

Trabajo Fin de Grado

Ingeniería Electrónica, Robótica y Mecatrónica

Detector de distancia entre deportistas en competiciones de triatlón sin drafting

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

Detector de distancia entre deportistas en competiciones de triatlón sin drafting

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: Detector de distancia entre deportistas en competiciones de triatlón sin drafting

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, y más 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 el deportista en competición, y herramientas para el staff de la prueba para comprobar la cumplimentación de la normativa.

Abstract

... -translation by google-

Índice

Trabajo de Fin de Grado: Detector de distancia entre deportistas en competiciones de triatlón sin drafting	4
Agradecimientos	6
Resumen	7
Abstract	8
Índice	9
Índice de Tablas	11
Índice de Figuras	12
1 Prefacio	1
1.1 <i>Introducción</i>	1
1.1.1 El triatlón	1
1.1.2 El drafting	2
1.1.3 Normativa	3
1.1.4 Situación actual	3
1.2 <i>Estado del arte</i>	3
1.2.1 Metodos	4
1.2.2 Dispositivos	4
1.3 <i>Propuesta</i>	4
2 Selección de tecnología	2
2.1 <i>Hardware</i>	2
2.1.1 Microcontrolador	2
2.1.2 Detector	3
2.1.3 Periféricos	6
2.2 <i>Software</i>	7
2.2.1 Comunicaciones	7
2.2.2 Interfaz gráfica y tratamiento de datos	9
3 Esquema general de funcionamiento	12
3.1 <i>Funcionalidad en competición</i>	12
3.2 <i>Esquema Comunicaciones</i>	13
3.3 <i>Estructura de datos</i>	14
3.3.1 Hoja Excel	14
3.3.2 Ficheros de configuración	14
4 Desarrollo aplicación	16
4.1 <i>Interfaz gráfica</i>	16
4.1.1 Uso de Tkinter	16
4.1.2 Aplicación	16
4.2 <i>Conexión y datos</i>	21
5 Desarrollo detector	24
5.1 <i>Estructura y configuración</i>	24
5.1.1 Flasheo y configuración	24

5.1.2	Carga de archivos a dispositivo.	25
5.2	Montaje	26
5.3	Lectura distancia, ciclo principal	27
5.4	Conexión a SSID	31
5.5	GPS	31
5.6	Pantalla	34
6	Demostradores y pruebas	36
6.1	Aplicación organización	36
6.2	Servicio usuario	39
7	Conclusiones	40
Anexo 1.	Instalación y problemas	2
	Instalación de Windows subsystem for Linux	2
	Problemas al lanzar servicio mosquito	4
Anexo 2.	Código	7
	read.py	7
	gps.py	7
	sendInfo.py	7
	main.py	7

r! Marcador no definido.

r! Marcador no definido.

r! Marcador no definido.

r! Marcador no definido.

1 PREFACIO

1.1 Introducción

1.1.1 El triatlón

El triatlón es un deporte que consta a su vez de tres disciplinas distintas, natación, ciclismo y atletismo; y estas tres disciplinas se encadenan una tras otra, sin descanso entre ellas, dando así forma a lo que se conoce hoy en día como el deporte del triatlón.

Entrando un poco más en detalle, pasamos a resumir como es el funcionamiento de una competición de triatlón.

La carrera siempre tiene lugar en un lugar cercano a un embalse, río o playa, que es donde se realiza el sector de natación en aguas abiertas, que es la primera disciplina deportiva que se lleva a cabo. Cercano al agua tendremos el área de transición, muy importante en cualquier competición de triatlón. Aquí cada deportista tendrá un pequeño espacio en el que antes de comenzar la competición habrá dejado preparados su bicicleta y sus zapatillas para correr, cada cambio de disciplina pasa por esta zona, por lo que suele ser el centro de la competición, donde suele además estar la salida, meta, centro de organización, mayoría de público, etc...

Habiendo realizado este paso previo, da comienzo la competición, todos los participantes toman la salida a la vez y se lanzan al agua. Ahí realizan un recorrido marcado que acaba en la entrada a el área de transición, por orden de llegada los deportistas van dirigiéndose cada uno a su bicicleta, sin descanso de ningún tipo, una vez hecho el cambio de material se dirigen bicicleta en mano a la salida de la transición, la carretera. Aquí comienza el sector de ciclismo, y es donde nuestro proyecto entra en juego.

El triatlón es un deporte multidistancia como se puede ver en la imagen [Reglamento apéndice 23](#) donde vienen

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

recogidas las distintas composiciones de un triatlón. Hay dos modalidades principales de ciclismo, con y sin *drafting*. Nuestro proyecto solo abarca las competiciones que se desarrollan bajo la modalidad sin *drafting*, esto significa que en el sector de ciclismo hay que mantener una distancia, normalmente de 12 entre participantes, para así no poderse aprovechar del rebufo de otro participante y reducir tu coeficiente de drag, lo que comúnmente se conoce como “*ir a rueda*”. Estas competiciones sin *drafting* son las de media y larga

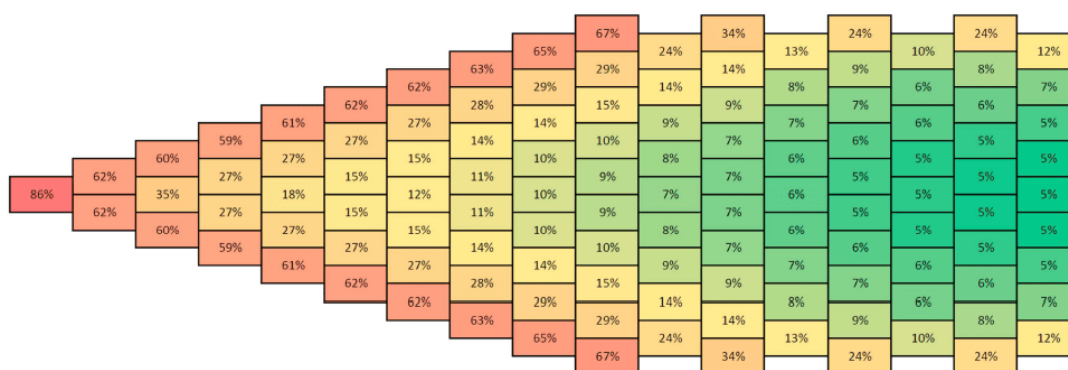
distancia, ya que el recorrido más extenso hace posible que haya espacio suficiente entre deportistas, aunque en algunas situaciones excepcionales puede aplicarse a otras distancias más cortas.

Una vez se ha completado el recorrido de ciclismo, los triatletas vuelven a su puesto del área de transición, donde dejarán la bici y se calzan la vestimenta para realizar el sector de atletismo, con el cual concluye la prueba.

1.1.2 El drafting

Una vez tenemos una visión general de que es una prueba de triatlón, podemos 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 respondería: *sin duda alguna, sí.*

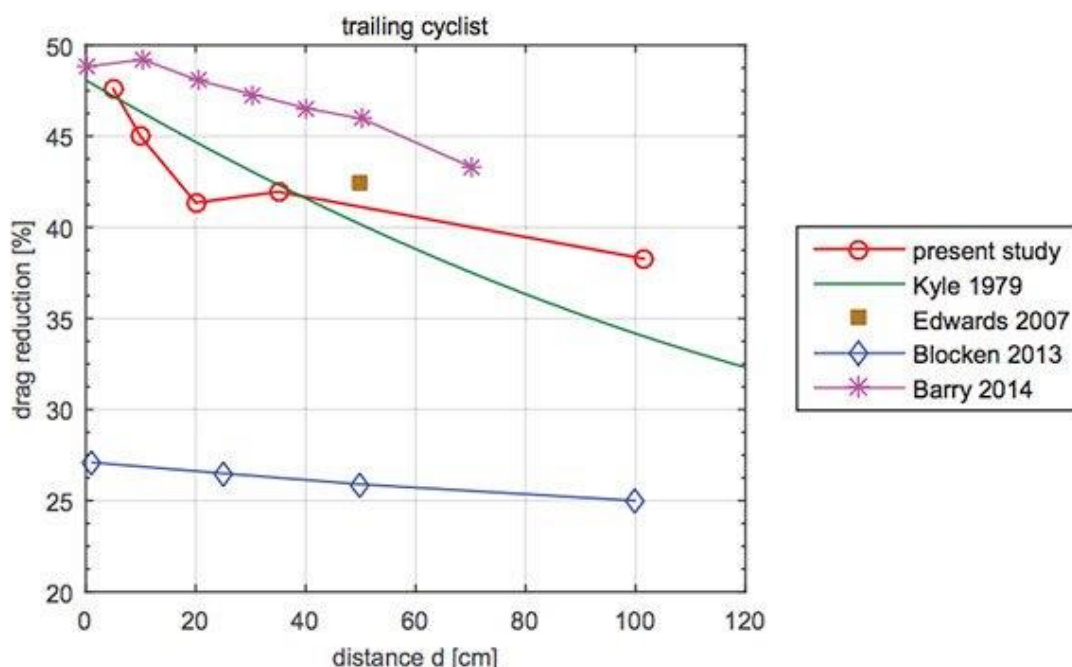
Comúnmente en una competición de ciclismo podremos ver que los ciclistas permanecen casi todo el tiempo en un gran grupo muy pegados unos a otros, y que, aunque muchas veces esto produzca situaciones de peligro y caídas, lo continúan haciendo sin parar. Esto es sin duda porque les merece mucho la pena, como se puede ver en la imagen <http://blog.endurancegroup.org/perfil-de-potencia-velocidad-modelo-de-rendimiento-para-ciclismo-de-ruta-parte-1/> 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.

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 imagen <https://www.adnciclista.com/ir-a-rueda/> donde se muestran los resultados de diferentes estudios acerca de la reducción del coeficiente aerodinámico del ciclista que va a rueda de otro en función de la distancia a el mismo.

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 del mismo 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, 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, esto es 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; y fuera de reglamento, pero normalmente suele aplicarse, 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, y que, debido a esta baja velocidad, la distancia representa un espacio de tiempo muy alto entre deportistas, lo que puede ser contraproducente en competición.

