Trabajo Fin de Grado

Ingeniería Electrónica, Robótica y Mecatrónica

Mención en Robótica y Automatización

Desarrollo de sistema de medida de distancia para competiciones de ciclismo en ruta

Autor: Daniel Seco Morales

Tutor: Manuel Perales Esteve

Dpto. Ingeniería Electrónica

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2020

Trabajo de fin de grado
Ingeniería Electrónica, Robótica y Mecatrónica

Desarrollo de sistema de medida de distancia para competiciones de ciclismo en ruta

Autor:
Daniel Seco Morales

Tutor:
Manuel Perales Esteve
Profesor titular

Dpto. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2020

Trabajo de Fin de Grado: Desarrollo de sistema de medida de distancia para competiciones de ciclismo en ruta

Autor: Daniel Seco Morales

Tutor: Manuel Perales Esteve

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2020

El Secretario del Tribunal

"El que ha llegado tan lejos que ya no se confunde,
ha dejado también de trabajar"

Max Planck

Agradecimientos

Agradezco.

A mi familia.

A mis amigos.

A mis profesores.

Resumen

En este trabajo de fin de grado se busca solucionar la problemática surgida con la distancia de drafting que rodea a un deporte en pleno auge en como es el triatlón, en concreto en sus versiones de larga distancia, conocidas por el nombre de la franquicia *Ironman*.

Se desarrollará un concepto de dispositivo detector del incumplimiento de la distancia de drafting, que sirva tanto para informar al deportista como a la organización de la infracción que se esté llevando a cabo, con el objetivo de evitar el incumplimiento por desconocimiento de esta normativa y la capacidad de sancionar a los deportistas que reiteren en este incumplimiento.

La solución será una solución completa, que ofrezca un elemento detector de la distancia con capacidad para ser portado por le deportista en competición; y herramientas para el staff de la prueba para comprobar la cumplimentación de la normativa.

Abstract

In this project it is intended to solve some problems are happening around drafing distance in a heyday sport as it is triathlon, and more specifaclly on its long-distance competitions, well knonw for its leader franchise name *Ironman*.

A concept of drafting distance detector dispositive will be developed, which can be used to inform both, athletes, and organization, about this distance normative breach. The desired goal of the project is avoiding the breach of the normative for unknowledge and punishing those athletes whose fall on it.

There will be a complete solution, which offers a detector element that can be wore by the athlete on his bicycle, and a useful tool which allows the organization staff check the normative compliance.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 PREFACIO

1.1 Introducción

1.1.1 El triatlón

Según la RAE, se define triatlón como: "*Conjunto de tres pruebas olímpicas que consisten en 1500m a nado, 40km de ciclismo en carretera y 10 km de carrera a pie*". Y bien es cierto, que en su origen, esta era la definición correcta, pero hoy en día el triatlón es un deporte en auge, y que no para de crecer y reinventarse, por lo que una definición más ajustada a la actualidad podría ser "*Competición deportiva formada por tres diciplinas, natación, ciclismo y carrera a pie, encadenadas una tras otra por este orden*", Hoy en día el triatlón se practica en multitud de distancias, y versiones, habiendo versiones incluso con el sector de ciclismo que se realiza en bicicleta de montaña.

Para ser propiamente llamado triatlón se respeta el orden, y el hecho de que las tres disciplinas se encadenan una tras otra, en las llamadas transiciones, sin descanso, y siendo este tiempo parte del tiempo de competición. Por ello muchas personas consideran a las transiciones la cuarta disciplina del triatlón, ya que, sobre todo en las distancias más cortas, la transición es un punto clave de la competición.

Entrando un poco más en detalle en las distancias, en el panorama competitivo se pueden ver multitud de variedades, pero si nos referimos al triatlón con ciclismo de carretera, nos podemos referir al reglamento oficial de la ITU (International Triathlon Organization), que es la institución reguladora a nivel mundial, donde en punto 23, apéndice A, se muestra la tabla a continuación con las distancias reguladas:

23. APPENDIX A: COMPETITION DISTANCES AND AGE REQUIREMENTS:

Triathlon:

	Swim	Bike	Run	Minimum age required
Team relay	250m to 300m	5km to 8km	1.5km to 2km	15
Super Sprint Distance	250m to 500m	6.5km to 13km	1.7km to 3.5km	15
Sprint Distance	Up to 750m	Up to 20km	Up to 5km	16
Standard Distance	1500m	40km	10km	18
Middle Distance	1900m to 3000m	80km to 90km	20km to 21km	18
Long Distance	1000m to 4000m	100km to 200km	10km to 42.2km	18

Tabla 1-1, Distancias en triatlón

De entre todas estas distancias, se pueden diferenciar dos tipos de modalidad de competición, con drafting, y sin drafting. Se considera drafting permitido cuando en el sector de ciclismo se permite rodar en grupos, aprovechando así el esfuerzo del grupo para reducir el esfuerzo individual que hay que realizar contra el aire para avanzar en la bicicleta. Normalmente, son la media y larga distancia las que se desarrollan siempre sin drafting, mientras que las otras distancias, ~~y~~ a excepción de campeonatos específicos sin drafting o

competiciones tipo contrarreloj por equipos, se desarrollan con la normativa de drafting permitido.

1.1.2 El drafting

Una vez tenemos una visión general de que es una prueba de triatlón, se puede pasar a lo que realmente nos ataña, el drafting en el sector de ciclismo. Una pregunta muy común de toda persona que no ha practicado nunca ciclismo deportivo en carretera es: *¿Realmente es tan importante ir a rueda?* Seguro que cualquier ciclista mínimamente experimentado en la carretera respondería: *sin duda alguna, sí.*

Comúnmente en una competición de ciclismo se pueden ver que los ciclistas permanecen casi todo el tiempo en un gran grupo muy pegados unos a otros, y que, aunque muchas veces esto produzca situaciones de peligro y caídas, lo continúan haciendo sin parar. Esto es sin duda porque les merece mucho la pena, como se puede ver en la imagen en la que se muestra el porcentaje del esfuerzo total que realiza un ciclista para batir el efecto aerodinámico en un grupo a 52 km/h en función de su posición en el grupo. Teniendo en cuenta que todos los deportistas van a una misma velocidad, se puede ver que el primer ciclista destina el 86% de su esfuerzo a batir la resistencia aerodinámica, mientras que los ciclistas mejor posicionados destinan sólo un 5% del total, por lo que están realizando un esfuerzo similar a si fuesen a 12km/h de manera individual.

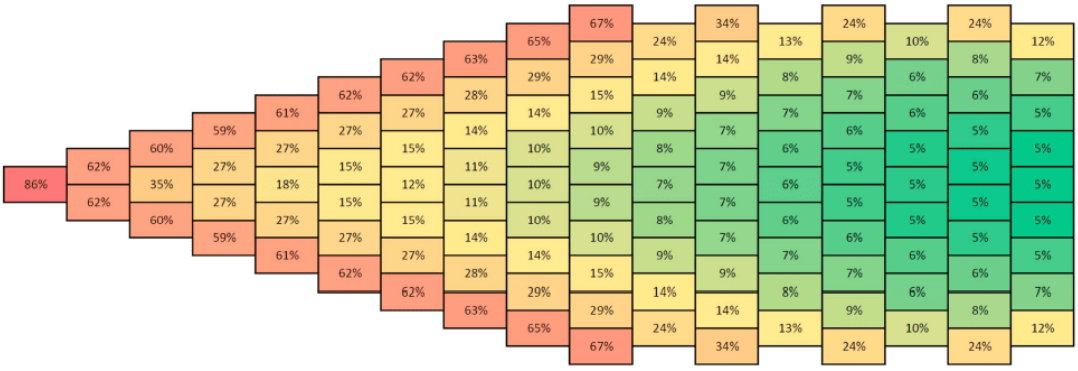


Ilustración 1-1, Esfuerzo aerodinámico en grupo ciclista a 52km/h

El ejemplo anterior es teniendo en cuenta un gran grupo, situación que pocas veces llega a ocurrir en un triatlón sin drafting; pero en cambio si que es más común que se vea a un ciclista a rueda de otro, situación igualmente prohibida por la normativa. Tal y como podemos ver en la siguiente imagen, donde se muestra los efectos acerca de la reducción del coeficiente aerodinámico de un ciclista que va a rueda de otro, no en grupo, en función de la distancia al mismo, hablamos de una reducción de hasta el 40% del coeficiente de rozamiento, o de unos 90 watios, teniendo en cuanta que la élite del deporte rueda a velocidades medias de unos 40-45km/h en el sector de cliclismo, y entre 250-300 watios ponderados.

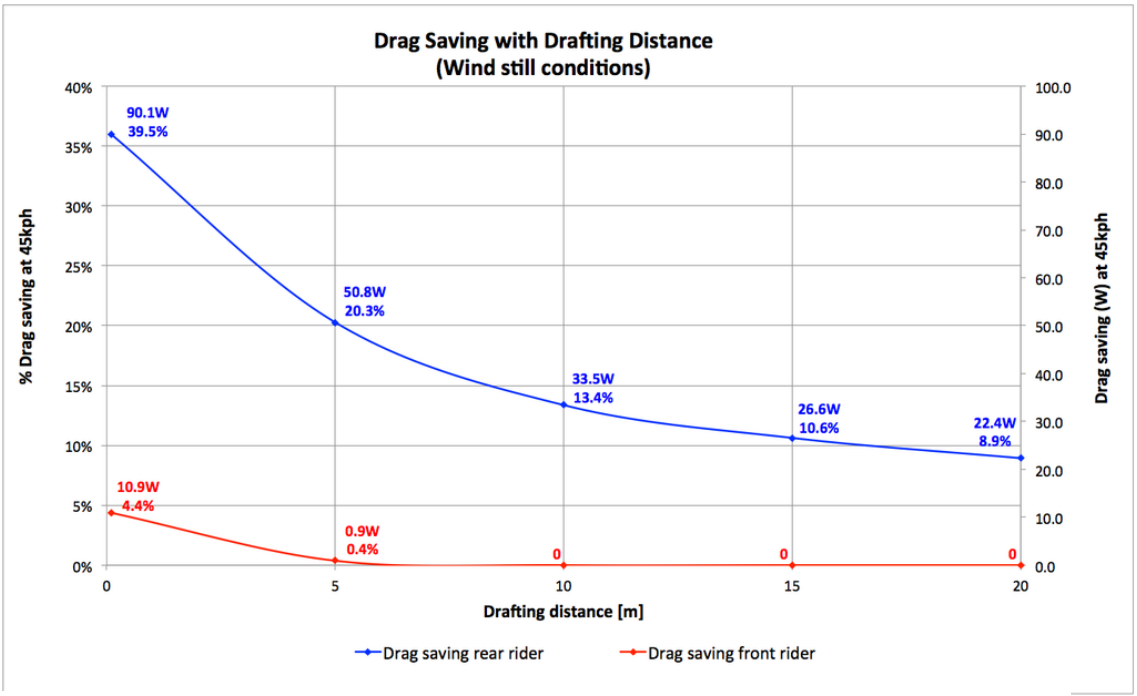


Ilustración 1-2, Estudios de reducción de coeficiente aerodinámico

1.1.3 Normativa

En la actualidad la normativa internacional de competiciones de la ITU (*International Triathlon Union*), que es la federación internacional de triatlón encargada de toda regulación oficial en cuanto al triatlón, en el punto 5.5.c de la misma donde regula el caso de competiciones sin drafting (https://www.triathlon.org/uploads/docs/itusport_competition-rules_2019.pdf) podemos ver que se define un área de drafting ~~entre deportistas de~~ entre 12 y 15 metros tras el deportista que va en va en cabeza y el perseguidor, dependiendo del tipo de carrera que se esté disputando, y se permite estar un máximo de entre 25 a 20 segundos dentro de esa área, siempre a efectos de realizar un adelantamiento.

Pero también se puntualiza, que bajo ciertas condiciones el drafting será permitido. Esto suele ser en áreas especialmente estrechas, en obras, zonas de avituallamiento o cerca del área de transición. Fuera de reglamento, en zonas con una pendiente positiva muy pronunciada, se suele rebajar el criterio de drafting ya que las bajas velocidades hacen que el efecto drafting sea mucho menor.

1.1.4 Situación actual

Hoy en día los métodos de control son bastante deficientes. Por una parte, la organización no dispone de los medios para controlar todo el trazado ciclista debido a la longitud de estas pruebas (muchas de ellas de hasta 180km), y por otra, la gran cantidad de participantes hace que haya muchos kms del trazado ocupado por deportistas al mismo tiempo, lo cual hace que sean necesarios todavía más recursos. Hay que tener en cuanta que en las competiciones más multitudinarias se dan varias salidas espaciadas en el tiempo por las diferentes categorías con el objetivo de no crear multitudes excesivas de deportistas.

Esta situación es, en la actualidad, un tema que da mucho que hablar en el mundo del triatlón sin drafting, ya que en algunos casos ha habido reproches entre deportistas por aprovecharse del esfuerzo. Además, en varias ocasiones, medios especializados que cubren las competiciones han podido grabar situaciones de drafting multitudinarios.

Tengamos en cuenta que esta regla del drafting, además de tratar de hacer competir a los deportistas en igualdad de condiciones teniendo que hacer todos el mismo esfuerzo en el sector ciclista, es una medida de seguridad, ya

que la conocida como bicicleta de triatlón o contrarreloj que se usa en estas pruebas dispone de unos acoples aerodinámicos sobre el manillar, lo que hace que la posición estándar de esta bicicleta deje al deportista con las manos lejos de los frenos y con una agilidad muy limitada, con lo que ir cercano a la rueda de otro deportista puede producir múltiple caídas ante cualquier imprevisto

1.2 Estado del arte

1.2.1 Métodos

En la actualidad la organización de una competición de triatlón sin drafting dispone de pocos métodos para el control de esta distancia. Por regla general son dos las formas en las que lleva a cabo este control:

- Jueces dispuestos a lo largo del recorrido en puntos estratégicos.
- Motos de organización con un conductor y un juez montados en ellas, que van moviéndose entre los participantes a lo largo del recorrido

El problema de los sistemas de control actuales radica en que generalmente las competiciones sin drafting son las conocidas como de media o larga distancia, es decir, que su sector de ciclismo es de 90 o 180km de distancia; y que debido al auge de popularidad que ha sufrido el triatlón en la última década, es bastante difícil ver competiciones en las que haya menos de un centenar de participantes, llegando en muchas de ellas a superar el millar holgadamente.

Estos métodos de detección resultan escasos y, pese a que permiten detectar a algunos infractores, gran parte de ellos consiguen pasar desapercibidos.

1.2.2 Dispositivos

Tal y como he podido comprobar con el departamento de competiciones de la Federación Española de Triatlón (FETRI), no existe ningún dispositivo específico para la detección de la distancia de drafting.

Tan sólo existe una prueba al respecto, realizada en el campeonato de España de triatlón larga distancia 2019, en Salamanca. En este evento se utilizó el dispositivo *Track the Race*, conocido en el mundo de las carreras de orientación ya que se usa como tracker de seguridad. Este dispositivo dispone de un GPS y transmite la información con una tarjeta SIM como la de nuestros móviles a través de red GSM/GPRS. Al estar basado únicamente en señal GPS consigue una precisión de entorno a 2m, y actualiza la señal cada 15-20”, esto tal y como ha indicado la federación, es insuficiente para tomarlo como una medida de control del drafting por si sola, sirve de apoyo para poder enviar a jueces en moto a controlar a ciertos deportistas que aparenten estar incumpliendo las reglas según la señal GPS.

1.3 Propuesta

El objetivo de este proyecto será la realización de una solución completa que permita la detección del incumplimiento de la distancia de no drafting en las competiciones de triatlón en las que aplique esta normativa.

Se diseñará un dispositivo detector de distancia de seguridad, que irá colocado en la bicicleta del deportista y que permitirá reconocer la distancia de drafting de manera autónoma sin tener que comunicarse con ningún otro dispositivo o humano.

Este detector deberá tener la capacidad de comunicar la información a la organización, la cual dispondrá de un elemento, ya sea físico o una aplicación en su ordenador con el que interpretar los datos y verificar la información de cara a la posterior sanción de los deportistas que hayan incumplido la normativa.

Se ha diseñado una hoja Excel con los requerimientos *objetivos.xlsx* los cuales se dividen en 4 niveles de restricción: MUST, para los que se tienen que cumplir; SHOULD, para los que son recomendables que se cumplan, pero no estrictamente necesarios; COULD, como opcionales que añadirían funcionalidad; y WON'T, que son situaciones que no deben ocurrir. El desarrollo del proyecto girará entorno a ir completando estos requerimientos, con lo que se irá rellenando conforme se vayan satisfaciendo.

		Requerimientos		Prioridad
#ID	Elemento	Requerimiento	Descripción	Tipo
#1	Detector	Detección de distancia drafting	Reconocer, en caso de estar incumpliendo la distancia de drafting, el valor en metros de esta sanción.	MUST
#2	Ambos	Comunicación de resultados	Comunicar con la aplicación de la organización de manera inalámbrica	MUST
#3	Detector	Feedback deportista	Informar al deportista acerca de cuando está incumpliendo la distancia de drafting	MUST
#4	Ambos	Información en directo	Informar a la organización sobre si ha incumplido durante el desarrollo del sector ciclista y los casos	COULD
#5	Detector	Peso liviano	El peso del conjunto el dispositivo no superará los 300g	SHOULD
#6	Detector	Aerodinámica	Integración en bicleta tipo contrarreloj con impacto aerodinámico sea leve	SHOULD
#7	Detector	Anclaje	Se podrá poner/quitar de la bicicleta de manera sencilla	COULD
#8	Aplicación	Interpretación de datos	Los datos de los usuarios se interpretarán de manera gráfica para la organización	MUST
#9	Aplicación	Configuración de dispositivos	Cconfigurar el dispositivo de forma que no sea neceario tocar código fuente	SHOULD
#10	Detector	Detección drafting viento lateral	Reconocer drafting bajo condiciones de viento lateral con abanicos.	SHOULD
#11	Detector	Batería	La duración de la batería del dispositivo será como mínimo de 6h, 180km a 30km/h.	SHOULD
#12	Detector	Costo	El costo del dispositivo no debe superar los 50€ en caso de producción.	SHOULD
#13	Aplicación	Configuración servidor	La configuración del servidorno requerirá de conocimientos informáticos específicos.	SHOULD
#14	Aplicación	Guardado de datos	Los datos de los usuarios quedarán guardados en algún tipo de fichero o BBDD como histórico.	MUST
#15	Detector	Detección de adelantamientos	No tomar adelantamientos como infracciones.	SHOULD
#16	Detector	Zonas de drafting permitido	Reconocer las zonas en las que el drafting esté permitido en el rocorrido y no sancionar por ello.	SHOULD
#17	Detector	Zona transición	No reconocer infracciones en zona de transición	MUST
#18	Aplicación	Configuración detector simple	La organización dispondrá de alguna herramienta para configurar los dispositivos de los participantes.	COULD
#19	Detector	Aviso encendido	El dispositivo informará de si está encendido y funcionando de manera visual	SHOULD
#20	Ambos	Seguridad	Método de seguridad para evitar que un usuario externo suplante a los dispositivos	SHOULD
#21	Aplicación	Comprobación a posteriori	La aplicación podrá interpretar los datos de un deportista infractor en cualquier momento.	COULD
#22	Ambos	Codigo	El código fuente debe estar encriptado o compilado en el dispositivo final	SHOULD
#23	Detector	Cambio de sentido	El detector pordrá interpretar en que sentido se circula en las áreas seguras	SHOULD

Tabla 1-2, Hoja de objetivos

2 SELECCIÓN DE TECNOLOGÍA

2.1 Hardware

2.1.1 Microcontrolador

Como dispositivo microcontrolador se ha seleccionado el conocido ESP32. No se han barajado otras opciones en cuanto a microcontroladores ya que desde el momento en que nació el proyecto se decidió usar este dispositivo en proposición del tutor de este trabajo.

Este dispositivo nos aporta una gran capacidad de conectividad, bajo costo, módulos de bajo consumo, y trabajar con una tecnología muy actual y con aparentemente buen futuro.

Ilustración 2-1, ESP32 Huzzah

Tomando más a fondo las propiedades hardware del ESP32 podemos destacar de cara a nuestro proyecto:

- Microprocesador de 32 bits en punto flotante Xtensa LX6 a 240MHz. Co-procesador de ultra bajo consumo
- 448KB ROM, 520KB SRAM
- 34 GPIOs programables
- ADC 12 bits
- 2 puertos I2C
- 3 puertos UART
- Conectividad wifi y bluetooth

Salida PWM El ESP32 se puede programar de distintas maneras:

- De manera nativa, se puede trabajar con el mediante comandos AT
- Python, se le puede cargar el firmware que nos permite usar el ESP con un intérprete de microPython y usarlo mediante scripts y comandos de Python o creando nuestro propio firmware microPython personalizado
- C++, se puede programar con Arduino IDE, con lo que podemos apoyarnos en la multitud de librerías que se dispone para Arduino.

- Espressif IoT Development Framework, es el IDE de programación oficial, en C y C++, aunque al ser de pago no está tan estandarizado
- JavaScript, existen versiones firmware para programarlo usando Espruino, el IDE para microcontroladores con javascript
- Existen otra multitud de formas de programar el ESP32, ya que es un micro bastante popular, algunas dan soporte por ejemplo a NodeMCU, .NET, LUA…

El lenguaje de programación seleccionado ha sido microPython, hoy en día Python es el lenguaje de programación más utilizado a nivel mundial, y aunque en el mundo de los microcontroladores todavía no ha tenido un largo recorrido, facilita mucho el trabajo y aporta una tecnología novedosa y de buen futuro. La comunidad de microPython es extensa y activa, la web oficial es https://micropython.org/, y a raíz de ella nacen distintas ramas en función de los microcontroladores a usar y otras para las distintas funcionalidades y periféricos que se pueden conectar con el firmware microPython.

El intérprete microPython es una distribución opensource disponible en GitHub (https://github.com/micropython/micropython) que apareció en 2013 con la cual podemos usar el ESP de manera similar a cualquier intérprete Python para computadora. Hay que tener en cuenta que esta distribución no contiene todos los paquetes de Python3, y que hay algunos otros específicos, tal y como veremos más adelante, para las funcionalidades específicas de un microcontrolador como protocolos de comunicación o GPIOs…

2.1.2 Detector

El detector es una parte imprescindible del proyecto, es el elemento que permite detectar a los infractores. En este aspecto se han barajado varias opciones tecnológicas que se pasan a comentar:

- **Reconocimiento de imagen:** existen módulos que añaden visión y almacenamiento de esta al ESP32. El proceso de detección sería tomar fotografías aproximadamente cada segundo y tratar esas imágenes. Apoyandose en que una bicicleta tiene siempre unas cotas que son constantes en todas ellas (principalmente el diámetro de rueda) se puede procesar esta imagen y realizar un cálculo de la distancia a la misma. O en caso de incluir más de una cámara incluso podríamos realizar un procesado de la distancia en base a la triangulación. El reconocimiento de distancia mediante imagen sería factible, y el costo y tamaño de la solución encajan en las especificaciones, pero la capacidad de cómputo en dispositivo de estas dimensiones, para darle toda la fiabilidad necesaria no sería viable, por lo que se decide desestimar esta metodología.
- Otra de las opciones barajadas es la de el uso de un sistema tipo radar Doppler, esta opción quedó descartada al poco de comenzar a estudiarla debida a que el coste y tamaño de estos dispositivos no se ajusta a las necesidades de nuestro proyecto.
- **Señal GPS:** con una geolocalización GPS bien afinada podríamos llegar a conseguir una precisión aceptable del orden del metro. El gran inconveniente de esta tecnología es que no es autónoma, se tendría que poder comparar la geolocalización e instante de tiempo de todos los deportistas, lo que

Ilustración 2-2, Max Sonar MB1360XL

obliga a enviar la señal GPS de manera constante durante el transcurso de la prueba a un servidor que compare las localizaciones, lo que nos obligaría a usar comunicación GSM/GPRS, ya que es la única comunicación disponible a lo largo de todo un trazado de ciclismo (y en ciertas zonas tampoco); o a procesar la información a posteriori. Todo esto unido a la falta de precisión de un GPS ~~nos~~ hace descartar esta tecnología.

