SELECCIÓN DE TECNOLOGÍA

# Dispositivo detector

## 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 tales como protocolos de comunicación, GPIOs…

## Detector

El detector es en cierta medida la esencia del proyecto, es el elemento que nos va a permitir detectar a los infractores. En este aspecto hemos barajado varias opciones tecnológicas que paso 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 procesado de la distancia en base a la triangulación. El reconocimiento de distancia mediante imagen sería factible, y el costo y tamaño de la solución encajan en las especificaciones, pero la capacidad de cómputo en dispositivo de estas dimensiones 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 prometen dar hasta 10m de distancia de detección. Se ha realiza 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. (Incluir pruebas?).

Sensor óptico de distancia, esta ha sido la opción final usada para el proyecto, en concreto se ha decidido usar el TF Mini Lidar de Seed Studio. Este sensor según especificaciones detecta hasta objetos hasta ha 12m de distancia en condiciones interiores y hasta 7m en condiciones exteriores. Las pruebas realizadas comprueban que supera la barrera de los 10m fácilmente en condiciones estándar externas. Trabaja a una tensión de alimentación de 5V con un consumo de 0.6W. Da lecturas de distancia a 100Hz con una resolución de 1cm, una precisión del 1% por debajo de 6m y del 2% a más de 6m, y un ángulo de apertura de 2.3 grados. Sus dimensiones son de 42x15x16mm, pesa 5 gramos y puede trabajar de -20 a 60 grados centígrados. La comunicación es bajo el protocolo UART. A excepción del ángulo de apertura, el cual sería deseable que fuese mayor, todas estas especificaciones se adecuan bastante bien a las necesidades de nuestro proyecto.

El funcionamiento del sensor óptico se basa en un emisor laser y un receptor, 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. (Incluir fotos, datasheets, protocolo etc…)

## 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 usará 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*. Para ello se ha usado el dispositivo GPS GY-NEO-6MV2, este pequeño y económico sensor GPS nos da una señal de entorno a 2.5 metros de precisión. Está formado por una antena cerámica de 27x27x6.5 mm y el sensor en si de 36x26x4 mm. Cabe destacar que el dispositivo tiene un gran condensador que mantienen hasta dos semanas alimentada la memoria, esto es para poder realizar el arranque en caliente, ya que en frío, al no tener información alguna de la localización puede tardar hasta varios minutos en encontrar una primera localización. Una vez consequida la localización nos refresca datos de localización a una frecuencia máxima de 5Hz. La información que obtenemos por UART son tramas GPS estándar, las cuales, pese a no disponer de una librería específica para este sensor en microPython (mientras si que las hay para Arduino o Rasbperry Pi) son fácilmente interpretables ya que es un código ASCII muy conocido.

Buzzer

Leds

# Aplicación organización

Para desarrollar la aplicación que de soporte a la organización se pretende continuar con el uso del lenguaje de programación Python, 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.

El framework elegido para el desarrollo de la interfaz gráfica ha sido Tkinter para Python3. Este framework es gratuito, bastante sencillo de usar y se puede usar el IDE Spyder de Anaconda para su desarrollo, lo que nos facilita bastante el desarrollo.

Además de esta librería, para algunas tareas específicas que iremos viendo a lo largo de desarrollo técnico se han utlilzado librerías de tratamiento de datos como Pandas y NumPy, así como la interfaz para comunicaciones con el dispositivo paho-mqtt.

# 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.

Ahora bien, bajo el amparo del wifi, se va a implementar el uso de del protocolo MQTT. Este protocolo nos permite comunicar múltiples dispositivos con la arquitectura bróker-cliente sobre TCP/IP . Dentro de una red local wifi, en una de las IPs que tenemos en esta red habrá lo que se conoce como un bróker mqtt activo. Este bróker lo que hace no es más que hacer lo que sería la función, si hacemos la analogía, de un tablón de anuncios. Los clientes tendrán topics sobre los que pueden publicar y/o subscribirse, los topics serían los distintos apartados del tablón, cuando algún cliente publica algo sobre en un topic, el bróker se encarga de distribuírselo a todos los clientes subscritos a ese topic.