1.1.4 Situación actual

En la actualidad 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 cantidad de participantes hace que, al mismo tiempo, haya muchos kms del trazado ocupado por deportistas al mismo tiempo, lo cual hace que sean necesarios todavía más recursos. Hay que tener en cuenta 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 quede 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 Metodos

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 llevan 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 por regla general 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, y la utilidad que tiene es la de servir de apoyo para poder enviar a jueces en moto a controlar a ciertos deportistas que aparenten estar incumpliendo 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 van 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 bicicleta 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	Configurar el dispositivo de forma que no sea necesario 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 servidor 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 recorrido 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	Código	El código fuente debe estar encriptado o compilado en el dispositivo final	SHOULD
#23	Detector	Cambio de sentido	El detector podrá interpretar en que sentido se circula en las áreas seguras	SHOULD

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.



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, ya que en la actualidad 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 como con la cual podemos usar el ESP de manera similar a cualquier intérprete Python para computadora, teniendo 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 nos va a permitir 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. Apoyándonos en que una bicicleta tiene siempre unas cotas que son constantes en todas ellas (principalmente el diámetro de rueda) podríamos 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 procesamiento 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, tendríamos que poder comparar la geolocalización e instante de tiempo de todos los deportistas, lo que nos deja en la situación de la necesidad de enviar de manera constante durante el transcurso de la prueba la señal GPS 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.

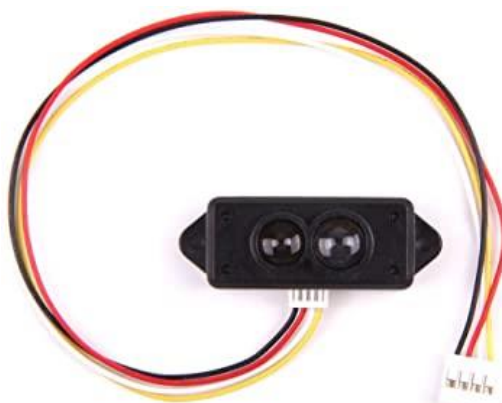


Sensor de distancia ultrasonido, estos sensores nos permiten detectar objetos y son muy conocidos en el mundo de la robótica y proyectos caseros debido a su bajo costo y buen resultado. Por regla general estos sensores tienen la capacidad de detectar objetos hasta una distancia de 6m con un haz de unas pocas decenas de grados. Esta distancia queda escasa para nuestro proyecto, pero existen algunas versiones más sofisticadas de estos sensores que indican conseguir 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, **como veremos más adelante**.

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

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 obtener la detección de ningún tipo de objeto a más de 3 metros.

Sensor óptico de distancia, este ha sido la opción final usada para el proyecto, en concreto se ha decidido usar el TF Mini Lidar de Seed Studio.



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

Se ha comprobado, con pruebas específicas en terreno, que la distancia máxima de lectura en condiciones externas para un día soleado son similares a 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 laser 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. Éste cálculo lo realiza el sensor internamente, y proporciona la información directamente en unidad de metros.

El protocolo UART es un protocolo entre dispositivos, lo cual lo limita bastante en comparación a los 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 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 tenemos 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.

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.

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 adaptador 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.

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.

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 caracteres 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 coincide con el área geográfica el proceso de conexión a satélites GPS puede tardar hasta 10 minutos.

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

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 ubicación resulta bastante lenta.

Buzzer

Leds

2.2 Software

2.2.1 Comunicaciones

En cuanto a la comunicación del dispositivo detector con la organización disponemos 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, un organización sin ánimo de lucro, por lo que el uso de este protocolo es gratuito y existen distribuciones básicas de código abierto y algunas con más funcionalidades de propietarios privados Queda regulado bajo la norma ISO/IEC 20922, y en los últimos años su uso está aumentando drásticamente ya que se presenta 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.

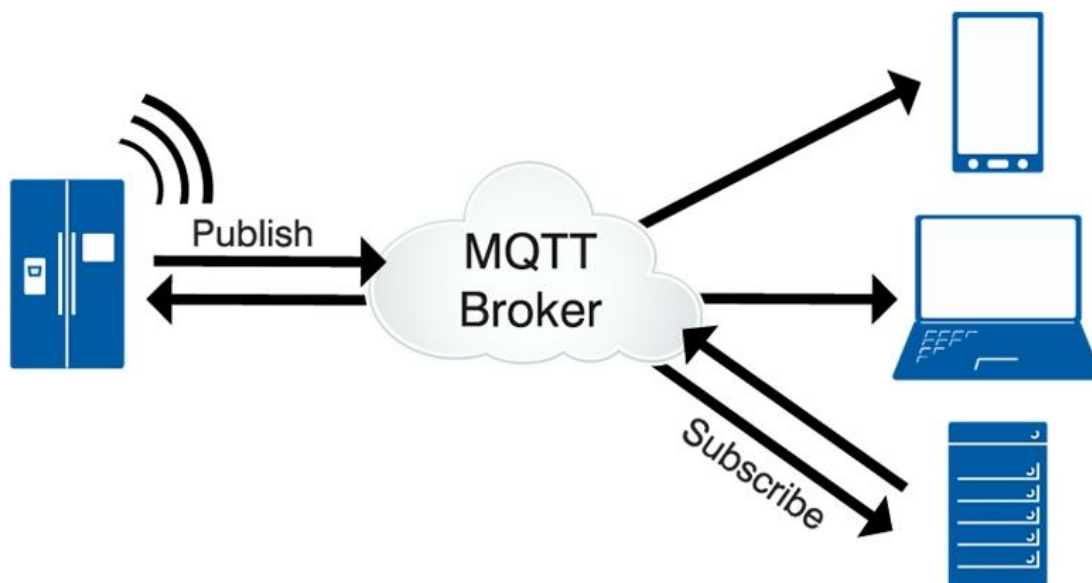
El protocolo por regla general corre sobre TCP/IP, aunque podría correr sobre cualquier otra capa que asegure las conexión bidireccionales como por ejemplo Zigbee. Esta ideado para proporcionar comunicación entre

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, aunque en 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 encontramos todos nuestros 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.

El otro elemento que conforma la arquitectura mqtt son los clientes. Estos clientes pueden ser cualquier dispositivo, computadoras, servidores, 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 actúa de broker.

Cuando tenemos un broker y una serie de clientes en una misma red, se pasa a comentar como es el funcionamiento del protocolo. El broker funciona como intermediario entre todos los clientes. Un cliente puede suscribirse y publicar sobre un topic. Este 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 los 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 nuestro caso se ha usado el broker de mosquitto para Ubuntu (*Ver instalación en Anexo 1*) En un nuevo terminal de Ubuntu, ejecutamos 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 habremos instalado el broker en nuestro computador, y para lanzarlo, ejecutamos:

- *sudo service mosquito start*
- *sudo service mosquito status*

Con la primera instrucción lanzamos el servicio, y con la segunda comprobamos que el servicio ha sido lanzado correctamente. Para ello nos debería devolver el siguiente mensaje

- ** mosquito is running*

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

Como clientes, ya hemos 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 eclipse que integran clientes mqtt sobre multitud de lenguajes de programación. En nuestro caso vamos a usar la librería de Python. Podemos obtenerlas mediante la utilidad pip para la gestión 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 ejecuta:

- *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 para el correcto funcionamiento es la dirección IP del broker.

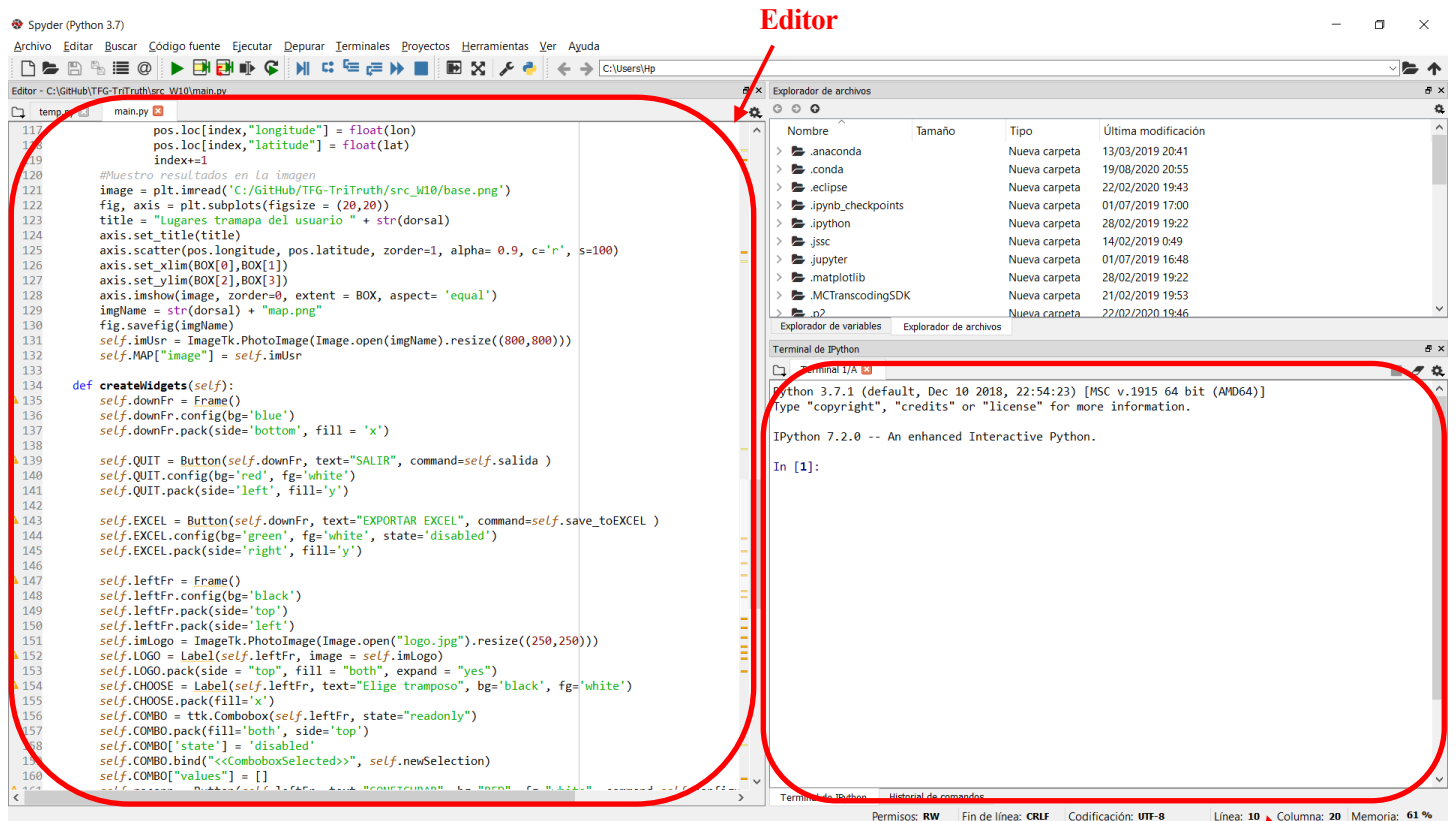
En cuanto al segundo de los clientes, los dispositivos detectores, se ha utilizado una utilidad para microPython llamada umqtt-simple, que nos da una API para usar un cliente de mosquito sobre TCP/IP. Esta se encuentra como parte de la librería de utilidades de microPython externas a la distribución básica. Puede ser instalada en algunos dispositivos mediante la emulación de pip para microPython o, como se ha realizado en nuestro 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 tan 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 de los que tenemos 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 sobre Python y R, diseñado para el machine learning y ciencia de datos, que proporciona un entorno libre y abierto. Incluye multitud de utilidades, de las cuales para el desarrollo de la aplicación vamos a usar Spyder, que es un IDE de desarrollo que pese a ser ideado para la ciencia de datos, es bastante sencillo de usar para el desarrollo de una pequeña utilidad como la nuestra, 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 interfaces bastante más completas, también gratuitas para Python como wxPython o PyQt, pero se ha elegido Tkinter por que proporciona lo necesario para la solución que se va a desarrollar, y 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, cabe 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 Anaconda. En concreto se va a utilizar la clase dataframe de *Pandas*, pandas es un 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, y 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 de 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.