- **Sensor de distancia ultrasonido:** estos sensores nos permiten detectar objetos y son muy conocidos en el mundo de la robótica y proyectos caseros debido a su bajo costo y buen resultado. Por regla general estos sensores tienen la capacidad de detectar objetos hasta una distancia de 6m con un haz de unas pocas decenas de grados. Esta distancia queda escasa para nuestro proyecto, pero existen algunas versiones más sofisticadas de estos sensores que indican conseguier hasta 10m de distancia de detección sin ampliar en exceso sus dimensiones o consumo. Se han realizado pruebas con uno de estos sensores, concretamente con un MB1360 XL Max Sonar, que según su hoja de especificaciones es específico para la detección de humanos hasta 10m de distancia. Lamentablemente las pruebas realizadas con este sensor han sido negativas consiguiendo unos resultados que para nada coinciden con la realidad. Aunque no se incluyen librerías específicas para usar este sensor con microPython, al tener una comunicación UART ha sido sencillo decodificar la información del sensor siguiendo las especificaciones del datasheet.

Especificaciones MB1360XL (Max Botix)	
Tensión de alimentación	5 voltios
Consumo	< 1 watios
Resolución	1 cm
Peso	7 gramos
Dimensiones	22x20x25mm
Temperatura de trabajo	-40/+70 ºC
Distáncia máxima	10.68 metros
Distancia mínima	0.25 metros
Ángulo de apertura	30º
Salida	Protocolo UART Salida analógica
Refresco de lecturas	10 Hz

Tabla 2-1. Especificaciones MB1360XK

Las pruebas realizadas con este sensor han arrojado un resultado muy inferior frente a las cualidades prometidas en las especificaciones. Como referencia no se ha conseguido obtenere la detección de ningún tipo de objeto a más de 3 metros.

Ilustración 2-3, TF Mini Lidar

- Sensor óptico de distancia, este ha sido la opción final usada para el proyecto, en concreto se ha decidido

Especificaciones TF mini Lidar (Seed Studio)	
Tensión de alimentación	5 voltios
Consumo	0.6 watios
Resolución	1% hasta 6 metros 2% de 6 a 12 metros
Peso	5 gramos
Dimensiones	42x15x16mm
Temperatura de trabajo	-20/+60 ºC
Distáncia máxima	12 metros (Condiciones interiores) 7.5 metros (Condiciones exteriores)
Distancia mínima	0.3 metros
Ángulo de apertura	2.6 º
Salida	Protocolo UART
Refresco de lecturas	100 Hz

Tabla 2-2, Especificaciones TF Mini Lidar

usar el TF Mini Lidar de Seed Studio.

Se ha comprobado, con pruebas específicas en terreno, que la distancia máxima de lectura en condiciones externas para un día soleado es similar a la obtenida para las condiciones en ambiente interior, con lo que alcanza los 12 metros de distancia máxima. Aunque la distancia es ideal para el proyecto, el mayor punto negativo de este dispositivo es su ángulo de apertura excesivamente pequeño, lo que lo hace muy sensible a pequeños movimientos en inclinaciones.

Como dato, el dispositivo posee un certificado de compatibilidad electromagnética tipo *EN 55032 Class B*, el cual indica que es óptimo para su uso en un ambiente tipo residencial. Para el este proyecto, esto es óptimo, ya que el uso de dispositivo se realizará al aire libre en un entorno muy aislado en el que es de esperar una muy baja interferencia electromagnética con otros dispositivos externos.

El funcionamiento del sensor óptico se basa en un emisor láser y un receptor, que transmiten una onda de 850nm, y que en función del tiempo que haya tomado la señal en volver al receptor se puede realizar un cálculo de la distancia a la que estamos. Este cálculo lo realiza el sensor internamente, y proporciona la información directamente en unidad de metros.

Data encoding interpretation	
Byte1	0x59, frame header, all frames are the same
Byte2	0x59, frame header, all frames are the same
Byte3	Dist_L distance value is a low 8-bit. Note: The distance value is a hexadecimal value, for example, Distance 1,000= 03 E8 (HEX)
Byte4	Dist_H distance value is a high 8-bit.
Byte5	Strength_L is a low 8-bit.
Byte6	Strength_H is a high 8-bit.
Byte7	Int.time Integration time
Byte8	Reserved bytes
Byte9	Checksum parity bit is a low 8-bit, Checksum = Byte1 + Byte2 + ... + Byte8, Checksum is the sum of the first 8 bytes of actual data; here is only a low 8-bit.

Tabla 2-3, Trama datos sensor distancia

El protocolo UART es un protocolo entre dispositivos, lo cual lo limita bastante en comparación a los de tipo bus de campo. Está basado en dos pines, uno de emisión (TX) y otro de recepción (RX), teniendo en cuenta que ambos hilos van cruzados entre dispositivos. Es decir, el RX de un dispositivo va conectado al TX del otros, y viceversa.

Para usar el sensor de distancia TM Mini lidar, tan solo es necesario utilizar el pin TX del dispositivo, ya que, aunque le enviemos información por RX, esta no será procesada por el dispositivo. Con tan sólo alimentarlo a 5 voltios, comienza a genera información a una frecuencia de 100Hz por la UART. La configuración de la UART deberá de de ser de 115200 baudios, sin paridad, tramas de 8 bits con 1 bit de parada, esta es la configuración por defecto de dispositivo y no es modificables, por lo que el lector que comunique con el tendrá que implementar esta configuración.

La información que se obtiene tiene una estructura concreta, la cual hay que decodificar para poder obtener la medida de distancia en metros. En concreto se envían tramas de 9 bytes, todas las tramas comienzan con dos bytes a valor *0x59* en hexadecimal. Los bytes tercero y cuarto son, respectivamente, el valor bajo y alto de la lectura, la cual es unsigned y de dos bytes de longitud. El resto corresponden a

la información relativa a la intensidad de la señal, checksum y tiempo de integración, además de un byte de información interna.

Además, de manera gratuita existe un software para la interpretación de la señal del dispositivo en un PC. Este software es propio de la marca y depende un adapatador USB-UART para su uso. Esto, aunque para labores de testeo puede resultar útil, no parece muy interesante para un uso real del detector, ya que no facilita las comunicaciones.

2.1.3 Periféricos

Son varios los periféricos que se conectan al ESP32 para complementar la funcionalidad del dispositivo detector. Se pasan a comentar cuales han sido los elementos usados, algunas de sus propiedades, y que función realizan en el dispositivo.

Ilustración 2-4, GPS GY NEO 6MV2

Sensor GPS; se va a usar para poder conocer la ubicación del deportista y de esa manera descartar los falsos positivos en las zonas en las que no haya que detectar la distancia, lo que de ahora en adelante se conocerá como *SafeZone*. Además, nos proporcionará información en cuanto a la fecha y hora actual, con lo cual podremos diferenciar entre la primera y segunda transición en las competiciones en las que ambas sean en el mismo lugar.

Especificaciones GPS GY-NEO-6MV2 (U Blox)	
Tensión de alimentación	3 a 5 voltios
Resolución	2.5 metros
Peso	15 gramos
Dimensiones	36x26x4 mm (Sensor) 27x27x6.5 mm (Antena)
Salida	Protocolo UART – Tramas NMEA
Refresco de lecturas	5 Hz

Tabla 2-4, Especificaciones GPS

Se ha usado el dispositivo GPS GY-NEO-6MV2 , este pequeño y económico sensor GPS está formado por una placa receptora a la que se le conecta una antena cerámica. La comunicación es a través de UART a 9600 baudios con tramas tipo NMEA de caráctaeres ASCII, lo que hace que pese a no disponer de librerías específicas para microPython, (mientras si que existen para Raspberry Pi y Arduino) sea bastante sencillo obtener la información deseada.

Cabe destacar que dispone de un supercondensador que permite guardar la última ubicación durante varias semanas y así agilizar el proceso de conexión el orden del minuto. Si no hay información de última conexión, o esta no coincide con el área geográfica el proceso de conexión a satélites GPS puede tardar hasta 30 minutos.

Las pruebas realizadas con el dispositivo arrojan buenos datos en cuanto a precisión de la ubicación y rapidez de conexión una vez ha encontrado la primera ubicación y almacenado en memoria. La primera ubiación resulta bastante lenta.

Además, para el aviso a usuario, se ha implementado un Buzzer CEM-1203(42), un dispositivo de muy bajo costo que tiene dos pines y que se puede regular el tono que proporciona con un PWM. Si se desea una mejor calidad de audio se puede impolementar un filgro RC paso baja, con lo que se suaviza la señal rectángular del PWM. La potencia de sonido con un PWM de 3V de amplitud es suficiente para el proyecto.

Tabla 2-5, CEM 1203(42)

El otro elemento es un diodo led genérico de color amarillo, que también se usa para informar al usuario de la infracción. Su tensión nominal es de 2.2V, y se ha montado en serie con una resistencia de 220 Ohm para así hacerlo funcionar a los 3V que proporciona el micro.

2.2 Software

2.2.1 Comunicaciones

En cuanto a la comunicación del dispositivo detector con la organización dispone de dos opciones ya integradas en el ESP32, bluetooth y wifi. Por sencillez y capacidad de alcance a la hora de distribuir una red de comunicaciones se ha elegido usar Wifi.

Bajo la conectividad wifi, se va a implementar el uso de del protocolo MQTT. Este es un protocolo creado en 1999 por OASIS, una organización sin ánimo de lucro, por lo que el uso de este protocolo es gratuito y existen distrubuciones básicas de código abierto y algunas con más funcionalidades de propietarios privados Regulado bajo la norma ISO/IEC 20922. En los últimos años su uso está aumentando drásticamente al presentarse como una de las mejores soluciones para implementar redes IoT. La última distribución estable del broker mqtt es la 5.0 de Marzo de 2019.

El protocolo, por regla general, corre sobre TCP/IP, aunque podría correr sobre cualquier otra capa que asegure las conexiones bidireccionales como por ejemplo Zigbee. Esta ideado para proporcionar comunicación entre dispositivos que se encuentran dentro de una misma red en diferentes hosts, aunque también puede ser usado para comunicar distintos procesos dentro de un mismo host local, para esta situación existan otras soluciones más eficientes. Permite ampliar la seguridad con una encriptación con usuario y password sobre la capa de

transporte, lo que se muestra casi imprescindible para redes abiertas.

El protocolo se basa en dos tipos de elementos principales.

- El primero de ellos, único e imprescindible es el broker mqtt. Este broker correrá sobre una estación dentro de la red local en la que se encontrarán todos los sensores, puede correr sobre cualquier máquina servidora. En nuestro caso correrá sobre un PC Windows 10, que de cara a la arquitectura mqtt actuará como servidor y cliente. Esto es óptimo para el proyecto ya que la cantidad de mensajes a enviar no es muy grande con lo que una computadora personal será más que suficiente.

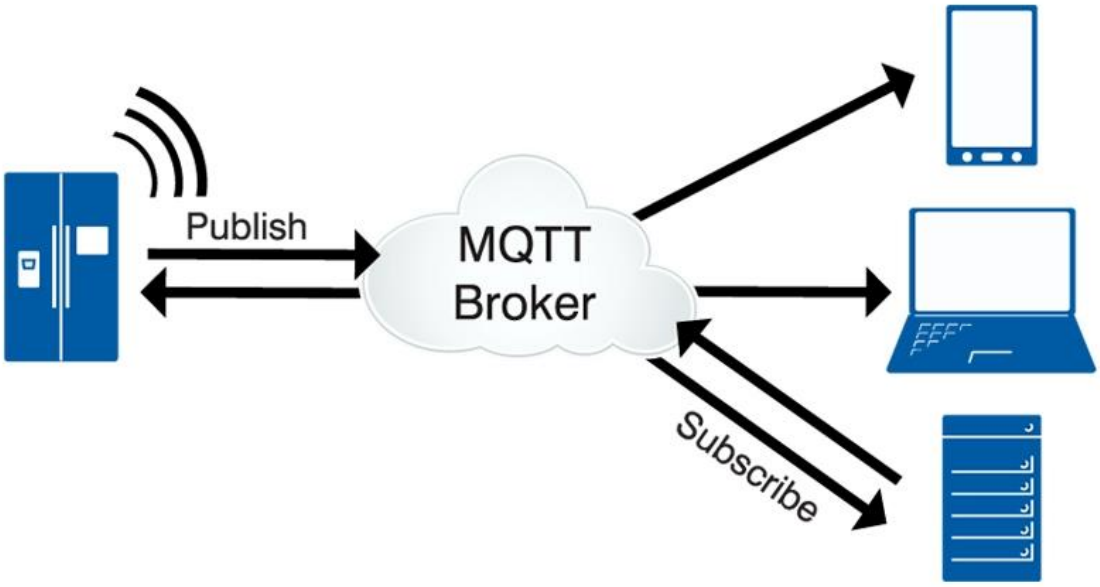


Ilustración 2-5, Arquitectura Mosquitto

- El otro elemento que conforma la arquitectura mqtt son los clientes. Esto clientes puede ser cualquier dispositivo, computadoras, sevidores, sensores… cualquier elemento de la red que genere o requiera una información concreta y que contenga algún tipo de cliente mqtt implementado. En nuestro caso los clientes serán los dispositivos detectores de los deportistas, y además el pc sobre el que corra la aplicación de la organización, que por regla general será el mismo que actue de broker.

A continuación, se pasa a comentar como es el funcionamiento de protocolo para un broker y clientes que comparten una misma red.

El broker funciona como intermediario entre todos los clientes.

Un cliente puede suscribirse y publicar sobre un topic.

Éste topic es el identificador de un tipo de información, cada publicación que se realiza tiene, por un lado, el valor o mensaje que se quiere publicar, y por otro, el topic al que va asociado este valor.

Cuando un cliente publica algo en un topic, el broker mqtt se encarga de distribuir este mensaje a todos lo clientes que están suscritos a este topic.

Por ello es importante tener en cuenta que los clientes deben estar previamente suscritos a un topic para recibir actualizaciones sobre la información que su publica en él.

Una manera clásica y sencilla de explicar el funcionamiento de mqtt es el uso de la analogía del tablón de anuncios:

Cualquier persona (cliente) puede publicar algo en una casilla (topic) de un tablón de anuncios (broker).

Cuando esto se publica, el mensaje queda expuesto a cualquiera que le interese esta casilla del tablón (clientes suscritos), con la única novedad de que mqtt se encarga de que todo el que le interese esa información la reciba justo cuando llega.

En este caso se ha usado el broker de mosquitto para Ubuntu (*Ver instalación enAnexo 1*) En un nuevo terminal de Ubuntu, se ejecuta el siguiente conjunto de instrucciones:

- *sudo apt-add-repository ppa:mosquito-dev/mosquito-ppa*
- *sudo apt-get update*

sudo apt-get install mosquito Con esto se ha instalado el broker en el computador, y para lanzarlo, se ejecuta:

- *sudo service mosquitto start*
- *sudo service mosquitto status*

Con la primera instrucción se lanza el servicio, y con la segunda se comprueba que el servicio ha sido lanzado correctamente. Para ello ~~nos~~ debería devolver el siguiente mensaje

- ** mosquitto is running*

Si este no fuese el caso, el servicio no habrá sido lanzado correctamente. (*Ver otros errores en Anexo 1*)

Como clientes, ya se ha comentado que hay dos tipos diferentes en el proyecto:

- El primero de ellos es la aplicación para la administración desde la organización de la competición. Ya que esta aplicación se ha desarrollado sobre Python3, se ha usado un cliente Mqtt para TCP/IP de Python3 sobre Windows 10, en concreto el cliente paho-mqtt.

  Paho es un conjunto de librerías desarrolladas por la fundación ecliplse que integran clientes mqtt sobre multitud de lenguajes de programación. En nuestro caso vamos a usar la librería de Python. Se puede obtener mediante la utilidad pip para la gestioón de paquetes de Python. Para ello se abre la consola sobre la que se ejecutará luego el script que lo use, en nuestro caso la consola de Anaconda, y se ejectua:

  - *Conda install -c sci-bots paho-mqtt*

  La API que ~~nos~~ ofrece paho para la programación es muy sencilla como veremos más adelante en el desarrollo técnico y el único parámetro de configuración necesario parra el correcto funcionamiento es la dirección IP del broker.

- En cuanto al segundo de los clientes, los dispositivos detectores, se ha utilizado una utilidad para microPython llamada umqtt-simple, que ~~nos~~ da una API para usar un cliente de mosquitto sobre TCP/IP.

  Esta se encuentra como parte de la librería de utlitilades de microPython externas a la distrubuición básica. Puede ser instalda en algunos dispositivos mediante la emulación de pip para microPython o, como se ha realizado en este proyecto, usarla directamente mediante el script que conforma la utilidad.

  El script es de código abierto y de libre distribución en el propio proyecto raíz de microPython. De manera similar a la librería paho, umqtt tán solo necesita de la IP del broker para su correcto funcionamiento, siempre y cuando el broker se encuentre en la misma red que el cliente. Internamente, umqtt usa los módulos socket y network de micropython, que, en su uso, son bastante similares a los de cPython.

2.2.2 Interfaz gráfica y tratamiento de datos

Para desarrollar la aplicación que de soporte a la organización se pretende continuar con el uso del lenguaje de programación Python3, ya que ~~nos~~ aporta una plataforma cómoda de trabajar, estandarizada en las diferentes arquitecturas de computadoras lo que la hace portable sin muchas complicaciones entre los distintos sistemas operativos disponibles en la actualidad.

Como IDE de desarrollo se ha utilizado Spyder, incluido dentro de Anaconda. (*Ver instalación en Anexo 1*) Anaconda es un entorno de desarrollo para Python y R, diseñado para el machine learning y ciencia de datos, es conocido por proporcionar un entorno libre y abierto, incluye multitud de utilidades. Para el desarrollo de la aplicación se va a usar Spyder, un IDE de desarrollo que pese a ser ideado para la ciencia de datos, es bastante

Ilustración 2-6, Icono Anaconda y Spyder

sencillo de usar para el desarrollo de una pequeña utilidad como la que se propone en este proyecto: y el Anaconda Prompt, que es un command prompt como el de Windows que ~~nos~~ da acceso a la herramienta pip para instalar los paquetes sobre la distribución de Python3 que usa Spyder.

En cuanto a Spyder, es muy útil ya que implementa en el propio editor un command prompt de Python, lo cual a la hora de desarrollar y testear funcionalidades concretas puede resultar bastante útil.

La herramienta para el desarrollo de la interfaz gráfica ha sido Tkinter para Python3. Esta librería es gratuita y, de hecho, viene por defecto con la instalación de Python para Windows.

Existen intefaces bastante más completas, también gratuitas para Python como wxPython o pyQt, pero se ha elegido Tkinter porque proporciona lo necesario para la solución que se va a desarrollar. Es la que ha resultado más sencilla de obtener y comenzar a usar. En concreto, para usarla en Spyder hay que instalarla desde el Anaconda Prompt:

- *conda install -c anaconda tk*

Además de esta librería, se va a utilizar alguna de las utilidades específicas de Python para machine learning, que viene ya instaladas por defecto en la distribución de Python para Ananconda. En concreto se va a utiliza la clase dataframe de *Pandas*. Una librería para manipular datos que facilita mucho el uso de matrices y en concreto tiene extensiones para obtener información de múltiples archivos, como bbdd, csv, o Excel, Además permite que esos datos, una vez tratados vuelvan a ser exportados a estas extensiones externas a Python.

La otra utilidad reseñable de Python que se utilizará será el módulo *matplotlib*. Este módulo es muy similar en su uso al plot de MatLab, con lo que resulta bastante familiar en su uso. Al igual todo lo anterior, es un módulo gratuito y, en este caso, suele venir instalado por defecto con Python. Se utilizará para mostrar los puntos de infracción a lo largo del recorrido de manera visual.

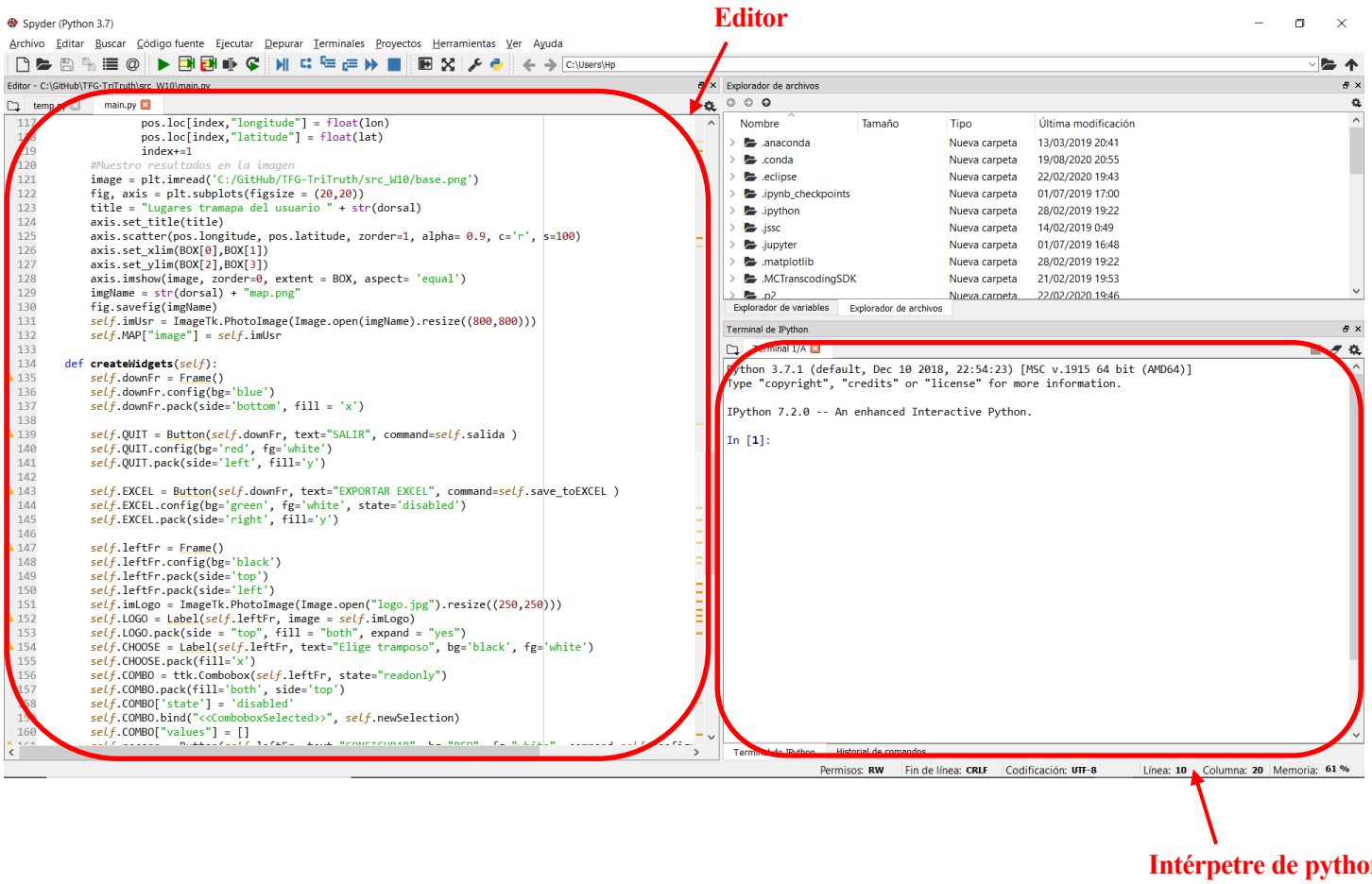


Ilustración 2-7, IDE Spyder

3 ESQUEMA GENERAL DE FUNCIONAMIENTO

3.1 Funcionalidad en competición

Para analizar el modelo de funcionamiento de la solución se va a pasar a explicar los distintos estados por los que pasan los elementos que están involucrados a lo largo de una competición.