La única restricción para que un cliente pueda contactar con el bróker es que, una vez iniciado el bróker, se conozca la IP del mismo, y cualquier otro dispositivo que esté en su misma red local tendrá alcance a ese bróker, lo que hace que la red inalámbrica sea ampliable a, por ejemplo, todo un área de transición en una competición muy multitudinaria. En ningún momento necesitamos conocer ninguna IP o MAC específica de ningún dispositivo que no sea el bróker, lo que nos facilita dos aspectos de la solución: la capacidad de tener un bróker en una computadora o servidor distinto al que se use para detectar la aplicación de la organización, y la posibilidad de dejar todas la IP’s del resto de dispositivos a asignación dinámica, lo que simplifica significativamente la comunicación y configuración.

El protocolo es muy liviano en cuanto a la capacidad de cómputo para hacerlo funcionar, y permite crear una red muy distribuida de clientes, además, existe un script que facilita el trabajo con este protocolo en microPython, para la parte organizativa también existe una completa librería para python3, y el bróker puede correr en un cualquier computadora, en nuestro caso un ordenador portátil con Windows 10, aunque como ya explicaremos más adelante el bróker en concreto corre sobre una arquitectura Linux para Windows. Todas las soluciones utilizadas son de código abierto tal y como iremos viendo según avancemos en el desarrollo técnico.

Esquema general de funcionamiento

# Funcionalidad general 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, por lo que la organización sólo necesitará de un juez para verificar los resultados generados por el dispositivo.

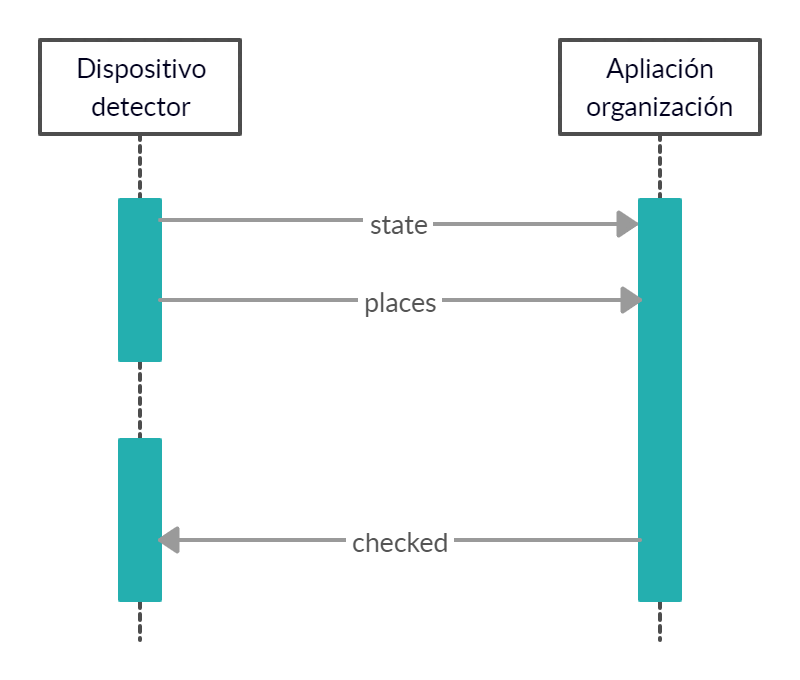
Poniéndonos en situación, fin de semana de competición de triatlón sin drafting, y se está usando nuestra solución detectora de la distancia de drafting. En el momento de la entrada en transición, cuando se realizan las diferentes verificaciones técnicas, al igual que se suele hacer con el chip, se entregará 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. En este punto nuestro 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.

Concluye la prueba, y el deportista regresa al área de transición, en la cual el abandona su bicicleta y se va a realizar el recorrido a pie. Es en este punto en el que nuestro dispositivo conectará con la red local que habrá montada en el área de transición, y mandará la información que ha recabado a lo largo de la prueba. 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 en que zona y la continuidad de esta infracción sobre el mapa base de la prueba. Aquí tendrá la oportunidad de descartar en caso de que considere que no es un positivo relevante, o aceptar la infracción. A mediad que vayan llegando participantes infractores irá recibiendo esta información en su aplicación. La información será exportable a un fichero Excel junto con la información del participante, creando así un registro de lo acontecido en la prueba de fácil interpretación.

