Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Red de sensores para la gestión de aparcamiento en vía urbana

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Álvaro Martínez Fernández

Tutor: Pietro Manzoni

2020/2021

Resumen

La continua evolución de las tecnologías de comunicación ha cambiado totalmente nuestro estilo de vida. Este cambio está marcado principalmente por la aparición de una comunicación global, ya sea entre personas o entre dispositivos. Esta evolución tiene como meta general hacer más fácil la vida de los humanos; claros ejemplos de estas mejoras son lo fácil que resulta hacer una videollamada con una persona esté donde esté, lo rápido que funciona la logística mundial o la cantidad de redes sociales que existen para poder comunicarnos entre nosotros.

Además, existe una tendencia que lleva a las personas jóvenes a mudarse a las ciudades más grandes de España, impulsadas, mayoritariamente, por la gran cantidad de ofertas de trabajo y la ilusión por una mejora en el nivel de vida. Esto se traduce en una superpoblación en las ciudades españolas y provoca que una gran parte de los núcleos urbanos de pequeño tamaño estén sufriendo una creciente despoblación.

Junto a estos factores, la poca costumbre del uso del transporte público entre la población española provoca que haya demasiados vehículos privados. Esto genera problemas en la gestión del tráfico dentro de la ciudad. Uno de esos problemas es la escasez de plazas de aparcamiento en la mayoría de calles céntricas de cualquier ciudad.

Estos tres motivos han sido las razones que han impulsado a la creación de este proyecto, que consiste en un sistema para buscar plazas de aparcamiento disponibles en una ciudad. A lo largo de este trabajo se explicará el proceso de creación del proyecto, desde el estudio y análisis de las necesidades hasta las pruebas finales para comprobar que el funcionamiento es el correcto.

Este sistema consistirá en una red de microncontroladores con los sensores necesarios para detectar si hay un coche en su plaza asignada. Estos dispositivos se comunicarán entre ellos y crearán su propia red. En cada red habrá un nodo máster que se encargará de gestionar toda la comunicación desde nuestra red a internet. Toda la información sobre las plazas libres se guardará en una base de datos en un servidor dedicado.

El usuario tendrá acceso al sistema mediante una aplicación Android, que mostrará en el mapa todas las plazas libres que hay, indicará cual es la más cercana y, si se desea, empezará la navegación hasta ese punto.

Abstract

The ongoing evolution of communication technologies have completely changed our way of life. This change is mainly marked by the introduction of a global communication, among people or digital devices. The aim of these developments is to make human life easier; some of the clearest examples are: how easy it is to make a videocall with someone wherever they are, how rapid worldwide logistics work and the number of social networks that enable us to communicate with one another

Furthermore, there is a tendency that leads young people to move to the biggest cities in Spain, pushed, mainly, by the great number of job offers and the hope for an improvement in their way of life. This is reflected in overpopulation in Spanish cities and causes smaller urban centres to suffer from depopulation.

Along with these factors, the fact that a great number of Spanish citizens are not used to public transport entails an excess of private vehicles in city centres. As a consequence, this causes problems regarding traffic management within a city. One of those problems is the shortage of car parking spaces in the majority of main streets of any city.

These are the three reasons that have motivated this project, a sensor network which looks for free parking spaces in a city. Along this dissertation, the process of creating this project will be explained, from the study and analysis of needs to the final tests to verify that the functioning is good.

This system consists of a network of microcontrollers with the necessary sensors to detect if there is a car in its assigned space. These devices will communicate with each other and create their own network. In each network there will be a master node that will be in charge of managing all the communication from our network to the internet. All the information about free spaces will be stored in a database on a dedicated server.

The user will have access to the system through an Android application, which will show on the map all the free spaces available, indicate which is the closest and, if desired, start the navigation to that point.

Tabla de contenidos

1. Introducción

1.1. Motivación

A lo largo de toda la carrera he trabajado con diferentes tecnologías y metodologías que me han ido haciendo crecer como persona y han cambiado mi forma de trabajar. Como broche final a estos cuatro años, considero que el trabajo de fin de grado será la forma más directa y sincera de plasmar todo lo que he aprendido y de transmitir todas las ganas y el cariño que he dejado en él.

El momento de escoger una temática siempre es difícil, ya que será algo en lo que se tendrá que trabajar durante muchas horas y puede que incluso tenga trascendencia después de acabar la carrera. En mi caso, esta decisión ya estaba tomada desde hace tiempo, ya que era un proyecto que me parecía muy interesante y, sobre todo, ideal para una ocasión como esta.

Los motivos se pueden dividir claramente en personales y profesionales.

El primer motivo y más importante es el de intentar superarme a mí mismo con un nuevo reto que, sin lugar a dudas, ha estado lleno de dificultades, pero que, a lo largo de su desarrollo, me ha aportado mucho conocimiento y experiencia en este campo. Desde el primer momento supe que enfrentarme a un proyecto de este tipo me haría aprender mucho y de muchas disciplinas diferentes.

Durante estos últimos años, en mi tiempo libre y como hobby, he estado desarrollando múltiples proyectos relacionados con microcontroladores de software libre. Entre otras cosas he realizado sistemas de domótica, contadores de aforo o robots móviles, pero lo que siempre tuve pendiente, era hacer una red sensores autónoma. Hacer un trabajo sobre algo que considero mi hobby es algo que me motiva mucho.

Esta idea vino causada por la cantidad de tiempo que perdía diariamente buscando sitio donde aparcar mi coche en las calles del centro de Valencia. Durante todas estas ocasiones fui pensando cómo haría para solucionar este problema, que como a mí, le ocurre a muchas personas. Tras mucho tiempo pensándolo y muchas charlas con profesores y amigos debatiendo cuál sería la mejor solución, creo que he llegado a una solución realmente viable.

En cuanto a las motivaciones profesionales, destacaría la oportunidad de hacer un proyecto a gran escala que me pueda servir como “carta de presentación”. Destacar en un campo como este no es tarea fácil, ya que actualmente hay mucha gente en el mundo de la ingeniería con mucho talento. Me gustaría servirme de este proyecto para demostrar todos los conocimientos que tengo en diferentes campos de las tecnologías de comunicación, para así, en un futuro cercano, poder facilitar mi incorporación al mundo laboral.

Durante los últimos años, hemos entrado en una nueva revolución industrial llamada “Industria 4.0”, basada en la conversión de la industria de lo físico a lo digital. Una de las tecnologías que está haciendo posible esta transformación es “IOT” (Acrónimo de “Internet Of Things”, que se traduce como “Internet de las Cosas” en español), que hace referencia a la interconexión de los dispositivos electrónicos, las personas e internet. Esto

hace que cada vez haya más dispositivos inteligentes que funcionen de forma autónoma y, además, que estén conectados a la red para enviar información y actualizarse en todo momento. Una red de sensores es un ejemplo perfecto de lo que sería un sistema de IOT.

También es una gran oportunidad para presentar el proyecto a ayuntamientos que estén buscando diferentes métodos de automatizar su núcleo urbano para poder convertirse en una Smart City. A día de hoy son bastante comunes los concursos propuestos por ciudades para financiar proyectos para actualizarse utilizando nuevas tecnologías. Un proyecto como este encaja perfectamente con este perfil de concursos, lo que hace que sea un motivo más para seleccionarlo.

1.2. Objetivo

Este proyecto tiene como objetivo crear un sistema que sea capaz de gestionar las plazas libres de aparcamiento en la vía pública de cualquier "Ciudad inteligente" para que cualquier habitante sepa cuál es el sitio más cercano libre a través de su teléfono móvil.

Uno de los objetivos principales del proyecto es que el sistema sea lo más fácil de instalar posible, incluso pudiendo ser "*Plug & play*". Además, el coste de mantenimiento podría llegar a ser muy bajo o prácticamente nulo.

Además, en este proyecto se busca una solución que sea autónoma, que no necesite mantenimiento ninguno durante todo el tiempo que pueda estar funcionando.

Otro de los objetivos principales del proyecto es intentar hacerlo lo más económico posible, para que pueda ser accesible para el máximo de personas posible, o incluso convertirse en una oportunidad de negocio real.

En resumen, los objetivos generales del proyecto son los siguientes:

- Crear un sistema que pueda contabilizar las plazas libres de una calle
- Ser capaz de actualizar esa información en una base de datos
- Diseñar una aplicación para dispositivos móviles para que los usuarios puedan saber qué plazas están libres
- Hacerlo con el mínimo presupuesto posible
- Que el sistema sea autónomo

1.3. Impacto esperado

De este proyecto puede resultar un producto muy atractivo para todas aquellas ciudades que estén en el proceso de convertirse en una "Smart City". En el caso de poder instalar nuestro sistema por todas las calles o la gran mayoría de ellas, significaría un gran paso en la dirección correcta. De este modo, se mejoraría a vida en la ciudad a la hora de circular con el coche y por lo tanto se podría llegar a disminuir la contaminación.

A nivel de usuario queremos ofrecer una aplicación intuitiva y rápida de usar. Creemos que la clave para que nuestro producto sea útil es la eficacia. Simplemente con dos toques en la pantalla, el usuario debería saber cuál es la plaza más cercana y sería inmediatamente guiado hasta allí.

2. Estructura del proyecto

En documento se estructurará de la siguiente manera:

En primer lugar, empezaremos con un pequeño estudio del problema que queremos solucionar. A continuación, investigaremos cuáles son los puntos más difíciles de abarcar con nuestra solución y enumeraremos todos los problemas que nos podemos encontrar en el desarrollo de la misma.

Una vez tengamos claro el problema que queremos solucionar y este se haya explicado correctamente, lo siguiente será empezar a hacer un estudio de todos los posibles diseños que se podrían desarrollar para solucionarlo. En esta parte, las analizaremos una por una con mucho detalle, incluso haciendo una pequeña comparación entre ambas para poder saber cuál sería la opción más viable. También se hará un pequeño estudio de todos los riesgos que supone cada opción.

Lo siguiente será explicar qué opción se ha escogido a la hora de hacer el proyecto y por qué nos hemos decidido por esa. A continuación, se expondrá el plan de trabajo por el que ha pasado el proyecto a lo largo de su desarrollo y una explicación de todos los detalles de las etapas y el presupuesto utilizado en cada una de ellas, así como el total utilizado para el proyecto.

A partir de este punto se hará una descripción exhaustiva de todo el diseño del proyecto. Empezaremos explicando cómo hemos ido solucionando los problemas presentados al principio del documento, y cómo esta reflexión hará que lleguemos a cumplir nuestros objetivos. Se presentará cada parte que comprenda el sistema al completo, qué función tiene dentro de la solución y cómo se conectan todas las tecnologías entre sí.

Seguidamente, se explicará cómo se realizará la implantación del sistema, especificando cómo se deben colocar los sensores, cómo configurarlos... También se hará una guía de instalación, para que se pueda comprender a la perfección cómo poner en marcha el sistema y dejarlo funcionando.

Una vez se haya hecho la instalación nos quedará el apartado de pruebas, en el que llevaremos al límite nuestro sistema para saber hasta dónde es capaz de llegar. Entre otras cosas, se comprobará que el funcionamiento es el esperado, se estudiará cuál es el rango máximo del sistema y cuánto tiempo de autonomía tiene.