> La descripción del dispositivo detector será individual, siendo esta extensible a todo un conjunto de participantes que usarán el dispositivo en una misma prueba al mismo tiempo.
>
> Mientras que para la aplicación de la organización se ha diseñado para que solo sea ejecutada desde un PC a la vez, y no más, de manera que la organización sólo necesitará de un juez para verificar los resultados generados por los dispositivos de los triatletas.

Poniendose en situación de una competición real sin drafting. En el momento de la entrada en transición~~, cuando~~ para las verificaciones técnicas previo a la realización de la prueba, y al igual que se suele hacer con el chip de control de paso, se entregará y colocará sobre la bicicleta del participante el dispositivo detector encendido. Este dispositivo habrá sido configurado mediante la carga de los distintos ficheros *.conf* con los datos de conexión específicos para esta prueba y participante.

Comienza la prueba, y se realiza el sector de natación, al finalizar este el participante se dirige a su bicicleta y sale del área de transición donde se monta en ella. Hasta este punto el dispositivo ha estado encendido, pero no comprobando la distancia de seguridad ya que el área de transición ha sido configurada como una safeZone.

Al salir de transición, el dispositivo detector comenzará a comprobar la distancia de seguridad, y lo hará así durante todo el transcurso del recorrido ciclista, a excepción de las áreas en las que esté permitido el drafting. En estas zonas del circuito ciclista en las que no tenemos que detectar la distancia, el dispositivo estará en un estado de espera al igual que en el área de transición. Durante el sector de ciclismo el deportista irá recibiendo alarmas sonoras y visuales acerca de el estado de infracción en el que encuentra.

Concluye el sector de ciclismo, y el deportista regresa al área de transición, en la cual el abandona su bicicleta y se va a realizar el recorrido a pie. Cuando el deportista llega al área de transición por segunda vez (en caso de que ambas transiciones estén ubicadas en la misma zona) nuestro dispositivo conectará con la red local, y mandará la información que ha recabado a lo largo de la prueba.

Es aquí donde se realizan todas las comunicaciones con la organización. Una vez la información ha llegado a la aplicación del organizador de manera satisfactoria, habiendo obtenido una confirmación de esta recepción, el dispositivo se apaga.

Ya está toda la información en la computadora del organizador, ahora el técnico juez responsable, si el deportista ha realizado alguna infracción, verá de manera visual sobre un mapa del recorrido en que zonas se ha producido ~~la infracción~~ y la continuidad de esta a lo largo del recorrido. Aquí, de manera manual podrá verificar la infracción o descartarla en función del criterio que desee aplicar.

A medida que vayan llegando más participantes irá recibiendo la información en su aplicación. En cualquier momento, el técnico que utiliza la aplicación podrá exportar los datos a un fichero Excel para realizar un recuento intermedio. De todas maneras, antes de cerrar la aplicación se realizará una exportación al fichero Excel, con lo que nos aseguraremos no perder ningún dato.

Aquí finaliza el funcionamiento de la solución, su última disposición es proporcionar la información obtenida en el ya mencionado fichero Excel. Una vez este ha sido generado, se deja en manos del equipo técnico de carrera informar de la infracción a los deportistas durante la competición o al final de la misma.

3.2 Esquema Comunicaciones

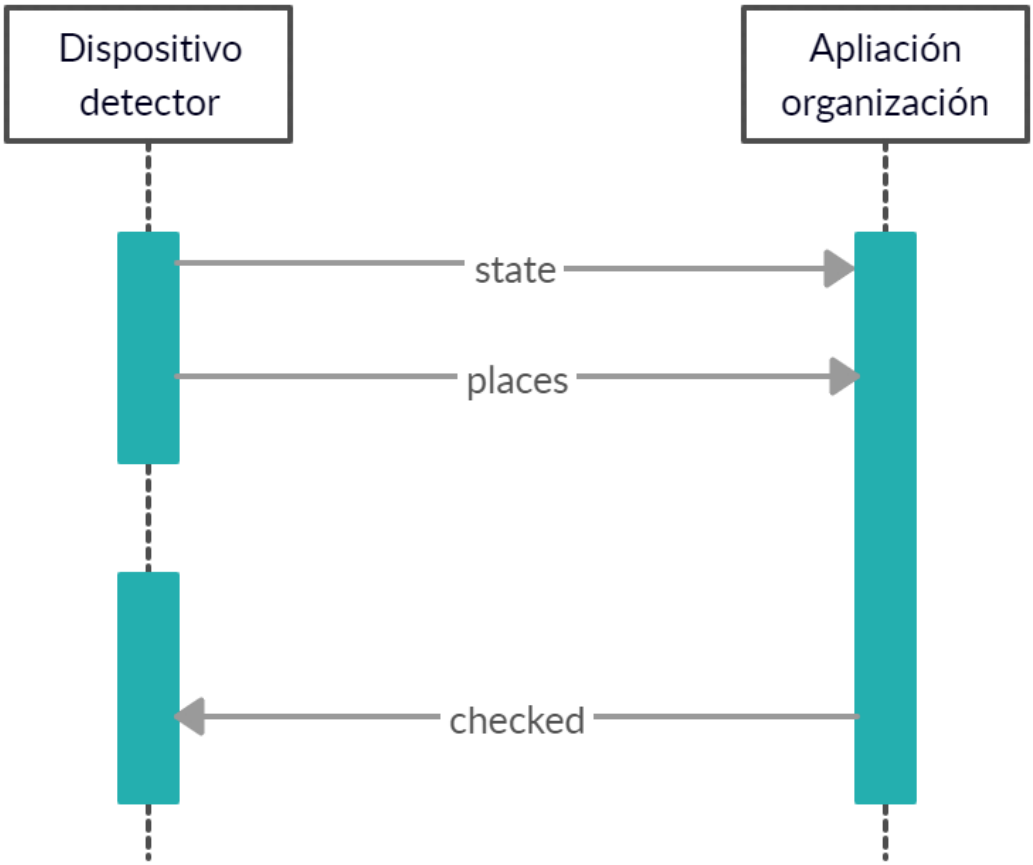


Ilustración 3-1, Esquema de mensajería

Tal y como se ha comentado, el dispositivo detector y la aplicación del organizador sólo se comunican una vez concluido el sector ciclista. Ahora se pasa a comentar el esquema de comunicación que hay entre estos dos dispositivos. Como se puede ver en el diagrama de la arquitectura de mensajería, son sólo tres los mensajes que fluyen de un lado a otro del dispositivo, aunque al ser una arquitectura bróker-cliente, todos pasarán por el bróker común.

Cabe mencionar, de manera general, que ~~a~~ todos los mensajes que se van a utilizar en la comunicación del proyecto comenzarán por el id del deportista, normalmente será su número de dorsal., que es el identificador de cada participante. A continuación irá un espacio y después el contenido del mensaje.

Esto se ha realizado de esta forma para simplificar la cantidad de topics a los que subscribirse, ya que en caso de crear un topic por deportista, muchos de ellos al no ser infractores caerían en desuso.

Una vez establecida la comunicación wifi y conectado al bróker MQTT, el dispositivo detector publicará un mensaje bajo el topic "*state*", el contenido podrá ser ***SI*** o ***NO***. En caso de recibir un positivo, pasamos a esperar en la aplicación la recepción del mensaje "*places*", en caso negativo pasamos a enviar el "*checked*".

El mensaje "*places"* contendrá la información relacionada a los lugares en los que el deportista ha infringido la distancia diferenciando cada bloque de información por un espacio. La información de cada lugar de infracción será ***#longitud#latitud,*** de manera que ambos valores serán valores numéricos que representan la posición exacta en coordenadas cardinales de los lugares en los cuales se ha detectado una infracción de la distancia de drafting.

Una vez recibido este mensaje, pasamos a enviar el mensaje de "*checked".*

El mensaje "*checked*" es el único que espera el dispositivo detector, cuando lo recibe se da por concluido el trabajo del dispositivo y se apaga. Este mensaje no tiene contenido más allá del número del identificador del deportista, que es lo que reconocerá el dispositivo detector para verificar la recepción de toda la información.

3.3 Estructura de datos

3.3.1 Hoja Excel

La información relativa a los participantes se obtendrá y almacenará una vez se haya procesado los datos recibidos del dispositivo detector. Esta hoja de Excel tendrá una serie de columnas necesarias, y además se permite que tenga información extra, no relevante para el funcionamiento del proyecto, pero sí para la competición.

Es imprescindible la columna con encabezado "*Dorsal*", en la cual irá el número de dorsal de los deportistas, que como ya se ha comentado, será además el identificador de su dispositivo detector. En la hoja de datos de configuración, la cual recibe el nombre de "*infoTriathletes.xlsx*", no hay más información relativa a los deportistas, aunque se ha trabajado con un caso más realista y se ha añadido la columna "*Nombre*", "*Categoría*", "*Teléfono*" y "*email*", para darle así una estructura más cercana a la realidad de una hoja de datos en una prueba.

Este archivo se abre por la aplicación de la organización al arrancar la misma, cargamos todos los datos que contiene, pero no se sobrescribe, la salida de la aplicación irá al archivo "*TriTruthOutput.xlsx*", el cual será el mismo archivo que el de origen al cual se le han añadido cuatro columnas más:

- "*Infractor*": En esta columna se almacenará un SI o NO, e indica si el deportista de esa fila ha incurrido en una infracción de la distancia de drafting a lo largo de la prueba
- "*Places*": Almacena la información de los lugares en los que el deportista ha infringido la distancia de drafting, no se espera que sea un dato legible a simple vista, si no una información para almacenar por seguridad. Esta columna sólo estará rellena en caso de infracción
- "*DropTime*": En esta columna se almacena el instante temporal en el que la información completa ha sido recepcionada por la aplicación y se ha enviado el mensaje "*checked*" al dispositivo detector. Sea un caso positivo o no, esta casilla siempre se rellenará.
- "*CheckTime*": Aquí tendremos la marca temporal correspondiente al instante en el que el Juez verificador ha comprobado la información correspondiente a un deportista infractor, por lo que esta columna sólo tendrá contenido en caso de positivo en infracción.

En la aplicación se dispone de un botón con el que hacer una exportación al Excel de salida en cualquier momento. Además, si se sale de la aplicación también se guardará por defecto la información, para así evitar una posible pérdida de datos.

3.3.2 Ficheros de configuración

Hay una serie de información que debe ser configurada en cada prueba, tanto en el dispositivo detector como en la aplicación de la organización. Esta configuración se realiza mediante ficheros de texto plano, con la extensión .conf, los cuales tienen una estructura determinada que se pasa a comentar a continuación.

De manera general, se ha decidido que en los ficheros puede haber comentarios que expliquen la estructura que debe seguir el archivo. Estas líneas tendrán como primer carácter el símbolo '#' siguiendo así una estructura

similar a la de un script de Python. Toda línea que con comience por el símbolo '#' será tomada como una línea de datos y todos los archivos se han creado con una pequeña explicación de la información que contendrán y el formato en el que hay que mostrarla.

Más en detalle, cada uno de los archivos tiene la siguiente estructura en particular:

- *safeZone.conf*: Este archivo contendrá una serie de coordenadas que identifican las áreas en las que el dispositivo no detectará la distancia de drafting. Cada línea contendrá 2 coordenadas, correspondientes a las dos esquinas opuestas, la noroeste en primer lugar y la sureste en segundo, de un rectángulo que identificará un área de seguridad. Podrá haber tantos rectángulos como sea necesario.
  La estructura de este archivo es **latitud1#longitud1#latitud2#longitud2**, los valores cardinales serán puramente numéricos.
- *connZone.conf*: Aquí se configura la emisión de datos a la aplicación servidora, la información contendrá un área rectangular similar a la del archivo safeZone, pero además tendrá una marca temporal, que indica el instante a partir del cual si se entra en esa área se procederá a realizar la conexión. Esta marca temporal es necesaria, porque en el caso más general, el área de transición será la misma para al salir de la natación tanto como al salir a correr, pero la conexión sólo se debe realizar en el segundo caso, por lo que se estimará un instante de tiempo en el cual los participantes estén fuera de transición, y que, por lo tanto, cuando vuelvan a entrar, hayan superado ese instante temporal y el dispositivo procederá a enviar la información.
  La estructura de este archivo será **tiempo#latitud1#longitud1#latitud2#longitud2**, siendo la variable tiempo un conjunto de números enteros, que tendrá la estructura hhmmss, en formato horario de 24h. Por ejemplo, las 09h 58' 38'' corresponde al conjunto de números 095838, importante destacar la importancia del cero al principio del conjunto en caso de que el valor de horas sea menor de 10.
- *wireless.conf*: En este archivo, tendremos la información relativa a la conexión a la red wifi y al broker MQTT. Este archivo tendrá tres líneas de contenido, y todas tendrán un contenido de tipo texto. La primera línea corresponderá al nombre del SSID, es decir, el nombre de la wifi en la que se configurará el bróker; la segunda línea contendrá el password de esa conexión wifi; y por último, tendremos la IP, en la que correrá el broker MQTT, que tal y como se verá más adelante, debido al diseño de la aplicación, será la misma IP que la del pc del Juez verificador, pero de cara al dispositivo detector esto podría no ser así y seguiría funcionando de manera apropiada.
- *user.conf:* Este archivo es el que contiene el número identificativo del deportista, por lo que es imprescindible que sea cargado. A diferencia de los otros, este es especificio de cada usuario, por lo que, dentro de la utilidad para cargar los archivos, se crea y cumplimenta con el identificador de deportista (pasado por argumentos) y se carga en el dispositivo. Este archivo es invisible al responsable de la configuración. Cómo es automático, no contempla el uso de comentarios.
- *map.conf*: Este es el único archivo de configuración correspondiente a la aplicación de la organización. Debe encontrarse en el mismo directorio que el ejecutable, y posee la misma funcionalidad en cuanto a los comentarios que los archivos anteriores. El contenido de este archivo es, por este orden, el nombre del fichero (y su extensión) que contiene el mapa base, y las cuatro coordenadas que identifican los limites de este mapa, siendo el primer conjunto el límite superior izquierdo, y el segundo el límite inferior derecho. Es importante que las coordenadas tengan una alta precisión, y deben estar en formato de grados decimales, con signo positivo o negativo en función de su orientación. El orden de las coordenadas será: longitud1, longitud2, latitud1, latitud2.

4 DESARROLLO APLICACIÓN

El desarrollo de la aplicación para el control de la organización de los resultados obtenidos por los dispositivos detectores se ha llevado a cabo en un solo script de Python 3. Este script está compuesto por una clase la cual contiene todo lo relacionado al uso de la interfaz gráfica, y una serie de funciones adicionales que permiten el tratamiento de datos y gestión del cliente mosquitto.

El script y todos los módulos dependientes han sido contenidos en un ejecutable, el cual es portable entre computadoras Windows 10 sin necesidad de instalación previa, aunuque si depende de una configuración específica como es el terminal de Ubuntu para Windows 10 y el servicio mosquitto como broker instalado. Junto con el programa ejecutable, deben ir una serie de archivos para su correcto funcionamiento, como son los ficheros de configuración, imágenes de mapa base o Excel de información de participantes.

4.1 Interfaz gráfica

4.1.1 Uso de Tkinter

Como ya se ha comentado, para la interfaz gráfica se va a utilizar la librería Tkinter para Python3. El uso de esta librería no difiere mucho de otras APIs gráficas en cuanto al diseño, aunque tiene alguna particularidad.

El diseño de la aplicación tan sólo tiene una ventana, esta será el que se llama *frame* principal, el cual, vamos a enviar como argumento al constructor de la apliación. De esta manera, siendo ese objeto un argumento externo a la clase que gestiona la interfaz, llamada *Application*, podemos gestionar el bloqueo de la ventana para que todos los elementos se creen en el orden deseado.

A continuación, otra serie de frames se crean como dependientes de este principal, actúan como contenedores que diferencian las distintas zonas de la ventana. A cada uno de estos contenedores es a lo que iremos adjuntando los diferentes widgets como botonoes, labels, boxs de introducción/selección o imágenes. La función principal que gestiona la dependencia de objetos dentro de la apliaciones es la función *pack.* Todos los objetos Tkinter tienen un método llamado pack, el cual debemos invocar tras la creación del objeto, y se le pasa como argument, si se desea, la posición en la que se posicionará el objeto dentro del frame del que depende. El frame del que depende se le pasa como un argumento en el momento de su creación. Con esta estructura quedan totalmente jerarquizados los objetos de la aplicación.

El resto de objetos de Tkinter tienen cada uno unas propiedades específicas. El único método común a todos es el método *config,* que sirve para configurar las diferentes opciones de los objetos, pero las propiedades internas de los objetos no son comunes a todos ellos. Además, si se prefiere, en lugar de usar el método config para cambiar las propiedades de un objeto se puede asignar el valor de manera directa a la propiedad de dicho objeto.

4.1.2 Aplicación

Tal como se ha comentado la clase *Application* es la que se ha creado para la gestión de todo lo relacionado con

la interfaz. Se pasa a comentar el funcionamiento de esta aplicación.

Comenzando por el constructor, que en Python es el método *__init__* de la clase, lo primero que se realiza es inicializar el frame de la ventana principal recibido por argumentos. La inicialización se realiza ejecutando es método *pack,* sin argumentos, ya que es el contenedor principal.

Lo siguiente es ejecutar la función interna createWidgets, la cual, como su propio nombre indica, crea todos los elementos que componen la aplicación. La ventana principal está compuesta por tres frames, uno ubicado en todo el ancho de la parte inferior de la ventana, llamado *downFr*; y dos superiores, uno ubicado en el margen derecho, llamado *rightFr*, y otro que ocupa el restante izquierdo, llamado *leftFr*.

Una vez conocidos los diferentes contenedores, se pasa a analizar cuáles son los objetos que se adjuntan a cada uno de ellos, y cuál es la configuración y funcionalidad de cada uno de ellos.

La configuración en cuanto a apariencia no se va a comentar de manera escrita, irá documentada gráficametne con cada uno de los frames.

Zona inferior de pantalla (*downFr*), el color de fondo será azul. De aquí dependen:

Ilustración 4-1, Botón Salir

- QUIT: es un objeto *Button*, está asociado al método *salida* de la propia clase. Cuando se pulsa este botón se ejecuta su función, que, a su vez, llama al método *save_toEXCEL*, el cual devolverá un mensaje '*Ok*' o de '*Error*' en función de si ha sido posible cerrar guardar la información en el archivo de salida. En caso de que la respuesta sea positiva, se ejecuta el método *quit*, el cual cierra el diálogo principal. Pero, si la respuesta no ha sido positiva, este método no se ejecturá, de esta manera se evita la pérdida de datos antes de un posible cierre indeseado.

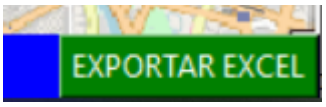


Ilustración 4-2, Botón Exportar Excel

- EXCEL: objeto *Button*, asociado al método *save_toEXCEL*. Destacar, que en el momento de su creación el botón aparece deshabilitado, más adelante se verá, que, tras una configuración imprescindible, este botón, y algunos otros se habilitarán, para poder usar la apliación con normalidad. El método *save_toEXCEL* asociado al click, lo que realiza es abrir el archivo de salida, llamado *TriTruthOutput.xlsx* como un pandas.ExcelWriter, es decir, como un objeto de pandas para escribir en Excel. Después, en el dataframe de pandas llamado *info*, en el cual se va guardando la información de los deportistas, se llama a su método to_excel, y se le pasa como objeto de salida el write de Excel anterior, y se crea la información en la hoja *Results*. Tras esto, se ejecuta el método *save* del writer, con lo que tendremos una copia actualizada de la información.

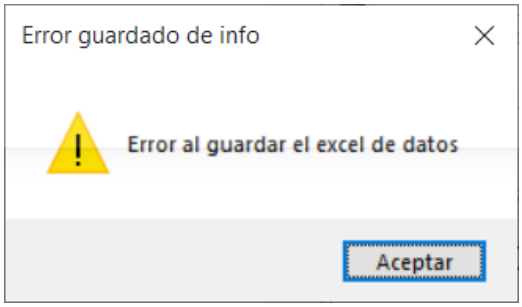


Ilustración 4-3, Mensaje error en guardado

Hasta aquí, si todo va bien se devuelve el mensaje '*Ok*', en cambio, si algo no funciona guardamos la excepción producida, que se informa con una ventana tipo pop-up de adverntencia con el mensaje : *Error al guardar el excel de datos*. Y devolvemos el mensaje '*Error*'.

Zona izquierda de pantalla (*leftFr*), el color de fondo será gris. De aquí dependen:

Ilustración 4-4, Logo y selección de deportista

- LOGO: es un objeto *Label*, que en lugar de texto tiene una imagen asociada, que se corresponde con el logo del proyecto.
- CHOOSE es un objeto *Label* con el texto "*Elige deportista*", que se sitúa justo debajo del logo.
- COMBO: objeto *Combobox* en el que, en caso de haber deportistas pendientes de revisión aparecerán en un desplegable,-se puede seleccionar el identificador que se quiera. Se sitúa justo debajo del label CHOOSE. En su inicio por defecto aparece deshabilitado y vacío. Tiene asignada, mediante el método *bind*, una callback que salta cuando el usuario realiza una nueva selección en el objeto. El método asociado a esta interacción es el *newSelection*.
  La funcionalidad de este método es importante. Primero descarga el valor seleccionado en el combo en un número entero Se obtiene la variable global *BOX*, que como se puede ver contiene una pareja de coordenadas cardinales. Las coordenadas serán los límites de un *subplot*, obtenido de la libería matplotlib. Aquí se introducen una serie de puntos, dentro de esa escala, que los obtendremos de la casilla Places del dataframe *info*. Esta información es decodificada y ploteada sobre los ejes customizados.
  Una vez se tienen los puntos, lo que se realiza es extender la imagen como fondo de esta figura, en los límites establecidos y guardar el resultado como una imagen con el nombre de dorsal mas el sufijo "*map.png*". Una vez almacenada esta imagen en local, para tenerla como referencia de cara a una futura verificación postcompetición, se muestra la imagen en pantalla, actualizando el label MAP.