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 tenemos involucrados a lo largo de una competición. La descripción del dispositivo detector será individual para un solo dispositivo, siendo esta extensible a todo un conjunto de participantes que usen 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 el dispositivo.

Poniéndonos en situación de una competición real sin drafting. En el momento de la entrada en transición, cuando se realizan las diferentes verificaciones técnicas, 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.

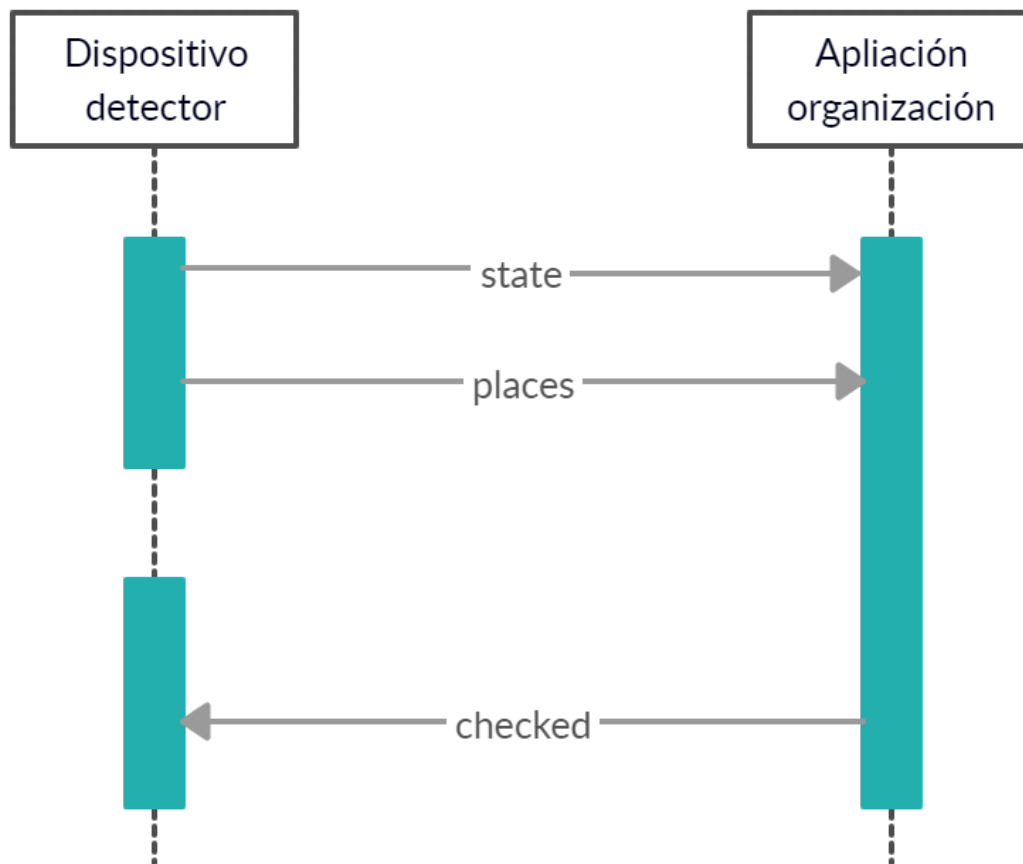
Comienza la prueba, y se realiza el sector de natación, al finalizar este el participante se dirige a su bicicleta y sale del área de transición donde se monta en ella. Hasta este punto el dispositivo ha estado encendido, pero no comprobando la distancia de seguridad ya que el área de transición ha sido 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 transcurso 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 que hay allí, 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 tenemos la información en la computadora del organizador, ahora el técnico juez responsable, en caso de que el deportista haya 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 infractores irá recibiendo esta 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, de manera que nos aseguraremos no realizar 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



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. Tal y 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 a todos los mensajes que se van a utilizar en la comunicación del proyecto comenzarán por el id del deportista, que por regla general será su número de dorsal. A continuación irá un espacio y después ya lo que a partir de ahora trataremos como contenido del mensaje. Esto se ha realizado de esta manera 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 espaciando 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 ninguna 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, y 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, ya 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 disponemos de un botón con el que hacer una exportación al Excel de salida en cualquier momento. Además, si salimos de la aplicación también se guardará por defecto la información, para así evitar una posible pérdida de datos.

3.3.2 Ficheros de configuración

En general, 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 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 archivos, este es específico de cada usuario, por lo que, dentro de la utilidad para cargar los archivos, se crea, cumplimenta con el identificador de deportista (pasado por argumentos) y se carga en el dispositivo. Por lo que este archivo es invisible al responsable de la configuración. Como este archivo 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 límites 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 mosquito.

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, aunque si depende de una configuración específica como es el terminal de Ubuntu para Windows 10 y el servicio mosquito 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, este será el que se llama *frame* principal, el cual, vamos a enviar como argumento al constructor de la aplicación. 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, estos frames 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 botones, labels, boxes de introducción/selección o imágenes. La función principal que gestiona la dependencia de objetos dentro de la aplicación 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 hacemos es asignar inicializar el frame de la ventana principal que nos han enviado por argumentos, y ejecutar 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 conocemos los diferentes contenedores, vamos a ir analizando cuales son los objetos que se adjuntan a cada uno de ellos, y cual 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 gráficamente con cada uno de los frames.

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



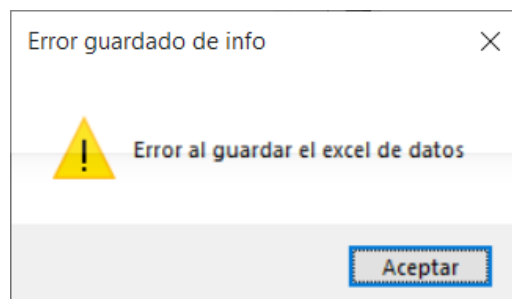
- QUIT: es un objeto *Button*, está asociado al método *salida* de la propia clase. Cuando se pulsa este botón se ejecuta su función, que, a su vez, llama al método *save_toEXCEL*, el cual devolverá un mensaje 'Ok' o de 'Error' en función de si ha sido posible cerrar guardando la información en el archivo



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 ejecutará, de esta manera evitamos la pérdida de datos antes de un posible cierre indeseado.

- 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 aplicación 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.

Hasta aquí, si todo va bien se devuelve el mensaje 'Ok', en cambio, si algo no funciona guardamos la excepción producida, y la informamos con una ventana tipo pop-up de advertencia 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:



- 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, del cual podemos seleccionar el identificador que queramos. 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 podemos ver contiene una pareja de coordenadas cardinales, y es que serán los límites de un *subplot*, obtenido de la librería *matplotlib*, en el cual introduciremos una serie de puntos, dentro de esa escala, que los obtendremos de la casilla *Places* del dataframe *info* que contiene la información de los deportistas. Esta información es decodificada y plotada sobre estos ejes customizados.

Una vez tenemos 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 más 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.

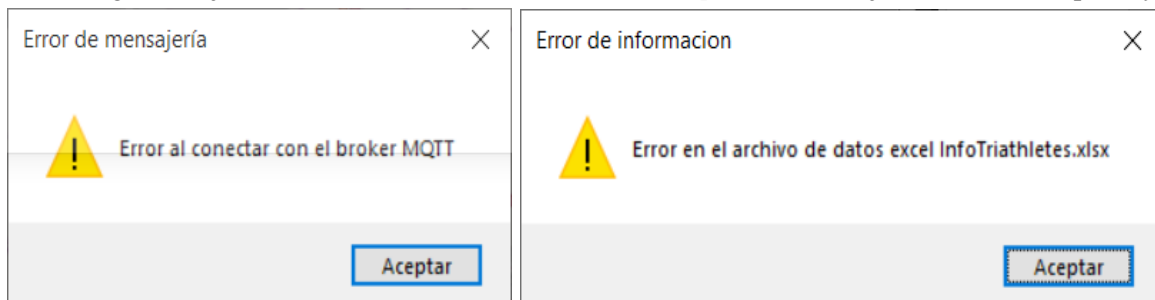
*/*Hacer diagrama de orden de función*/*

- recon: objeto *Button* que aparece debajo del combobox 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.