Para finalizar, expondremos las conclusiones del proyecto, presentando todos los pros y los contras del sistema, y si sería una solución viable dentro del mundo comercial. Este apartado tendrá una gran relación con el apartado anterior, ya que todas las pruebas realizadas anteriormente nos servirán para su perfecto estudio y serán la base para detectar mejoras para una nueva versión.

3. Análisis del problema

En Valencia, cómo en la mayoría de ciudades, existe una centrificación que provoca que la movilidad en vehículo sea bastante complicada. Esto, sumado al uso excesivo que tendemos a hacer de nuestro vehículo personal, genera otros muchos problemas. En este caso, el problema que nos incumbe es la falta de espacio o sitios para aparcar el coche en la vía pública.

Una gran parte de la población deja su automóvil aparcado en la vía pública, pero dependiendo de la zona en la que el usuario resida, esta tarea puede ser bastante complicada y puede llevar demasiado tiempo encontrar un lugar donde se pueda dejar el coche.

A parte de ser una gran pérdida de tiempo, se trata de otra pequeña fuente de contaminación que esperamos solucionar con nuestro sistema.

3.1. Identificación de posibles soluciones

Como ya hemos adelantado anteriormente, nuestra solución se podría dividir en varias partes: el método de detección de las plazas libres, la interfaz para interactuar con los usuarios y un sistema donde guardar toda la información.

De todas estas tres partes la que sin lugar a dudas nos ha dado más problemas ha sido el método de detección. Se nos han ocurrido multitud de maneras de controlar las plazas libres, pero solo dos han evolucionado como posibles soluciones.

La primera de ellas sería un sistema de cámaras con visión artificial. El funcionamiento estaría basado en el reconocimiento de imágenes captadas por las diferentes cámaras colocadas en la calle para saber y reconocer diferentes sitios libres en los que puedas dejar el coche. Este sistema sería bastante complejo computacionalmente, ya que se necesitarían bastantes cálculos para reconocer las diferentes plazas. Además, cada cámara debería ser programada de manera diferente a las demás, puesto que ninguna calle sería igual a la anterior, así como tampoco sería igual la colocación de la cámara o incluso la iluminación. Su gran punto fuerte es que un solo dispositivo sería capaz de gestionar una calle entera o al menos una gran porción de esta, además de que su sistema de comunicación no debería de ser muy complicado.

Como segunda opción tendríamos una red de sensores. Esta red se colocaría en la acera, al lado de cada aparcamiento, y sería capaz de detectar si hay un sitio libre o no gracias a un sensor de proximidad. En este caso, la lógica del software sería mucho más sencilla, dado que simplemente habría que tratar una señal digital recibida desde un sensor para saber si un sitio está libre o no.

Por otra parte, nos encontramos con que una instalación así sería bastante compleja dado que no solo hay que preocuparse de cómo subir los datos a nuestro servidor, sino que también hay que tener en cuenta como realizar la conexión entre todos los sensores para que tengan comunicación entre ellos.

	Fácil instalación	Complejidad computacional	Complejidad infraestructura	Precio	Escalabilidad	Fiabilidad

Visión artificial	Sí	Mucha	Baja	Bajo	Costosa	Baja
Red de sensores	Sí	Poca	Mucha	Bajo	Fácil de escalar	Alta

Tabla 1 Comparativa de soluciones

Otra parte de la solución es la interfaz con el usuario. En este caso, estábamos dudando entre realizar una página web o una aplicación para dispositivos móviles. Según nuestro punto de vista, una aplicación para móviles tendría mucho más sentido, simplemente por la diferencia de rapidez y compatibilidad que hay entre las dos alternativas. En este caso, simplemente abriendo la aplicación e indicando la plaza más cercana, el usuario ya estaría recibiendo las indicaciones sobre cómo llegar hasta ella; sin embargo, con una página web todo este proceso pasaría a ser mucho más complejo y lento.

En nuestro caso en concreto realizaremos una aplicación para dispositivos Android, ya que es accesible para un público más amplio y tiene menos complicaciones a la hora de publicar una aplicación, a diferencia, por ejemplo, de otros sistemas como puede ser Apple.

La selección de la base de datos está bastante influenciada por la selección de la interfaz. Hemos seleccionado una base de datos en formato JSON que lo hace mucho más fácil de interactuar a diferencia de una base de datos SQL. Concretamente nos hemos decidido por utilizar Firebase. Al tratarse de un producto de Google (al igual que Android), facilitará la integración de la base de datos en nuestra aplicación. Además, cuenta con una gran cantidad de documentación en su página web.

3.2. Análisis de la posible solución

Una vez realizado el estudio y comparativa de las diferentes soluciones hemos decidido lo siguiente. Utilizaremos una red de sensores a la hora de detectar las plazas libres en la calle. Esta red estará directamente conectada con la base de datos de Firebase para que esté actualizada en todo momento. Y como ya hemos comentado antes, se utilizará una aplicación Android como interfaz de usuario, que también estará conectada a la base de datos.

Al habernos decidido por una red de sensores, será necesario hacer un estudio de qué tipo de sensores serán utilizados, así como qué tipo de topología conviene más para este tipo de aplicación.

Una red de sensores consiste en varios nodos conectados entre sí para ofrecer una información de su entorno y subirla al servidor. Estos nodos se podrían dividir en tres partes fundamentales: el microcontrolador, el sensor y la fuente de alimentación. A continuación, haremos un pequeño análisis de las diferentes posibilidades de las tres partes para saber qué tecnologías se adaptan mejor a nuestras posibilidades.

3.3. Análisis de los microcontroladores

La parte más importante de cada nodo es el microcontrolador. Como su nombre indica, es el encargado de gestionar todas las señales eléctricas y convertirlas en información válida para el usuario final. En nuestro caso, este microcontrolador tiene que ser capaz de leer una señal eléctrica analógica (*) o, en su defecto, ser compatible con interfaces de

comunicación bastante usadas como pueden ser UART o I2. Además, debería de contar con la tecnología necesaria para realizar comunicaciones inalámbricas. Hay una gran variedad de tipos y tecnologías de comunicación, pero se le dedicará un apartado completo para hablar de cuál es la mejor para esta aplicación y porqué.

Un factor que se debe tener en cuenta en cuenta es la comunidad que tiene cada microcontrolador; aunque no es lo más importante, es de agradecer cuando una comunidad puede ayudar mucho en cualquier problema que pudiera surgir y lo agradable que resulta trabajar con sistemas en los que la gente ha desarrollado librerías con numerosas utilidades.

3.3.1. Arduino

A la hora de hablar sobre microcontroladores es muy común que se referencie a Arduino y, más concretamente, a su modelo Arduino UNO, el microcontrolador más usado durante la última década. Este modelo tiene numerosos puntos a favor, como la facilidad para programarlo, la gran comunidad y su precio. Su falta de comunicación inalámbrica hace imposible poder seleccionarlo para nuestra aplicación.

3.3.2. ESP8266

Otro grupo de microcontroladores muy conocidos son los de la familia ESP, fabricados por la empresa Espressif. Destacan sobre el resto de competencia por su reducido tamaño, la gran conectividad y su precio. Dentro de este grupo destacan los dos SOCs (*) más usados, que son el ESP8266 y el ESP32. Ambos son claros candidatos para nuestra aplicación, así que haremos un pequeño estudio sobre sus pros y sus contras.

En primer lugar, en cuanto al chip ESP-8266, destacan las siguientes características:

- Procesador de 32 bits de bajo consumo
- Velocidad de 80 MHz
- 64 Kb de memoria RAM
- Conectividad WiFi 802.11 b/g/n 2.4GHz
- 16 pines GPIO
- 1 pin entrada analógica
- UART, SPI y I2C
- Consumo medio de 80mA

Existe una gran variedad de módulos que integran este chip. Los dos más interesantes para nuestra aplicación son el ESP-01 y el ESP-12.

El módulo ESP-01 es muy conocido por ser uno de los módulos más pequeños de todo el mercado. Además, se suele utilizar mucho para dotar de conexión WiFi a otros microcontroladores como puede ser la Arduino Uno, citada anteriormente. Tiene un total de 8 pines, de los cuales solo se pueden utilizar 2, ya que el resto están reservados para funcionalidades básicas de la placa. Como, por ejemplo: la alimentación, la señal de encendido, la señal de reseteo y la comunicación serie para programarla. La falta de

pines, acompañado de la ausencia de entradas analógicas es uno de los grandes puntos en contra del módulo.

En el caso del otro módulo utilizado por el ESP8266, el ESP-12, se soluciona el problema de los pines, ya que cuenta con 9 entradas/salidas digitales, que en casos de usos comunes para estos dispositivos suele ser más que suficiente.

3.3.3. ESP 32

Dada la gran acogida que tuvo el chip ESP8266, Espressif decidió diseñar un nuevo microcontrolador, que salió al mercado en el año 2016. En este caso se trataba de un chip parecido al anterior pero mejorado en todos los aspectos. Estas son sus características principales:

- Procesador de 32 bits
- Velocidad de 240 MHz
- 520 KiB de memoria RAM
- Conectividad WiFi 802.11 b/g/n 2.4GHz
- Conectividad Bluetooth v4.2 compatible con BLE (*)
- 36 pines GPIO
- 16 entradas analógica
- UART, SPI y I2C

Como se observa en sus características, este chip es una versión mejorada del ESP8266. Entre todas las características destacan dos por encima del resto: la gran cantidad de pines de entrada analógica y la conectividad Bluetooth.

Al tener tantos pines de entrada analógica, facilita la posibilidad de instalar más sensores y aumentar la posible escalabilidad del sistema.

El Bluetooth abre una nueva posibilidad de conexión entre sensores que el anterior chip no podía ofrecer.

Es por estos motivos que se ha escogido el ESP-32 como el microcontrolador más efectivo para llevar a cabo este proyecto.

3.4. Análisis de los sensores

A la hora de detectar si en una posición hay un coche o no lo más lógico es usar un sensor de distancia y, dependiendo de lo que nos devuelva el sensor, seremos capaces de descubrir si está libre o no. De todos modos, existen una gran variedad de sensores de distancia diferentes. En este punto, haremos una comparación de las tres tecnologías más usadas y sus dispositivos más conocidos para estudiar cuál es el que mejor se adapta a nuestras necesidades.

Estas tres tecnologías son las siguientes: ultrasónicos, infrarrojos de triangulación y TOF (Time of Flight). Explicaremos cuál es el funcionamiento de cada sensor y cuáles son sus especificaciones.

3.4.1. HC-SR04. Ultrasónico

Estos sensores se asemejan mucho a lo que hacen los murciélagos para detectar obstáculos: lanzan un haz de sonido por uno de sus actuadores y cuentan el tiempo que pasa hasta que el receptor recibe ese sonido de vuelta. Conociendo la velocidad del sonido y sabiendo cuánto tiempo tarda en ir y volver esa señal, se puede saber a qué distancia se encuentra el obstáculo que tenemos delante.