# Esquema de comunicación

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 la imagen x, 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 trataremo 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.

# Estructuras de datos

## 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 si 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 de ”*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.

## Ficheros de configuración

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

De manera general, se ha decidido que en los ficheros puede haber comentarios que expliquen la estructura que debe seguir el archivo. Estas líneas tendrán como primer carácter el símbolo ‘#’ siguiendo así una estructura similar a la de un script de Python. Toda línea que con comience por el símbolo ‘#’ será tomada como una línea de datos.

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.

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.

# Tecnología utilizada

Se ha utilizado IDE abierto Spyder, incluido en Anaconda. Incluir texto sobre anaconda

Se han instalado los paquetes paho, pandas, matplotlib, tkinter, PIL. Añadir q es cada paquete y como se instalan

Por último, se ha creado un ejecutable con código fuente, con lo cual conseguimos que esta aplicación sea ejecutable en cualquier PC con Windows 10 sin necesidad de tener la instalación de Python con todos los paquetes para su ejecución. Explicar como se hace con anaconda prompt pyinstaller –onefile main.py

Aquí queda x meter un poco la explicación general de como se usa Tkinter. Y nose si de mqtt

# Estructura de la aplicación

La aplicación se basa en dos partes principales, por una parte, está la interfaz gráfica, y por otra la comunicación por Mqtt.

Al lanzar el ejecutable de python, internamente lo primero que hacemos es crear las variables globales (Completar vbles glob) que existen, y luego, crear una instancia de la clase Application, que es la que contiene la interfaz gráfica.

El constructor de la clase, ejecuta la función interna createWidgets, la cual, como su propio nombre indica, crea los elementos que componen la aplicación. En resumen, la aplicación se compone de tres Frame, uno inferior (downFr) y dos superiores (rightFr y leftFr), todos los elementos van asociados a un frame, y dentro del mismo tendrán una colocación específica. Vamos a pasar a ver uno a uno los elementos que forman la solución:

* QUIT: es el botón de salida de la aplicación, se le asocia el método salida y al downFr.
* EXCEL: este botón se usará para guardar de manera durante el uso la información actual en el archivo de Excel de salida. Se asocia al método save\_toExcel y al downFr.
* LOGO: es un label, con una imagen asociada, que se corresponde con el logo del proyecto. Se asocia al leftFr en su parte superior.
* CHOOSE: es un label con el texto “Elige deportista”, que se sitúa justo debajo del logo, asociado al leftFr.
* COMBO: este es el combobox en el que, en caso de haber deportistas pendientes de revisión aparecerán en un desplegable. Se sitúa justo debajo del label CHOOSE y asociado al leftFr. En su inicio por defecto aparece deshabilitado y vacío.
* reconn: este botón aparece en la parte inferior del leftFr, se utiliza para configurar la aplicación en caso de que no haya habido algún problema en la primera configuración.
* IMAGEN: este label contiene el título de la imagen de mapa mostrada, por defecto contiene el nombre del circuito. Este asociado al borde superior del rightFr.
* MAP: label que contiene la imagen del mapa. Se sitúa justo debajo del título, asociado al rightFr.
* Verify: botón que tiene asociado el método verificar, se sitúa en el margen inferior izquierdo del rightFr.
* Discard: botón que tiene asociado el método descartar, se sitúa en el margen inferior derecho del rightFr.