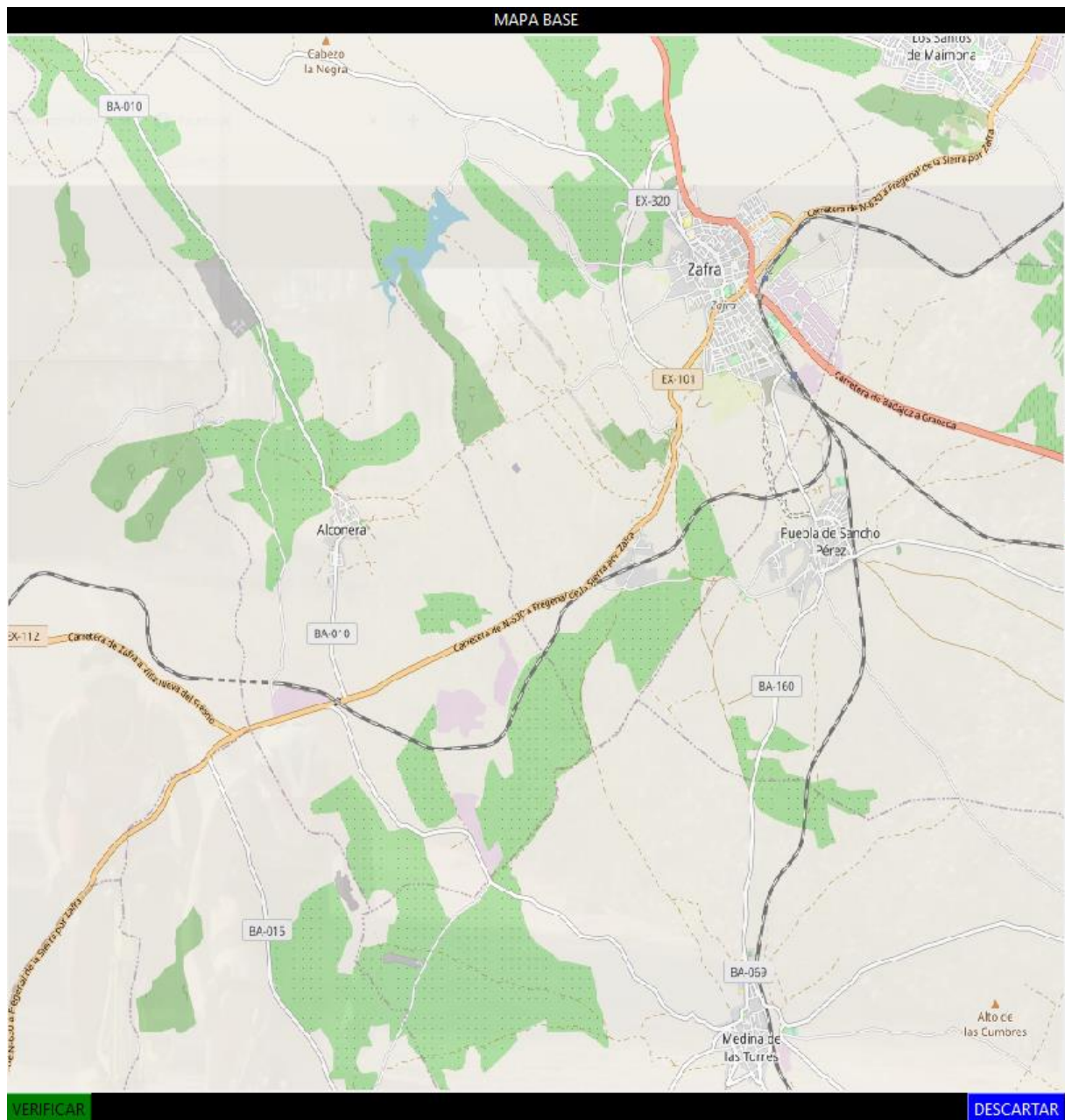
En caso de que alguna excepción ocurra se recoge, y se trata de manera diferente en función del código de excepción. Si este es *mqtt error* se muestra una advertencia pop-up informando de que no se ha logrado la conexión con el broker; si es *data_error* el mensaje informará de que no se ha podido cargar la hoja de información de los usuarios; si es *map error* el mensaje 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*' corresponde a que 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.

El método *configurar* se llama justo tras crear los widgets, con lo que si este método funciona bien, no deberíamos ver el botón en ningún momento, ya que se destruye antes de que se fije la ventana como visible. De esta manera, si todo funciona 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:



- IMAGEN: objeto *Label* contiene el título de la imagen de mapa mostrada, por defecto contiene 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 conoce 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 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 verificar, se elimina el deportista del combobox, pero en este caso, en lugar de rellenar el campo CheckTime lo que se hace es poner *Infractor* a “NO” y vaciar 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, en concreto más en detalle sobre el funcionamiento de la conexión al broker mosquito y la gestión de datos de participantes.

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

- **connMqtt:** en esta función configuramos 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 mosquito, 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 mosquito start')

lo que se realiza es ejecutar, desde un command prompt de Windows, en el terminal Ubuntu de Windows 10 el comando “*sudo service mosquito 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 el 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 asignamos a la variable global cliente, hasta ahora vacía, un objeto cliente mosquito. 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, le asignamos la función propia *on_message* como callback a la llegada de mensajes, realizamos la conexión a la IP obtenida, arrancamos el bucle para la recepción de mensajes y nos suscribimos 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 nuestra red local, esta función fallará. En este caso capturamos la excepción, y lanzamos una excepción propia llamada *mqtt error* usando la siguiente instrucción:

raise Exception('mqtt error')

- **getData:** en esta función cargamos en el dataframe global *info*, hasta ahora vacía, la información del Excel de información de triatletas. Para ello se usa la utilidad 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, y, las líneas que no comiencen por el carácter '#', se toman como líneas de contenido, con lo que el valor de la primera línea de contenido se almacena en la variable global *mapName*, habiéndole quitado la terminación de nueva línea '\n' aplicando el método genérico que tienen 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 atributo contiene la información publicada. Indiferentemente de topic que se nos envíe, decodificamos la información de payload a texto plano, y la dividimos 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 tendremos un mensaje 'SI' o 'NO'. Además, si no ha infringido, el otro campo a rellenar será el *DropTime*, en el que incluiremos la fecha y hora actual y, automáticamente publicamos 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 anexa, de manera similar, pero en el campo *places* la información de las coordenadas de infracción y, si todavía no se conocía el estado de infracción porque se hubiese perdido el mensaje, se fuerza el campo *Infractor* a *SI*. A continuación, mandamos el mismo mensaje para apagar el dispositivo detector y cumplimentamos 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

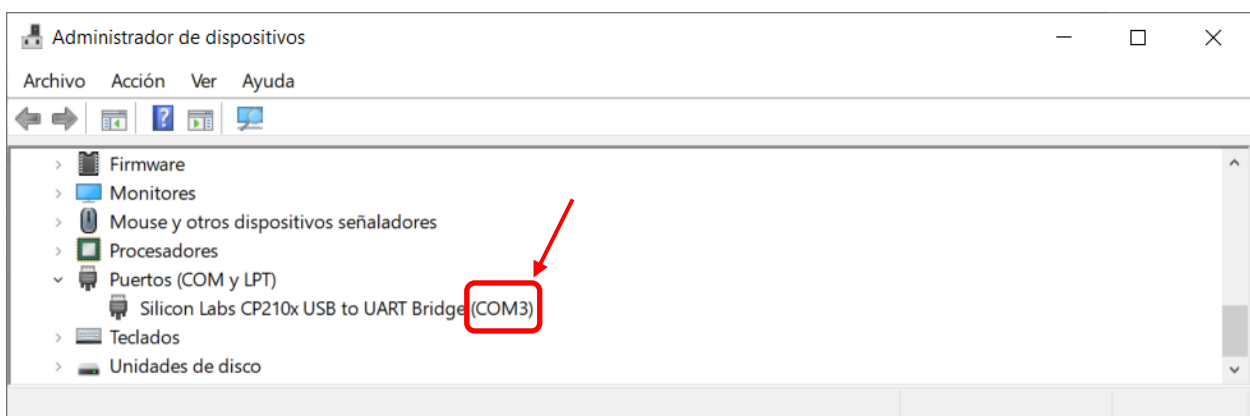
El desarrollo del dispositivo detector está formado por tres scripts, estos son *sendInfo.py*, *gps.py* y *read.py*. Los dos primeros se usarán como módulos dependientes del tercer script, que será el principal, que gestione el ciclo de funcionamiento del dispositivo. Cada uno de estos archivos, así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

En cuanto al arranque del dispositivo, por regla general, las distintas distribuciones de microPython tienen un proceso de arranque específico. En concreto, todas poseen un script llamado *boot.py*, es más, en caso de que este script se elimine manualmente, automáticamente en el siguiente reinicio lo crea, aunque vacío. Bien, este archivo se ejecuta automáticamente 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, debemos conectar el 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 que número de puerto serie, nos podemos dirigir al administrador de dispositivos de Windows, y en el buscar el número de puerto COM asignado.



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 debemos tener disponible Python3 en el entorno de trabajo, con lo que tan sólo hay que ejecutar 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 de nuestro 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 hemos descargado antes (o su ruta completa si no está en el directorio de trabajo).

Una vez se ha cargado el nuevo firmware, podemos 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, nos aparece un intérprete de Python que permite ejecutar sentencias básicas como la importación de un módulo y chequear los archivos disponibles en un directorio:

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

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

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, aunque además de se puede usar para eliminar, obtener, gestionar directorios, resetear o correr archivos. Estas otras utilidades se pueden realizar directamente en el terminal de microPython, 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 lo vemos 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 distinto 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 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 le fabricante del producto. Para la carga de los archivos de configuración por parte de la organización se ha creado un script Shell para Ubuntu, este se llama *loadConfig.sh*. Al ser una tarea bastante repetitiva, ya que debe ser realizada 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 encuentran 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 puedes 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 script ejecuta la instrucción de *ampy* correspondiente. Para el archivo *user.conf*, lo primero que se realiza es borrar un posible archivo con el mismo nombre antiguo, esto se hace con la instrucción:

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

se vuelca el número identificador a un nuevo archivo:

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

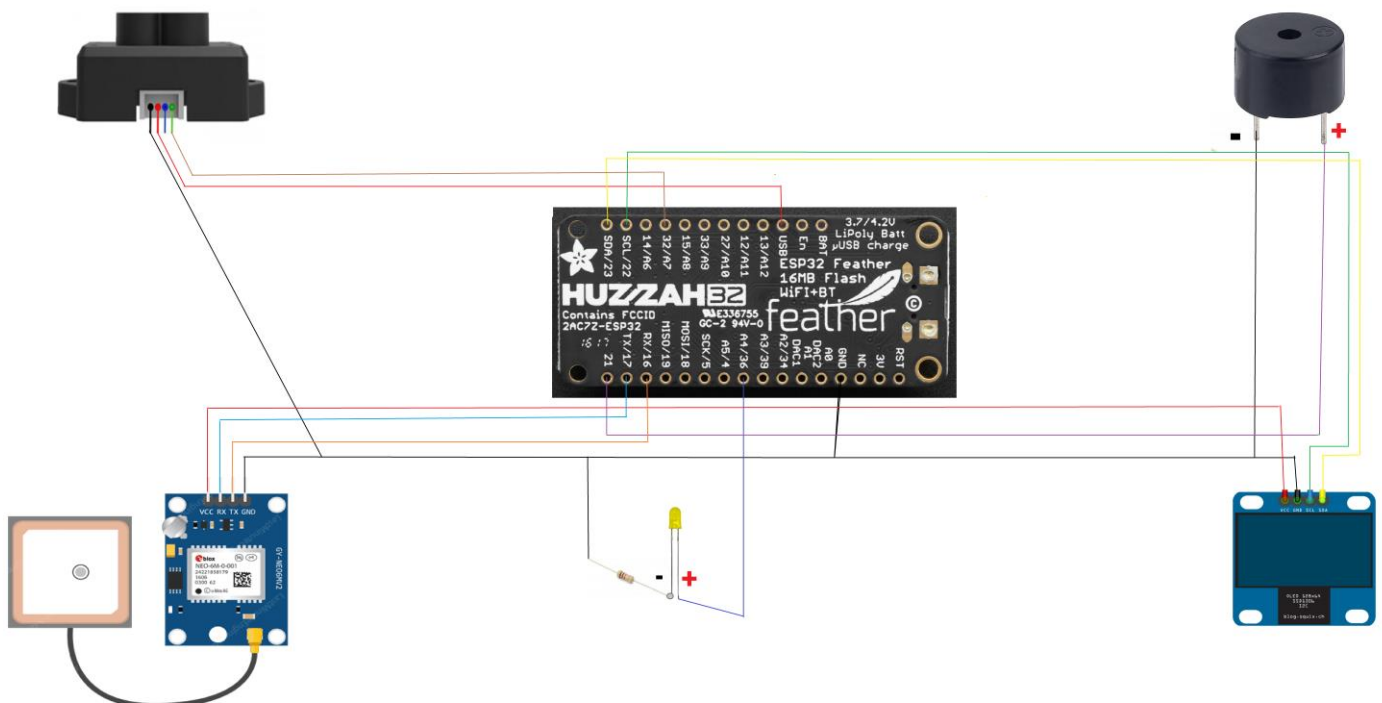
en la variable *user* se ha guardado antes internamente el número identificador, y por último cargamos mediante *ampy* el archivo, y lo eliminamos de la computadora.

Con este script se simplifica bastante la carga de archivos, a, una vez creados los archivos comunes, ejecutar una sola 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



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 efectúa 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, por lo que 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 recibimos de la función, para trabajar más adelante con el sensor. Además, también se fija como pin de salida el pin 33, ya 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 aplicación 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 infringir, es decir, 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; de manera contraria, si el tiempo libre es menor al cincuenta por ciento del total del aviso, se considerará una infracción, de manera que se añade la ubicación (correspondiente al comienzo del aviso) en la variable *cheatInfo*.

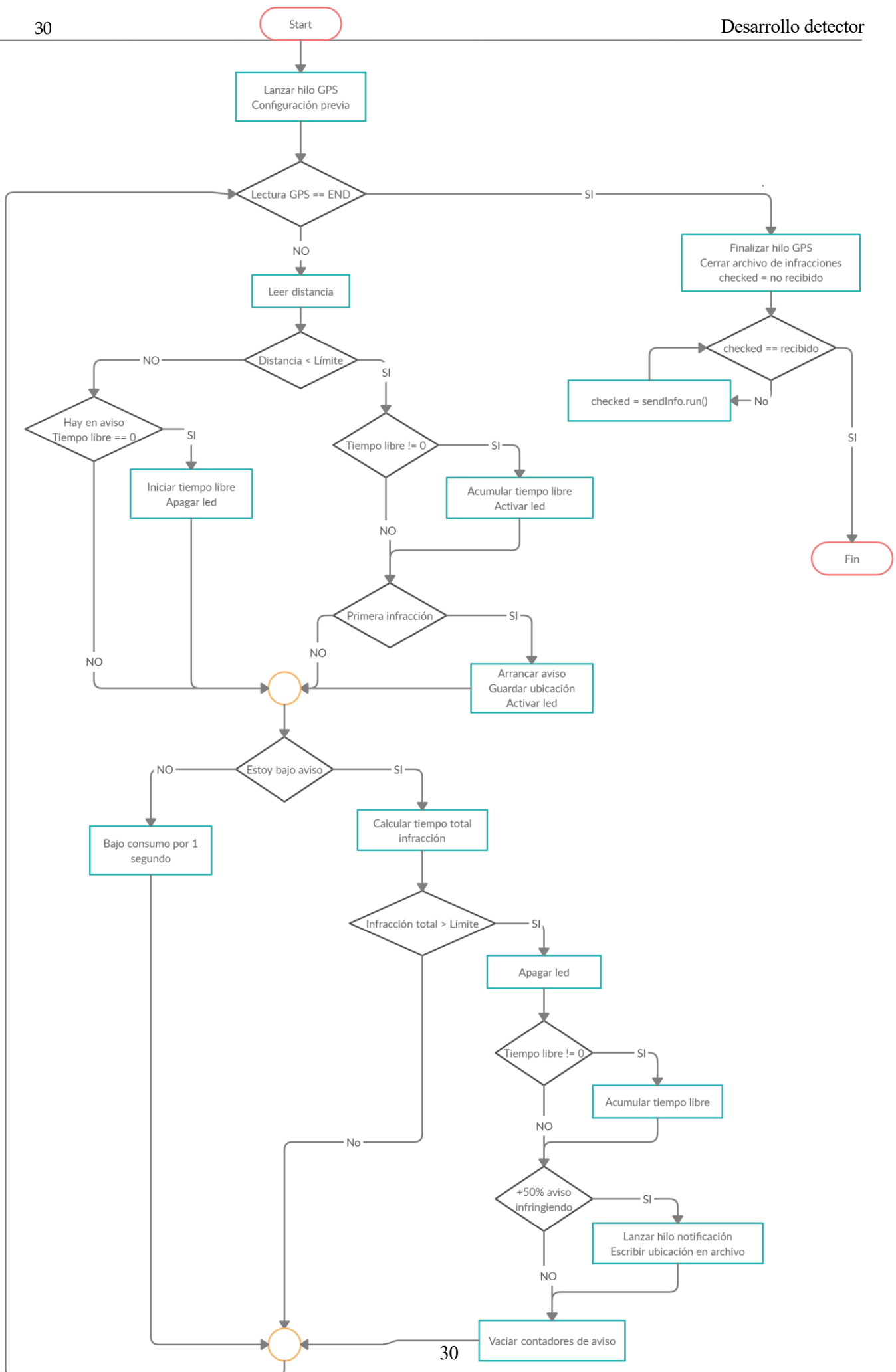
Además, durante este ciclo de reconocimiento de positivos, siempre que bajo un aviso estemos infringiendo la distancia de drafting, se iluminará el led del pin 33 a modo de información para el deportista. El tiempo de no infracción, bajo un aviso, el led estará apagado. Una vez finalizado un aviso, si 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*, tan solo hay llamar a su método *exit*.

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



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, y para aplicarlo en el ESP se ha usado la utilidad *umqttsimple*, ya comentada previamente.

Al importar el script, lo primero que se 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 obtenemos, 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 realizamos la conexión a la Wifi, y a continuación al broker Mqtt; Para ello tenemos dos funciones dedicadas, que en caso de funcionar correctamente devuelven el objeto correspondiente. Pero 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 creamos un objeto llamada *wifi* y ejecutamos 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 pasa a comentar 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, ejecutamos *machine.deepsleep()*, con lo que ponemos en el modo bajo consumo más estricto al dispositivo de manera indefinida, siendo esto lo más similar a estar apagado.

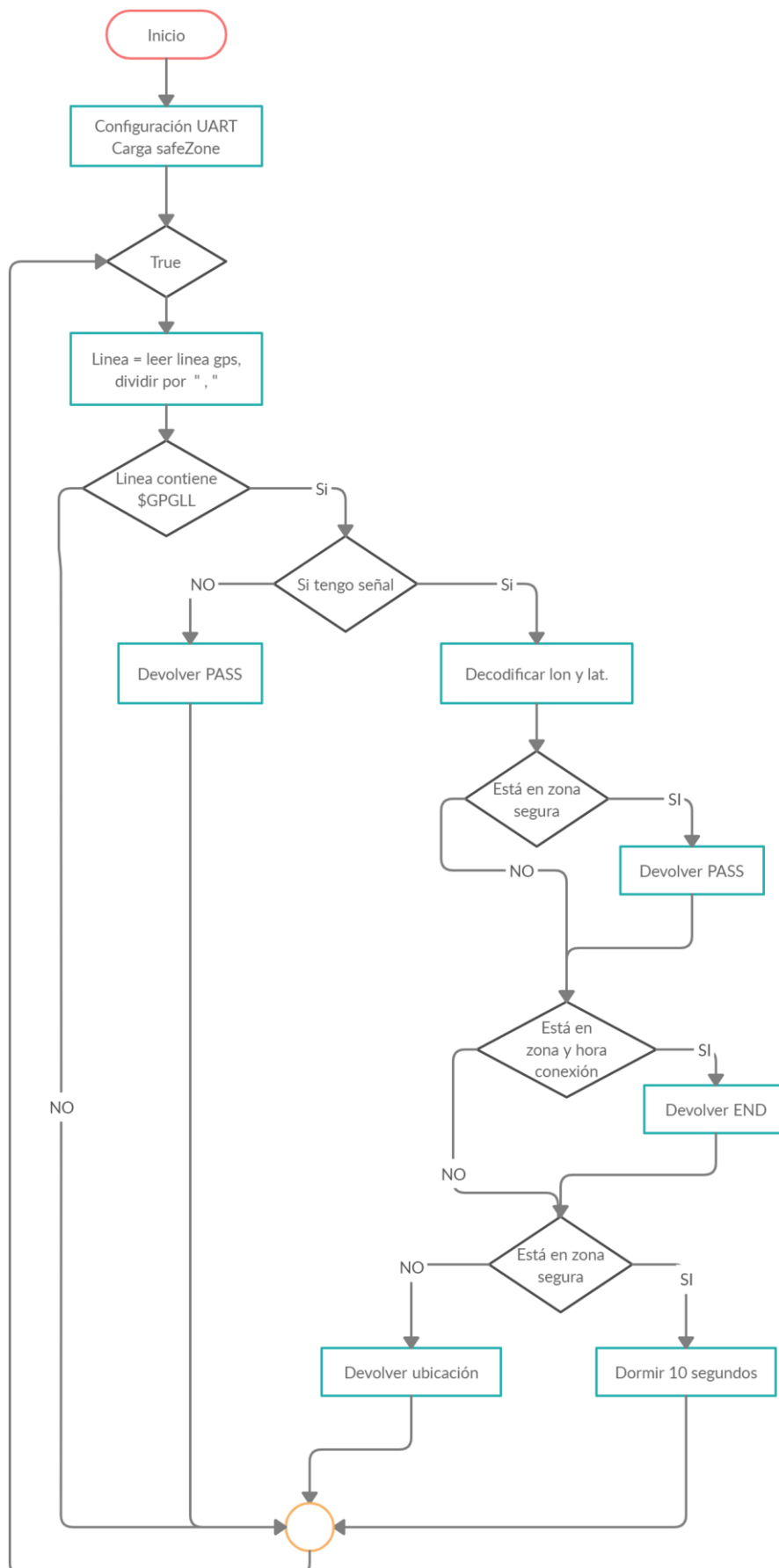
Una vez se ha configurado satisfactoriamente todo, usando la variable *cheatInfo* recibida por argumentos, en la que se han almacenado los casos de infracción, enviamos, a través de la función creada para ello, el contenido y estado de dispositivo. Si ha habido infracción se envían mensajes a los topics *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 hemos 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 él cargamos dos archivos de configuración, *safeZone.conf* y *connZone.conf*, en ellos respectivamente tenemos 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 llamaremos desde el hilo principal, 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 pasamos al bucle infinito. En este bucle leemos la información de la UART que nos viene en caracteres ASCII. Esto son tramas bajo el protocolo de comunicaciones NMEA (*National Marine Electronics Association*) 0183.