Esta sería la fórmula para calcular la distancia:

$$distancia(m) = \frac{tiempo(s) * velocidad\ del\ sonido\left(\frac{m}{s}\right)}{2}$$

Estos sensores son muy utilizados en robótica educativa por la gran documentación que hay sobre ellos, el bajo coste que tienen y lo fácil de integrar que resultan en casi cualquier sistema. Cuentan con cuatro pines y solo tiene una salida, en este caso es una salida digital que es activada cuando recibe la señal de vuelta.

Como desventajas, este tipo de sensor tiene una baja calidad de medición y un bajo ratio de lectura, ya que hay que poner un mínimo tiempo de espera para evitar que lea rebotes anteriores.

3.4.2. Sharp GP2Y0A02. Infrarrojo de triangulación

Este sensor también consta de un actuador y un receptor. El actuador es un led infrarrojo que emite una señal luminosa, imperceptible por el ojo humano, que rebotará en el obstáculo más cercano y será recibida por el receptor. En este caso, el sensor no utiliza el tiempo de recorrido de la señal, si no que triangula su posición a partir de donde sale la señal (en este caso el led) y sabe en qué parte del sensor lineal se encuentra rebotada la luz. De esta manera se obtiene la distancia con el objeto delantero.

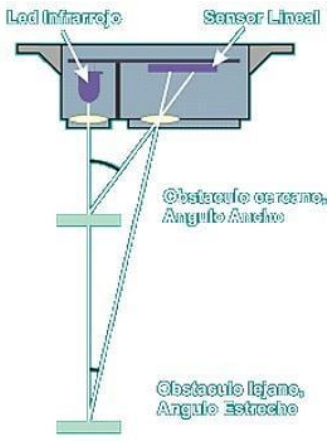


Ilustración 1 Sensor de distancia mediante triangulación

Este sensor utiliza una señal analógica para enviar los datos a nuestro microcontrolador. Esto es un punto importante, dado que algunos microcontroladores no tienen entradas analógicas y se tendría que instalar un conversor analógico-digital (también conocido como ADC) para que este fuese capaz de interpretar esta señal. Que la salida sea analógica hace que el ratio de lectura se limite al máximo ratio de lectura del microcontrolador, ya que el sensor en todo momento está devolviendo un resultado.

Uno de los aspectos más significativos que están en su contra es el rango de detección del propio sensor. Sharp en este tipo de sensores presenta una variedad con diferentes rangos de distancia; en nuestro caso hemos escogido el que va de 20 cm hasta 150 cm. Además de tener un rango muy corto, una lectura fuera de esos rangos puede provocar una lectura errónea debido a su forma de representar los datos. En la siguiente gráfica se puede observar que, en una lectura analógica concreta, puede tener dos opciones. Esto puede llevar a lecturas erróneas y, por lo tanto, a un error en el sistema.

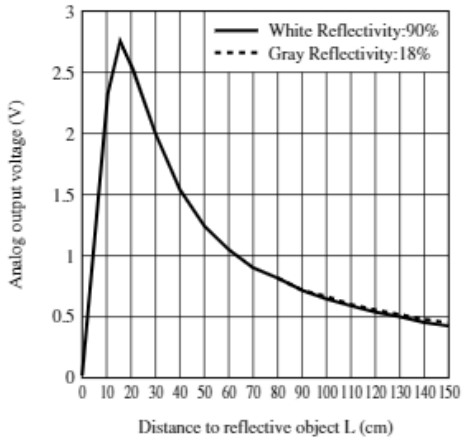


Ilustración 2 Relación entre el voltaje de salida y distancia detectada

3.4.3. VL53L0X. Infrarrojos de TOF

Este tipo de sensores tiene una funcionalidad muy parecida a los ultrasónicos analizados anteriormente. El sensor envía un haz de luz que rebotará y será recibido otra vez por el sensor. En este caso, también se trata de una luz infrarroja que no es apreciable por el ojo humano.

Por este motivo el sensor es mucho más rápido que el ultrasónico por una simple cuestión de física, ya que la luz viaja a 299 792 458 m/s mientras que el sonido viaja a 331 m/s. Este sensor es mucho más preciso que los dos analizados anteriormente, ya que devuelve las distancias en milímetros. Además, tiene un rango desde 50 mm hasta 2000 mm y, a diferencia del sensor de triangulación, este nos dará lecturas mucho más precisas.

La comunicación con el sensor se hace mediante I2C, que es un protocolo de comunicación serie muy conocido en el mundo de los microcontroladores, que además permite conectar hasta un total de 127 dispositivos en paralelo. Esto facilita la integración, ya que la mayoría de microcontroladores son compatibles con este protocolo.

Un punto más a su favor es el pequeño tamaño del sensor.

3.4.4. Comparativa

En la siguiente tabla compararemos las características más relevantes para nuestro caso de uso:

	Rango (m)	Precisión	Salida	Consumo	Frecuencia	Precio
HC-SR04	0,02 - 4	-	Digital	15mA	40Hz	Bajo
VL53L0X	0,05 - 2	1mm	I2C	19mA	400KHz	Bajo
Sharp	0,20 - 1,5	-	Analógico	33-50mA	Microcontrolador	Medio

Tabla 2 Comparativa entre los sensores

3.5. Análisis de tecnologías de comunicación

El análisis de tecnologías de comunicación es uno de los puntos más críticos de todo el proyecto, ya que la elección de la tecnología de comunicación correcta dependerá directamente en el tipo de microcontrolador que vayamos a escoger. Además, tendrá una gran relación con el gasto energético, ya que hay grandes diferencias de consumo eléctrico de un caso a otro. Dicho esto, el siguiente paso es comparar las opciones que se han barajado para este proyecto.

3.5.1. WiFi

Es la tecnología inalámbrica más conocida hoy en día en todo el mundo. Su nombre viene de Wireless Fidelity (traducido "Fidelidad inalámbrica"), aunque también se conoce por su estándar IEEE802.11. Transmite los datos de la misma forma que las ondas de radio, pero en este caso en una frecuencia diferente: 2.4GHz y 5GHz.

Puede transmitir a una velocidad de hasta 9.6Gbps pero no suele alcanzar más de 100 metros de alcance, debido a la dificultad que tienen las ondas para atravesar obstáculos.

3.5.2. Bluetooth

Es una de las tecnologías con las que estamos más familiarizados, sobre todo al hablar de transmisión de sonido y datos a corta distancia. A ojos del desarrollador es una tecnología muy cómoda, ya que permite conectar varios dispositivos entre sí de forma muy sencilla.

Tiene un pico de transferencia de 2Mbps. El alcance depende mucho del tipo de transmisor que tenga cada dispositivo, pero es menor al del WiFi. Está pensado para conexiones punto a punto o mallas de pocos nodos.

3.5.3. Zigbee

Este, al contrario que los dos anteriores, no es tan conocido. Utiliza ondas de baja energía para comunicar los dispositivos, lo que hace que su consumo sea menor. Es capaz de crear redes de dispositivos entre sí, sin necesidad de que todos estén conectados al mismo punto.

Un gran inconveniente de esta tecnología es que necesita de un dispositivo que haga de puente entre internet y la red Zigbee.

Su alcance es pequeño (entre 10 y 20 metros) y su velocidad de transferencia ronda los 250 kbts. Se usa especialmente en la domótica.

Ilustración 3 Logo ZigBee

3.5.4. LoRa

El significado del concepto LoRa dice mucho de esta tecnología: "LoRa" viene de "Long-Range" (largo alcance), y esta es, precisamente, una de sus principales características.

LoRa utiliza un tipo de modulación en radiofrecuencia como lo AM o FM, más conocidas por ser el tipo de frecuencia de las radios comerciales. Gracias a esto es capaz de comunicarse en rango de hasta 20km de longitud. Además, tiene un consumo muy bajo, lo que hace posible que muchos dispositivos que utilizan LoRa puedan funcionar con la misma batería durante varios años.

La parte más floja de esta tecnología es el ancho de banda, ya que olo es capaz de enviar un total de 255 bytes por segundo.

Ilustración 4 Logo LoRa

3.5.5. ESP-NOW

Es una tecnología desarrollada por Espressif para sus microcontroladores de la versión ESP32. ESP-NOW utiliza un protocolo similar al WiFi de 2.4GHz, pero de baja potencia. Los dispositivos necesitan ser emparejados antes de la comunicación, pero una vez que están emparejados no necesitan "handshake".

ESP-NOW puede crear una comunicación encriptada de punto a punto. El máximo tamaño para un mensaje que puede ser enviado es de 250 bytes. Se puede utilizar la función de callback que activa al dispositivo receptor con un mensaje en la capa de aplicación.

Una de las grandes ventajas de este sistema es la facilidad que tienen estos dispositivos para conectarse entre ellos. Lo único que se necesita para conectar varios dispositivos a la misma red es indicar el papel que van a tener (ya sea maestro o esclavo) y, en caso de que fuese necesario, una dirección MAC para conectarse.

Ilustración 5 Logo ESP-NOW

3.5.6. Comparativa

Esta es una comparativa de las diferentes tecnologías y sus características más importantes para nuestro proyecto:

	WiFi	Bluetooth	Zigbee	LoRa	ESP-NOW
Rango	100m	20m	15m	20km	200m
Velocidad	9,6Gbps	2Mbps	250kbps	255bytes	255bytes
Red	Todo tipo	P2P, malla pequeña	Malla	P2P	Todo tipo
Consumo	Alto	Medio	Bajo	Bajo	Bajo

Tabla 3 Comparativa de tecnologías de comunicación

Habiendo analizado todas las posibilidades, se ha decidido escoger ESP-NOW para comunicar toda la red de sensores. El hecho de que esté específicamente creado para los microcontroladores que hemos escogido en apartados anteriores hace que sea muy fácil de implementar. Además, el mismo sistema es capaz de gestionar él mismo todas las conexiones. El dispositivo solo necesita una dirección MAC para saber dónde se tiene que conectar y a quién enviar los datos.

3.6. Análisis energético

Este es uno de los mayores problemas a los que nos enfrentamos a la hora de hacer este proyecto. Dado que todos los sensores son inalámbricos, no tienen ningún tipo de fuente de alimentación externa; por lo tanto, cada dispositivo tendrá que estar alimentado por su propia batería.

Para este caso hemos decidido utilizar las baterías de litio NCR18650 de 2000 mAh por varios motivos. Por una parte, es una de las baterías más utilizadas hoy en día; de hecho, la mayoría de pequeños electrodomésticos las usan, incluidos algunos coches eléctricos. Además, son menos propensas a deformarse por desgaste o uso inadecuado y tienen una muy buena relación entre tamaño y capacidad.