A continuación, y para concluir el constructor de clase, se llama al método configurar. Pasamos a comentar la funcionalidad de este y de los otros métodos y funciones restantes:

* **Configurar:** lo primero que haces es llamar a las funciones externas connMqtt y getData, y, en caso de que no generen ninguna excepción, se habilitan los botones de verify, discard, EXCEL y el desplegables COMBO. En caso de que salga alguna excepción se muestra un mensaje tipo pop up de advertencia, indicando que error es el que ha sucedido.
* **connMqtt:** en esta función configuramos la conexión al broker Mqtt. Por las peculiaridades de este proyecto, se ha decidido que el broker correrá sobre la misma IP en la que se abre la aplicación, aunque hay que tener en cuenta que el broker no se lanza al abrir la aplicación, debe haber sido lanzado previamente. En la función se configura el cliente, y se subscribe a los dos topics que nos llegarán de los dispositivos (state y places). En caso de que alguna de estas funciones no funcione apropiadamente, por regla general será la conexión al broker, generamos una excepción “Mqtt error”.
* **getData:** en esta función abrimos el Excel de información de triatletas, lo indexamos por la columna dorsal, y le añadimos las nuevas columnas en las que iremos añadiendo la información recibida de los dispositivos (Infractor, Places, DropTime y CheckTime).
* **on\_message:** esta función es la que se configura como callback cuando lleguen nuevos mensajes a los topics subscritos. Una vez decodificado el mensaje, trateremos los mensajes en función del topic del que provengan.   
  Si es del topic state, se guarda la información sobre si ha infringido o no en la distancia de seguridad, y si es que no, automáticamente publicamos el correspondiente mensaje en checked para apagar el dispositivo, y anexamos la fecha de DropTime.  
  Si es del topic places, se anexan en el campo correspondiente la información de las coordenadas de infracción, se fuerza si todavía no se conocía la información de infracción a que sea positiva, y se apaga el dispositivo confirmando la fecha de Droptime. Además, se llama al método de la interfaz newCheater, con el argumento del número de dorsal, para anexar la información para ser revisada.
* **save\_toExcel:** Este método de la aplicación está asociado al botón EXCEL, y también es llamado antes de cerrar la solución. Aquí guardamos la información en el archivo de salida. Para ello usamos el método ExcelWriter de pandas. En caso de que haya algún problema, como por ejemplo que ese archivo esté abierto en otra aplicación y no pueda ser sobrescrito, mostramos un mensaje de error y retornamos el código ERROR.
* **Verificar:** Esté método está asociado al botón verify, en el se obtiene el valor seleccionado del combobox, se elimina este dorsal del comobobox, completa el campo CheckTime, y muestra el mapa base en pantalla.
* **Descartar:** Este método es similar al de verificar, sólo que en este caso lo tratamos como un falso positivo, por lo que en lugar de limitarnos a completar el campo checkTime, lo que hacemos es limpiar la información de la infracción y poner el campo Infractor a NO.
* **Salida:** Está asociado al botón QUIT, aquí lo que se hace es, llamar al método sabe\_toExcel, y en caso de que este funcione bien, cerrar la app con el método propio quit.
* **newCheater:** este método lo único que hace es añadir a las opciones del combobox el número de dorsal que se le pasa por argumentos.
* **newSelection:** Es el método de interacción con el combobox, configurado mediante la opción bind en la creación del widget. Lo que hacemos aquí es mostrar la imagen de mapa base, con los puntos en los que se ha cometido infracción superpuesto sobre ella. Para ello tenemos el conjunto de coordenadas en la variable Box, estas coordenadas son las correspondientes a los extremos del mapa base. Las coordenadas almacenadas en el campo places, se superponen usando la herramienta pyplot para superponer los puntos como puntos de una gráfica cuyo fondo es el mapa base. La figura de pyplot, se guarda como una imagen, que además es guardada en el directorio de trabajo a modo de complemento de información para la infracción, y se muestra en la apliación. El nombre de la imagen guardada será el número de dorsal más el apóstrofe map con extensión png.

DESARROLLO DETECTOR

# Estructura del detector

El desarrollo sobre microPython para el equipo detector se ha realizado en tres scripts diferentes. Vamos a pasar a comentar cada uno de ellos. Además en el fichero boot.py se inicializan los componentes correspondientes.

### Conexión a SSID

El script sendInfo.py es imporado desde el script principal, read.py 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 desarrollada por xxxx y que, tal y como veremos a continuación facilita mucho el trabajo ya que implementa el protocolo a más bajo nivel de Python, abriendo los sockets y codificando en el método correcto.