El protocolo NMEA 0183 es usado por múltiples dispositivos de la industria marítima, y por la gran mayoría de dispositivos GPS. Este protocolo propone una comunicación por puerto serie, 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 satélites 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'
```

- GLL: latitud, longitud y hora.
- RMC: Datos mínimos recomendados (posición, 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 los 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 último, 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, horas, minutos y segundos.

Para localizar la trama que se busca entre los mensajes del sensor, se va leyendo por líneas completas la

	Grados	Minutos	Sentido	Grados	Minutos	Sentido	HH:MM:SS.SS
\$GPGLL,	38	25.22	967,N	006	24.93	251,W	102651.00,A,A*72\r\n
	LATITUD			LONGITUD			

información, para 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 comprobación 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, lo primero que 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 las coordenadas es más común y fácil de tratar a posteriori que la que ofrece la trama por defecto. Lo primero que se hace es obtener los grados y minutos en número independientes, y a continuación, a los grados, sumarle los minutos divididos por 60. Por último, si el signo debe ser negativo se aplica el valor.

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 las zonas seguras. Si es así devolvemos el mensaje *PASS*. A continuación, comprobamos si estamos en el área de conexión y en la marca temporal requerida; si es así, devolvemos el mensaje *END*, para que el script principal llame al bloque de importación. Si tan sólo estamos 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 nos referimos a 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, con lo que se consigue que el hilo tan sólo haya que lanzarlo una vez, y la lectura de valores sea mucho más sencilla.

5.6 Pantalla

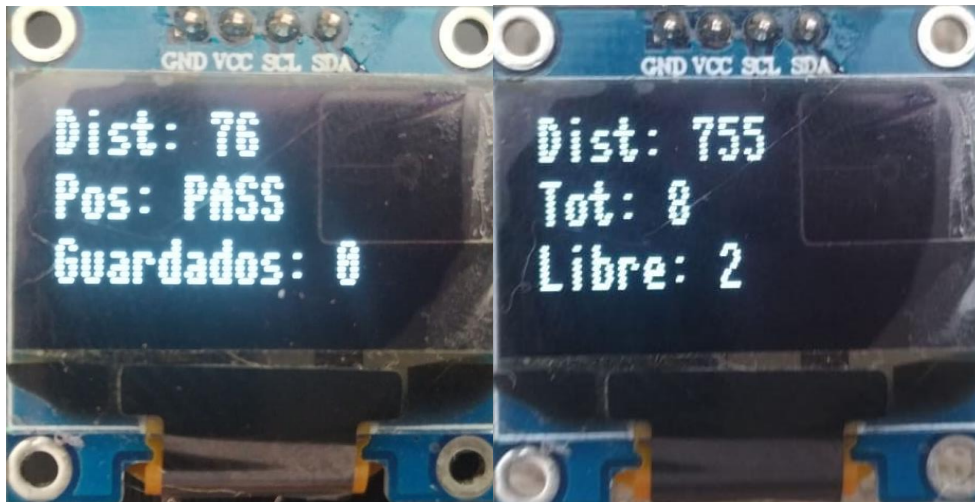
A modo de ayuda para las pruebas, el dispositivo desarrollado cuenta con una pantalla OLED SSD1306 de 32x128 bits. Su uso va enfocado al desarrollo de pruebas para comprobar el correcto funcionamiento tanto del sensor de distancia como del resto del proceso de detección. Este elemento no se incluye como 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 los 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 librería *ssd1306*, incluida en el conjunto de librerías de apoyo en microPython, al igual 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, limpia el contenido anterior de la pantalla y muestra el actual, esta es la función *dispMsg*. Tienen 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, el primero es, mientras se encuentra detectando la distancia, (ya estemos 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 aplicación 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 *mosquitto_pub*:

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

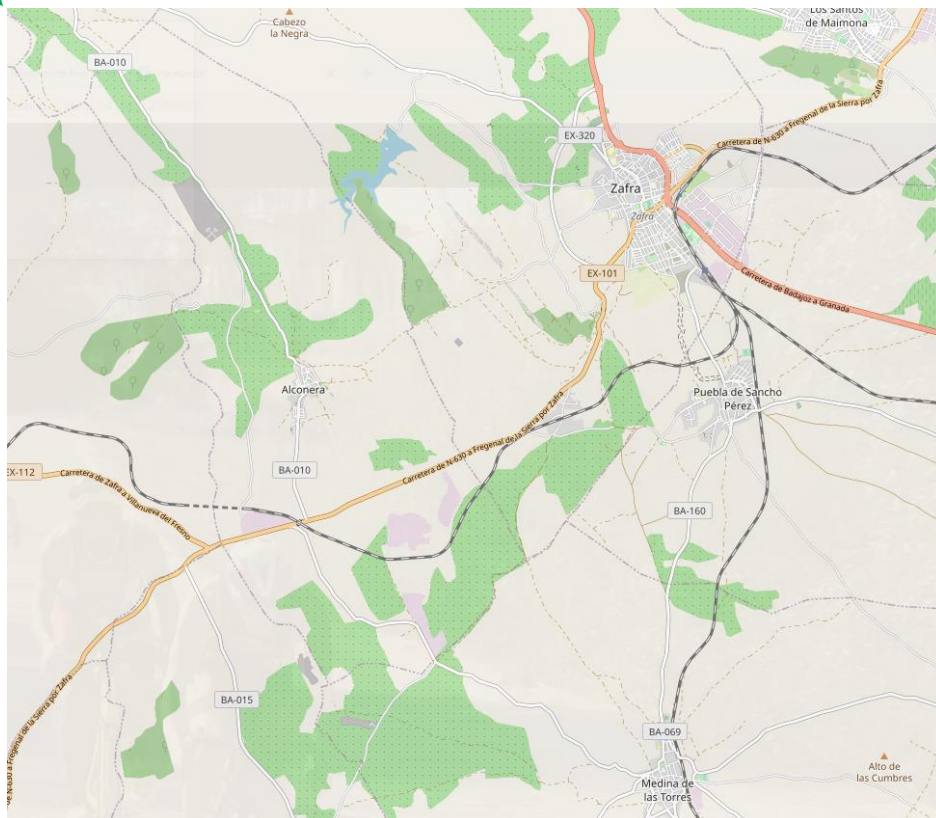
donde el argumento *-h* es la IP del bróker, *-t* el topic y *-m* el mensaje asociado. Para suscribirse a un topic, en un terminal nuevo, se ejecuta:

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

e irán apareciendo los mensajes en ese terminal.

Lat: 38.447095

Long: -6.493933



Lat: 38.353592

Long: -6.364823

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, nos dirigimos 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 se recomiendan 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 ejecutable de la aplicación.

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

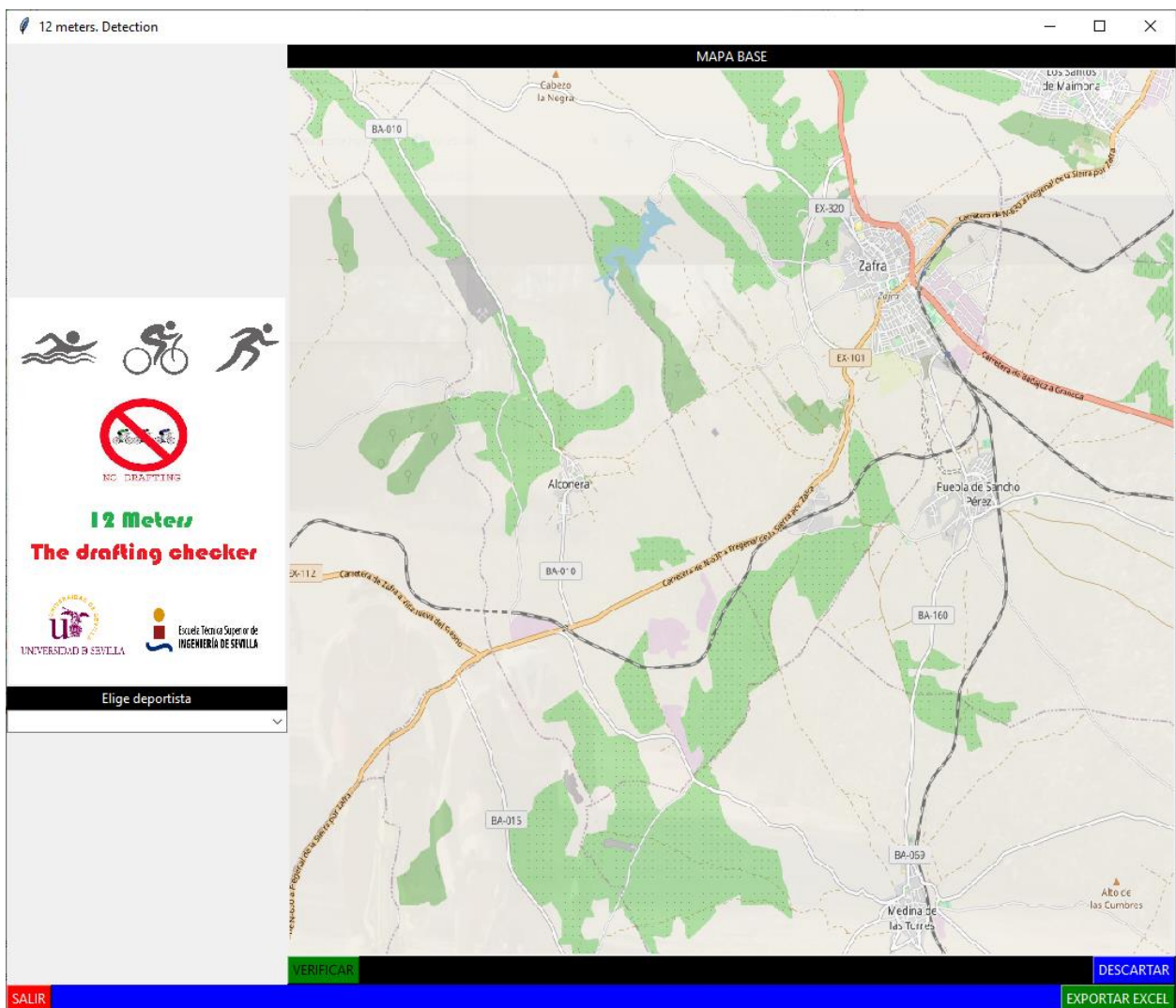
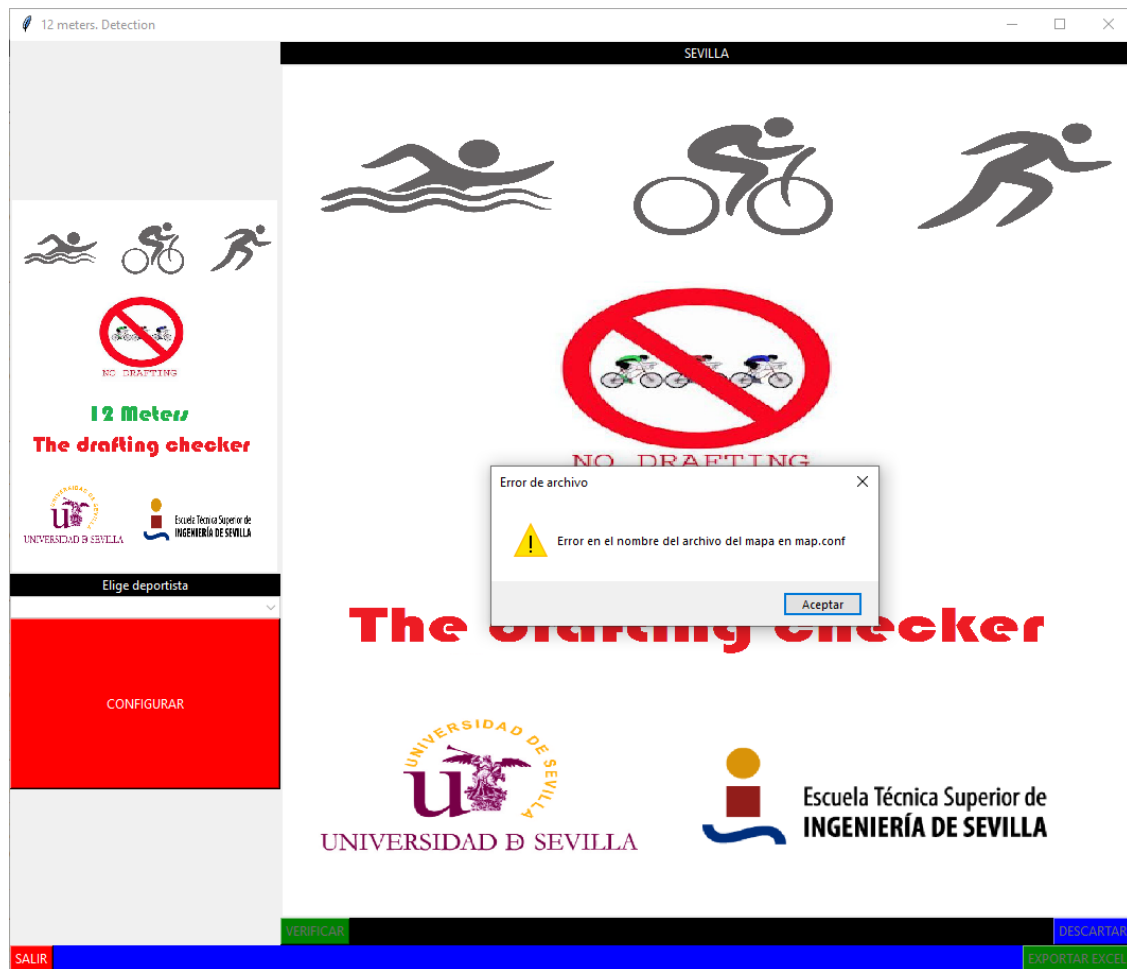


Si toda la configuración ha ido correctamente, aparecerá la pantalla principal. En caso de que alguna de las configuraciones previas haya sido defectuosa, se muestra el mensaje de advertencia pop-up correspondiente, y aparecerá el botón de CONFIGURAR habilitado y en color rojo, bloqueando el resto de 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. Tan sólo 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 **/*Imagen Desplegable*/**, y en caso de ser el primer deportista, será automáticamente seleccionado para su verificación. 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 **/*Imagen mapa custom*/**.