Llegados a este punto, se debe estudiar la autonomía que se podría tener con todas las elecciones hechas hasta el momento. La autonomía de un dispositivo se mide haciendo la división de la capacidad que tiene la batería (mAh), partido por el consumo medio que se estima en el sensor (mA).

$$Autonomía\ (h) = \frac{Capacidad\ de\ la\ batería\ (mAh)}{Consumo\ medio\ del\ dispositivo\ (mA)}$$

Para calcular el consumo medio teórico del nodo tendremos que tener en cuenta los componentes más importantes de la placa, en nuestro caso lo haremos del microcontrolador y del sensor. Estas características siempre están explicadas en la hoja de datos de cada elemento. Estas son la de nuestros componentes:

- ESP32: 30 mA sin utilizar ESP-NOW y 240 mA utilizando ESP-NOW
- Vl53l0x: 20 mA midiendo y 1 mA en reposo

Es importante a la hora de hacer este estudio saber cuál será el comportamiento de los componentes durante su uso. Definiremos las siguientes pautas:

- Se hará una medición de aproximadamente 1 segundo en períodos de 30 segundos
- Cada vez que se utilice ESP-NOW se hará durante 10 segundos
- Se estimará que la plaza cambiará de estado cada 15 minutos

Se calculará el consumo de una hora para estimar el consumo medio. En ese rango de una hora sabemos que se activará el sensor un segundo cada treinta, por lo tanto, una 1/30 parte del tiempo. Habrá un cambio de plaza cada quince minutos lo que conlleva cuatro a lo largo de una hora. El resto del tiempo el microcontrolador funcionará de forma normal. A continuación, este tiempo lo pasaremos a porcentajes para facilitar el cálculo.

- ESP32 con ESP-NOW: 40 segundos / 3600 segundos = 1.1%
- ESP32 sin ESP-NOW: 3560 segundos / 3600 segundos = 98.9%
- Vl53l0x midiendo: 120 segundos / 3600 segundos = 3.3%
- V53l0x en reposo: 3480 segundos / 3600 segundos = 96.7%

$$Consumo\ medio(mA) = 240 * 0.011 + 30 * 0.989 + 20\ *\ 0.033 + 1 * 0.967 = 33.937\ mA$$

$$Autonomía\ (h) = \frac{2000\ mAh}{33.937\ mA} = 58.93\ h$$

Tal y como está pensado nuestro sistema no sería rentable tener que cambiar las baterías o cargarlas cada poco tiempo. Una solución a este problema sería que cada dispositivo pudiese generar energía para autoabastecerse. Sería posible conseguirlo instalando una placa fotovoltaica en la parte superior de cada sensor, la cual estará alimentando continuamente la batería, y podrá, de este modo, cargarse durante todas las horas de luz que tenga.

Esta placa fotovoltaica según el fabricante es capaz de generar 100 mA en perfectas condiciones. Teniendo en cuenta las horas de luz que se puede tener al día, que aproximadamente doce horas al día, estaríamos generando el 50% del tiempo. Por lo

tanto, tendríamos una corriente media de 50 mA. Según los datos teóricos la placa podría estar trabando durante todo el tiempo deseado sin necesidad de cambiar la batería.

3.7. Solución propuesta

Para resolver el problema propuesto se va a crear una red de sensores basados en los microcontroladores ESP-32 que se comunicarán mediante ESP-NOW. Cada calle o parcela tendrá su propia red de sensores. En cada red habrá un nodo (nodo máster) que será el maestro encargado de conectar esa red a internet, este será el único nodo que será dependiente energéticamente hablando. Pero, a cambio, podrá estar en todo momento pendiente en el caso.

Todos estos nodos están conectados en topología de estrella y tendrán comunicación directa con el nodo máster. Lo más lógico al hacer una red de sensores sería hacerla en topología de malla por cuestión de seguridad y duplicidad; sin embargo, en nuestro caso, si quisiéramos implementar ese tipo de topología, todos los nodos deberían estar completamente activos en todo momento para que hubiese una correcta comunicación en toda la red. Además, ESP-NOW no proporciona una solución para gestionar una red en malla, por lo que deberíamos gestionarla nosotros mismos, tarea que sería demasiado costosa y que, por lo tanto, no sería rentable.

Además, se diseñará una placa de circuito impreso donde quedará embebido todo el sistema que será la base de cada nodo. Tendrá una salida para conectar su placa fotovoltaica. Para tapar todos los componentes y que no queden al aire libre se diseñará una carcasa que pueda ser impresa en 3D.

Ilustración 6 Renderizado del primer prototipo

Primer prototipo del sensor

Es necesario que el nodo máster tenga conexión a internet, y será este el que publique todas las actualizaciones de los nodos de su red en la base de datos de Firebase. Todos los nodos estarán añadidos a la base de datos con su correspondiente latitud y altitud, para que sea mucho más fácil de visualizar en la interfaz.

El usuario final utilizará una aplicación en su Smartphone que le indicará cuál es el aparcamiento más cercano e incluso le dará indicaciones de navegación para llegar hasta dicho sitio.

3.8. Plan de trabajo

A continuación, presentamos el plan de trabajo del proyecto. Hemos dividido entre las tareas que se van a realizar y el tiempo que durarán.

Nombre de la tarea	Fecha de inicio	Fecha de finalización	11.2020	12.2020	01.2021	02.2021	03.2021	04.2021	05.2021	06.2021
PROYECTO										
Estudio posibles soluciones	11.2020	01.2021								
Ánalisis distintas tecnologías	12.2020	02.2021								
Pruebas de sensores	01.2021	02.2021								
Diseño del hardware	02.2021	02.2021								
Diseño del software	03.2021	04.2021								
Diseño de la apliación Android	04.2021	05.2021								
Pruebas comunicación	05.2021	06.2021								
Pruebas de autonomía	03.2021	06.2021								
Pruebas de rango	06.2021	06.2021								
Documentación	05.2021	06.2021								

Tabla 4 Diagrama de Gantt del proyecto

La duración estimada del proyecto era de 8 meses. Debido a ciertas complicaciones con el diseño del hardware esta tarea finalmente se retrasó un mes más.

En resumen, podemos decir que la cronología del proyecto se ha cumplido con éxito.

4. Arquitectura del sistema

Una de las características del proyecto es la simpleza y modularidad de su arquitectura. Ésta es fácilmente expandible.

Lo más conveniente a la hora de explicar la arquitectura del sistema es verla en un diagrama. Este sería el de nuestro sistema

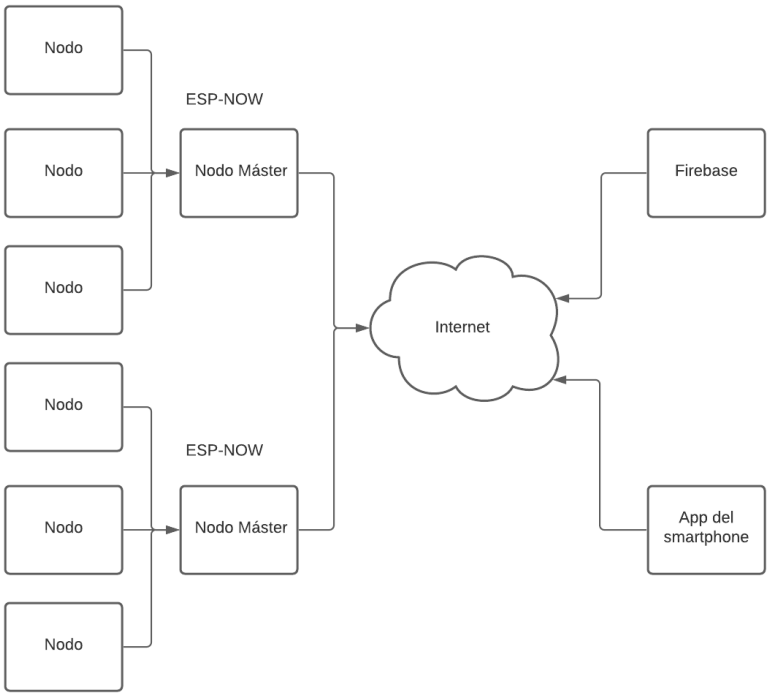


Ilustración 7 Diagrama de la arquitectura del sistema

Lo primero que nos encontramos en este diagrama son los nodos de nuestra red de sensores. Los nodos son los encargados de recoger la información de su entorno. En este caso nos interesa saber si hay un coche en su posición o no, para eso utilizaremos dos sensores de distancia por cada nodo que se conectarán al microcontrolador en su bus de I2c en paralelo. Una vez hecha la lectura y dependiendo de si ha cambiado el estado es el propio microcontrolador el que enviará mediante ESP-NOW un paquete con toda la información necesaria al microcontrolador que hace de nodo maestro. El microcontrolador que ha hecho el envío recibirá una confirmación cuando el mensaje que haya enviado se haya recibido con éxito. Este mensaje será gestionado por la librería de ESP-NOW.

Una vez la información llegue al nodo máster este será el encargado de subirlo a la base de datos. El acceso a Firebase se hará a través de unas credenciales fijas dentro del código del microcontrolador y, además, tendrá que estar indicado en la propia página de configuración de la base de datos. Para actualizar la base de datos se hará una petición a la dirección de nuestro servidor Firebase con los datos recibidos del resto de nodos.

La base de datos dentro de este proyecto tiene un gran peso, ya que absolutamente toda la información se almacena aquí. A la hora de escoger nuestra base de datos, como se ha explicado en apartados anteriores, se tuvo en cuenta un factor muy importante que es la

fácil incorporación de esta en nuestros sistemas. Como ya sabemos, tanto Android como Firebase son propiedad de Google, esto hace increíblemente fácil trabajar con los dos sistemas a la vez.

Firebase utiliza archivos Json para guardar toda su información. Los archivos Json (acrónimo de JavaScript Object Notation) son un estilo de ficheros pensado para realizar intercambio de datos. Ganó mucha popularidad junto al uso de JavaScript ya que la forma de guardar los datos era mucho más sencilla de procesar que un XML.

En nuestra base de datos toda la información se almacena dentro del objeto raíz llamado sensors. Dentro de esta se encuentran enumerados los diferentes nodos/sitios que están controlados por nuestro sistema. Cada objeto sensor tiene 4 campos, el id, longitud, latitud y si está libre.

Es posible que, en un futuro, si la aplicación se quiere escalar a otras ciudades diferentes se creen más objetos raíz. Sería interesante crear uno por cada ciudad por el simple hecho de reducir la cantidad de datos que se envían en cada consulta de la aplicación.

Este sería un ejemplo de cómo está estructurada la base de datos.

```
{
  "sensors" : {
    "1" : {
      "free" : true,
      "id" : 2,
      "lat" : 39.476751,
      "lon" : -0.356259
    },
    "2" : {
      "free" : true,
      "id" : 2,
      "lat" : 39.481029,
      "lon" : -0.34887
    },
    "3" : {
      "free" : true,
      "id" : 3,
      "lat" : 39.481029,
      "lon" : -0.341684
    }
  }
}
```

Nuestra aplicación Android será la encargada de hacer de interfaz con el usuario final. Al igual que el microcontrolador, la aplicación accederá a la base de datos con unas credenciales ya predefinidas. Una vez la aplicación haya descargado toda la información utilizará el GPS para poder localizarse en el mapa y mostrar cuales son las plazas libres cercanas a nuestra posición. A partir de aquí el sistema estará listo para su uso.

5. Desarrollo de la solución propuesta

5.1. Desarrollo del circuito impreso

Nuestros nodos están basados en el microcontrolador ESP-32, así que nuestro diseño del circuito empezará por ahí. En este caso vamos a utilizar la aplicación EasyEDA para diseñar la placa, por lo intuitiva que es y fácil que es de usar.