/\*Describrir umqtt\*/

Al importar el script, lo primero que hacemos es leer el archivo de configuración wireless.conf /\*Añadir timeout de 5’ como parametro\*/ el cual sigue las estructura ya comentada para estos archivos. 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, se abre el archivo cheat.log, en el que se ha almacenado los casos de infracción., y 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 /\*Meter esto como param configurable\*/ 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.

### GPS

El script de gps.py está diseñado para ser importado desde el script principal en un nuevo hilo. En el 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 las 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. /\*Añadir diagrama de flujo\*/

Se configura la UART del dispositivo en los pines 17 y 16, a 9600 baudios, 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 ACII. Esto son tramas GPRSM, las cuales son del tipo /\*Añadir biográfico GPRSM\*/.

Cuando llegamos a la trama de tipo GPLL, que es de la que estamos extrayendo la información de la ubicación, comprobamos que esta no esté proporcionando información sobre la ubicación. El sensor GPS siempre proporciona información por la UART, pero las celdas correspondientes a la latitud y longitud solamente vendrán rellenas en caso de que esté posicionado. Si no tiene cobertura devolvemos el mensaje “PASS”, que indica que no se deben detectar positivos.

En caso de que tengamos cobertura, se parsea la información, para devolver la latitud en grados decimales. Es decir, en lugar de tener Grados, minutos, segundos y sentido (Este, Oeste, Norte o Sur), trabajaremos con grados con un valor decimal y signo positivo o negativo en lugar de la orientación.

Una vez calculada la ubicación, se comprueba si estamos dentro de alguna safeZone, si es así devolvemos el mensaje “PASS”, y dormimos al micro por 10 segundos /\*Meter los 10” en archivo config\*/. A continuación, comprobamos si estamos en el área de conexión, en la marca temporal requerida; si es así, devolvemos el mensaje “END”, para que el script principal llame al bloque de importación.

Por último, si tenemos cobertura, y no estamos en ningún área especial, se devuelve la ubicación y la fecha en la que nos encontramos.

Lo más destacable de este hilo, es que cuando nos referimos a devolver un mensaje, el hilo no acaba, porque para ello hemos usado la función yield, que va devolviendo valores al hilo principal (más adelante se verá como tratamos la información), pero no finaliza la función, con ello conseguimos crear una comunicación entre ambos hilos sin la necesidad de abrir algún tipo de pipe de comunicación o relanzar el hilo más de una vez.

### Lecturas distancia, ciclo principal

Este hilo es el principal, el que se lanza desde el fichero boot.py al conectar a alimentación el dispositivo, y desde el que se importan los otros bloques del proyecto para su correcto funcionamiento.

Al iniciar la función run, lo primero que hacemos es lanzar en un nuevo hilo, la función actGPS, la cual va recogiendo los valores que son aportados períodicamente por el bloque GPS, y los va copiando a la variable global currGPS, con lo que lo hacemos accesible desde el hilo principal del programa. La función del GPS se va recorriendo como si de un vector se tratase, de esta manera no finaliza nunca, ya que el GPS aporta valores de manera indefinida.

Lo siguiente que se realiza es la configuración del sensor de distancia, para ello llamamos a la función setLidar, la cual nos devuelve el objeto UART asociado al detector. El lidar, está configurado en la UART 2, a 115200 baudos y en los pines 14 y 32 del ESP.

Lo último que configuramos es el pin 33 como un pin de salida, a él irá conectado el led indicador de infracción, y abrimos el fichero cheat.log /\*Revisar w+ del cheat.log\*/.

Una vez en el bucle principal /\*Incluir diagrama de flujo del bucle\*/, cuya condición de salida es que el detector GPS devuelva el mensaje END, se realiza el siguiente proceso:

/\*incluir bucle\*/

Al finalizar el bucle principal. Se fuerza la finalización del hilo GPS, se cierra el fichero cheat.log y se llama a la función run del bloque sendInfo, hasta que este finalice el trabajo del dispositivo durmiéndolo de manera indefinida