(bno.read_gyroscope())
        # Acelerometro (metros por segundo al cuadrado):
        print(bno.read_accelerometer())
        # Aceleracion lineal (movimiento, no gravedad)
        # (metros por segundo al cuadrado):
        print(bno.read_linear_acceleration())
        # Aceleracion de la gravedad
```

```
15          # (metros por segundo al cuadrado):
16          print(bno.read_gravity())
```

De este modo se puede hacer un print para mostrar los datos por terminal como en el ejemplo anterior, o guardarlos en un archivo externo (en el caso de este proyecto se ha guardado en un archivo CSV) y encerrar dicho código en un bucle para obtener lecturas periódicamente y guardarlas en consonancia.

4.7. Guante

Para la realización del diseño del soporte que usará el atleta se tuvieron varios factores en cuenta, como la ergonomía, espacio para los componentes, ubicación óptima de los mismos, seguridad, sujeción sin influenciar el desempeño del atleta, etc.

Como resultado se optó por un guante estilo muñequera completa que cubre la mano hasta los dedos. De esta manera se podría satisfacer de la mejor forma los requisitos anteriormente enunciados.

Figura 4.5: Guante

La placa, junto con la batería, se ubican en la parte exterior del antebrazo, en un bolsillo especialmente diseñado que permite el movimiento de la muñeca y también su extracción e inserción sin demasiado problema. La placa se mantiene segura y sujeta durante los lanzamientos, y permite la salida de los cables que van al sensor cuando el bolsillo esta cerrado.

Figura 4.6: Conexiones de la placa

La ubicación del sensor se decidió que fuera lo mas cercana a la mano posible, por lo que se colocó en la parte superior, prácticamente en el nudillo. Para ello se diseñó un espacio que, al igual que en la placa, permitiera su extracción e inserción sin demasiado problema, manteniendo el sensor fijo durante los lanzamientos y permitiendo la salida de los cables que van a la placa.

Figura 4.7: Conexiones del sensor

Capítulo 5

Resultados

El resultado de las pruebas realizadas se guarda en un archivo .csv, éste archivo contiene las lecturas tomadas por el sensor en el lanzamiento monitorizado.

A	B	C	D	E	F	G	H
timestamp	euler heading	euler roll	euler pitch	Quaternion x	Quaternion y	Quaternion z	Quaternion w

Figura 5.1: Parte de la tabla de las lecturas del sensor

Tal y como se puede observar en la figura 4.1 (aunque debido al tamaño, no se ha mostrado completa en dicha figura), la tabla esta formada por las siguientes columnas:

- **Timestamp**: Contiene el instante de tiempo en el que se realiza la lectura.
- **Euler heading, roll, pitch**: Estas 3 columnas contienen los ángulos de Euler, que sirven para especificar la orientación de un sistema de referencia normalmente móvil, respecto a otro sistema de referencia normalmente fijo.
- **Quaternion x, y, z, w**: Estas 4 columnas están formadas por las 4 componentes del cuaternión, el cual sirve para representar las orientaciones y las rotaciones de objetos en tres dimensiones.
- **Magnetómetro x, y, z**: Contiene las componentes x, y, z del magnetómetro en cada columna respectivamente.

- **Giroscopio x, y, z**: Información de los 3 ejes del giroscopio, divididos en 3 columnas.
- **Acelerómetro x, y, z**: Datos de cada eje del acelerómetro en cada una de las 3 columnas.
- **Aceleración lineal x, y, z**: Aceleración lineal en cada uno de los ejes (3 columnas, una para cada eje).
- **Aceleración de la gravedad x, y, z**: Información de los 3 ejes de aceleración gravitacional, cada eje en 1 columna.

Una vez obtenidos estos datos, es necesario realizar una serie de cálculos matemáticos, obteniendo así los puntos en el espacio 3D y la velocidad para poder representarlos. Por desgracia, el alumno de la Facultad de Matemáticas cuyo proyecto consistía en realizar esos cálculos, no lo ha terminado a tiempo, por lo que dicha parte queda pendiente.