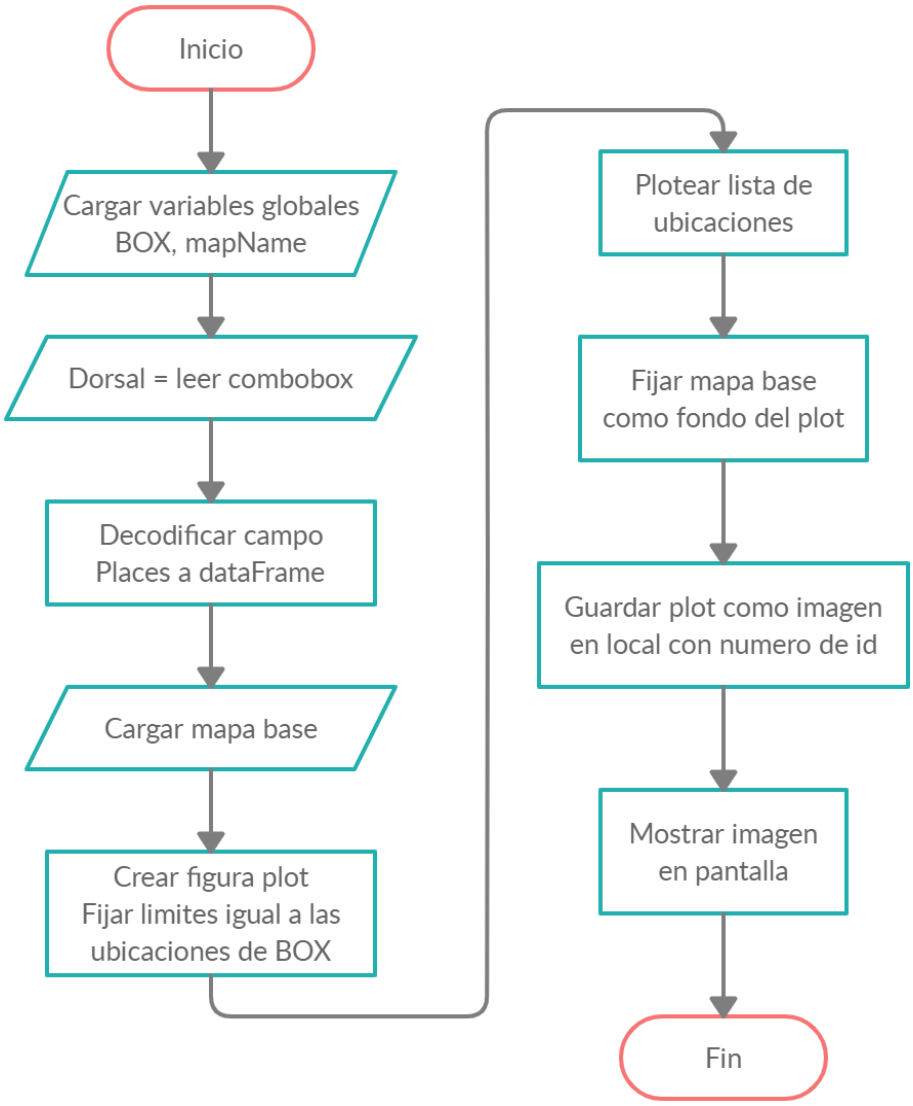


Ilustración 4-5, Flujo función *newSelection*

- reconn: objeto *Button* que aparece debajo del comobox de selección, se utiliza para configurar la aplicación en caso de que no haya habido algún problema en la primera configuración. En el momento de su creación el botón aparece visible, habilitado, y con el método *configurar* asociado.
  Este método llama a las funciones externas *connMqtt*, *getData* y *getMapConfig*. Si ninguna de las tres produce una excepción se fija la variable booleana interna a la clase *configured* a True, se destruye el botón y se habilitan el botón de verificar, descartar, excel y el comobox. Además, se actualiza la imagen del objeto MAP al mapa base; y se guarda este elemento en el objeto *imBase* para poderlo usar más adelante sin recargarlo desde el archivo.

Ilustración 4-6, Botón CONFIGURAR

En caso de que alguna excepción ocurra, se recoje, y se trata de manera diferente en función del código de excpeción:

- Si este es *mqtt error* se muestra una advertencia pop-up informando de que no se ha logrado la conexión con el broker.
- Si es *data_error* el mensaje informará de que no se ha podido cargar la hoja de información de los usuarios.
- Si es *map error* el mensjae informará de que hay algún problema con el fichero de configuración; si es otro se muestra un mensaje de error genérico para código desconocido.
- Además, si la excepción es de tipo '*No such file or directory*' se ha introducido un nombre imagen de mapa base que no se encuentra en el directorio. Esta excepción ocurre en la propia función de de reconfigurar, pero se trata como si fuese externa.

Con esto, en total se contemplan cuatro causísticas diferentes de excepciones.

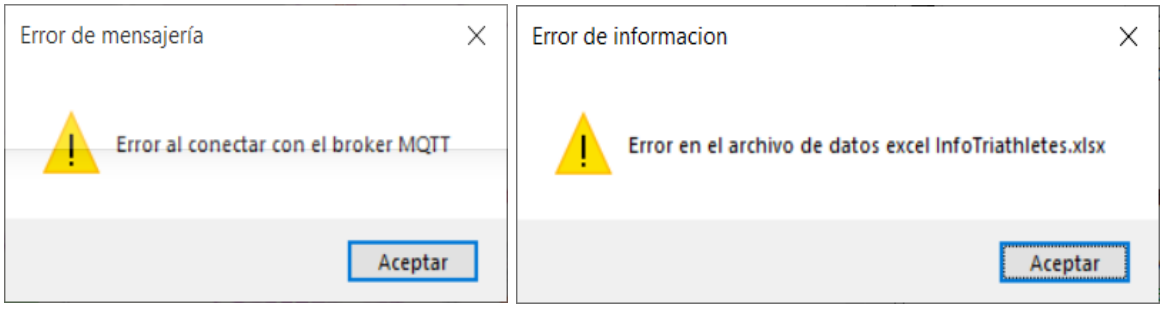


Ilustración 4-7, Mensajes de advertencia en configuración

El método *configurar* se llama justo tras crear los widgets, con lo que si este método funciona bien, no se debería ver el botón en ningún momento, ya que se destruye antes de que se fije la ventana como visible. De esta manera, si todo opera correctamente la configuración será automática, y en caso de que algo falle, al arrancar la aplicación lo hará con alguno de los mensajes de advertencia, teniendo la posibilidad de solucionar los problemas de configuración sin necesidad de reiniciar la aplicación.

Zona derecha de pantalla (*rightFr*). De aquí dependen:

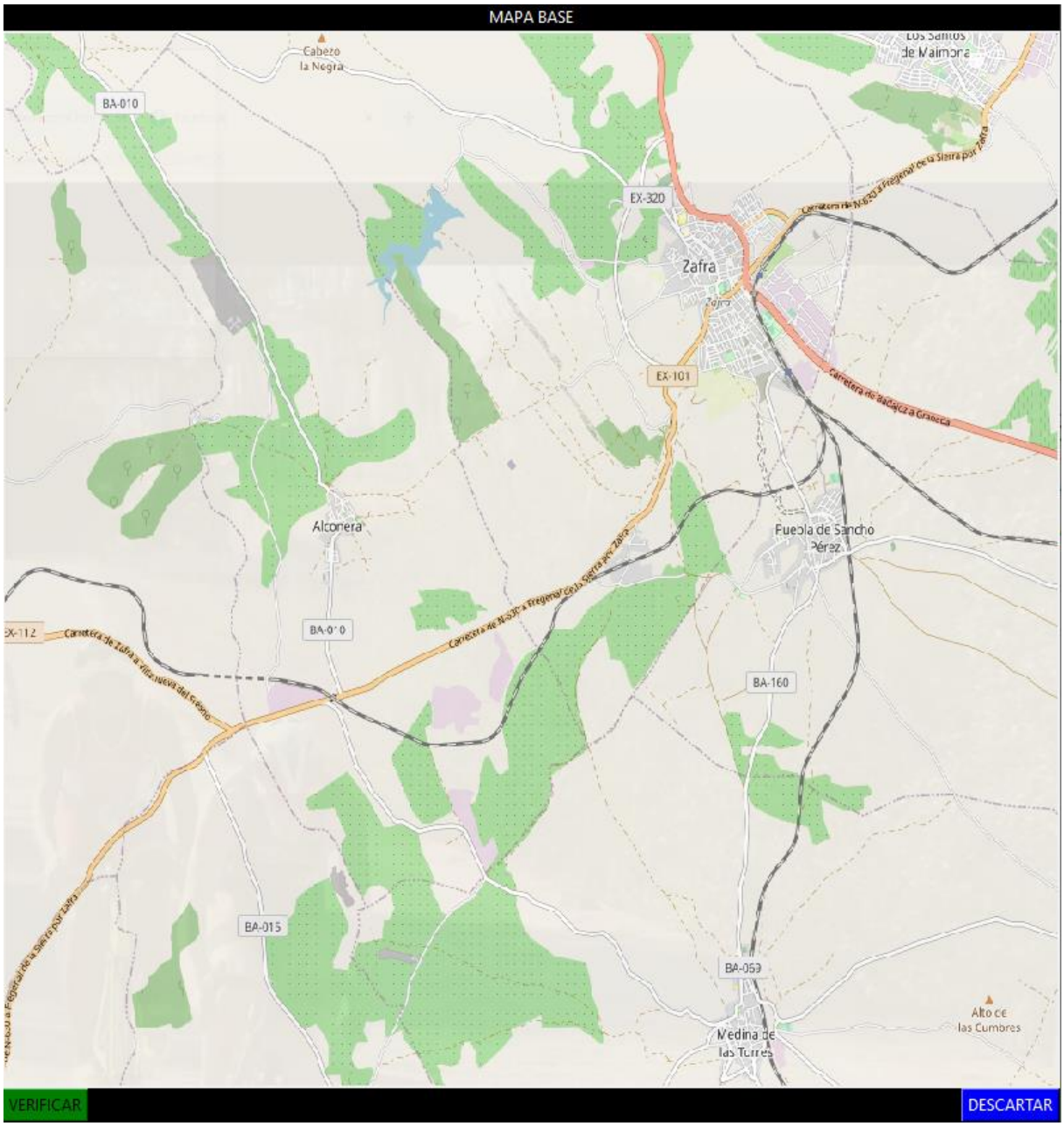


Ilustración 4-8, Mapa base, verificación y descarte

- IMAGEN: objeto *Label* que contiene el título de la imagen de mapa mostrada. Por defecto tiene el título *MAPA BASE* y cuando se realiza una nueva selección se cambiará el nombre al identificador de ese deportista.
- MAP: objeto *Label* que contiene la imagen del mapa. Ocupa la mayor parte de la pantalla y si no hay ningún deportista seleccionado mostrará por defecto el mapa base de la competición, y cuando haya deportista seleccionado, mostrará este mapa customizado con los puntos de infracción. En el momento de su creación, se le asigna la imagen del logo, ya que el nombre del fichero que contiene el mapa base no se conce hasta más adelante, cuando carguemos la configuración correspondiente. Es aquí cuando se mostrará el mapa base, por lo que, si todo funciona correctamente, debería mostrarse directamente el mapa base en primera instancia.
- Verify: objeto *Button* que tiene asociado el método *verificar*. Este método se usa para guardar un positivo en el dataframe de información, en esta estructura lo único que realiza es guardar en en el campo *CheckTime* la fecha actual, como referencia para futuras verificaciones. Además, actualiza el

combobox eliminando el valor que ha sido verificado. Al finalizar se muestra en MAP la imagen del mapa base.

- Discard: objeto *Button* que tiene asociado el método *descartar*. Este método se usa para eliminar la información de una infracción que no pasa los criterios de técnico responsable del control de la aplicación. Al igual que con la opción verificar, se elimina el deportista del combox, pero en este caso, se pone *Infractor* a "*NO*" y se vacia la información de *Places*. Al finalizar se muestra en MAP la imagen del mapa base.

4.2 Conexión y datos

Se pasa a comentar el funcionamiento del resto de la solución, detallando el funcionamiento de la conexión al broker mosquitto y la gestión de datos de participantes.

Primero se pasa a explicar el trabajo realizado por las dos funciones de configuración que se ejecutan al iniciar la aplicación.

- **connMqtt:** en esta función se configura la conexión al broker Mqtt. Para simplificar la configuración se ha decidido que el broker correrá sobre la misma IP en la que se abre la aplicación. Cada vez que se llama a esta función se lanza el broker mosquitto, esto se realiza usando el método *system* del módulo de Python *os*. Se ejecuta la siguiente instrucción:

  os.system('cmd /k ubuntu run sudo service mosquitto start')

  lo que se realiza es ejecutar, desde un command prompt de Windows, en el terminal Ubuntu de Windows 10 el comando "*sudo service mosquitto start*", con lo que se lanza el servicio. Se verá gráficamente como se abre un terminal, que ~~nos~~ preguntará por el password del usuario root de Ubuntu, en caso de tenerlo. Una vez introducido se lanza el servicio y se debe cerrar manualmente la ventana del terminal para seguir usando con normalidad la aplicación.

  Para simplificar la configuración, se ha importado el módulo socket, y con él se ejecuta la siguiente instrucción:

  IP = socket.gethostbyname(socket.gethostname())

  En la variable IP tendremos almacenado un string, con la IP actual de nuestro computador. La idea, es que, en una situación real de competición, esta IP sea estática, dentro del dominio de red local implementado en el área de transición. Pero, en cualquier caso, con esta instrucción se consigue automatizar el proceso.

  A continuación, se asigna a la variable global cliente, hasta ahora vacía, un objeto cliente mosquitto. Este objeto se obtiene de la librería paho-mqtt, habiendo realizado la importación

  import paho.mqtt.client as mqtt

  y asignando:

  client = mqtt.Client()

  Al objeto cliente, se le asigna la función propia on_message como callback a la llegada de mensajes, se realiza la conexión a la IP obtenida, se aranca el bucle para la recepción de menajes y se suscribe a los topics *state* y *places*.

  El posible error llega en la instrucción:

  client.connect(IP)

  en caso de que el broker no este activo en la IP correspondiente a la red local, esta función fallará. En este caso se captura la excepción, y se lanza una excepción propia llamada *mqtt error* usando la siguiente

instrucción:

raise Exception('mqtt error')

- **getData:** en esta función se carga en el dataframe global *info*, hasta ahora vacía, la información del Excel de información de triatletas. Para ello se usa la ulilidad de pandas read_excel. Además, se indexa por la columna dorsal, y se le añaden las nuevas columnas en las que irá cumplimentándose la información recibida de los dispositivos. Las nuevas columnas son: *Infractor, Places, DropTime* y *CheckTime*.
  En caso de que surgiese algún problema, que puede deberse a una estructura errónea del Excel de partida, o a la no existencia del mismo, se lanza de manera similar a la función anterior una excepción propia, de identificador *data error*.
- **getMapConfig:** en esta función se carga el fichero de configuración que contiene toda la información relativa al mapa base, es decir, el nombre del mapa, y la pareja de coordenadas que lo definen. El fichero se recorre línea por línea, las que no comienzan por el carácter '#', se toman como líneas de contenido. El valor de la primera línea de contenido se alamacena en la variable global *mapName*. Para quitar la terminación de nueva línea '\n' se aplica el método genérico que poseen todos los strings en Python *rstrip*.
  mapName = line.rstrip()
  Los 4 valores siguientes, se convierten de string a float, y se almacenan en la variable global BOX.
  En caso de que surja algún problema durante la carga, se lanza una excepción con identificador *map error.*

Por último, queda ver el funcionamiento de la callback a la llegada de nuevos mensajes para los topics mqtt a los que está suscrito el cliente:

- **on_message:** Esta es la función que se ejecutará ante la llegada de un nuevo mensaje. Como argumentos le llega el cliente, información de usuarios y el mensaje en si. En esta solución, como es bastante cerrada, lo más relevante es el mensaje, el resto de argumentos, aunque no les daremos usos tienen que estar presentes en la firma de la función, ya que van a ser enviados igualmente por el broker.
  El mensaje está formado por el atributo *topic*, y el atributo *payload*, este segundo atrubuto contiene la información publicada. Indiferentemente de topic que se envíe, se decodifica-la información de payload a texto plano, y se divide por cada espacio que contiene. A continuación, se tratará de manera diferente en función del topic que haya llegado.
  Si es del topic *state*, se guarda la información sobre si ha infringido o no en la distancia de seguridad en el campo *Infractor* del dataframe *info*, para ello, la información obtenida de *payload* habrá dado lugar a un vector de dos componentes tipo string, en la posición 0 estará el número identificador del deportista, que se usará para indexar la información, y en la posición 1 se tendrá un mensaje '*SI'* o '*NO'*. Además, si no ha infringido, el otro campo a rellenar será el *DropTime,* en el que se incluirá-la fecha y hora actual y, automáticamente se publicará en el topic *checked* un mensaje con el número de identificación para que ese dispositivo se apague.
  Para publicar el mensaje, se usa la variable global del cliente, con una llamada con la siguiente estructura:
  client.publish('checked',msg[0])
  En cambio, si el mensaje que llega es del topic *places*, se anexan, de manera similar, colocando en el campo *places* la información de las coordenadas de infracción. Si todavía no se conocía el estado de infracción porque se hubiese perdido el mensaje, se fuerza el campo *Infractor* a *SI.* A continuación, se manda el mismo mensaje para apagar el dispositivo detector y se cumplimente el campo *DropTime* en el dataframe de información de deportistas.
  Por último, se llama al método de la aplicación gráfica newCheater, pasándole como argumento el número de identificación del deportista.
- **newCheater:** este método lo único que hace es añadir al desplegable del combobox el número que recibe por argumentos.

5 DESARROLLO DETECTOR

5.1 Estructura y configuración

El desarrollo del dispositivo detector está formado por tres scripts: *sendInfo.py, gps.py* y *read.py*. Los dos primeros se usarán como módulos dependientes del tercero. Cada uno de estos archivos, así como el orden de funcionamiento, se comentará en detalle en este apartado.

En cuanto al arranque del dispositivo, por regla general, las distintas distrubuciones de microPython tienen un proceso de arranque específico. En concreto, todas poseen un script llamado *boot.py* que se ejecuta atomáticamente tras cada inicio de un dispositivo con firmware microPython, y es aquí donde se ha incluido la importación del script principal, y se llama a su método *run()*. De esta manera, con cada reinicio, sin necesidad de interactuar, el dispositivo comienza a trabajar de manera automática.

5.1.1 Flasheo y configuración

El microcontrolador ESP32 Huzzah usado en este proyecto, necesita ser flasheado a una versión de microPython, ya que de fábrica no trae el intérprete cargado.

Para flashear el dispositivo, se debe conectarle mediante el puerto microUSB del ESP a uno de los USB del PC desde el que se va a realizar el flasheo. Esta conexión será detectada como un puerto serie. Para ver qué número de puerto serie, nos podemos dirigir al administrador de dispositivos de Windows, y buscar el número de puerto COM asignado.

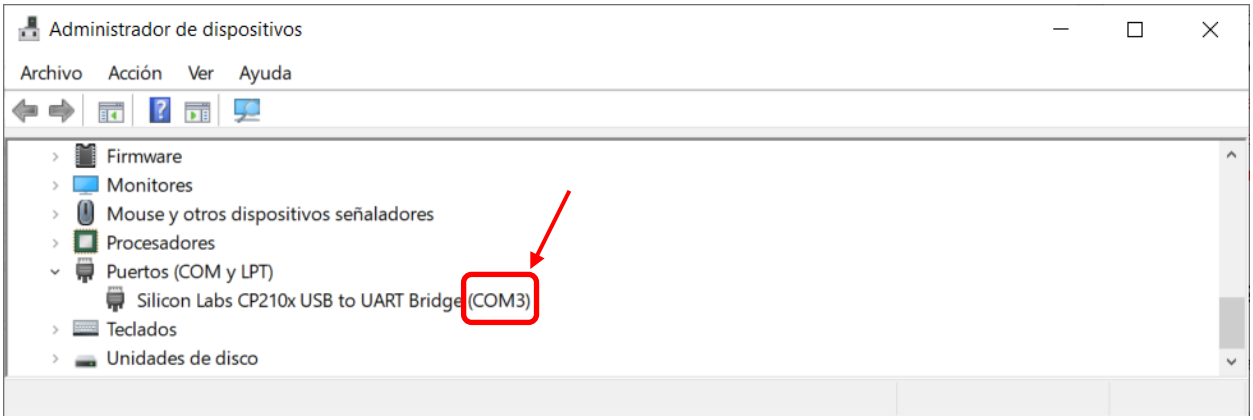


Ilustración 5-1, Búsqueda de puerto COM

Una vez conocido el número de puerto, se instala en el terminal de Ubuntu la herramienta *esptool* desarrollada por espressif de manera gratuita y de código abierto (publico en GitHub). Para su instalación se debe tener disponible Python3 en el entorno de trabajo, con lo que tan sólo hay que ejecturar la siguiente orden en el terminal:

- ***sudo pip3 install esptool***

con esto debe quedar disponible en el path la herramienta para su uso posterior.

A continuación, hay que obtener una versión del intérprete de microPython. Existen versiones para múltiples dispositivos de libre descarga en la web *https://micropython.org/* , en el directorio de descargas de la misma, se selecciona el dispositivo ESP32, y, en concreto se ha descargado la versión V1.12, de la distribución V3x, que es la que da soporte para el uso de conectividad LAN en el micro. Se descarga un archivo con extensión *.bin*, el cual debemos guardar en un directorio conocido.

Una vez descargado, ejecutamos el siguiente comando:

- ***esptool.py --chip esp32 –port /dev/ttyS3 erase_flash***

, donde */dev/ttyS3* es el identificador del puerto COM. Con esto se limpia el anterior contenido de la memoria del ESP, y lo dejamos listo para poder instalar el nuevo intérprete.

La instalación se realiza con la instrucción:

- ***esptool.py –chip esp32 –port /dev/ttyS3 write_flash -z 0x1000 esp32-idf3-20191220-v1.12.bin***

donde se indica que la memoria comience en la dirección 0x1000, esta es la configuración recomendada para no modificar memoria protegida por el dispositivo, y el último argumento es la versión del intérprete que se ha descargado antes (o su ruta completa si no está en el directorio de trabajo).

Una vez se ha cargado el nuevo firmware, ya se puede comprobar el funcionamiento abriendo en un cliente de terminal, como puede ser *TeraTerm* o *Putty*, el puerto serie correspondiente. La configuración por defecto son 9600 baudios, 8 bits de mensaje, 1 bit de parada y sin paridad. Si una vez abierto, aparece un intérprete de Python que permite ejecutar sentencias básicas como la importación de un módulo y checkear los archivos disponibles en un directorio:

Si esto funciona, se habrá instalado correctamente el intérprete, y ya se puede comenzar a desarrollar sobre él.

```
>>> import os
>>> os.listdir()
['umqttsimple.py', 'sendInfo.py', 'wireless.conf', 'gps.py', 'currGPS.log', 'read.py', 'ssd1306.py', 'cheat.log', 'user.conf']
>>>
```

Ilustración 5-2, Comprobación de instalación microPython

5.1.2 Carga de archivos a dispositivo.

Para cargar los distintos archivos necesarios para el funcionamiento del proyecto, se ha usado la utilidad Ampy (Adafruit MicroPython Tool). Esta es una herramienta para ser usada en línea de comandos, existente tanto para Windows como para Linux, al cual se ha instalado en este caso en un terminal Ubuntu. Al correr sobre Python es necesario tenerlo instalado e incluido en el path de trabajo, partiendo de aquí, para su instalación se usa la herramienta *pip* de Python, con lo que se ejecuta la siguiente instrucción:

- ***sudo pip3 install adafruit-ampy***

con esto se habrá instalado la herramienta. El uso principal que se le va a dar, y el único que se comentará en específico, es el de cargar archivos al dispositivo. Además, se puede usar para eliminar, obtener, gestionar directrios, resetear o correr archivos. Estas otras utilidades se pueden realizar directamente en el terminal de micoPython, por ello no se les ha dado uso.