Una vez se ha seleccionado un deportista, se puede seleccionar 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 infracción seguirá guardado en local. Este es el ciclo de funcionamiento en la segunda transición.



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.

6.2 Servicio usuario

Meter imagenes de positivos negativos

Como buscar los puntos.

Meter pruebas de batería

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.

• **Añadir análisis de costo.**

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 entorno 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 transcurso 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 viabilidad 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 esta áreas no deberán tener, de manera obligatoria, forma rectangular.

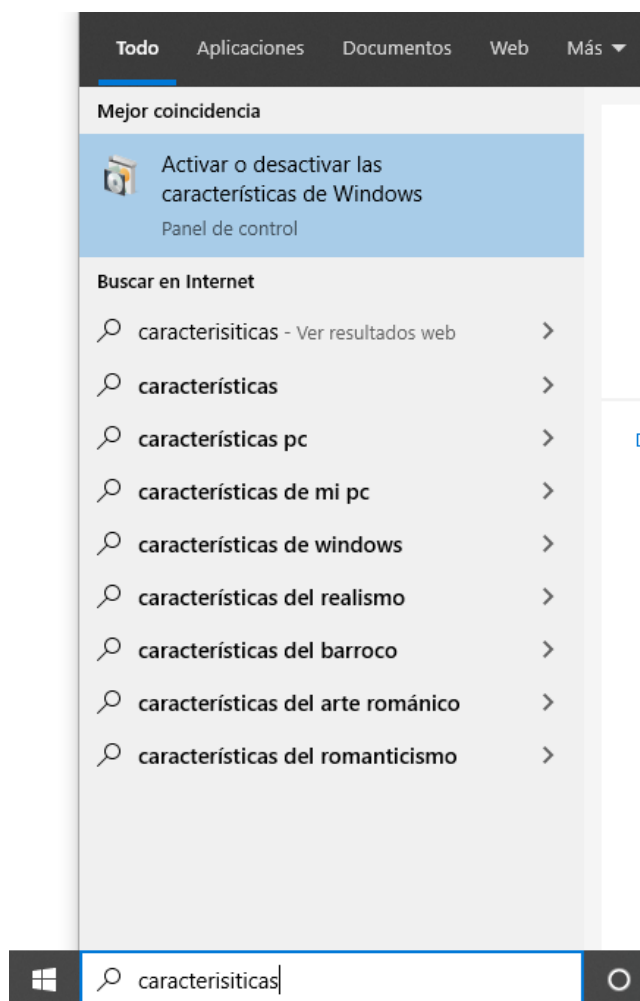
Requerimientos				Prioridad	Comprobación	
#ID	Elemento	Requerimiento	Descripción	Tipo	Check	Comentarios
#1	Detector	Detección de distancia drafting	Reconocer, en caso de estar incumpliendo la d	MUST		Sensor óptico
#2	Ambos	Comunicación de resultados	Comunicar con la aplicación de la organización	MUST		WiFi+Mosquitto
#3	Detector	Feedback deportista	Informar al deportista acerca de cuando está in	MUST		Led indicador e alarma sonora
#4	Ambos	Información en directo	Informar a la organización sobre si ha incumpl	COULD		
#5	Detector	Peso liviano	El peso del conjunto el dispositivo no superará	SHOULD		Sin incluir batería se cumple
#6	Detector	Aerodinámica	Integración en bicicleta tipo contrarreloj con im	SHOULD		Configurable
#7	Detector	Anclaje	Se podrá poner/quitar de la bicicleta de mane	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		
#12	Detector	Costo	El costo del dispositivo no debe superar los 50	SHOULD		
#13	Aplicación	Configuración servidor	La configuración del servidor no 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 transición	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	Código	El código fuente debe estar encriptado o comp	SHOULD		Ejecutable en aplicación organización
#23	Detector	Cambio de sentido	El detector podrá interpretar en que sentido	SHOULD		

Anexo 1. Instalación y problemas

Instalación de Windows subsystem for Linux

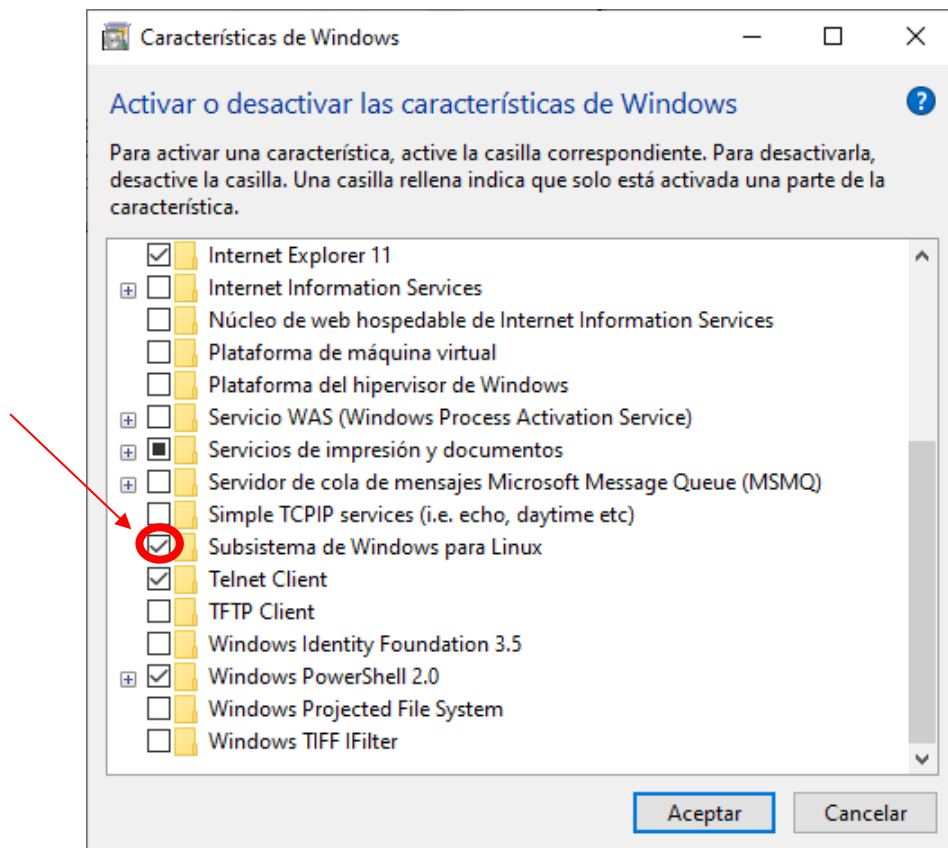
Para la instalación del terminal de Ubuntu para Windows 10, debemos tener una versión de este sistema operativo posterior a 2016, lo cual es casi una necesidad debido al asistente Windows Update que fuerza las actualizaciones de Windows. En concreto este proyecto se referencia con una versión de Windows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

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

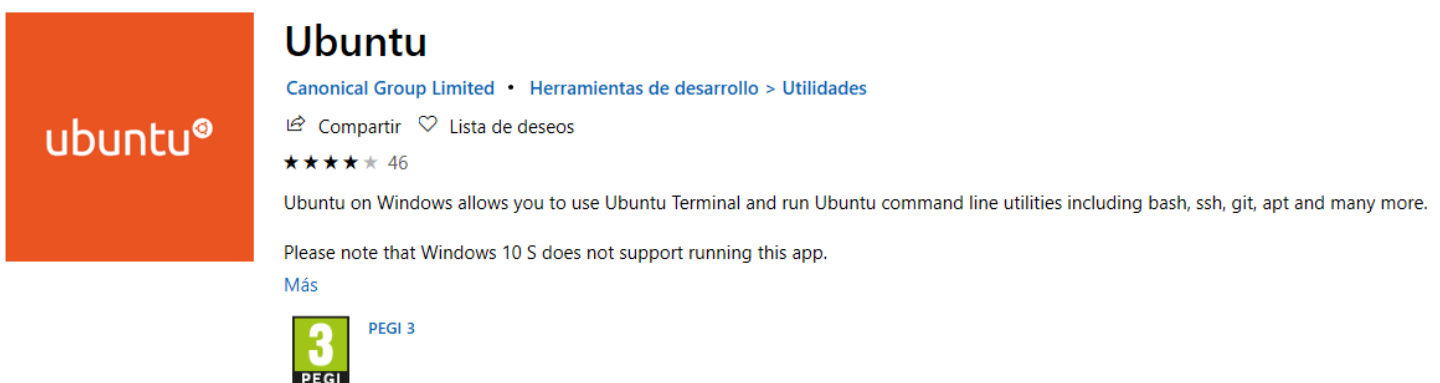


2. En esta ventana, se activa la opción del subsistema de Windows para Linux. En inglés se puede buscar

por “*Windows subsystem for Linux*”.



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



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 recomienda 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 globales para el acceso a las carpetas más comunes, como mínimo, ya que no resulta nada intuitivo compaginar el sistema 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

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

pueden resultar problemáticas) habrá que conocer su ruta de acceso desde Windows Explorer.

Problemas al lanzar servicio mosquito

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 mosquito.

Esto ocurre cuando, se lanza el broker, pero si se comprueba su estado, no está funcionando correctamente. Para comprobar el estado se puede ejecutar el comando:

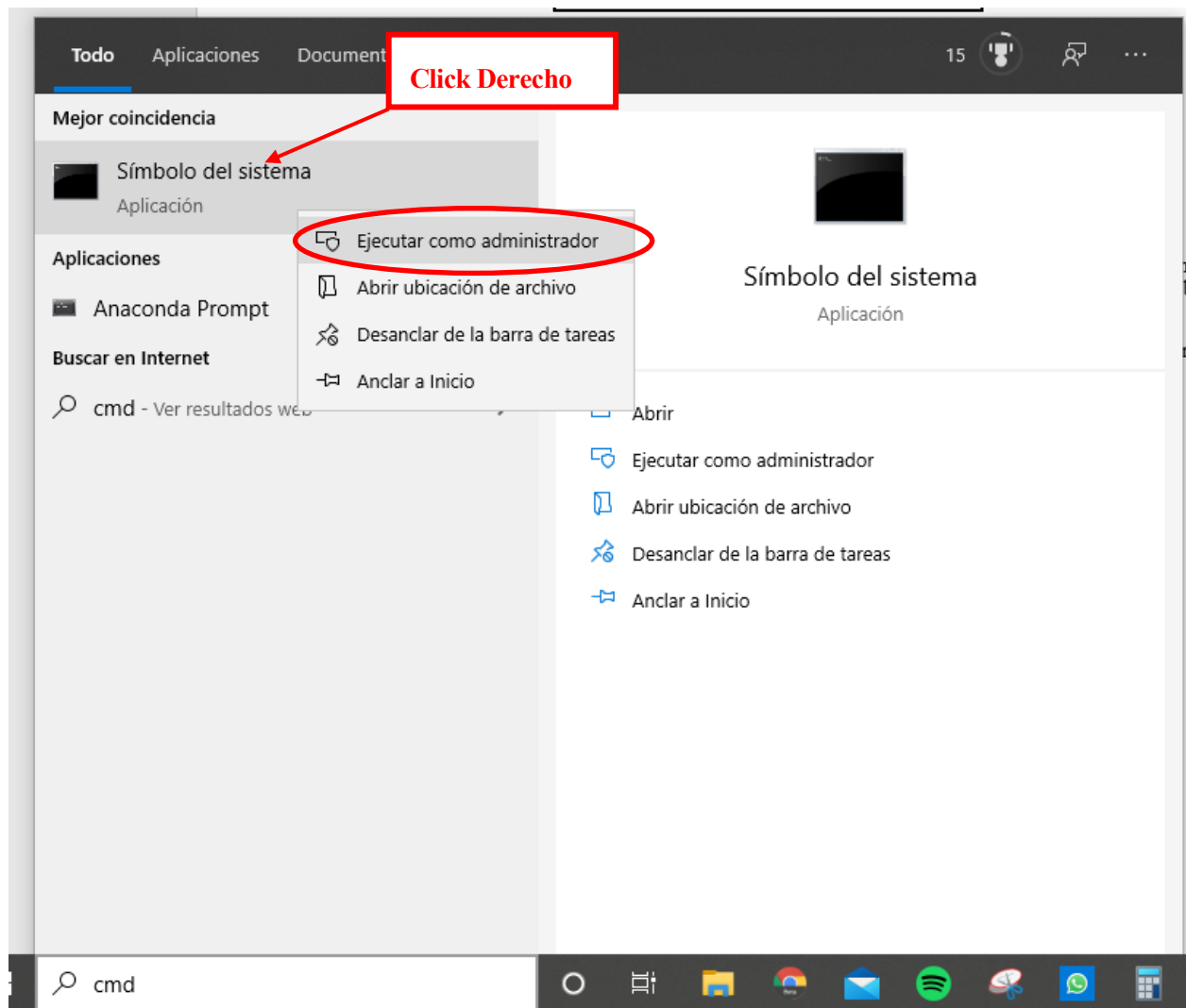
➔ ***sudo service mosquito status***

si devuelve:

➔ ****mosquito is not running***

Habiendo lanzado previamente el servicio será porque ha quedado bloqueado el puerto.

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



➔ **netstat -abo**

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:

```
Administrador: Símbolo del sistema
C:\WINDOWS\system32>netstat -abo

Conexiones activas

Proto Dirección local Dirección remota Estado PID
TCP 0.0.0.0:135 LAPTOP-E6MQN0ID:0 LISTENING 1204
RpcSs
[svchost.exe]
TCP 0.0.0.0:445 LAPTOP-E6MQN0ID:0 LISTENING 4
No se puede obtener información de propiedad
TCP 0.0.0.0:902 LAPTOP-E6MQN0ID:0 LISTENING 4820
[vmware-authd.exe]
TCP 0.0.0.0:912 LAPTOP-E6MQN0ID:0 LISTENING 4820
[vmware-authd.exe]
TCP 0.0.0.0:1883 LAPTOP-E6MQN0ID:0 LISTENING 4888
[mosquitto.exe]
TCP 0.0.0.0:5040 LAPTOP-E6MQN0ID:0 LISTENING 6972
CDPSvc
[svchost.exe]
TCP 0.0.0.0:8733 LAPTOP-E6MQN0ID:0 LISTENING 9688
[HPJumpStartBridge.exe]
TCP 0.0.0.0:49664 LAPTOP-E6MQN0ID:0 LISTENING 952
[lsass.exe]
TCP 0.0.0.0:49665 LAPTOP-E6MQN0ID:0 LISTENING 860
No se puede obtener información de propiedad
TCP 0.0.0.0:49666 LAPTOP-E6MQN0ID:0 LISTENING 1692
EventLog
[svchost.exe]
TCP 0.0.0.0:49667 LAPTOP-E6MQN0ID:0 LISTENING 2356
Schedule
```

➔ ***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 mosquito en Linux.

Anexo 2. Código

Pegar aquí todo el código como anexo

read.py

gps.py

sendInfo.py

main.py

Meter tb los ficheros de configuración.