Sin embargo, si que se ha creado la función para representar los datos. Para ello se ha usado la librería de Python llamada Matplotlib. Mediante dicha librería es posible representar de manera gráfica los datos y guardarlos si así se desea.

La instalacion de la libreria se realiza mediante pip:

```
python -m pip install -U pip
python -m pip install -U matplotlib
```

Para la representación, una vez hechos los import necesarios para el uso de la biblioteca, y desactivado el modo interactivo de la misma de la siguiente manera:

```
1 from mpl_toolkits import mplot3d
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 plt.ioff()
```

Se han creado dos figuras independientes usando `plt.figure()` en cada una. Para la figura tridimensional es necesario especificarlo mediante `ax = plt.axes(projection='3d')`, para luego representar los puntos x, y, z con la función `ax.plot3D(x, y, z)`.

Para la gráfica de la velocidad, no es necesario especificar que es bidimensional, y usando la función `plt.plot(x, y)` quedaria representada.

Por último, una vez representados los datos correctamente, `plt.show()` crea las 2 ventanas con las gráficas correspondientes a los datos.

En las figuras 4.2 y 4.3, se puede observar el resultado obtenido en las pruebas, utilizando para ellas, datos simulados.

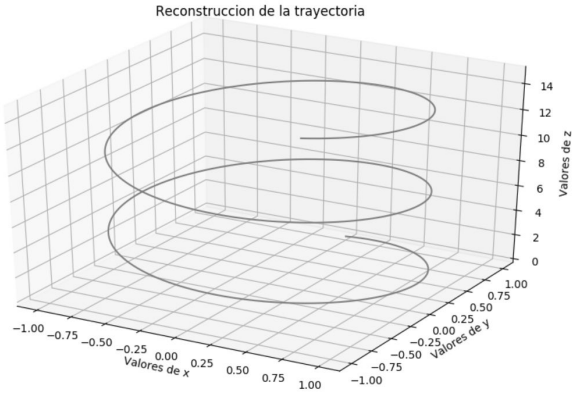


Figura 5.2: Representación de la trayectoria de un lanzamiento con datos simulados

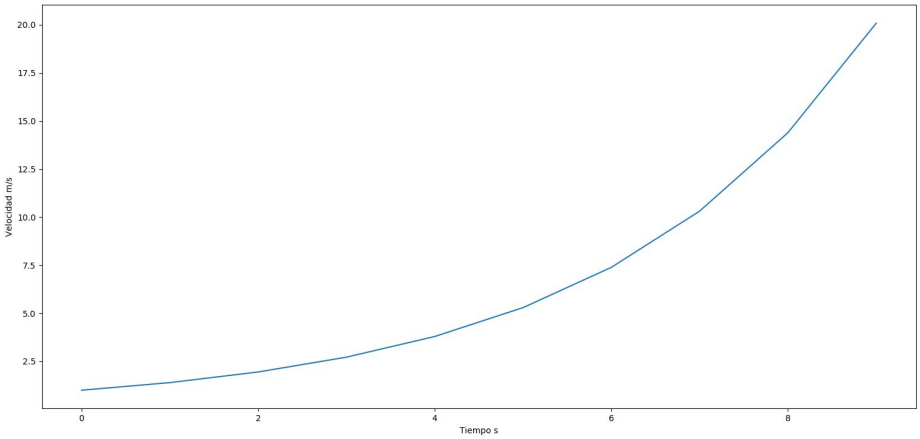


Figura 5.3: Representación de la velocidad de un lanzamiento con datos simulados

El software desarrollado para la realización de este TFG[9] se puede encontrar en el siguiente enlace:

https://github.com/CRP8/TFG

Capítulo 6

Presupuesto

En este apartado se desglosarán los costes de desarrollo, así como los costes de los materiales, investigación, montaje, diseño, etc.

6.1. Componentes

En este apartado se presenta el gasto de los componentes, cantidad y materiales que componen el hardware del proyecto. No se han tenido en cuenta las herramientas utilizadas para su desarrollo, como por ejemplo, un teclado para la Raspberry Pi, adaptador HDMI y cable, etc.