En concreto, para la carga de archivos se ejecuta el siguiente comando

- ***ampy -p /dev/ttyS3 put directorio/archive.file***

donde, */dev/ttyS3* representa el puerto COM al cual está conectado el dispositivo. Si se ve desde el explorador de Windows, será el puerto COM3, lo que el sistema Linux trata como /dev/ttyS3. *archive.file* se corresponde con el nombre del archivo, del cual, si está en un directorio distintio al actual habrá que pasar su ruta relativa o

completa. Si ejecutando el comando, no da ningún mensaje, es indicativo de que ha funcionado correctamente, mientras que, si deja algún tipo de información por pantalla, que, por regla general, no es fácilmente legible para alguien no habituado a esta tecnología, no habrá funcionado bien. Los fallos más comunes suelen ser, un error en el número de puerto, que el puerto ya está abierto por otra aplicación, o que el archivo no existe o no está en el directorio correcto.

Esta utilidad se ha usado para cargar los archivos *.py, los cuales solo deben ser cargados una vez, por el fabricante del producto. Para la carga de los archivos de configuración por parte de la organización se ha creado un script Shell para Ubuntu, llamado *loadConfig.sh.* Al ser una tarea bastante repetitiva, ya que debe ser realizadad para cada usuario de manera individual, se ha intentado automatizar lo máximo posible.

El script depende de tres parámetros. El primero de ellos es el path en el que se encuentrasn los archivos de configuración. Aquí es importante tener en cuenta que estos archivos son los mismo para todos los usuarios de una competición, por lo que la idea es tenerlos todos en un mismo directorio, y guardar la ruta de este directorio en la variable global *TargetPath*. Para ello ejecuta:

- ***export TargetPath='/mnt/c/myPath'***

en caso de que los archivos se encuentren en el mismo directorio que el script, es prescindible crear esta variable.

Los otros dos parámetros son, por este orden, el número de puerto COM y el número de identificador de usuario. Estos parámetros pueden ser pasados como argumentos al lanzar el script, por este orden, y siempre los dos juntos. En caso de que alguno, o ambos falten, se pedirán específicamente por el terminal.

Una vez se disponen de todos los parámetros se cargan los tres archivos de configuración comunes al dispositivo. Internamente, el scirpt ejecuta la instrucción de *ampy* correspondiente. Para el archivo *user.conf*, lo primero que ser realiza es borrar un posible archivo con el mismo nombre antiguo con la instrucción:

- ***rm -f user.conf***

se vuelca el número identificativo a un nuevo archivo:

- ***echo $user >> user.conf***

en la variable *user* se ha guardado antes internamente el número identificativo. Por último, se carga mediante *ampy* el archivo, y se elimina de la computadora.

Con este script se simplifica bastante la carga de archivos; para, una vez creados los archivos comunes, ejecutar una sóla instrucción en un terminal Linux.

5.2 Montaje

Pin dispositivo	Pin ESP32
TF Mini Lidar	
Vcc	USB
Rx	--
Tx	32
Gnd	GND
GPS Gy Neo 6MV2	
Vcc	3V
Rx	17
Tx	16
En el Gnd	GND
Buzzer (Cem 1203-42)	
+	21
-	GND
Led	
+	33
-	GND
SSD 1306	
Vcc	3V
Gnd	GND
SDA	23
SCL	22

Tabla 5-1, Pinout montaje

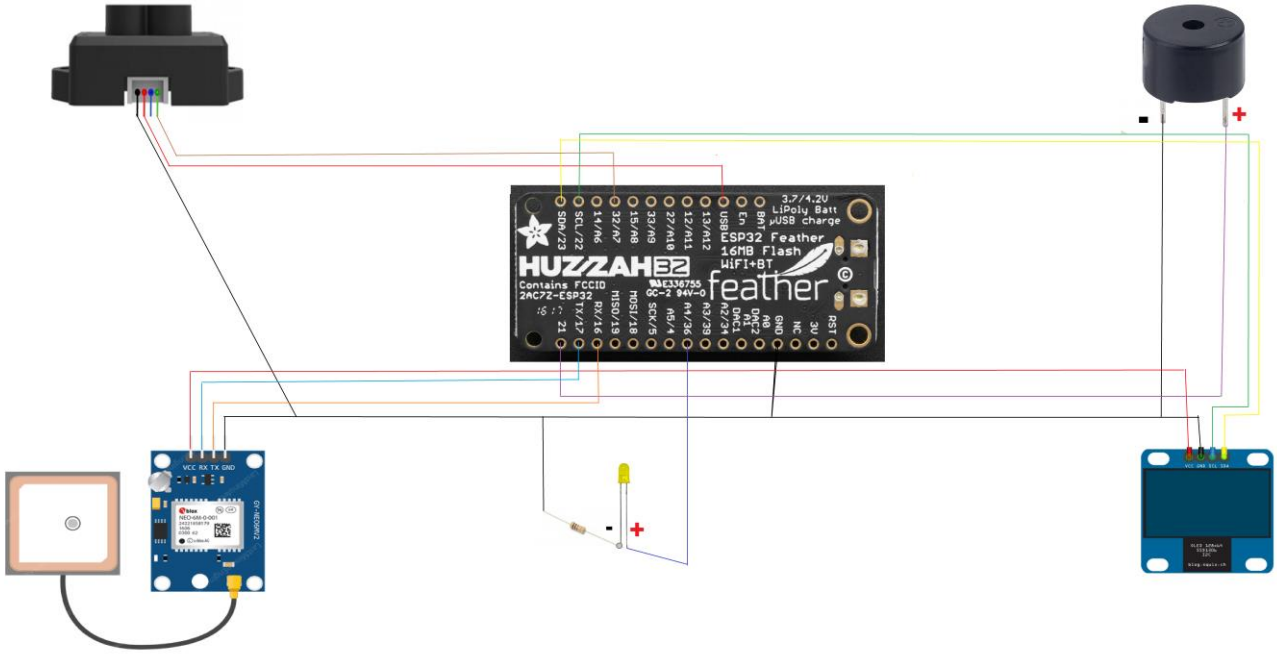


Ilustración 5-3, Esquema de montaje y pinout

5.3 Lectura distancia, ciclo principal

Lo primero que se realiza en este bloque es importar los módulos gps y sendInfo. Al estar en el mismo directorio, es suficiente con poner:

import gps

import sendInfo

al comienzo del archivo.

Una vez en la función run, se lanza el hilo que va a recoger la información del GPS. Para ello se usa el módulo thread de micropthon, con la siguiente instrucción:

gpsThread = _thread.start_new_thread(actGPS, ())

donde, en gpsThread se obtiene el objeto identificador del hilo. La función actGPS que se pasa como argumento tiene una estructura concreta. Va recorriendo como si de un vector se tratase la función que gestiona el gps, y va actualizando los datos que proporciona en la variable global currGPS.

Esto es debido a que los datos del gps se van añadiendo con la instrucción *yield* de Python, la cual efectua el mismo efecto que el clásico return, pero sin finalizar la misma, por lo que va adjuntando valores. El bucle resulta como el siguiente:

for current in gps.run():

currGPS = current

El ciclo run del módulo gps es infinito. La manera de finalizar este proceso será matando al hilo una vez este nos envíe el mensaje '*END*', que indica que el deportista ha llegado a la segunda transición. Además, esta condición es también la que se usa para finalizar el bucle principal del módulo inicial.

Por último, antes de iniciar el bucle principal, se configura el sensor de distancia con la función *setLidar,* la cual abre la UART 2 en los pines 14 y 32 del ESP a 115200 baudios, 8bits, 1 bit de parada y sin paridad. Esta UART

es la que se recibe de la función, para trabajar más adelante con el sensor. También se fija como pin de salida el pin 33, que será al que irá conectado el led para informar de la distancia.

Antes de iniciar el bucle principal, se crea la variable *cheatInfo* como cadena vacía. A esta cadena se le irá adjuntando la ubicación de las infracciones de los deportistas, y se enviará como argumento al módulo de envío de información a la apliación organizativa.

El bucle principal será el encargado de detectar las situaciones de infracción. Lo primero que se debe tener en cuenta es la ubicación GPS, esto lo conocemos en función del valor de la variable currGPS. Si su valor es 'PASS' indica que, o bien no hay conexión y no se conoce la ubicación, o bien estamos en una zona segura. Bajo cualquiera de estas situaciones, no se va a proceder a acumular tiempo de infracción.

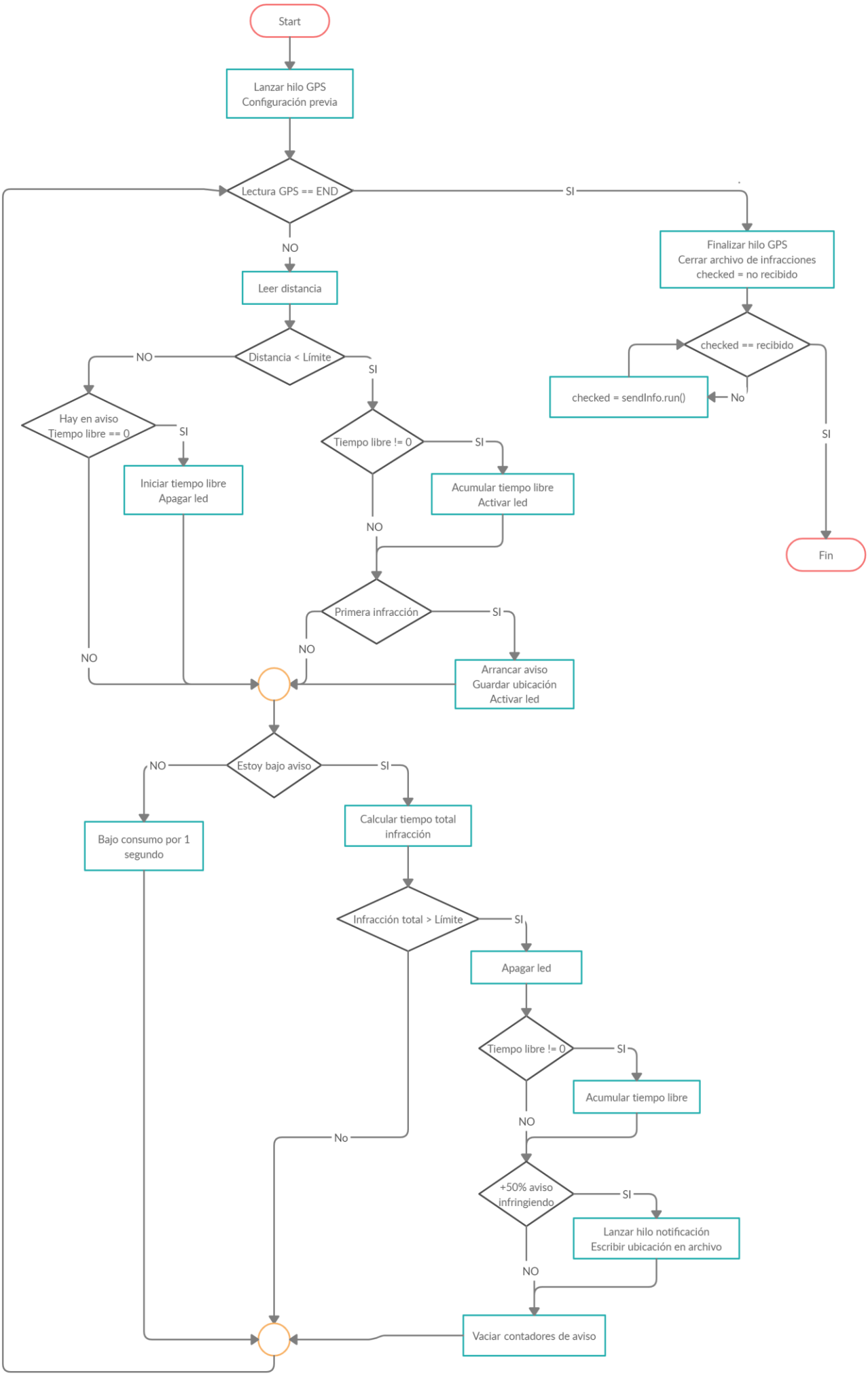
Suponiendo que el gps está proporcionando una ubicación, si el deportista comienza a infrigir, la distancia leída es menor del límite, pasamos a estar en un estado que podemos llamar de aviso. Durante 10 segundos el dispositivo va a estar comprobando con la máxima frecuencia posible la medida del sensor, si es menor al límite, se acumula el tiempo de infracción, y si no lo es, se acumula en lo que llamaremos tiempo libre. Para ello, tal como se puede ver en el diagrama de flujo se ha diseñado una estructura que va acumulando el tiempo libre, y el tiempo total del aviso. Cuando este último llega al límite, se desactiva el aviso, y se pasa a comprobar el resultado. Si durante más de la mitad del aviso se ha estado acumulando tiempo libre, o lo que es lo mismo, durante más de la mitad del tiempo no se ha infringido la distancia de seguridad, se descarta el positivo. En caso contrario, si el tiempo libre es menor que el total del aviso, se considerará una infracción. En caso de infracción se añade la ubicación (correspondiente al comienzo del aviso) en la variable *cheatInfo*.

Además, durante este ciclo de recnocimiento de positivos, siempre que bajo un aviso estemos infringiendo la distancia de drafting, se iluminará el led del pin 33 a modo de información para el deportista. El tiempo de no infracción, bajo un aviso, el led estará apagado. Una vez finalizado un aviso, si el resultado ha sido positivo, en un nuevo hilo, de manera similar al GPS, usando el módulo thread, se lanzará una alarma sonora de duración limitada. Este hilo ejecutará la función *userAdvice*.

La función de alarma a usuario usa un pwm en el pin 21 del ESP. La configuración es muy sencilla, basta con importar el módulo *PWM* que viene en el bloque *machine*, y crear un objeto con argumento otro objeto tipo *Pin* (también un módulo de *machine*) número 21. Después fijamos el ciclo al cincuenta por ciento y, durante los segundos que dura el aviso, vamos variando la frecuencia del pwm de 500 a 1000 Hz, de manera que resulta una alarma reconocible. Una vez superado el tiempo de duración de la alarma, se elimina el pwm y el hilo finaliza.

Con la llegada del mensaje *END* del gps, finaliza este bucle principal, con lo que se pasa al proceso de finalización del ciclo funcionamiento del dispositivo. La primero que se realiza es matar al hilo del GPS, como antes se ha obtenido el identificador del hilo en la variable gpsThread, tán solo hay llamar a su método *exit*.

Por último, se entra en un bucle infinito, hasta que el método *run* del módulo *sendInfo* devuleva el mensaje '*checked'*. Si todo va bien, este bucle sólo tendrá un ciclo, y nunca se cumplirá la condición. El dispositivo se apagará dentro de la función a la que está llamando.


Start
Lanzar hilo GPS
Configuración previa
Lectura GPS == END
SI
NO
Finalizar hilo GPS
Cerrar archivo de infracciones
checked = no recibido
Leer distancia
checked == recibido
checked = sendInfo.run()
No
SI
Distancia < Límite
NO
SI
Hay en aviso
Tiempo libre == 0
SI
Tiempo libre != 0
SI
Iniciar tiempo libre
Apagar led
Acumular tiempo libre
Activar led
Fin
NO
Primera infracción
SI
NO
NO
Arrancar aviso
Guardar ubicación
Activar led
Estoy bajo aviso
NO
SI
Bajo consumo por 1
segundo
Calcular tiempo total
infracción
Infracción total > Límite
SI
Apagar led
Tiempo libre != 0
SI
Acumular tiempo libre
NO
No
+50% aviso
infringiendo
SI
Lanzar hilo notificación
Escribir ubicación en archivo
NO
Vaciar contadores de aviso

5.4 Conexión a SSID

El script *sendInfo.py* es importado desde el script principal, que realiza las lecturas. Tal y como ya se ha comentado la comunicación se basa en el protocolo Mqtt. Para aplicarlo en el ESP se ha usado la utilidad *umqttsimple*, ya comentada previamente.

Al importar el script, lo primero que ser hace es leer el archivo de configuración wireless.conf el cual sigue la estructura común de todos los archivos de configuración del proyecto. De él, se obtiene, por este orden, el nombre del ssid wifi, su contraseña, y la IP en la que estará corriendo el broker Mqtt; estos datos se cargan como variables globales al bloque.

Una vez cargado al bloque, para lanzar un intento de comunicaciones, se realiza la llamada a la función run. En ella se realiza la conexión a la Wifi, y a continuación al broker Mqtt; Para ello se tienen dos funciones dedicadas, que en caso de funcionar correctamente devuelven el objeto correspondiente, mientras que en caso de haber algún error en el proceso, se lanza una excepción con el identificador de en cual de los bloques se ha producido.

Para la conexión a la WiFi, se usa la utilidad network de microPython, con ello se crea un objeto llamada *wifi* y se ejecuta su método connect con el ssid y password obtenido del fichero de configuración. Se da un tiempo de 30 segundos para alcanzar la conexión, y una vez se ha obtenido se devuelve el objeto *wifi*. Si no se consigue realizar la conexión se devuelve el código error.

El funcionamiento de la conexión al broker Mqtt es similar al de la wifi, en este caso, además de abrir un cliente, que será el objeto Mqtt con el que trabajar, se suscribe al topic y se configura la callback para publicaciones de los ítems suscritos. Esta callback es la función sub_cb, que se comenta a continuación:
El dispositivo detector, sólo estará suscrito al topic *checked*, cuando llegue un mensaje de este tipo, se comprueba si el identificador coincide con el del identificador asociado al dispositivo, si es así, se ejecuta *machine.deepsleep()*, poiniendo en el modo bajo consumo más estricto al dispositivo de manera indefinida, siendo esto lo más similar a estar apagado.

Una vez se ha configurado satisfactoriamente todo, usando la variable *cheatInfo* recibida por argumentos; en la que se han almacenado los casos de infracción, se envia,a través de la función creada para ello, el contenido y estado de dispositivo. Si ha habido infracción, se envían mensajes a los topics *state* y *places*, por este orden. En caso de que no haya habido infracción sólo se envía contenido al primer topic.

Una vez enviados los mensajes, durante un tiempo de 5 minutos se espera la recepción de mensajes al topic suscrito. Si tras este tiempo, no se ha–recibido la confirmación, se vuelve a enviar la información, y así recursivamente hasta que se reciba nuestro id en el topic checked y se desconecte el dispositivo.

5.5 GPS

El script de gps.py está diseñado para ser importado desde el script principal en un nuevo hilo. En el se cargan dos archivos de configuración, safeZone.conf y connZone.conf, en ellos respectivamente se tiene el conjunto de áreas en las que no se debe detectar la distancia entre deportistas, llamadas *zonas seguras*, y en el otro, el área en el que se debe realizar la conexión y la marca temporal para ello. Estos archivos mantienen la estructura ya comentada de los archivos de configuración del proyecto con la posibilidad de añadirles líneas de comentarios.

El script se basa en una sola función, la función *run()* . Esta función es la que se llama desde el hilo principal.

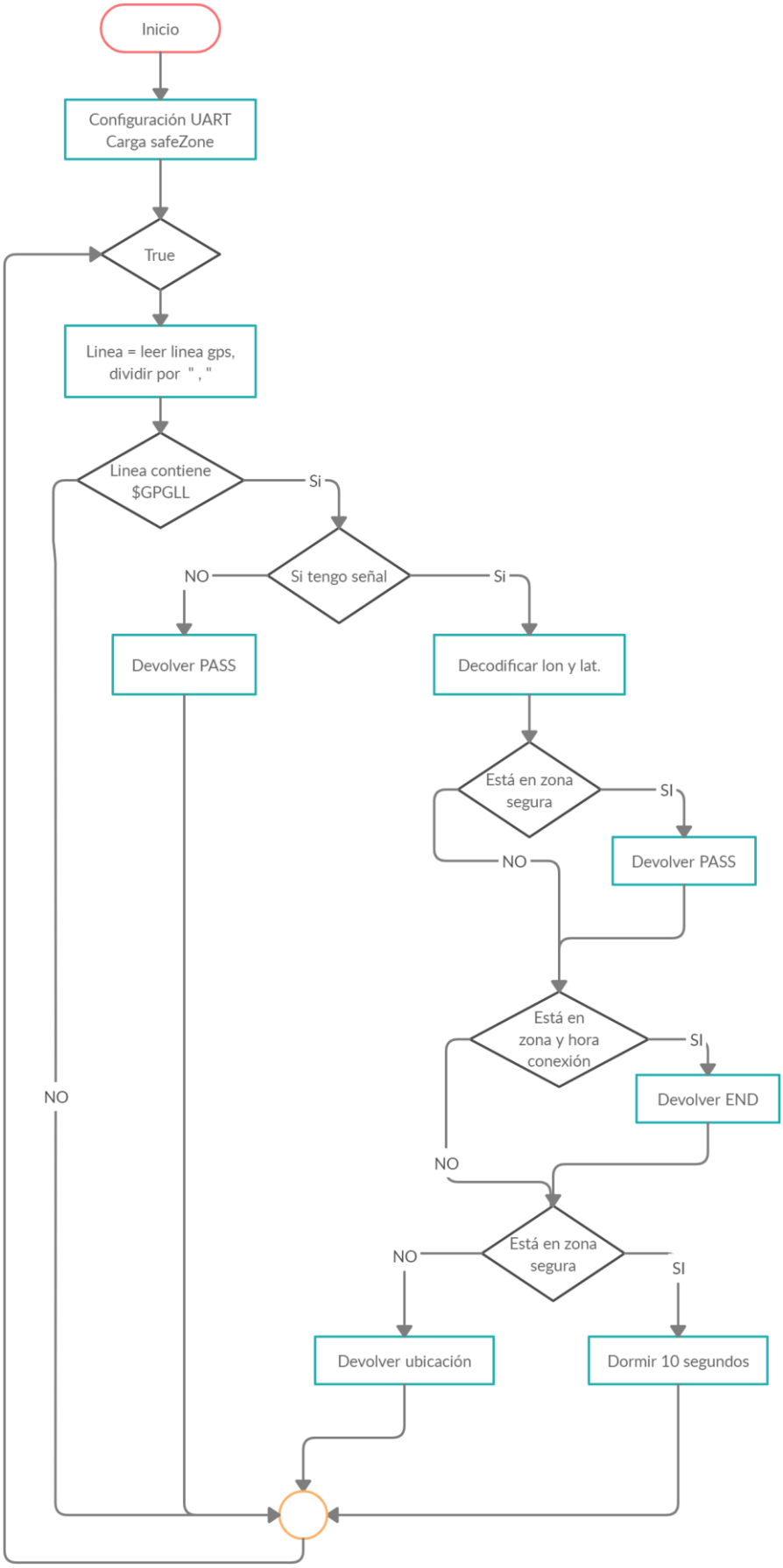


Ilustración 5-5, Flujo del hilo GPS en dispositivo detector

Se pasa a comentar el funcionamiento de la misma:

Se configura la UART del dispositivo en los pines 17 y 16, a 9600 baudios, sin paridad, con un bit de parada y 8 bits de mensaje. Se llama a la carga de los ficheros de configuración y se pasa al bucle infinito. En este bucle se lee la información de la UART que ~~nos~~ viene en caracteres ACII. Esto son tramas bajo el protocolo de comunicaciones NMEA (*National Marine Electronics Association*) 0183.