Lo fundamental a la hora de hacer un circuito para un sistema embebido como este es mirar cuáles son las conexiones mínimas necesarias para que funcione el microcontrolador. En este caso hay que tener en cuenta varios pines que son muy relevantes en el comportamiento del micro:

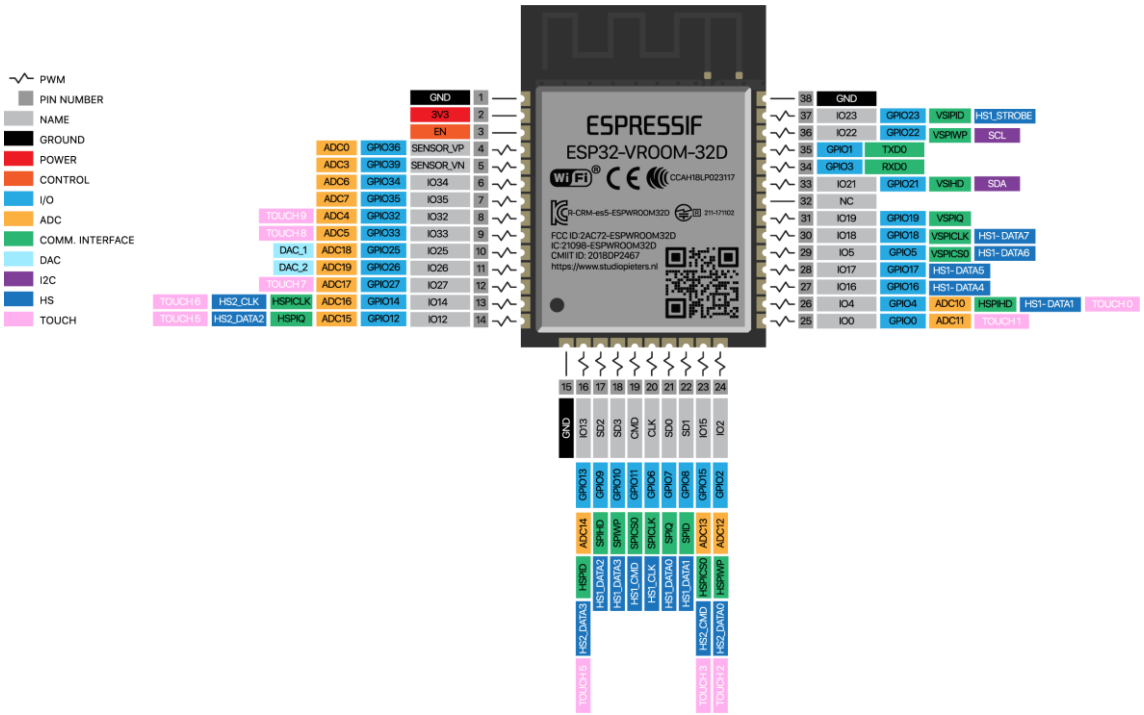


Ilustración 8 Pines del ESP-32 y sus funciones

- Pines 1, 15 y 38. Son los pines que están conectados a tierra.
- Pin 2. Es el pin de alimentación del sistema. Por este pin entra toda la corriente necesaria para que el sistema tenga energía.
- Pin 3. Este pin sirve para activar o desactivar el sistema. Suele estar conectado a un botón para poder reiniciar el sistema sin tener que quitar la alimentación total.
- Pin 25. El GPIO0 es el encargado de cambiar el modo de arranque del micro. Si al encenderse el pin tiene una entrada digital 0 entra en modo descarga, estará preparado para que se le cargue un programa nuevo. Si al encenderse tiene una entrada digital de 1, el micro cargará el programa de su memoria. Como en el caso anterior, también es común instalarle un botón para poder ir alternando entre modo descarga y modo ejecución.

- Pin 23, 24 y 10. Estos también cambian la configuración a la hora del arranque del sistema, como por ejemplo el modo debug. En este caso simplemente pondremos el que indique el datasheet.
- Pins del 17 al 22. Estos pines están conectados mediante SPI a la memoria interna del sistema. No se recomienda conectar nada en estos pines a menos que se quiera cambiar directamente la memoria del micro.
- Pin 34 y 35. Son los pines que se encargan de la comunicación en serie directa con el micro. Para cargar programas al sistema hay que utilizar estos dos pines.

Una vez explicados los pines más relevantes del sistema, vamos a explicar paso por paso el circuito de la placa para que se pueda entender con facilidad.

Este sería el diagrama de conexiones entero del sistema:

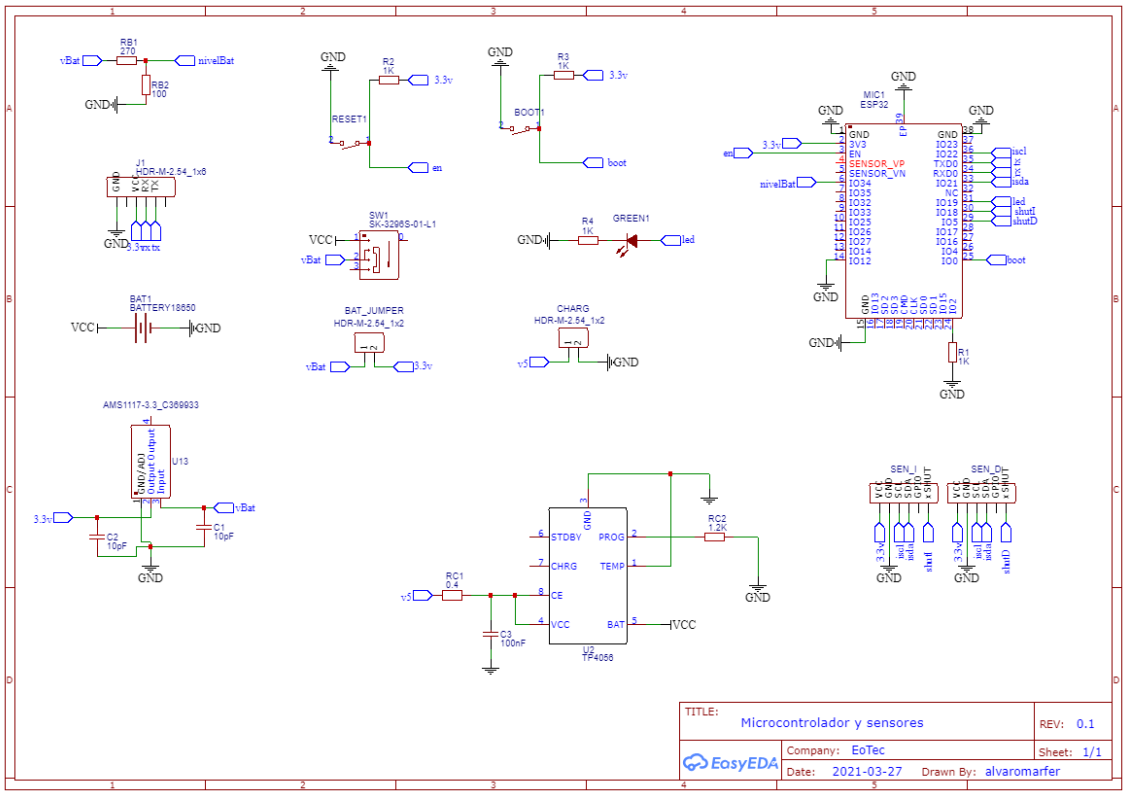
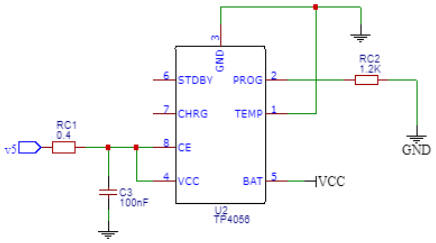


Ilustración 9 Diagrama de conexiones

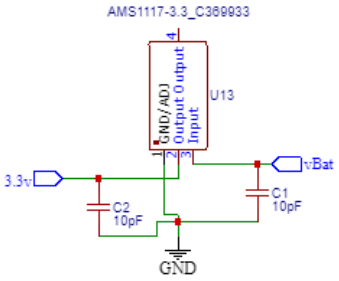
La fuente de alimentación del sistema es la batería 18650. En este caso se ha conectado entre VCC (Voltaje de corriente continua), que sería nuestro positivo y GND (Tierra) que sería el negativo común de todo el sistema.

La batería tendrá que ser cargada por una placa fotovoltaica para que sea energéticamente autónoma. Aquí se presentan dos problemas que impiden que se pueda conectar la placa directamente a la batería. Uno de ellos es la diferencia de voltaje, ya que

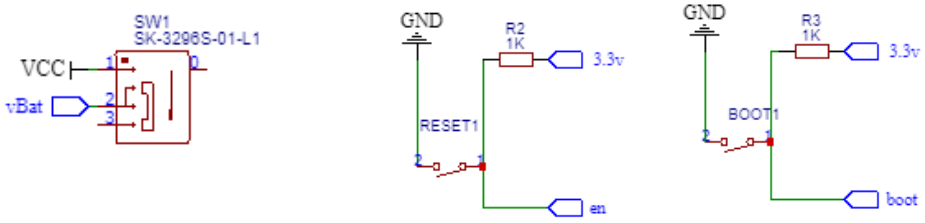
la placa funciona a 5V y la batería entre 3.7 y 4.2. El otro problema es que la placa siempre le estaría dando corriente en todo momento que estuviera expuesta a luz solar. Esto sería extremadamente peligroso, ya que, si la batería pasa de un voltaje, puede llegar a incendiarse o incluso explotar. Es por eso que hay que colocar un cargador de baterías entre la placa y la batería. Para este proyecto se ha escogido el chip TP4056, que permite cargar baterías de una celda, precisamente lo que necesitamos, con el control necesario de voltaje y amperaje para no dañar la batería. Los diferentes valores que tienen las resistencias de este circuito hacen que el amperaje de carga no supere cierto rango; en nuestro caso, al poner una resistencia de 1.2k ohmios se consigue que tenga un máximo de carga de 1A. De todos modos, es bastante improbable que la placa alcance ese valor.



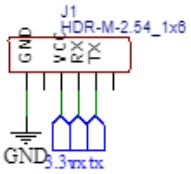
La batería 18650 tiene un voltaje entre 3.7 y 4.2 V, dependiendo de su nivel de carga. De esta manera, hace imposible conectar directamente ningún componente a este valor, ya que todos nuestros componentes funcionan a 3.3V. Por lo tanto, hay que instalarle un regulador de voltaje. Hemos escogido un AMS1117-3.3, que tiene un funcionamiento muy sencillo. Tiene tres patas: por una pata entran los 5V, por otra salen los 3.3V, y la otra está conectada a la tierra común de las redes. Además, se instalan un par de condensadores para evitar picos en la señal.



Se ha colocado un interruptor para cortar toda la corriente al dispositivo que lo apaga directamente. Asimismo, se han colocado dos botones, uno para reiniciar el microcontrolador y otro para poder entrar en modo descargar del microcontrolador.