La lista de precios (ver tabla 6.1) se ha elaborado con distintos proveedores para cada pieza, tomando en cuenta los precios, las fechas de envío, etc. Se ha incluido en los precios el importe por el envío, en el caso de que dicho producto no tuviera envío gratuito.

En el caso de la Raspberry Pi, se ha tenido en cuenta el precio del proveedor que figura en la pagina oficial, junto con los gastos de envío que indica dicho proveedor.

En el presupuesto, también se ha tenido en cuenta el sensor MPU-9250, que aunque más tarde fue sustituido por el BNO055, fué el que se utilizó en un primer momento, y tuvo importancia en el desarrollo.

Componentes			
Producto	***Precio***	***Unidades***	***Total***
Raspberry Pi Zero W	18.52 €	1	18.52 €
Tarjeta MicroSD Sandisk	9.68 €	1	9.68 €
Sensor BNO055	27.99 €	1	27.99 €
Sensor MPU-9250	4.90 €	1	4.90 €
TP4056 charger	1.16 €	1 pack 5 unidades	1.16 €
Fuente de alimentación Lipo-shim	12.30 €	1	12.30 €
Muñequera	5 €	2	10 €
Cables Dupont, hembra a hembra	3.99 €	1	3.99 €
Baterías	12.33 €	1 pack 4 unidades	12.33 €
Total (iva incluido)			***100.87 €***

Tabla 6.1: Tabla de precios de los materiales

6.2. Desarrollo e investigación

Para realizar el cálculo del precio por hora de trabajo, se ha supuesto que el trabajador contratado trabaja para la empresa CRP-SL. En la tabla 6.2 se especifica tanto el sueldo bruto, como el resto de cotizaciones que paga la empresa, con el fin de obtener el coste real para realizar dicho cálculo.

Calculo del trabajador a la empresa		
Salario bruto	*35000 €*	*35000 €*
Contingencias comunes	+ 23.60 %	8260 €
Desempleo	+ 5.50 %	1925 €
Formación profesional	+ 0.60 %	210 €
Fondo de garantía salarial	+ 0.20 %	70 €
Accidentes	+ 1 %	350 €
Total		**45815 €**

Tabla 6.2: Coste del sueldo del trabajador a la empresa

Una vez obtenida dicha cifra y teniendo en cuenta que el convenio especifica que el número de horas de trabajo anuales es de 1750, se obtiene un precio de 26.18 euros por hora de trabajo.

Teniendo en cuenta esta cifra, en la tabla 6.3 se ha realizado el calculo del precio en función de las horas empleadas para su desarrollo.

Tarea	***Estimación de horas***	***Precio/hora***
Desarrollo del software	190	26.18 €
Desarrollo del hardware	30	26.18 €
Investigación	250	26.18 €
Pruebas	90	26.18 €
Documentación	40	26.18 €
Total	***600***	***15.708 €***

Tabla 6.3: Tareas realizadas y coste de ellas

Para finalizar, en la tabla 6.4 se puede observar el coste total del proyecto, incluyendo tanto los materiales como el coste de desarrollo.

Coste de desarrollo	***Coste de los materiales***	***Precio total***
15.708 €	100.87 €	15.808,87 €

Tabla 6.4: Coste total

Capítulo 7

Conclusiones

Con el fin de analizar el desempeño de un atleta de lanzamiento de martillo y poder mejorarlo, surgió la necesidad de obtener datos de la actividad que este realiza para su análisis.

Actualmente, se recolectan datos para su posterior análisis de múltiples fuentes con objetivos como el análisis de mercado, con fines publicitarios, de seguimiento de salud, etc. Con esto en mente surgió la idea de este TFG, crear un dispositivo tipo *wearable* que recogiera los datos de un atleta de lanzamiento de martillo, y los enviara a otro dispositivo mediante una conexión bluetooth para, de esta manera, poder analizarlo.

Realizar este proyecto individualmente desde cero, sin apenas experiencia en Python y Raspberry Pi ha sido un reto. Han sido necesarias muchas horas de investigación sobre el hardware y el software de la Raspberry Pi, los sensores, Python, conectividad, y múltiples frameworks disponibles tanto para el uso de la conexión bluetooth como para crear una interfaz gráfica multiplataforma.