El protocolo NMEA 0183 es usado por mútiples dispositivos de la industria marítima, y por la gran mayoría de dispositivos GPS. Este protocolo propone una comunicación por puerto seríe, sin paridad, con un bit de parada a 4800 baudios de manera típica. Aunque el dispositivo en concreto se puede usar como una UART; y cambia la transmisión a 9600 baudios. Las tramas son líneas de texto, con una estructura concreta, en ellas se recibe información acerca de la posición en longitud y latitud, fecha y hora actual o satétiles visibles. La información concreta que proporciona el sensor se compone de un conjunto de líneas, todas ellas comienzan por *$GP*, y a continuación tres letras más que indican que mensaje es el que se está enviando. En el dispositivo, este código de tres letras puede ser:

```
'$GPGLL,3825.22967,N,00624.93251,W,102651.00,A,A*72\r\n'
'$GPRMC,102652.00,A,3825.22977,N,00624.93252,W,0.504,,270820,,,A*64\r\n'
'$GPVTG,,T,,M,0.504,N,0.934,K,A*2C\r\n'
'$GPGGA,102652.00,'
'3825.22977,N,00624.93252,W,1,06,1.44,528.2,M,48.2,M,,*42\r\n'
'$GPGSA,A,3,10,21,16,01,08,11,,,,,,,2.63,1.44,2.20*08\r\n'
'$GPGSV,3'
',1,09,01,46,278,37,08,80,060,30,10,26,044,37,11,63,317,39*76\r\n'
'$GPGSV,3,2,09,16,17,171,23,21,43,057,37,22,41,205,,23,00,0'
'38,22*7C\r\n'
'$GPGSV,3,3,09,39,35,135,30*48\r\n'
```

Ilustración 5-6, Mensajes NMEA del GPS GY NEO 6MV2

- GLL: latitud, longitud y hora.
- RMC: Datos mínimos recomendados (posicion, velocidad, hora)
- VTG: Vector velocidad sobre la superficie.
- GGA: Calidad y ubicación 3D.
- GSA: Información general de satélites
- GSV: Información detallada de satélites.

En concreto, la sentencia que se ha usado para obtener la información es la GLL, que contiene siete argumentos, de los cuales, son los cinco primeros los que se usarán. El primer argumento es el valor (decimal) de la latitud, será un valor de 0 a 90, siempre positivo y de cuatro números en su parte entera. Las dos primeras cifras de la parte entera indican los grados, y las dos últimas de la parte entera, y las de la parte decimal, los minutos. El segundo valor será un carácter N o S, que indica si son orientación norte o sur respectivamente. Los parámetros tercer y cuarto son los correspondiente a la longitud, la estructura es similar a la de la latitud, con la salvedad

que ahora los grados pueden valer de 0 a 180, con lo que serán cinco lo números de la parte entera y serán los tres primeros los que indiquen el valor de los grados. El carácter de la posición cuarta será E o W, indicando si es orientación este u oeste (*west* en inglés). Por útltimo, el quinto carácter corresponde a la hora, viene en formato de 24h, con lo que llegarán 6 caracteres en un entero siendo, en grupos de 2 y de izquierda a derecha,

Ilustración 5-7, Trama GLL del sensor GPS

horas, minutos y segundos.

Para localizar la trama que se busca entre los mensajes del sensor, se va leyendo por líneas completas la información, pare ello se utiliza el método *readline* de la UART. El mensaje leído se divide por el carácter " , ", con lo que obtenemos un vector de strings en los que vendrá la información. La primera comoprobación realizada es ver si el primer parámetro del vector, contiene el mensaje $GPGLL, si es así, se analiza el mensaje, en caso contrario, simplemente se descarta y se lee la siguiente línea.

Una vez encontrada la línea, se comprueba si hay conexión o no. Para ello tan sólo hay que comprobar si los campos correspondientes a la ubicación tienen información, en caso de no disponer de suficientes satélites para disponer de conexión, estos campos aparecerán vacíos. Si no hay conexión, automáticamente devolvemos el mensaje *PASS*.

En caso de que haya conexión, al comienzo se realiza es interpretar la ubicación, para que se convierta a su valor en grados decimales, con signo positivo o negativo en función de la coordenada cardinal correspondiente. Si es Norte o Este, será signo positivo, mientras que tanto Sur como Oeste llevará signo negativo. Esta estructura de coordenadas es más común y fácil de tratar a posteriori que la que ofrece la trama por defecto. Se comienza obteniendo los grados y minutos en número independientes, y a continuación, a los grados, sumarle los minutos divididos por 60. Por último, si el signo cardinal es negativo, se aplica al valor calculado.

Una vez calculada la ubicación, se comprueba si estamos dentro de alguna safeZone. Esto es solamente comprobar si ambas coordenadas se encuentran dentro de los márgenes que forman los rectángulos de la s zonas seguras. Si es así se devuelve el mensaje *PASS*. A continuación, se comprueba si se está en el área de conexión y en la marca temporal requerida. Si es así, se devuelve el mensaje *END*, para que el script principal llame al bloque de importación. Si tan sólo se está en el área de conexión, pero la marca temporal no se cumple, se devuelve el mensaje *PASS*.

Si tras este proceso, se conoce la ubicación, y no se encuentra en ninguna de las ubicaciones seguras o de conexión, se devuelve el mensaje con la estructura *#longitud#latitud#hora*. Con lo que podrá ser interpretado por el hilo padre. Si se ha concretado que la ubicación es una de estas zonas, se fija el dispositivo en un estado de bajo consumo por 10 segundos.

En este hilo hay que destacar que cuando se dice devolver un mensaje, el hilo no finaliza, porque para ello se está usando la función *yield*, que va adjuntando valores al hilo principal, como si de un vector se tratase. Se consigue que el hilo tan sólo haya que lanzarlo una vez; y que la lectura de valores sea mucho más sencilla.

5.6 Pantalla

El dispositivo desarrollado cuenta con una pantalla OLED SSD1306 de 32x128 bits. Su uso va enfocado al desarrollo de pruebas para comprobar el correcto funcionamiento tanto del sensor de distancia como del resto del proceso de detección. Este elemento no se incluye como elemento funcional en el dispositivo de cara a un uso en competición. Por ello no se ha incluido su desarrollo en el resto de las partes del proyecto, y se pasa a comentar su utilización en este apartado en exclusiva.

La pantalla comunica por bus I2C, este protocolo es uno de lo más comunes para la comunicación con dispositivos de un circuito integrado, y también uno de los más comunes en sensores y complementos de bajo

costo para proyectos embebidos en muchos casos no industriales. Se basa en una arquitectura maestro-esclavo, en la que, en nuestro proyecto el maestro será el ESP.

Para usarlo se crea un objeto de este bus, usando el módulo I2C del bloque *machine* de microPython, con el pin SCL como el 22 y el SDA como el 23, que son los que por defecto van asociados al módulo I2C del ESP32. A continuación, usando la libreriá *ssd1306*, incluida en el conjuntos de librerías de apoyo en microPython, al igul que se encontraba *umqttsimple*, se crea un objeto oled, en ese bus I2C, y se configura con la pantalla vacía con el método *fill(0)*. Todo esto se ha incluido en la función *setScreen*.

Se ha desarrollado una función, a la cual pasándole como argumento las líneas de texto a mostrar, que limpia el contenido anterior de la pantalla y muestra el actual. Esta es la función *dispMsg*. Tiene como argumentos, el objeto que representa la pantalla, y el mensaje a mostrar en la primera línea. Se pueden pasar dos argumentos más, opcionales, que muestren por pantalla información en las dos siguientes líneas de la pantalla. Esta función elimina el contenido anterior de la pantalla, y escribe el nuevo, usando el método *text("")*. Finalmente, muestra este nuevo contenido en pantalla.

El uso que se ha dado a la pantalla se divide en tres estados principales.

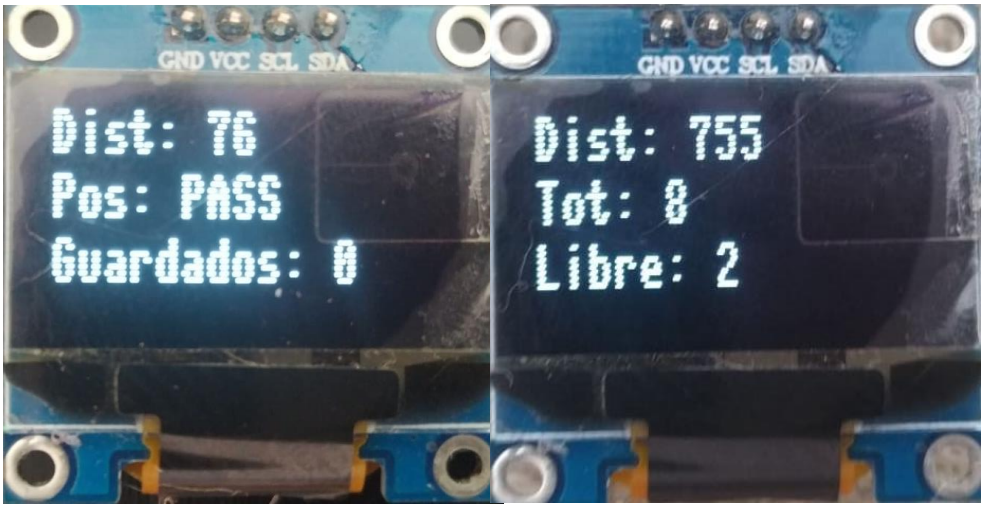


Ilustración 5-8, Pantallas de información en OLED

- El primero es, mientras se encuentra detectando la distancia, (ya se esté en zona segura o no) se muestra la distancia leída por el sensor, la información del GPS, y el total de infracciones acumuladas.
- El segundo de estado, ocurre cuando el dispositivo está bajo aviso, en ese caso se muestra la distancia leída, el tiempo de duración del aviso, y el tiempo libre dentro del aviso acumulado.
- El último de los estados corresponde al estado de conexión, aquí se mostrará el mensaje "FIN DETECCIÓN" en la pantalla.

6 DEMOSTRADORES Y PRUEBAS

6.1 Aplicación organización

Para el testeo del funcionamiento de la apliación en sintonía con el dispositivo detector se han desarrollado pruebas unitarias de la comunicación. Primero, simulando los mensajes que enviaría el dispositivo desde un terminal Linux, para ello se ha usado la utilidad *mosqitto_pub*:

➔ ***mosquitto_pub -h 192.168.43.22 -t 'state' -m 'SI'***

donde el argumento -h es la IP del bróker, -t el topic y -m el mensaje asociado. Para suscribirse a un topic, en un terminal nuevo, se ejecuta:

➔ ***mosquitto_sub -h 192.168.43.22 -t 'checked'***

e irán apareciendo los mensajes en ese terminal.

Ilustración 6-1, Límites de mapa base

También se han desarrollado pruebas tipo end-to-end, llamando manualmente a la función *run* del bloque *sendInfo* del proyecto, y pasándole la cadena de infracción por argumentos, tanto para casos positivos como negativos.

En ambos casos el funcionamiento es bueno.

Es importante haber lanzado la aplicación antes que la funcionalidad en el dispositivo detector, ya que, si no es así, el mensaje llega al broker, pero no a la aplicación. Esto no es un problema, ya que el dispositivo tampoco recibirá la confirmación y tras cinco minutos volverá a enviar su información, pero es un detalle a tener en cuenta.

La funcionalidad es correcta y rápida.

A continuación, se pasa a mostrar el funcionamiento de la aplicación durante su uso:

- Antes de lanzar la aplicación, hay que configurarla. La configuración se forma de dos partes, primero buscar el mapa base, y segundo buscar sus límites para copiarlos en el fichero de configuración.

  - El primer paso, buscar el mapa, es muy sencillo, tan sólo se necesita una imagen con todo el recorrido, esto suele estar disponible en la documentación de cualquier competición, por lo que no representaría una carga de trabajo extra, además que esta imagen suele tener resaltado la ruta a seguir. Si se prefiere se puede tomar un mapa de cualquier otro lugar, en este caso, para el ejemplo, se ha tomado la imagen a continuación. Sobre esta imagen, se va a un explorador geográfico, Google Maps es suficiente. En este explorador se identifica la ubicación de la esquina superior izquierda y la inferior derecha, por sus coordenadas geográficas, es importante tomar como mínimo 5 cifras decimales para que la precisión sea aceptable, y son recomendables 6 cifras para una mejor calidad de la ubicación.
  - Una vez conocida la ubicación, se rellena el archivo de configuración map.conf con la arquitectura ya comentada. Primero el nombre de la imagen que contiene el mapa base, y el valor de los límites. Tanto la imagen, como el fichero de configuración deben estar en el mismo directorio que el ejectuable de la apliación.

- Con la configuración previa realizada, se abre la apliación. Lo primero que aparecerá será una terminal en el que (si se ha establecido contraseña) se preguntará el password de un usuario root de Linux, se introduce y se cierra el terminal.

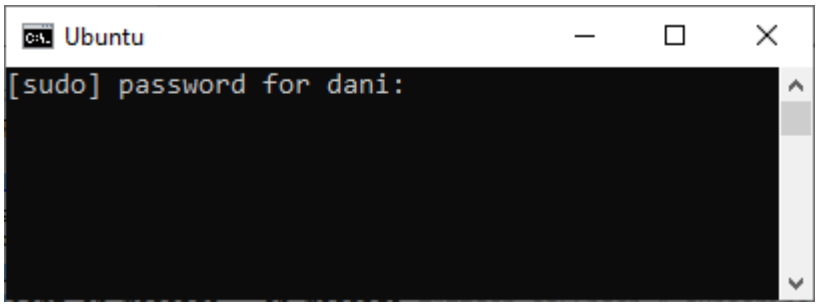


Ilustración 6-2, Terminal para lanzar mosquitto

- Si toda la configuración ha ido correctamente, aparecerá la pantalla principal. En caso de que alguna de las configuraciones previas haya sido defectuosa, se muestra el mensaje de advertencia pop-up correspondiente, y aparecerá el botón de CONFIGURAR habilitado y en color rojo, bloqueando el resto de funcionalidades hasta que la configuración no sea satisfactoria. Haciendo click sobre este botón, se vuelve a intentar configurar toda la aplicación, con lo que si se han solucionado los errores se eliminará el botón, y si el error persiste, o hay uno nuevo, saltará la advertencia correspondiente.
- Una vez se ha conseguido configurar de manera apropiada el dispositivo tan sólo queda espera a la recepción de datos por parte de los deportistas. Unicamente los deportistas que incurran en una infracción de la distancia de drafting aparecerán de manera visual en la aplicación, ya que los que no hayan acumulado ninguna infracción, serán automáticamente almacenados sin necesidad de interacción del técnico verificador.
- Cuando se recibe un caso de deportista infractor aparecerá en el desplegable para ser seleccionado, en caso de ser el primer deportista, será automáticamente seleccionado para su verficación. Con la selección de un deportista, el mapa base que aparece por defecto se sustituye por un mapa custom de deportista, y en el label superior, se mostrará el identificador del deportista seleccionado.

Ilustración 6-4, Inicio con configuración errónea de mapa base

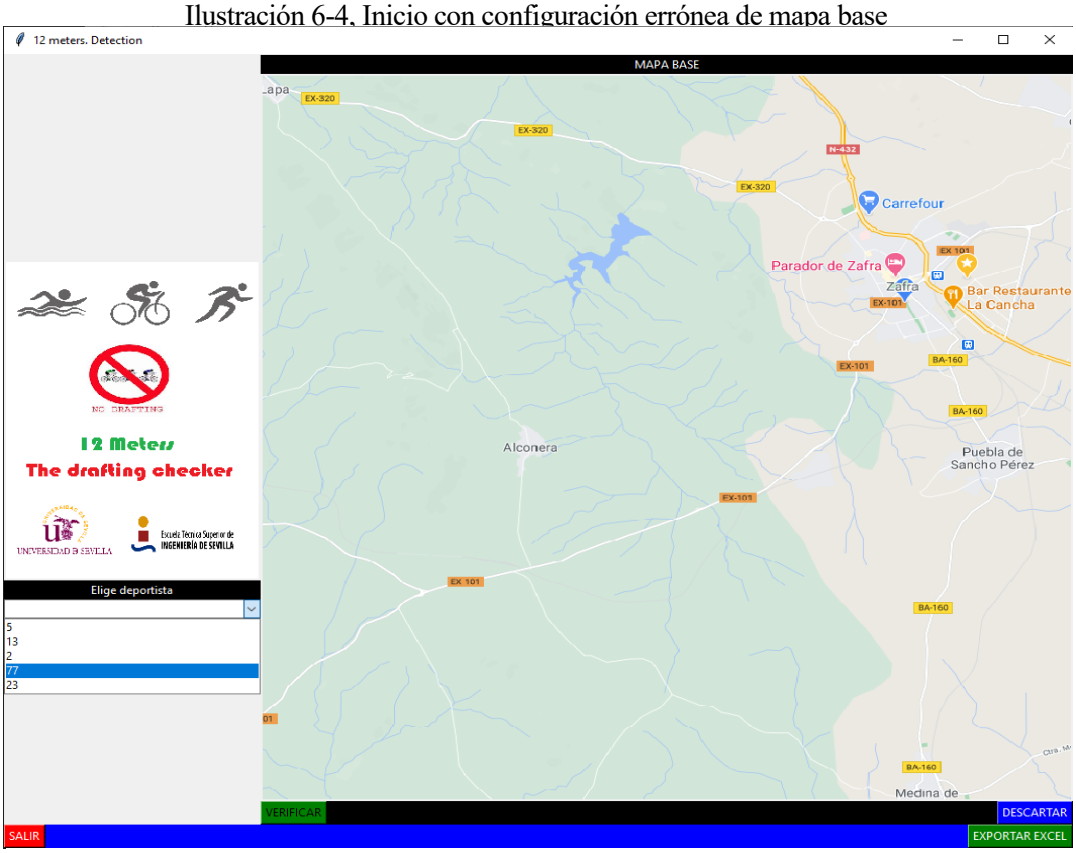


Ilustración 6-3, Selección de deportista

- Una vez se ha seleccionado un deportista, se puede clickar sobre la opción VERIFICAR, con lo que se almacena el positivo en la hoja de datos de salida, o la opción DESCARTA, con la que descartamos la información del positivo, aunque el mapa de las áreas de infraccion seguirá guardado en local. Este es el ciclo de funcionamiento en la segunda transición.

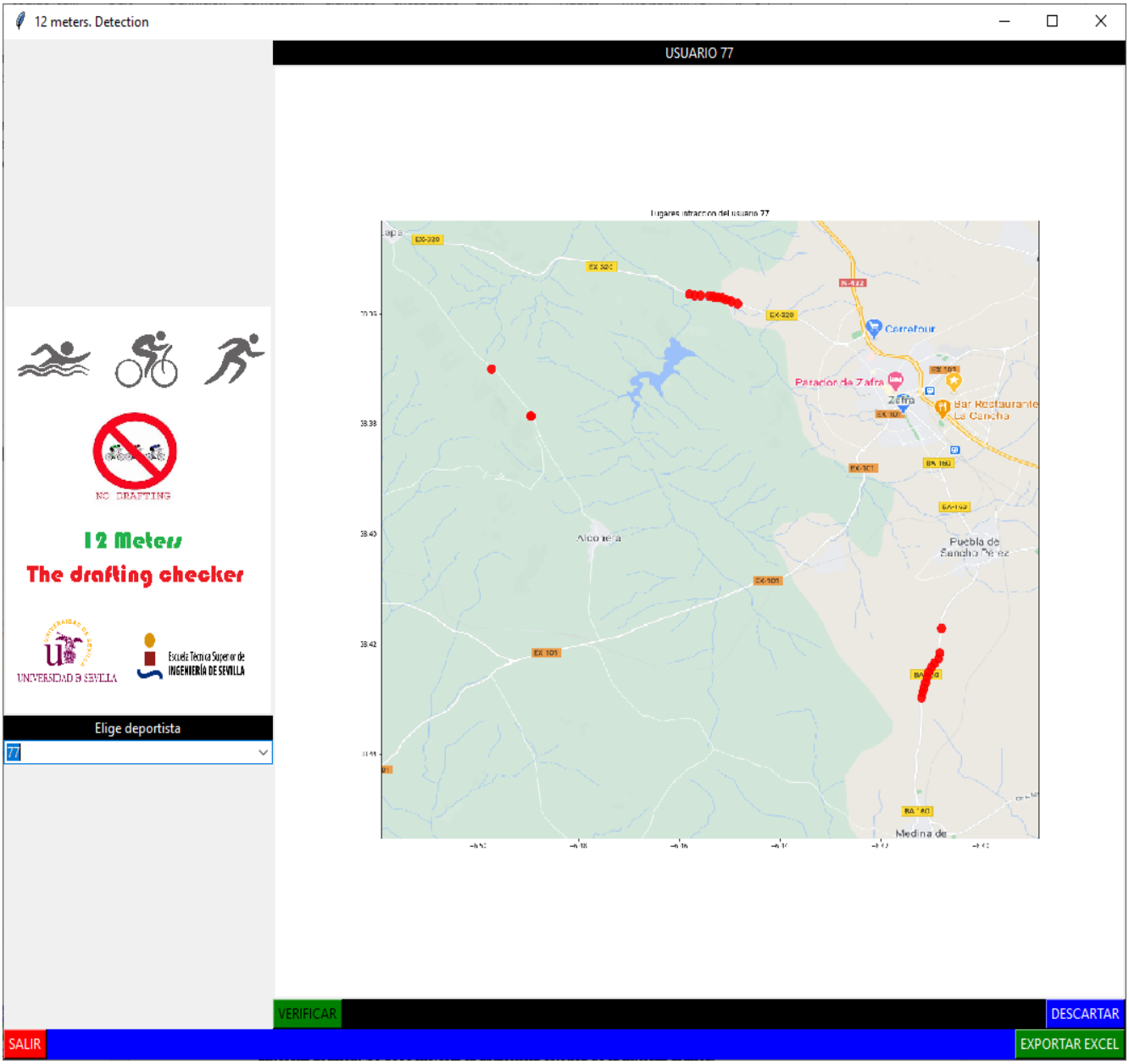


Ilustración 6-5, Mapa custom con infracciones

- Una vez han llegado todos los deportistas al área de transición, y se ha recepcionado su información, se puede salir de la aplicación. Para ello se puede pulsar tanto la el botón salir en la esquina inferior izquierda, como la cruz clásica en la esquina superior derecha, ambos llaman al mismo método, y no permiten cerrar la aplicación hasta que no se ha almacenado la información en la hoja de datos de salida, para así evitar la pérdida indeseada de información.

A modo de referencia, este sería el resultado de la hoja de datos de salida para el caso de las imágenes anteriores. Com se puede ver los dorsales 23 y 77 han sido descartados de la infracción, y los deportistas 2, 13, 5 han sido verificados. Además los tres últmos deportistas, 8, 99, 54, no han cometido ninguna infracción.