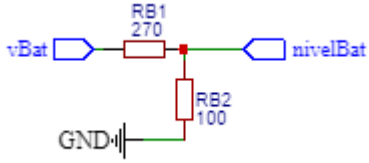
Anteriormente se explicó la importancia de los pines RX y TX, que son los encargados de la comunicación en serie. Sin conexión a estos pines seríamos incapaces de programar nuestro sistema. Además de estos pines, será necesario un conversor para conectar a nuestro puerto USB del PC. Para ello, utilizaremos una placa basada en el chip FT232RL, que cuenta con seis pines de los cuales solo utilizaremos cuatro: positivo, negativo, RX y TX. Hemos diseñado una entrada de seis pines específicamente para esta placa, aunque en realidad solo estarán conectados los cuatro mencionados anteriormente.



Para controlar el nivel de carga de la batería se ha diseñado un divisor de tensión, que consiste, simplemente, en dos resistencias entre las que se coloca una conexión a una entrada analógica del microcontrolador. La fórmula para calcular cuál sería el voltaje resultante en esa entrada analógica es la siguiente:

$$Vout = \frac{RB2}{RB1 + RB2} * Vin$$

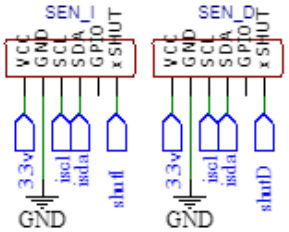
Este sería el divisor de tensión en nuestro diagrama:



Teniendo en cuenta que tendremos unos valores de voltaje entre 3.7V y 4.2V, esperaremos una salida entre 1.14V y 1V.

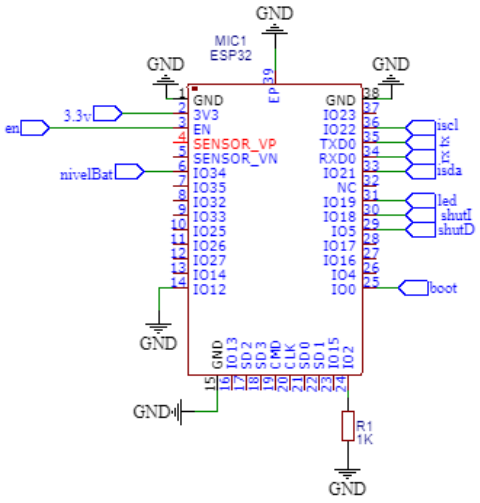
Los dos sensores van conectados a la placa con unos pines que forman un ángulo de 90º para que el sensor pueda quedar perpendicular al suelo. Están conectados al bus I2C en paralelo. Estos sensores tienen un pin para encenderlos y apagarlos desde el microcontrolador. Esto nos va a ser muy útil en el futuro, ya que los componentes que se comunican por I2C tienen una dirección asignada. En el caso de este sensor todos tienen la misma dirección. De todos modos, se puede cambiar por software, pero solo de uno en uno. Por lo tanto, lo que se hará más adelante será apagar uno mientras se configura el otro, de ahí la importancia del pin de encendido.

Por último, queda lo más importante: el microcontrolador. En este diagrama se pueden



diferenciar todas las conexiones citadas anteriormente. A la hora de hacer estas

conexiones en el microcontrolador, se debe tener en cuenta si el pin que está usando sirve para la idea planteada. Por ejemplo, no todos los pines pueden usarse como entrada, sino que algunos no admiten entradas analógicas y algunos especiales están reservados para el propio sistema.



Una vez ya tenemos todas las conexiones hechas en el esquema, falta transformarlo en una placa real con sus dimensiones. Esto se trata de un trabajo bastante tedioso, aunque no complicado. Cuando ya estemos seguros de nuestras conexiones, lo siguiente será indicarle al programa que lo pase a una PCB (Printed Circuit Board), nos preguntará las dimensiones de la placa y cuantas capas queremos. Normalmente se suelen utilizar 2 capas, a no ser que estemos hablando de un tipo de diseño muy complejo. Por último, aparecerá una pantalla como la siguiente:

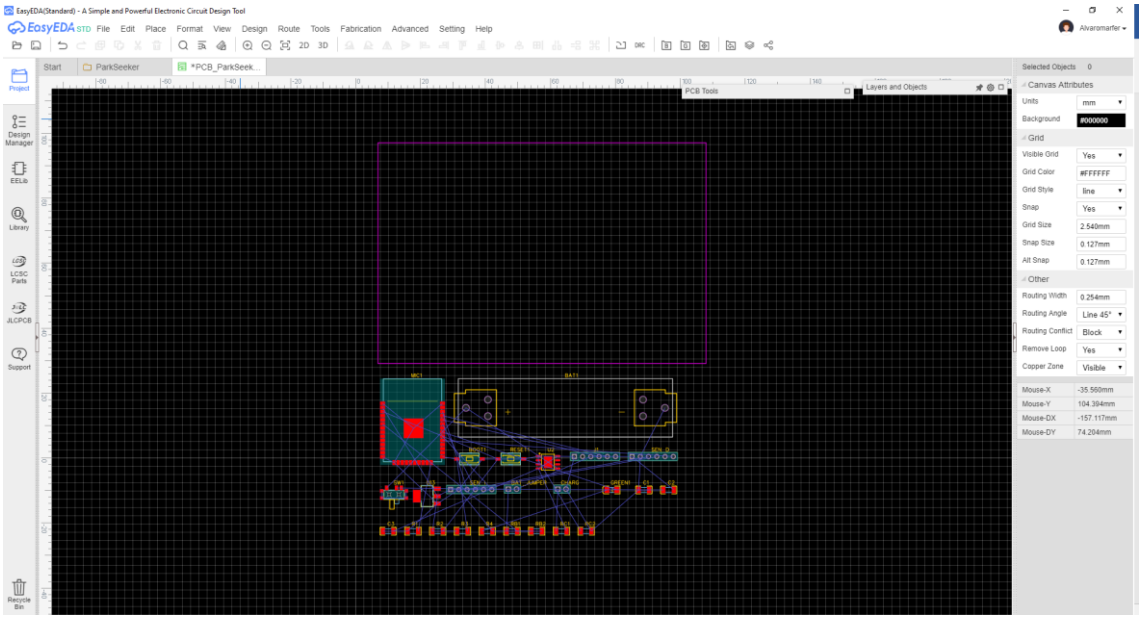


Ilustración 10 Pantalla de diseño de circuitos impresos

Aquí se encuentran todos los componentes que se han añadido al esquema. Ahora, nuestro trabajo será colocarlos como queremos y después establecer conexiones físicas

entre ellos. Además, se puede apreciar un tipo de líneas de color azul llamadas “ratlines” que nos indican las conexiones que debemos hacer según nuestro documento anterior.

Lo primero que debemos hacer es ir colocando los componentes como nos gustaría que quedasen en el diseño final, intentando siempre que los ratlines sean lo más cortos posibles, para facilitarnos así a la hora de dibujar nuestras pistas (las pistas serían los equivalentes de cables en una instalación normal).

En los circuitos impresos existe una técnica llamada zona de cobre, que cubre todo el hueco que estaría vacío con un material conductor. Estas zonas se conectan a una malla, ya que esta ayuda a simplificar mucho las pistas y, además, sirve de aislante entre las capas. Normalmente se suele conectar a GND, ya que es la capa que más conexiones tiene y es la que más pistas nos ahorraría.

A la hora de diseñar las pistas hay muchas recomendaciones o buenas prácticas. Entre ellas cabe destacar, por ejemplo, que no es recomendable hacer giros de 90 grados. También es recomendable un ancho mínimo de las pistas dependiendo de qué potencia vaya a soportar.

Para comprobar que todo está correcto se pueden crear una serie de reglas sobre el ancho de las pistas, el ángulo de giro de estas, el diámetro de las vías o la distancia de todos los objetos con el resto (en inglés “clearance”). En todo momento se puede comprobar que se cumplen estas normas, lo que hará mucho más fácil nuestro trabajo, ya que nos indicará todos los posibles errores. Además, nos avisará si dejamos alguna conexión sin hacer.

Así quedaría el diseño con todas las pistas hechas:

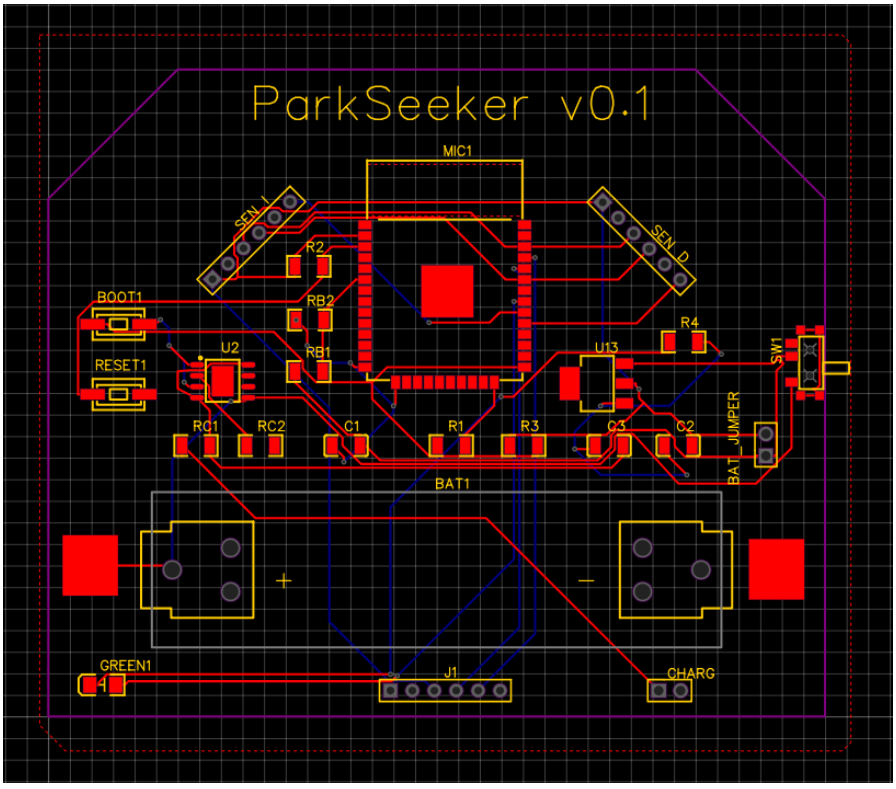


Ilustración 11 Circuito acabado

Y esta sería una renderización de la placa por EasyEDA:

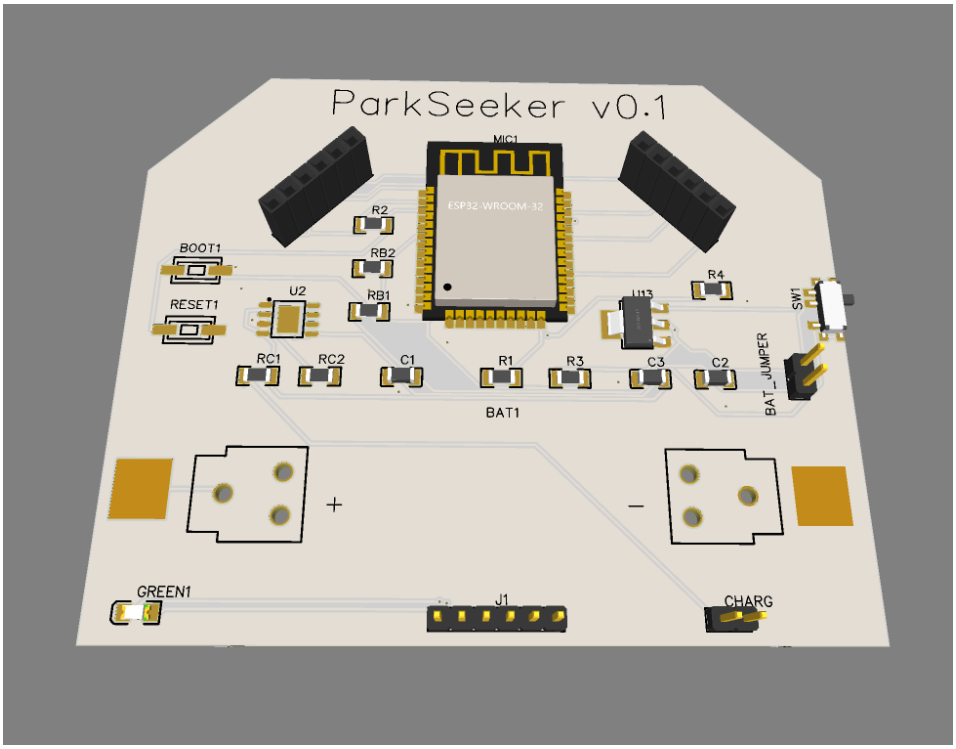


Ilustración 12 Renderización de la placa

Y este sería el resultado una vez fabricada y soldada con todos sus componentes:

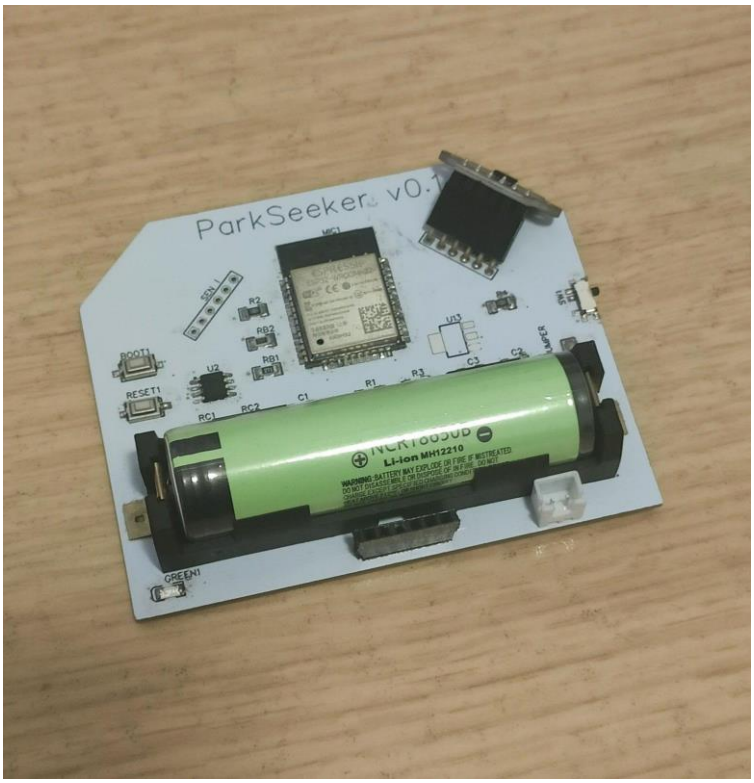


Ilustración 13 Nodo soldado

5.2. Diseño de la aplicación ESP-32

La aplicación del microcontrolador no tiene una lógica complicada, pero a la vez es un poco compleja a la hora de establecer las conexiones. Uno de los problemas que tiene este microcontrolador es que no permite utilizar el Bluetooth y el WiFi al mismo tiempo, ya que las dos tecnologías comparten la misma antena. Es por esto que la estructura del programa hace que se complique un poco, debido a que se deben alternar los dos tipos de comunicaciones para evitar que se solapen.

Dicho esto, vamos a repasar la lógica del programa. Este sería su diagrama de flujo:

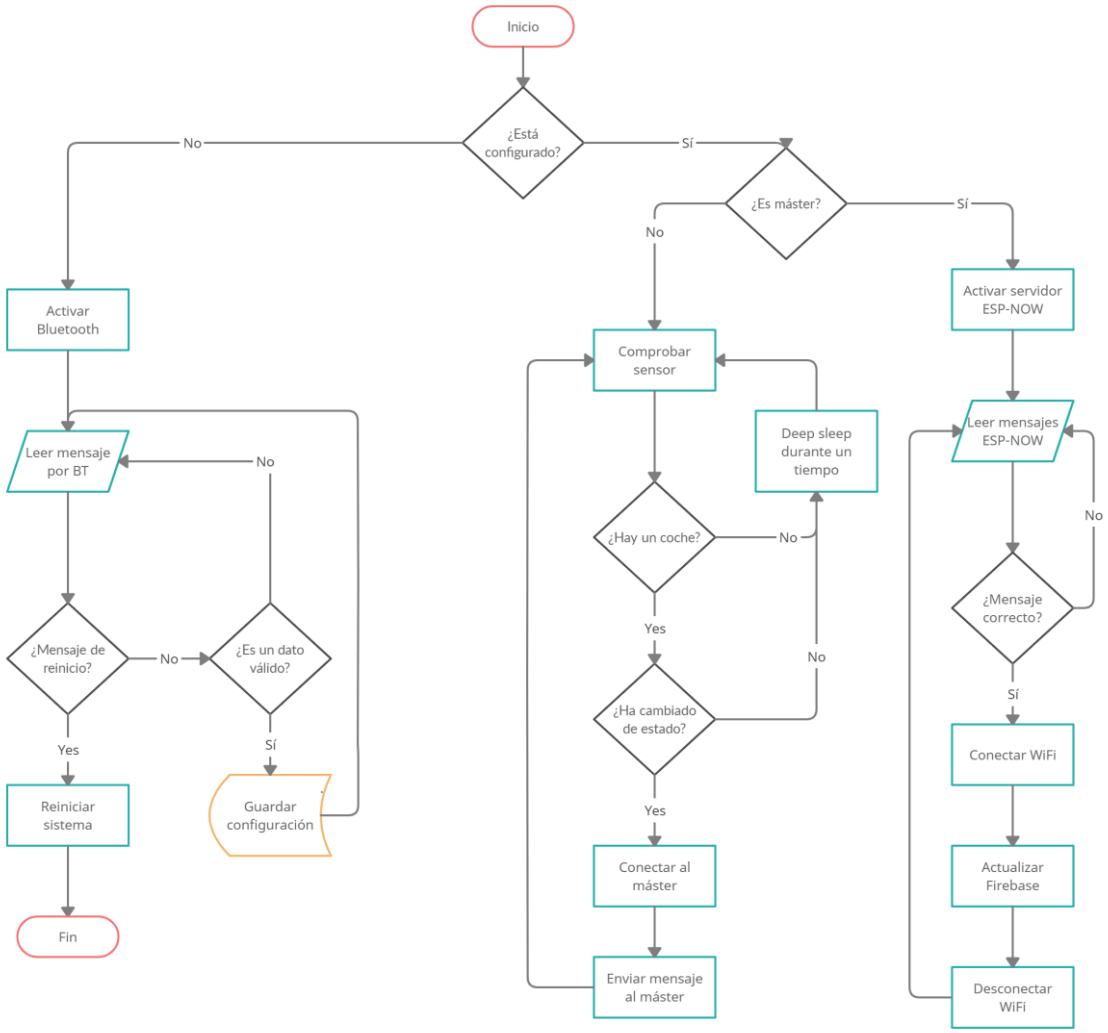


Ilustración 14 Diagrama de flujo de la aplicación del microcontrolador

Después de cargar el programa en un nodo, este no estará configurado. No sabrá a qué dirección MAC conectarse y tampoco qué ID tiene (el ID servirá posteriormente para relacionar cada dispositivo con una latitud y longitud). Si el dispositivo todavía no está configurado, encenderá el Bluetooth y abrirá un servidor de comunicación serie Bluetooth. Para configurar el nodo simplemente hay que conectarse con cualquier móvil

que tenga Bluetooth y una aplicación para comunicación serie Bluetooth. Para comunicarnos se ha creado un protocolo de comunicación bastante simple pero efectivo, que consiste en valores alfanuméricos separados por punto y coma; en este caso, la primera parte del mensaje sería la opción y la segunda sería el valor de esa opción. A continuación, una tabla con los valores de este protocolo:

Opción	Ejemplo	Descripción
ID	ID;1;	Cambia el ID del nodo
SSID	SSID;ssid_wifi;	Si se trata de un nodo máster, es a la red WiFi a la que se conectaría
PASS	PASS;password;	Si se trata de un nodo máster, es la contraseña para conectarse al WiFi
MASTER	MASTER;TRUE;	Indica al nodo si debe de funcionar como nodo máster o no
MAC	MAC;01:3A:1D:54:6B:32;	Si se trata de un nodo no máster, es la dirección MAC a la que se conectará con el ESP-NOW
RESTART	RESTART;	Reinicia el nodo para que el sistema vuelva a cargar la nueva configuración.

Tabla 5 Protocolo de comunicación Bluetooth

En cuanto el nodo recibe la información mediante la comunicación serie Bluetooth, la guarda en su memoria para no perderla cuando tenga que reiniciarla. Es en el momento que se reinicia cuando la configuración tiene efecto. Por lo tanto, los pasos para configurar cada nodo son los siguientes:

1. Encender el nodo
2. Emparejar el nodo que tendrá el nombre de “ParkingSeeker_NoConf” con un teléfono inteligente
3. Utilizar una aplicación de terminal serie Bluetooth para abrir una conexión con el nodo
4. Mandarle los mensajes necesarios para configurar el dispositivo
5. Reiniciar el sistema físicamente o enviándole un mensaje de “RESTART”

Si después de configurar el nodo da un error al arrancar después de ser configurado, volverá a abrir el servidor Bluetooth para que se vuelva a configurar.

Una vez que el dispositivo esté configurado y arranque sin problemas, comprobará si es un nodo máster o no. En caso de tratarse de un nodo máster, creará un servidor de ESP-NOW al cual se conectarán después el resto de nodos para enviar la información. Es

bueno recordar que los nodos máster están pensados para que estén todo el tiempo conectados a una fuente de alimentación externa. De esta manera los otros nodos pueden entrar en modo Deep-sleep para ahorrar energía sin que haya problemas en la red. Los nodos máster solo tienen que gestionar todos los datos entre la red de sensores y la base de datos en la nube.

El sistema de ESP-NOW tiene una función de callback al recibir un mensaje, lo que significa que es la misma librería la que gestiona la entrada de mensajes en el sistema. Nosotros no tendremos que estar haciendo una comprobación en cada iteración, sino que simplemente tendremos que preocuparnos de gestionar los datos que lleguen de otros nodos y enviarlos al servidor.