La gran comunidad y el número de desarrolladores que realizan proyectos con este tipo de hardware, permite que exista gran cantidad de información acerca de estas de arquitecturas de bajo coste.

Debido al reducido coste de los materiales, este tipo de hardware resulta especialmente interesante de cara a proyectos con una utilidad específica como el de este TFG, ya que no requieren de un alto presupuesto.

En cuanto al "guante" que lleva acoplados los componentes hardware, cabe destacar la importancia del diseño. Es importante que este dispositivo sea lo mas cómodo posible para el atleta, con el fin de evitar que éste interfiera con el desempeño de la actividad y no "contamine" la muestra de datos recogida por los sensores.

Como conclusión final, los datos recogidos por el dispositivo pueden ser muy útiles para un atleta (de lanzamiento de martillo en este caso), ya que el tipo de sensores utilizado en este TFG son capaces de obtener una gran cantidad de información de cada lectura. Se puede reconstruir la trayectoria y la velocidad de cada lanzamiento y, de esta manera, corregir errores o introducir mejoras en la técnica del atleta. Además, es posible estudiar el impacto que tiene en el atleta diferentes condiciones atmosféricas, estrés, etc. La precisión de las medidas y la cantidad de datos que pueden recoger este tipo de dispositivos es cada vez mayor y las posibilidades son enormes, por lo que el futuro es prometedor en cuanto a este tipo de entrenamientos "conectados".

Capítulo 8

Conclusions

In order to analyze the performance of a hammer throw athlete and be able to improve it, became necessary to obtain data of the activity to make an analysis.

Nowadays, data are collected for analysis from multiple sources with objectives such as market analysis, advertising, health monitoring, etc.. With this in mind it came to us the idea the of this TFG, build a device type *wearable* to collect data from an hammer throw athlete , and send them to another device via a bluetooth connection to analyze it.

Doing this project individually from scratch, with hardly any experience in Python and Raspberry Pi has been a challenge. It has taken many hours of research on the hardware and software of the Raspberry Pi, sensors, Python, connectivity, and multiple frameworks available for the use of bluetooth connection and to create a multi-platform graphical user interface.

The large community and the number of developers who carry out projects with this type of hardware, makes possible to find a lot of information about these low-cost architectures.

Due to the low cost of materials, this kind of hardware is very interesting for projects with a specific utility such as this TFG, because do not require a high budget.

The desing of the " glove " wich has the hardware components attached, is very important. It is important to be as comfortable as possible for the athlete, in order to prevent it from interfering with the performance of the activity and not " contaminate " the sample of data collected by the sensors.

As a final conclusion, the data collected by the device can be very useful for an athlete (hammer throw athlete in this case), since the kind of sensors used in this TFG are able to get a lot of information from each reading. It is possible to reconstruct the trajectory and speed of each throwing and, in this way, correct errors or introduce improvements in the technique of the athlete. In addition, it is possible to study the impact that different atmospheric conditions, stress, etc. have on the athlete. The accuracy of the measurements and the amount of data that this devices can collect is increasing and the possibilities are enormous, so the future is promising in terms of this type of "connected training ".

Bibliografía

[1] Advamation - know-how - raspberry pi i2c bug. URL: https://www.advamation.com/knowhow/raspberrypi/rpi-i2c-bug.html. Accessed: 2019-04-30. (Archived by WebCite® at http://www.webcitation.org/7821W5dNm).

[2] Bluetooth - wikipedia, la enciclopedia libre. URL: https://es.wikipedia.org/wiki/Bluetooth. Accessed: 2019-04-30. (Archived by WebCite® at http://www.webcitation.org/781fHj56w).

[3] Datasheet mpu-9250a. URL: http://www.invensense.com/wp-content/uploads/2015/02/PS-MPU-9250A-01-v1.1.pdf. Accessed: 2019-04-29. (Archived by WebCite® at http://www.webcitation.org/780IrzEN1).