Dorsal	Nombre	Categoría	Tlf.Contacto	Infractor	Places	DropTime	CheckTime
77	Daniel Seco	Elite	655900026	NO		Wed Sep 2 19:53:31 2020	
23	Manolo Sanchez	Vet1	687128963	NO		Wed Sep 2 19:53:46 2020	
2	Maria Martinez	Sub23	641287385	SI	#38.440286#-6.4	Wed Sep 2 19:53:28 2020	Wed Sep 2 19:57:31 2020
13	Juan Martin	Junior	648752169	SI	#38.440286#-6.4	Wed Sep 2 19:53:25 2020	Wed Sep 2 19:57:36 2020
5	Joana Roig	Vet2	698751389	SI	#38.440286#-6.4	Wed Sep 2 19:53:21 2020	Wed Sep 2 19:57:29 2020
8	Paco Salinas	Vet1	658315987	NO		Wed Sep 2 19:53:38 2020	
99	Jorge Saras	Elite	645873985	NO		Wed Sep 2 19:54:51 2020	
54	Marina Salas	Cad	648257896	NO		Wed Sep 2 19:52:58 2020	

Ilustración 6-6, Hoja de datos de salida

6.2 Servicio usuario

En cuanto a la calidad de la detección se han realizado pruebas con lo que se ha corroborado una precisión bastante buena a distancias de hasta unos 10 metros. Una vez en movimiento, el ángulo tan cerrado del sensor no ha resultado inestable en situaciones de drafting común, en el que el deportista se mantiene a rueda justo detrás del que va delante. La principal fragilidad de esta arquitectura es en situaciones de viento lateral, en la cuales el sensor no detecta la distancia con el otro participante ya que en esta situación se suele rodar por el costado del ciclista por el que no entra el viento, con lo que delante no existe ningún objeto que detectar. En traitlón de larga distancia esta situación es bastante poco común, ya que para que haga efecto en necesario ir muy pegado, y esto si es mucho más 'llamativo' de cara a la infracción.

Para la realización de las pruebas se ha diseñado una pequeña carcasa, que se ha imprimido con una impresora 3D y fijado al manillar de la bicicleta con una brida. Su peso es de 22g y las dimensiones de 32x61x56mm. La batería es externa a la carcasa y se fijado a la tija de sillín y conectado al dispositivo por un cable USB. Esta carcasa, aunque funcional, y sencilla de colocar, es muy poco aerodinámica, y permite que con baches e irregularidades del terreno vaya girando entorno al manillar poco a poco, con lo que puede acabar apuntando hacia abajo con lo que el sensor deja de detectar los objetos de delante y empieza a detectar el suelo, o nada.

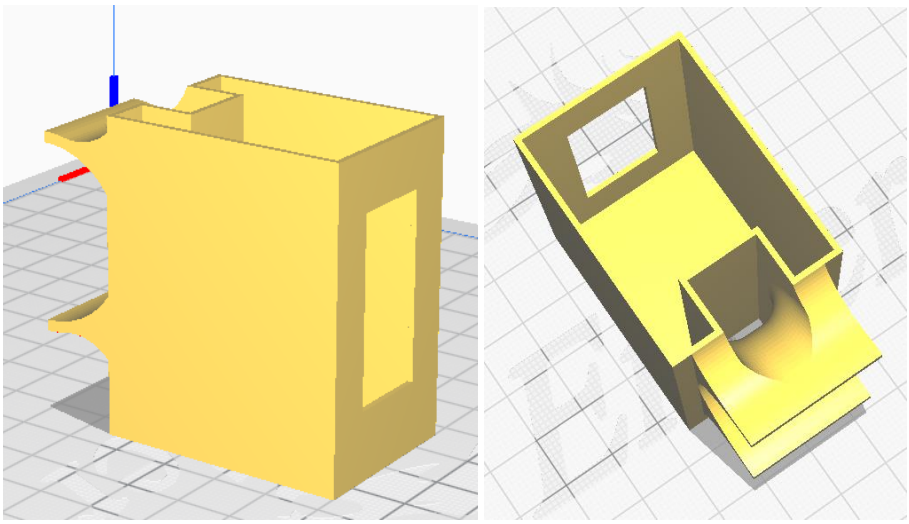


Ilustración 6-7, Modelo de la caja para pruebas

El funcionamiento del GPS ha sido muy preciso, y con la pantalla se ha ido verificando que detecta las zonas seguras y el área de transición. En algunas situaciones, sobre todo en días nublado y calles muy estrechas, la conexión es débil e inestable

.

Ilustración 6-8, Caja de pruebas montada

Durante las pruebas en carretera, el dispositivo se ha mojado por accidente, habiéndose estropeado el sensor de distancia. Esto, pese a no haber afectado al pryecto ya que ocurrió al final de este, hace ver que es necesarios que la caja sea resistente a salpicaduras algo contundentes.

Incluso en días soleados esta especificación es importante, ya que, tal y como ha ocurrido en este caso, la zona del manillar está expuesta al sudor del deporstista, y al vertido de líquido al beber del bidón. Sobre todo el vertido del bidón es algo que ocurre con cierta frecuencia, y puede llegar a ser mucho más contundente que el efecto del sudor.

7 CONCLUSIONES

Como resumen el se puede concluir con un resultado en cuanto a la solución del proyecto, en la que se han conseguido cumplir todos los objetivos marcados como de alta prioridad (MUST) en las especificaciones previas, y con gran parte del resto de los objetivos. Se puede comprobar el resultado de la hoja de objetivos en la imagen a continuación.

7.1 Industrialización

En este apartado se va a comentar las necesidades que requeriría una posible industrialización del producto, no necesariamente como producto comercial final, pero si como prototipo comercial, en una serie de puntos y pautas generales, sin realizar el desarrollo específico del contenido:

- **Montaje:** Se fija la necesidad de realizar un montaje integrado en una placa tipo PCB o directamente circuito integrado, de manera que no se utilicen cables para la unión de los componentes. Además, este tipo de montaje facilita la integración en menor tamaño. Estudiar la utilización de componentes sin encapsulado para el uso individual, integrar directamente los chips que forman los componentes en la fabricación electrónica.
- **Encapsulado:** Se diseña un encapsulado ligero, con posibilidad de una certificación tipo IP61 como mínimo, a prueba de polvo y de agua (columnas débiles de agua), como primer objetivo
- **Batería:** Se incluye la batería en el encapsulado del dispositivo, y se estudia la integración de una batería interna en el manillar o comunicación con la batería de los sistemas de cambio electrónico Shimano Di2
- **Interfaz gráfica:** Se debe mejorar la apariencia estética de la interfaz gráfica.
- **Instalación:** La instalación del sistema debe ser más automatizable, o menos dependiente del sistema operativo y entrono de trabajo.
- **Visión:** Se estudiará la viabilidad de integrar una microcámara para tomar imágenes en caso de infracciones, de cara a documentar y afianzar la viabilidad de la sanción a deportistas.
- **Información en directo:** Se debe desarrollar algún tipo de actualización de infracciones durante el transucurso de la prueba, ya sea con un checkpoint o con una comunicación con la moto de carrera via Wfi, NFC, BLE …
- **Robustez:** Se certificará su robustez a lo largo del tiempo, y a lo largo de su uso.
- **Licencias:** Se debe comprobar las viablilidad de usar licencias gratuitas para un uso industrial, y en su defecto, la compra de ellas.
- **Configuración de mapa:** Se buscará una manera más simple y automatizable de configurar tanto el mapa base, como las áreas de conexión y de no detección. A ser posible estas áreas no deberán tener, de manera obligatoria, forma rectangular.

7.2 Presupuesto

En cuanto a este presupuesto, para el proyecto se ha invertido un total de 55 €, ya que algunos elementos han sido proporcionados por la universidad, como el Buzzer y el ESP.

Dispositivo	Precio (€)	Notas
ESP32	**9**	Para una unidad (Europea)
TF Mini Lidar	**33.75**	A partir de 2 uds (30€)
Buzzer	**1**	Precio aprox
Led	**0.20**	Precio aprox
GPS	**13**	Una unidad (Europea)
Pantalla	**12**	Solo en dispositivo desarrollo
Caja	**0.50**	Coste del plástico (PETG)
TOTAL	**68.95**	

Tabla 7-2, Presupuesto de proyecto

De cara a un presupuesto, aproximado, para una industrialización en primera fase, que es lo que realmente interesa de cara a verificar el cumplimiento o no de objetivo, habría que tener en cuenta que la pantalla no estaría incluida en el presupuesto, y que, tanto el micro, como el sensor de distancia y el GPS bajan mucho de precios haciendo pedidos de varias unidades (con pedir 5 unidades suele bajar entorno a un 20%). Además, todos los elementos electrónicos, a excepción del sensor de distancia que proviene de EE.UU. puede buscarse directamente paginas web de Asia, o en sus proveedores, donde su precio es bastante más bajo. Además, para un montaje industrial se ha comentado que habría que ser realizaría sobre una PCB, hoy en día hay multitud de proveedores en China que trabajan con pequeños pedidos, con una latencia alta en la producción, y buena calidad. Se usa esta estimación para el presupuesto. Uniendo ambas carácteríasticas se estima el siguiente presupuesto:

Dispositivo	Precio (€)	Notas
ESP32	**2**	Pack 5 – (Aliexpress)
TF Mini Lidar	**27.50**	
Buzzer	**0.50**	Precio aprox
Led	**0.10**	Precio aprox
GPS	**4.50**	(Aliexpress)
Caja	**0.50**	Coste del plástico (PETG)
PCB	**1**	Proveedor Chino
TOTAL	**36.1**	

Tabla 7-1, Presupuesto para industrialización

Un presupuesto algo inferior a los 40€, para una pequeña producción, sin contar con mano de obra. Entra dentro de los límites aceptables. Aunque debería de ser más ajustado para una producción en masa.

Requerimientos				Prioridad	Comprobación	
#ID	Elemento	Requerimiento	Descripción	Tipo	Check	Comentarios
#1	Detector	Detección de distancia drafting	Reconocer, en caso de estar incumpliendo la d	MUST		Sensor óptico
#2	Ambos	Comunicación de resultados	Comunicar con la aplicación de la organización	MUST		WiFi+Mosquitto
#3	Detector	Feedback deportista	Informar al deportista acerca de cuando está ir	MUST		Led indicador e alarma sonora
#4	Ambos	Información en directo	Informar a la organización sobre si ha incumpli	COULD		
#5	Detector	Peso liviano	El peso del conjunto el dispositivo no superará	SHOULD		Sin incluir batéría se cumple
#6	Detector	Aerodinámica	Integración en bicleta tipo contrarreloj con im	SHOULD		Configurable
#7	Detector	Anclaje	Se podrá poner/quitar de la bicicleta de manei	COULD		Colocación con bridas.
#8	Aplicación	Interpretación de datos	Los datos de los usuarios se interpretarán de n	MUST		Aplicación organización
#9	Aplicación	Configuración de dispositivos	Cconfigurar el dispositivo de forma que no sea	SHOULD		Configuración mediante ficheros de configuración
#10	Detector	Detección drafting viento lateral	Reconocer drafting bajo condiciones de viento	SHOULD		El sensor óptico no detecta este tipo de situaciones
#11	Detector	Batería	La duración de la batería del dispositivo será c	SHOULD		Consumo calculado de 2W/h
#12	Detector	Costo	El costo del dispositivo no debe superar los 50	SHOULD		Se calcula un costo de 40€ aprox.
#13	Aplicación	Configuración servidor	La configuración del servidorno requerirá de c	SHOULD		Uso simple pero tras instalación de utilidades
#14	Aplicación	Guardado de datos	Los datos de los usuarios quedarán guardados	MUST		Hoja de Excel
#15	Detector	Detección de adelantamientos	No tomar adelantamientos como infracciones.	SHOULD		Tiempo máximo de adelantamiento configurado.
#16	Detector	Zonas de drafting permitido	Reconocer las zonas en las que el drafting esté	SHOULD		Configurable por hoja de coordenadas
#17	Detector	Zona transición	No reconocer infracciones en zona de transicio	MUST		Configurable por hoja de coordenadas
#18	Aplicación	Configuración detector simple	La organización dispondrá de alguna herramie	COULD		Script loadCongif.sh
#19	Detector	Aviso encendido	El dispositivo informará de si está encendido y	SHOULD		Uso de la pantalla en prototipo
#20	Ambos	Seguridad	Método de seguridad para evitar que un usuar	SHOULD		Limitación de direcciones MAC en router
#21	Aplicación	Comprobación a posteriori	La aplicación podrá interpretar los datos de un	COULD		Imagen guardada en local
#22	Ambos	Codigo	El código fuente debe estar encriptado o comp	SHOULD		Ejecutable en aplicación organización
#23	Detector	Cambio de sentido	El detector pordrá interpretar en que sentido s	SHOULD		

Tabla 7-3, Hoja de objetivos cumplidos

Anexo 1. Instalación y problemas

Instalación de Windows subsystem for Linux

Para la instalación del terminal de Ubuntu para Windows 10, se debe tener una versión de este sistema operativo posterior a 2016, lo cual es casi una necesidad debido al asistente Windows Update que fuerza las actualizaciones de Windows. En concreto este proyecto se refencia con una versión de Windows 1903.

1. Se abre el gestor de características de Windows, en caso de trabajar con el sistema operativo en inglés buscaremos "*Windows Features*".

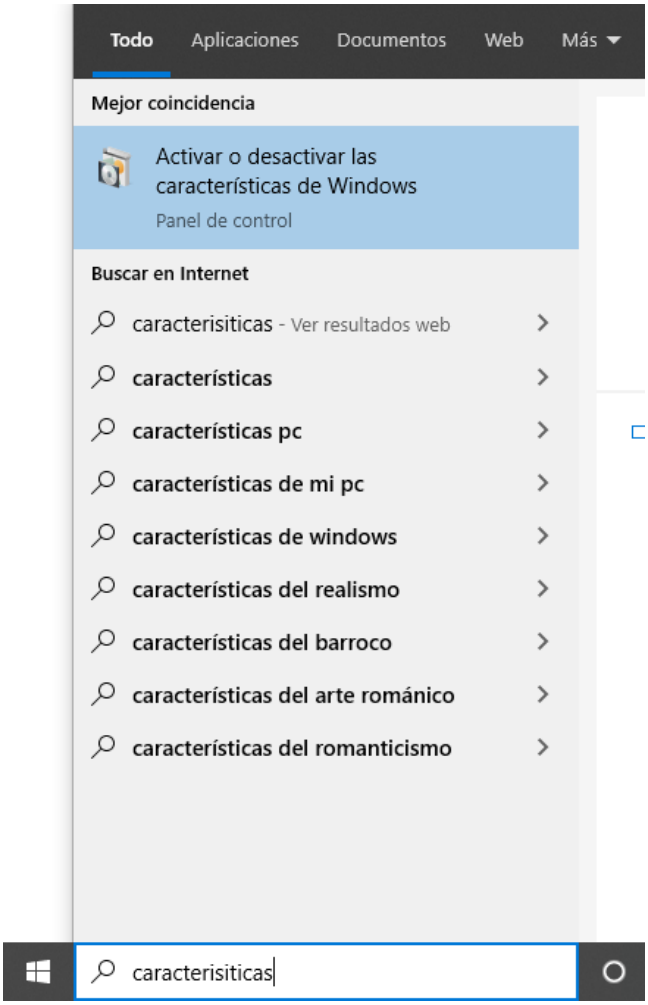


Ilustración 7-1, Abrir características de Windows

2. En esta ventana, se activa la opción del subsitema de Windows para Linux. En inglés se puede buscar por "*Windows subsystem for Linux*".

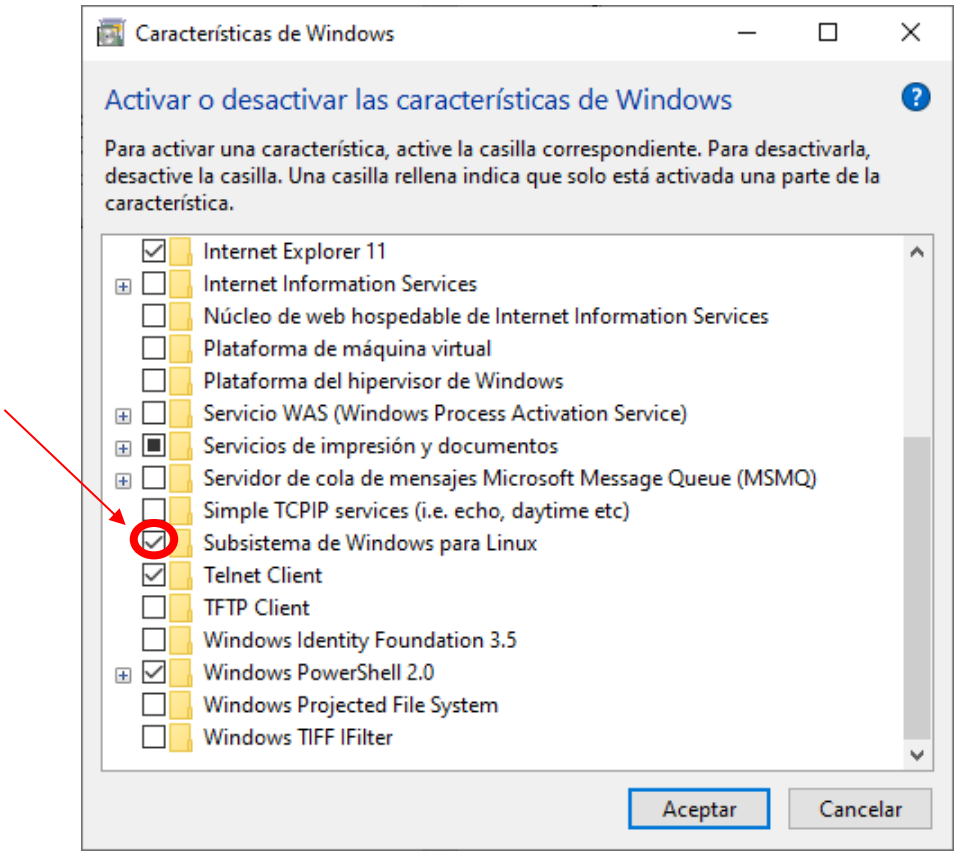


Ilustración 7-2, Habilitar característica terminal Linux

3. Una vez activada la característica se descarga de la Microsoft Store la aplicación *UBUNTU*. Tras instalarla, al lanzar esta aplicación aparecerá un terminal con arquitectura Linux. Habrá que crear un un nombre de usuario y password.

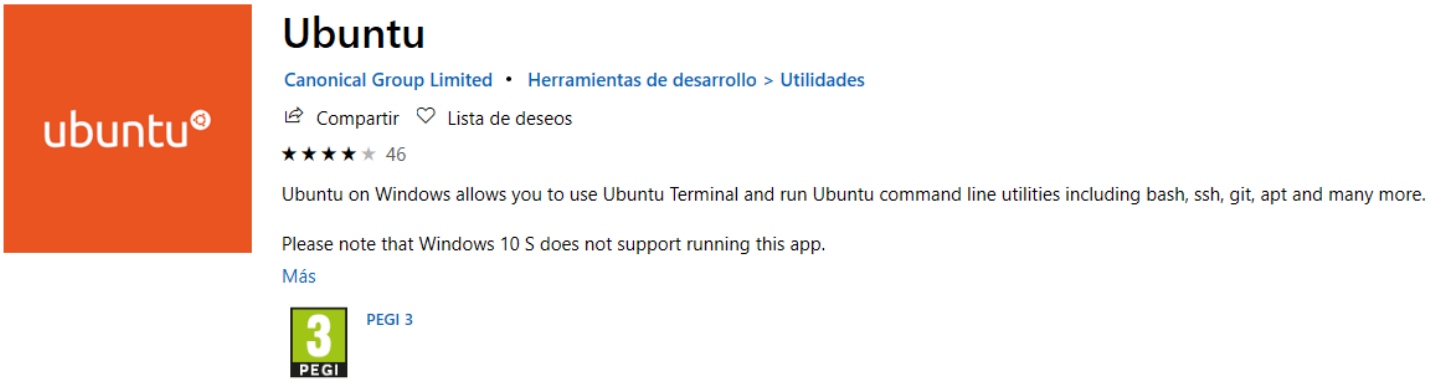


Ilustración 7-3, Terminal Ubuntu en Microsoft Store

4. Para acceder a los archivos que se encuentran en nuestro sistema de archivos de Windows, hay que

escribir la ruta "*/mnt/c/"*, si la unidad en la que se encuentran estos archivos es la por defecto "C:\", o la letra necesaria en otro caso. Se recomienza modificar el archivo "*.profile"* ubicado en "/home/*username*/.profile", siendo username el nombre de usuario elegido, que se ejecuta cada vez que el usuario abre un terminal. En este archivo es recomendable exportar variable globlales para el acceso a las carpetas más comunes, como mínmo, ya que no resulta nada intuitivo compaginar el sitema de ubicación de archivos Windows con el de Linux. Como referencia se adjunta las modificaciones añadidas al archivo ".profile" usado en este proyecto. Importante tener en cuenta que el terminal instalado no cuenta con soporte gráfico, por lo que para modificar archivos en un IDE gráfico externo (el terminal Linux proporciona la utilidad *vi* y *nano,* que si no se está habituado pueden resultar problemáticas) habrá que conocer su ruta de acceso desde Windows Explorer.

```
export WP=/mnt/c/
export TFG=/mnt/c/GitHub/TFG-TriTruth/
cd $TFG
```

Ilustración 7-4, Ejemplo de fichero *.profile* de Linux

Problemas al lanzar servicio mosquitto

Tras apagados forzosos del PC sobre el que se ejecuta el broker bajo la arquitectura de Linux en Windows, puede quedar un proceso *zombie* que bloquea el puerto 5040, que es el que usa por defecto el broker mosquitto.

Esto ocurre cuando, pese a lanzar el broker, al comprobar el estado no está funcionando correctamente. La comprobación del estado se realiza ejecutando el comando:

➔ ***sudo service mosquitto status***

y si devuelve:

➔ **** mosquitto is not running***

Se habrá quedado un proceso *zombie* bloqueando el puerto.

Para liberarlo se abre un CMD de Windows como administrador, y en él se ejecuta el comando

➔ ***netstat -abo***

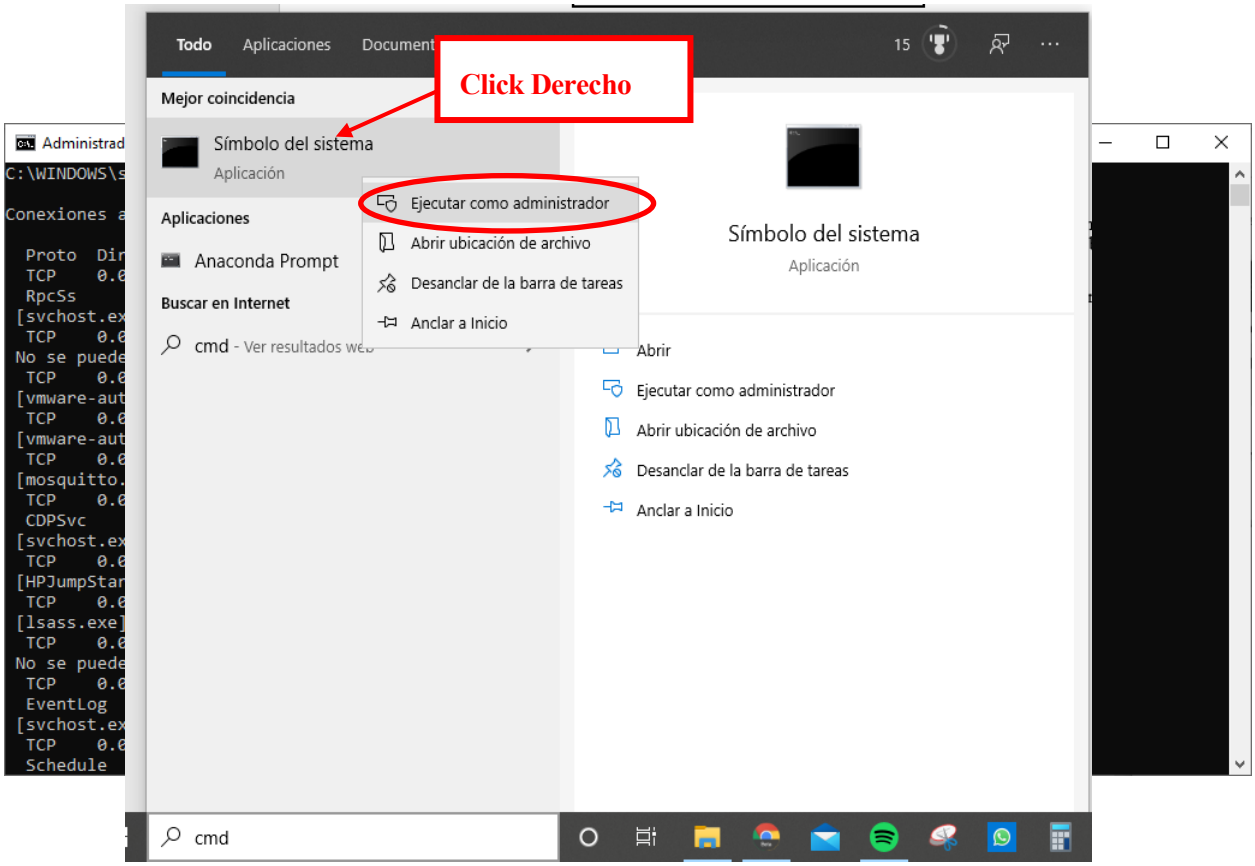


Ilustración 7-5, Abrir CMD Windows como administrador

Ahora se busca el puerto 5040, que aparecerá algo similar a lo de la imagen, buscamos el PID del proceso que lo bloquea, que será el proceso *zombie*, y se ejecuta el siguiente comando:

➔ ***taskkill /F /PID 6972***

donde el número 6972 es el PID del proceso *zombie*.

Una vez realizado esto se puede relanzar sin problemas el servicio mosquitto en Linux.

Anexo 2. Código

read.py

```
from machine import UART, I2C, Pin, PWM
import time
import ssd1306
import gps
import _thread
import sendInfo

currGPS = 'PASS'

def actGPS():
	global currGPS

	for current in gps.run():
		currGPS = current

def userAdvice(seconds):
	myPwm = PWM(Pin(21))
	myPwm.duty(512)
	start = time.time()
	i=0
	while(seconds > time.time()-start):
		if i==0:
			myPwm.freq(500)
			i=1
		else:
			myPwm.freq(2000)
			i=0
		time.sleep(0.01)
	myPwm.deinit()

def setLidar():
	#Configuro la UART
	try:
		Lidar = UART(2, baudrate=115200)
		Lidar.init(baudrate=115200, bits=8, parity=None, tx=14, rx=32)
		return [True, Lidar]
	except Exception as errCod:
		return [False, errCod]


def getLidar(Lidar):
	Lectura = bytearray(3)
	uno=bytearray([0])
	#Vacío lecturas
	Lidar.read()
	#Busco byte inicio
	uno = Lidar.read(1)
	while uno != b'Y':
		uno=Lidar.read(1)
```

```
        #Leo los 3 bytes siguientes
        Lectura=Lidar.read(3)
        if Lectura[0]==89:#Si lectura Ok
                #Convierto a metros
                Distancia=Lectura[1] + Lectura[2]*256
                return Distancia
        else:
                return 'Error'

def setScreen():
        try:
                i2c = I2C(scl=Pin(22),sda=Pin(23), freq=100000)
                oled = ssd1306.SSD1306_I2C(width=128, height=32, i2c=i2c)
                oled.init_display()
                oled.poweron()
                oled.fill(0)
                oled.show()
                return [True, oled]
        except Exception as errCod:
                return [False, errCod]

def dispMsg(screen, msg, msg2=None, msg3=None):
        screen.fill(0)
        screen.text(msg, 0, 0)
        if msg2 != None:
                screen.text(msg2,0,10)
        if msg3 != None:
                screen.text(msg3,0,20)
        screen.show()

def run():
        currCheat = 0
        currFree = 0
        totalFree = 0
        totalCheat = 0
        totAvisos = 0
        gpsThread = _thread.start_new_thread(actGPS, ())
        sensor = setLidar()
        screen = setScreen()
        led = Pin(33, Pin.OUT)
        led.value(0)
        cheatInfo =''
        if (sensor[0] == True):
                while(currGPS != 'END'):
                        try:
                                distance = getLidar(sensor[1])
                                if distance != 'Error':

                                        if currCheat == 0:
                                                dispMsg(screen[1], "Dist: "+str(distance),
"Pos: "+str(currGPS), "Guardados: "+str(totAvisos))
                                        else:
                                                dispMsg(screen[1], "Dist: "+str(distance),
"Tot: "+str(totalCheat), " Libre: "+str(totalFree))

                                        if distance < 800 : # Si a menor de la distancia
                                                if currFree != 0:#Limpio el tiempo de
liberación y lo acumulo si habia
                                                        totalFree += time.time()-currFree
                                                        currFree = 0
                                                        led.value(1)
```

```
                                    if currGPS != 'PASS' and currCheat == 0: #
Si estoy en zona prohibida x primera vez
                                          currCheat = time.time() #Arranco el
aviso
                                          lon, lat, tiemp = currGPS.split('#')
#Guardo donde comienzo
                                          led.value(1)

                              elif distance > 800 and currCheat != 0: #Si
salgo de zona prohibida en un aviso
                                    if currFree == 0:
                                          currFree = time.time()
                                          led.value(0)

                              if currCheat != 0: #Si estoy bajo un aviso
                                    totalCheat = time.time()-currCheat
                                    if totalCheat > 10 :#Si llevo + de 10" en
aviso
                                          led.value(0)
                                          if currFree != 0:#Limpio el tiempo
de liberación y lo acumulo si habia
                                                totalFree += time.time()-
currFree
                                                currFree = 0
                                          if totalFree < 5: # Si he estado mas
de la mitad del tiempo infringiendo

      _thread.start_new_thread(userAdvice, [3])
                                                cheatInfo += '
#{}#{}'.format(lon, lat) #Guardo el positivo
                                                totAvisos += 1
                                          totalCheat = 0 #Haya sido una
infraccion o not
                                          totalFree = 0  #reinicio todos los
contadores
                                          currCheat = 0
                              else:
                                    time.sleep(1)
                        print(totalCheat, totalFree, distance, totAvisos)
                  except Exception as e:
                        print(e)
            gpsThread.exit()
            dispMsg(screen[1],"FIN","DETECCION")
            checked = 'Unchecked'
            while (checked == 'Unchecked'):
                  try:
                        checked = sendInfo.run(cheatInfo)
                  except:
                        pass
      else:
            print('Whoops :( something went wrong')
```

gps.py

```
from machine import UART, Pin, deepsleep
from time import sleep

def loadSafeZone():
      safeZone = []
```

```
	connZone = []
	try:
		with open('safeZone.conf','r') as zoneFile:
			for line in zoneFile:
				if line[0] != '#':
					safeZone.append(line.split('#'))
	except:
		pass
	try:
		with open('connZone.conf','r') as connFile:
			for line in connFile:
				if line[0] != '#':
					connZone = line.split('#')
	except:
		pass
	return [safeZone, connZone]

def run():
	gps = UART(1)
	gps.init(baudrate=9600, bits=8, tx=17, rx=16)
	safeZone, connZone = loadSafeZone()
	#---------------------------------
	createdFile = False
	timeStamp = 11
	#---------------------------------
	while(True):
		try:
			inSafe = False
			line = str(gps.readline()).split(',')
			if '$GPGLL' in line[0]:
				if len(line)>5 and len(line[1]) > 9 and len(line[3]) >
10: #Si tengo señal
					dgLat = int(line[1][0] + line[1][1])
		#Calculo latitud
					minLat = float(line[1][2] + line[1][3] +
line[1][4] + line[1][5] + line[1][6] + line[1][7] + line[1][8] + line[1][9] )
					latitude = dgLat+minLat/60
					if line[2] == 'S':
						latitude = -latitude

					dgLon = int(line[3][0] + line[3][1] +
line[3][2]) #Calculo longitud
					minLon = float(line[3][3] + line[3][4] +
line[3][5] + line[3][6] + line[3][7] + line[3][8] + line[3][9] + line[3][10])
					longitude = dgLon+minLon/60
					if line[4] == 'W':
						longitude = -longitude

					time = line[5]
					for zone in safeZone: #Si tengo q detectar
						if (float(zone[0]) < latitude <
float(zone[2])) and (float(zone[1]) < longitude < float(zone[3])):
							yield('PASS')
							inSafe = True
					if  len(connZone)==5 and (float(connZone[1]) <
latitude < float(connZone[3])) and (float(connZone[2]) < longitude <
float(connZone[4])):
						if (time > float(connZone[0])):
							yield('END')
							debugFile.write("#{}#{}#{}
".format(longitude, latitude, 'END'))
							debugFile.close()
```

```
                                    break
                              else:
                                    yield('PASS')
                                    continue

                        if inSafe == False:
                              yield('{}#{}#{}'.format(longitude,
latitude, time))
                        else:
                              sleep(10) #Si estoy en zona segura, duermo
10s
                        #---------------------------------
                        if not createdFile:
                              debugFile = open('infoFile'+time, 'w')
                              createdFile = True
                        if timeStamp > 10:
                              timeStamp = 0
                              debugFile.write("#{}#{}#{}
".format(longitude, latitude, str(inSafe)))
                        #---------------------------------
                  else:
                        print("NO connection")
                        yield('PASS')

      except Exception as e:
            print(e)
            pass
```

sendInfo.py

```
from umqttsimple import MQTTClient
import ubinascii
import machine
import micropython
import network
import time
import esp
esp.osdebug(None)
import gc
gc.collect()
client_id = ubinascii.hexlify(machine.unique_id())
topics = ['checked']

#Previous global config load
with open('wireless.conf','r') as configFile:
      lineNumber=0                  #As in python, line beginning with '#' is
a comment very
      for line in configFile: #important adding the symbol as first character
and
            if line[0] != '#':      #all comments at the top of file whithout
empty lines
                  if lineNumber==0:
                        ssid = line.replace('\n','')
                        lineNumber+=1
                  elif lineNumber==1:
                        password=line.replace('\n','')
                        lineNumber+=1
                  elif lineNumber ==2:
                        mqtt_server=line.replace('\n','')
```

```
with open('user.conf','r') as idFile:
      id = idFile.read()

#Wireless connection
def connectWifi():
      global ssid, password
      wifi = network.WLAN(network.STA_IF)
      wifi.active(True)
      wifi.connect(ssid,password)
      waitTime=30
      startTime = time.time()
      while wifi.isconnected()==False and waitTime > time.time()-startTime:
            pass
      if wifi.isconnected()==False:
            return 'Error'
      else:
            return wifi

#Callback for topic subscribing
def sub_cb(topic,msg):
      if msg == id:
            #open('cheat.log','w').close()#Limpio el archivo
            machine.deepsleep()

#MQTT connection and topics subscribing
def connectMQTT():
      global client_id, mqtt_server, topics
      client = MQTTClient(client_id, mqtt_server)
      client.set_callback(sub_cb)
      try:
            client.connect()
      except:
            return 'Error'
      for topic in topics:
            client.subscribe(topic)
      return client

def publishContent(msgState, msgPlaces):
      mqtt.publish('state',msgState)
      time.sleep(0.5)
      if msgPlaces != '':
            mqtt.publish('places',msgPlaces)
      startTime = time.time()
      waitTime = 300 #5 minutos para esperar una respuesta de checked.
      while(waitTime > time.time()-startTime):
            mqtt.check_msg()

def run(cheatInfo):
      wifi = connectWifi()
      if wifi == 'Error':
            raise Exception('Wifi Error')
      mqtt = connectMQTT()
      if mqtt == 'Error':
            raise Exception('mqtt Error')

      if cheatInfo == '':
            msgState = '{} NO'.format(id)
            msgPlaces = ''
      else:
            msgState = '{} SI'.format(id)
            msgPlaces = '{} {}'.format(id, cheatInfo)
```

```
        while wifi.isconnected():
              publishContent(msgState,msgPlaces)
        return('Unchecked')
```

main.py

```
# -*- coding: utf-8 -*-
"""
Created on Mon Apr  6 14:25:30 2020

@author: Hp
"""

import paho.mqtt.client as mqtt
import socket
import pandas as pd
import matplotlib.pyplot as plt
from tkinter import *
import time
from tkinter import ttk
from PIL import ImageTk, Image
import tkinter.messagebox as warning
import os

#Callback para gestión de mensajes
def on_message(myClient, userdata, message):
    global client
    msg = str(message.payload.decode("utf-8")).split(' ')
    if message.topic == 'state':
        if len(msg) == 2:
            info.loc[int(msg[0]),'Infractor'] = msg[1]
            if msg[1] == 'NO':#Si no es un infractor
                client.publish('checked',msg[0])
                info.loc[int(msg[0]),'DropTime'] = time.asctime()
    if message.topic == 'places':
        msLen = len(msg)
        i = 1
        while i < msLen:
            info.loc[int(msg[0]), 'Places'] += (msg[i] + ' ')
            i+=1
        if info.loc[int(msg[0]),'Infractor'] == '':
            info.loc[int(msg[0]),'Infractor'] = "SI"
        client.publish('checked',msg[0])
        app.newCheater(msg[0])
        info.loc[int(msg[0]),'DropTime'] = time.asctime()



class Application(Frame):
    def save_toEXCEL(self):
        if self.configured == True:
            try:
                writer = pd.ExcelWriter('TriTruthOutput.xlsx',
engine='xlsxwriter')
                info.to_excel(writer, sheet_name='Results')
                writer.save()
                return ('Ok')
            except Exception:
```

```
                warning.showwarning(title='Error guardado de info',
message='Error al guardar el excel de datos')
                return ('Error')
        else:
            return ('Ok')

    def verificar(self):
        dorsal = self.COMBO.get()
        if dorsal != '':
            values = list(self.COMBO["values"])
            values.remove(dorsal)
            self.COMBO["values"] = values
            self.COMBO.set('')
            info.loc[int(dorsal),'CheckTime'] = time.asctime()
        self.MAP["image"] = self.imBase
        self.IMAGEN['text'] = "MAPA BASE"

    def configurar(self):
        try:
            connMqtt()
            getData()
            getMapConfig()
            self.imBase =
ImageTk.PhotoImage(Image.open(mapName).resize((800,800)))
            self.MAP["image"] = self.imBase
            self.configured = True
            self.reconn.destroy()
            self.verify['state'] = 'normal'
            self.discard['state'] = 'normal'
            self.EXCEL['state'] = 'normal'
            self.COMBO['state'] = 'normal'
        except Exception as e:
            if e.args[0] == 'mqtt error':
                warning.showwarning(title='Error de mensajería',
message='Error al conectar con el broker MQTT')
            elif e.args[0] == 'data error':
                warning.showwarning(title='Error de informacion',
message='Error en el archivo de datos excel InfoTriathletes.xlsx')
            elif e.args[0] == 'map error':
                 warning.showwarning(title='Error de archivo', message='Error
en el archivo de configuración map.conf')
            elif e.args[1] == 'No such file or directory':
                warning.showwarning(title='Error de archivo', message='Error
en el nombre del archivo del mapa en map.conf')
            else:
                warning.showwarning(title='Error', message='Error desconocido
\n {}'.format(e.args))
    def descartar(self):
        dorsal = self.COMBO.get()
        if dorsal != '':
            values = list(self.COMBO["values"])
            values.remove(dorsal)
            self.COMBO["values"] = values
            self.COMBO.set('')
            info.loc[int(dorsal),'Places'] = ''
            info.loc[int(dorsal),'Infractor'] = 'NO'
        self.MAP["image"] = self.imBase
        self.IMAGEN['text'] = "MAPA BASE"


    def newCheater(self, dorsal):
        self.COMBO["values"] = list(self.COMBO["values"]) + [dorsal]
```

```
    def salida(self):
        if self.save_toEXCEL() == 'Ok':
            self.quit()

    def newSelection(self, posArg):
        global BOX, mapName
        dorsal = int(self.COMBO.get())
        pos = pd.DataFrame()
        pos["longitude"] = 0
        pos["latitude"] = 0
        #Interpreto los lugares del excel
        strPlaces = info.loc[dorsal,"Places"]
        listPlaces = strPlaces.split(' ')
        index=0
        for place in listPlaces:
            if len(place.split('#')) == 3:
                waste, lon, lat = place.split('#')
                pos.loc[index,"latitude"] = BOX[3]+(BOX[2]-float(lon))
                pos.loc[index,"longitude"] = float(lat)
                index+=1
        #Muestro resultados en la imagen
        image = plt.imread(mapName)
        fig, axis = plt.subplots(figsize = (20,20))
        title = "Lugares infraccion del usuario " + str(dorsal)
        axis.set_title(title)
        axis.scatter(pos.longitude, pos.latitude, zorder=1, alpha= 0.9,
c='r', s=200)
        axis.set_xlim(BOX[0],BOX[1])
        axis.set_ylim(BOX[2],BOX[3])
        axis.imshow(image, zorder=0, extent = BOX, aspect= 'equal')
        imgName = str(dorsal) + "map.png"
        fig.savefig(imgName)
        self.imUsr =
ImageTk.PhotoImage(Image.open(imgName).resize((800,800)))
        self.MAP["image"] = self.imUsr
        self.IMAGEN['text'] = "USUARIO "+str(dorsal)

    def createWidgets(self):
        self.downFr = Frame()
        self.downFr.config(bg='blue')
        self.downFr.pack(side='bottom', fill = 'x')

        self.QUIT = Button(self.downFr, text="SALIR", command=self.salida )
        self.QUIT.config(bg='red', fg='white')
        self.QUIT.pack(side='left', fill='y')

        self.EXCEL = Button(self.downFr, text="EXPORTAR EXCEL",
command=self.save_toEXCEL )
        self.EXCEL.config(bg='green', fg='white', state='disabled')
        self.EXCEL.pack(side='right', fill='y')

        self.leftFr = Frame()
        self.leftFr.config(bg='black')
        self.leftFr.pack(side='top')
        self.leftFr.pack(side='left')
        self.imLogo =
ImageTk.PhotoImage(Image.open("logo.png").resize((250,350)))
        self.LOGO = Label(self.leftFr, image = self.imLogo)
        self.LOGO.pack(side = "top", fill = "both", expand = "yes")
        self.CHOOSE = Label(self.leftFr, text="Elige deportista", bg='black',
fg='white')
        self.CHOOSE.pack(fill='x')
```

```
        self.COMBO = ttk.Combobox(self.leftFr, state="readonly")
        self.COMBO.pack(fill='both', side='top')
        self.COMBO['state'] = 'disabled'
        self.COMBO.bind("<<ComboboxSelected>>", self.newSelection)
        self.COMBO["values"] = []
        self.reconn = Button(self.leftFr, text="CONFIGURAR", bg="RED",
fg="white", command=self.configurar, height=10, width=35)
        self.reconn.pack(side="top")

        self.rightFr = Frame()
        self.rightFr.config(bg='black')
        self.rightFr.pack(side='left')
        self.IMAGEN = Label(self.rightFr, text="MAPA BASE", bg='black',
fg='white')
        self.IMAGEN.pack(fill='x')
        self.imBase =
ImageTk.PhotoImage(Image.open("logo.png").resize((800,800)))
        self.MAP = Label(self.rightFr, image = self.imBase)
        self.MAP.pack(side = "top", fill = "both", expand = "yes")
        self.verify = Button(self.rightFr, text="VERIFICAR", bg="green",
fg="black", command=self.verificar, state='disabled')
        self.verify.pack(side="left")
        self.discard = Button(self.rightFr, text="DESCARTAR", bg="blue",
fg="white", command=self.descartar, state='disabled')
        self.discard.pack(side="right")

    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.pack()
        self.createWidgets()
        self.toCheck = []
        self.configured = False
        self.configurar()



#Iniciar proceso Mqtt
def connMqtt():
    try:
        global client
        os.system('cmd /k ubuntu run sudo service mosquitto start')
        IP = socket.gethostbyname(socket.gethostname())
        client = mqtt.Client()
        client.on_message = on_message #Attach callback 4 subscribed topics
        client.connect(IP)
        client.loop_start()
        client.subscribe('state')
        client.subscribe('places')
    except:
      raise Exception('mqtt error')
def getData():
    try:
        global info
        info = pd.read_excel(r'infoTriathletes.xlsx').set_index('Dorsal')
        info['Infractor']=''
        info['Places']=''
        info['DropTime']=''
        info['CheckTime']=''
    except:
        raise Exception('data error')

def getMapConfig():
```

```
    global mapName, BOX
    try:
        with open('map.conf','r') as mapInfo:
            lineNumber = 0
            for line in mapInfo:
                if line[0] != '#':
                    if lineNumber == 0:
                        mapName = line.rstrip()
                        lineNumber += 1
                    elif lineNumber == 1:
                        long1 = float(line)
                        lineNumber += 1
                    elif lineNumber == 2:
                        long2 = float(line)
                        lineNumber += 1
                    elif lineNumber == 3:
                        lat1 = float(line)
                        lineNumber += 1
                    elif lineNumber == 4:
                        lat2 = float(line)
                        lineNumber += 1
        BOX = (long1, long2, lat1, lat2)
    except:
        raise Exception('map error')

mapName = ''
BOX = None
info = None
client = None
root = Tk()
root.title("12 meters. Detection")
app = Application(master=root)
app.mainloop()

root.destroy()
```

wireless.conf

```
# Config file for wireless communication
# info MUST be in: ssid, password and IP from server
# each data MUST be in a new line, with no extra spaces or symbols
# any comments line should have as first character '#'
DSM-Point
ES4911FXN

192.168.1.66
```

map.conf

```
# Config file for base map
# info MUST be in: map name, long1, long2, lat1, lat2
# each data MUST be in a new line, with no extra spaces or symbols
# any comments line should have as first character '#'
base.png
-6.519302
-6.388540
38.455336

38.343094
```

safeZone.conf

```
# Config file for safe zones
# info MUST be in: lat1#long1#lat2#long2
# each data MUST be in a new line, with no extra spaces or symbols
# any comments line should have as first character '#'
-6.010196#37.413202#-6.001958#37.405740
```

connZone.conf

```
# Config file for connection zone
# info MUST be in: HHMMSS#lat1#long1#lat2#long2
# each data MUST be in a new line, with no extra spaces or symbols
# any comments line should have as first character '#'
200000#-6.001322#37.401352#-5.990725#37.394137
```

Referencias

[1] www.wikipedia.org

[2] www.triathlon.org, *Competition Rules*

[3] www.micropython.org, www.docs.micropython.com

[4] www.github.com/espressif, esptool

[5] www.github.com/micropython/micropython-lib, umqttsimple, ssd1306 drivers.

[6] www.learn.adafruit.com, ampy tool

[7] www.anaconda.org

[8] www.docs.python.org/3/library/tk.html

[9] www.stackoverflow.com/

[10] www.cyclingweekly.com, Paul Norman, *How close do you need to be to benefit from drafting?*

[11] www.planetatriatlón.com, Juan P. Vázquez, *Análisis de datos de potencia Kona 2018*

[12] Juan Carlos Díez, Técnico de la Federación Española de triatlón

[13] www.mqtt.org