[4] Datasheet tp4056. URL: https://dlnmh9ip6v2uc.cloudfront.net/datasheets/Prototyping/TP4056.pdf. Accessed: 2019-04-29. (Archived by WebCite® at http://www.webcitation.org/780I9rBdI).

[5] Github - adafruit/adafruit_python_bno055: Library for accessing the bosch bno055 absolute orientation sensor on a raspberry pi or beaglebone black. URL: https://github.com/adafruit/Adafruit_Python_BNO055. Accessed: 2019-04-30. (Archived by WebCite® at http://www.webcitation.org/781iT80bx).

[6] Github - blauret/rtimu. URL: https://github.com/blauret/RTIMU. Accessed: 2019-04-30. (Archived by WebCite® at http://www.webcitation.org/781i6feKK).

[7] Github - kivy/kivy: Open source ui framework written in python, running on windows, linux, macos, android and ios. URL: https://github.com/kivy/kivy. Accessed: 2019-04-30. (Archived by WebCite® at http://www.webcitation.org/781hvQMkX).

[8] Github - pybluez/pybluez: Bluetooth python extension module. URL: https://github.com/pybluez/pybluez. Accessed: 2019-04-30. (Archived by WebCite® at http://www.webcitation.org/781gc0bnz).

[9] Github del tfg. URL: https://github.com/CRP8/TFG.

[10] Hardware | bno055 absolute orientation sensor with raspberry pi, beaglebone black | adafruit learning system. URL: https://learn.adafruit.com/bno055-absolute-orientation-sensor-with-raspberry-pi-and-beaglebone-black/hardware. Accessed: 2019-04-30. (Archived by WebCite® at http://www.webcitation.org/781ivdIvQ).

[11] I2c clock stretching - raspberry pi forums. URL: https://www.raspberrypi.org/forums/viewtopic.php?p=146272. Accessed: 2019-04-30. (Archived by WebCite® at http://www.webcitation.org/7821mz7B5).

[12] Kivy: Cross-platform python framework for nui development. URL: https://kivy.org//#home. Accessed: 2019-04-30. (Archived by WebCite® at http://www.webcitation.org/781h6hQSe).

[13] Lipo-shim | the pi hut. URL: https://thepihut.com/products/lipo-shim. Accessed: 2019-04-29. (Archived by WebCite® at http://www.webcitation.org/780IYSJVf).

[14] List of bluetooth protocols - wikipedia. URL: https://en.wikipedia.org/wiki/List_of_Bluetooth_protocols. Accessed: 2019-04-30. (Archived by WebCite® at http://www.webcitation.org/781fdH54i).

[15] Python. URL: https://www.python.org/. Accessed: 2019-04-29. (Archived by WebCite® at http://www.webcitation.org/780J2qkdd).

[16] Python faq. URL: https://docs.python.org/3/faq/general.html. Accessed: 2019-04-29. (Archived by WebCite® at http://www.webcitation.org/780JCQL1c).

[17] Que es un gpio. URL: https://www.programoergosum.com/cursos-online/raspberry-pi/238-control-de-gpio-con-python-en-raspberry-pi/que-es-gpio. Accessed: 2019-04-29. (Archived by WebCite® at http://www.webcitation.org/780HsD9jI).

[18] Software | bno055 absolute orientation sensor with raspberry pi, beaglebone black | adafruit learning system. URL: https://learn.adafruit.com/bno055-absolute-orientation-sensor-with-raspberry-pi-and-beaglebone-black/software. Accessed: 2019-04-30. (Archived by WebCite® at http://www.webcitation.org/781iibSFW).

[19] What is a raspberry pi? URL: https://opensource.com/resources/raspberry-pi. Accessed: 2019-04-29. (Archived by WebCite® at http://www.webcitation.org/780H73ZrK).

[20] Wifi - wikipedia, la enciclopedia libre. URL: https://es.wikipedia.org/wiki/Wifi#Estándares_que_certifica_la_Alianza_Wi-Fi. Accessed: 2019-04-30. (Archived by WebCite® at http://www.webcitation.org/781fubRRd).