A la hora de comunicarse entre dos nodos diferentes, por protocolo ESP-NOW hay un factor importante que se debe tener en cuenta: ambos deben de tener el mismo tipo de datos. En nuestro caso, podríamos hacerlo de dos formas: enviando una cadena de caracteres plano separado con comas (al igual que hicimos en el protocolo de configuración Bluetooth) o bien creando una estructura que incorpore todos los datos necesarios para que esta comunicación sea exitosa. Hemos optado por la segunda opción, ya que nos facilitaba mucho a la hora de descifrar el mensaje.

En caso de que no esté configurado como máster, el dispositivo tendrá un comportamiento diferente. Lo primero que hará a la hora de arrancar será conectarse al nodo máster de su propia red, es decir, a la dirección MAC a la que se le ha configurado con anterioridad. Reintentará conectarse al nodo máster cada cierto tiempo hasta llegar a tener conexión.

Una vez haya realizado la conexión con el máster, el nodo tendrá un comportamiento repetitivo durante todo el funcionamiento. Cada cierto tiempo, determinado por la compensación entre ahorro de energía y tasa de actualización, comprobará cuál es la lectura de su sensor. Una vez tenga el valor del sensor, comprobará si es diferente al anterior, porque de ser así, significaría que la plaza de parking que está controlando ha cambiado de estado. En ese caso, actualizará la variable para compararla en la siguiente iteración y enviará la información al nodo máster con la estructura anteriormente citada.

A la hora de hacer las primeras pruebas con nuestro código para los nodos, nos hemos encontrado con un gran problema que nos ha complicado el proceso de desarrollo de toda la aplicación. Cómo se ha comentado con anterioridad, el microcontrolador ESP-32 utiliza la misma antena para todas sus comunicaciones inalámbricas y además también comparten la misma interfaz. Sabíamos que esto supondría un reto al que nos íbamos a enfrentar a la hora de programar el nodo máster, ya que este tiene que trabajar con dos tipos de comunicación inalámbrica a la vez. Por un lado, debe comunicarse con los nodos por ESP-NOW, y, por otro lado, con la base de datos mediante conexión WiFi. Para evitar este problema, cada vez que el máster reciba un mensaje de la red de sensores se cierra la conexión ESP-NOW y se abre la conexión WiFi. Este método resultó ser poco fiable. A partir del primer mensaje recibido de los nodos, la fiabilidad de la red se ve reducida drásticamente. Aunque no existe manera de comprobarlo, esto se puede deber a dos motivos. El primero es que algún tipo de servicio o hilo de la conexión WiFi siga utilizando la interfaz al cerrarse la conexión y está bloqueando o dificultando el trabajo al protocolo ESP-NOW, o bien también puede existir el problema de tener los nodos para volver a conectarse con la red una vez el máster cierra la conexión y la vuelve a abrir.

Una vez las placas de circuito impreso estaban impresas, teníamos que adaptarnos a eso, lo que hizo que nuestra solución estuviese bastante limitada por la parte hardware. Hemos conseguido encontrar una solución a este problema modificando sólo las placas máster. El gran problema de este microcontrolador es que solo tiene una interfaz, por lo que la solución será añadirle una interfaz más para que no tengan problemas de compatibilidad el uso de las dos comunicaciones a la vez. Pero desde luego, la tarea más complicada era como añadir esta nueva interfaz.

Como se puede ver en el capítulo anterior donde se ha explicado el diseño del circuito impreso, todos los nodos cuentan con una fila de 6 pines por los que se programan los nodos mediante una comunicación serie. A partir de estos pines se ha creado una nueva placa que se pueda acoplar a los nodos. En este caso, la hemos diseñado a partir del microcontrolador ESP-12 por cuestiones de inventario. Esta placa funcionará como puente entre la red de sensores e internet, y recibirá datos por comunicación serie utilizando los pines que se usan normalmente a la hora de programar el microcontrolador. Utilizaremos un protocolo propio muy parecido al que utilizamos a la hora de la configuración Bluetooth. Esta es la tabla de nuestro protocolo:

Opción	Ejemplo	Descripción	Respuesta
SSID	SSID;ssid_wifi;	Es a la red WiFi a la que se conectará.	OK;
PASS	PASS;password;	Es la contraseña para conectarse al WiFi.	OK;
CONNECT	CONNECT;	Es el mensaje para indicarle al ESP-12 que se tiene que conectar a la red.	OK; si se ha conectado con éxito FAIL; si no se ha podido conectar.
STATUS	STATUS;	El ESP-12 enviará de vuelta un mensaje con la información del estado de conexión.	CONNECTED; si está conectado a una red. DISCONNECTED; si no está conectado a ninguna red.
RESTART	RESTART;	Reinicia el ESP-12 en caso de que haya algún problema.	
UPDATE;	UPDATE;12;TRUE;	Este es el mensaje que indica que se ha de actualizar la base de datos. La primera parte de los valores es el ID del sensor y el segundo es el que indica si está libre o no.	OK; si se ha actualizado correctamente. FAIL; si no se ha podido actualizar la base de datos.

Tabla 6 Protocolo de comunicación par ESP-NOW

Una vez explicada toda la lógica de la aplicación, pasaremos a revisar el código de la aplicación. En este caso, la aplicación está programada en lenguaje c.

La primera parte del programa es la inclusión de las librerías para toda la aplicación. Utilizamos la librería de Arduino para gestionar la comunicación con la parte hardware; como se ha explicado en la comparación de microcontroladores, el ESP-32 es totalmente compatible con Arduino. Además, utilizaremos estas librerías:

- VL53L0X: Es una librería desarrollada por AdaFruit, el distribuidor oficial del sensor, por lo quenos facilita mucho su uso. Gracias a la librería somos capaces de cambiar la dirección I2C o recibir la lectura con tan solo una línea de código.
- Preferences: Se utiliza para guardar valores en una memoria no volátil, y es una especie de base de datos interna que funciona como si fuera un diccionario. Está pensado para guardar la configuración del dispositivo que es, en cierta medida, lo que estamos haciendo. Todos los datos recibidos mediante la configuración por Bluetooth se guardarán ahí.
- BluetoothSerial: Como su nombre indica, se encarga de gestionar la comunicación serie del Bluetooth.
- FirebaseESP32: Controla el acceso al servidor y gestiona las actualizaciones de la base de datos.
- esp_now: Se encarga de gestionar todas las conexiones que utilizan su protocolo. En nuestro caso, será la conexión entre el máster y el resto.
- esp_wifi: Esta es una de las pocas librerías que sustituye a las librerías propias de Arduino. Al utilizar un chip de Espressif el microcontrolador necesita su propia librería para utilizar la interfaz WiFi del sistema.

Los mensajes que se envían por la red de sensores seguirán una estructura común. En este caso, la estructura, llamada Packet, incorpora solamente dos datos. El primero es de tipo entero y es el que indica el número de identificación que tiene el nodo; cabe recordar que el id del nodo es bastante importante a la hora de saber ubicar el sensor. El segundo es de tipo booleano que indica si esa posición está ocupada o no.

```
//Estructrura del message que envían entre sensores
typedef struct Packet {
    int id;
    bool free;
} Packet;
```

En este programa, como en todas aplicaciones hechas para los microcontroladores de Arduino, cuando el sistema arranca, lanza la función llamada setup. Esta función está pensada para preparar todo el sistema antes de su ejecución continua. Lo primero que se hará será cargar todos los datos que estén dentro de las “preferences”, ya que esto indica qué tipo de nodo es y cómo debería ser su funcionamiento. Tal y como indicamos en el diagrama de flujo, la primera condición es saber si está configurado o no. Para ello, hemos reutilizado una variable para saber el estado del nodo. Por defecto, la variable que indica el id será puesta a -1; en caso de que no haya ninguna configuración cargada la variable seguirá siendo -1, por lo que sabremos que el dispositivo todavía no está configurado. En este caso, el dispositivo activará la configuración Bluetooth abriendo un servidor serie.

```
  if(idSensor == -1){
    bt.begin(nameBT);
    while (true){
      readBt();
      delay(1);
    }
  }
```

Dentro de la función readBt se gestionarán los mensajes del protocolo de configuración que hemos creado específicamente para esto. Como se ha explicado antes, el protocolo separa los mensajes por “;”, y lo que hará nuestro código es identificar las cadenas de texto que se encuentren entre los separadores. Dependiendo de qué tipo de mensaje reciba, el programa esperará un tipo de valor específico. Ya que, por ejemplo, si se tratase de un número, habría que transformar esa cadena de caracteres en una variable de tipo entera. Además de recibir los diferentes datos de configuración, el sistema dará información al Smartphone de todos los cambios que se estén haciendo. De esta forma nos podemos asegurar de que los cambios se están efectuando correctamente.

```
void readBt(){
  if(bt.available()){
    message = bt.readStringUntil(';');
    if(message.equalsIgnoreCase("ID")){
      message = bt.readStringUntil(';');
      preferences.putInt("id", message.toInt());
      sendBt("ID is now: " + message);
      preferences.putString("name", namePrefix + message);
      sendBt("Bluetooth name changed to: " + namePrefix + message);
    }else if(message.equalsIgnoreCase("SSID")){
      message = bt.readStringUntil(';');
      sendBt("SSID: " + message);
      preferences.putString("ssid", message);
    }else if(message.equalsIgnoreCase("PASS")){
      message = bt.readStringUntil(';');
      sendBt("Contrasena Wifi: " + message);
      preferences.putString("passwordWiFi", message);
    }else if(message.equalsIgnoreCase("MASTER")){
      message = bt.readStringUntil(';');
      if(message.equalsIgnoreCase("TRUE")){
        sendBt("This node is now a master");
      }else{
        sendBt("This node is now a slave");
      }
    }else if(message.equalsIgnoreCase("MAC")){
      message = bt.readStringUntil(';');
      sendBt("Master MAC address: " + message);
      preferences.putString("MAC", message);
    }else if(message.equalsIgnoreCase("RESTART")){
      sendBt("Node is gonna restart in 2 seconds...");
      delay(2000);
```

```
      ESP.restart();
    }
  }
}
```

Una vez nos aseguremos de que el sensor está configurado, seguimos con la preparación del nodo. Lo siguiente que se comprueba es si funcionará como máster o no. Esto es bastante importante, ya que se ha tomado la decisión de utilizar el mismo código para los dos nodos diferentes por facilitar a la hora de instalar el programa. Esto es un punto a favor si después se quiere reutilizar un nodo, pero con otra función sin necesidad de cambiar el código.

En el caso de tratarse de un nodo máster, tendremos que seguir tres pasos importantes. El primero será arrancar el servidor ESP-NOW, en caso de que diese error el sistema, entraría en modo configuración Bluetooth. Una vez esté iniciado el ESP-NOW, lo siguiente será arrancar la conexión WiFi. Para hacer esto enviaremos los mensajes para que el puente sepa a qué red se tiene que conectar y cuál es su contraseña. En caso de que se haya conectado correctamente, el puente enviará un mensaje de vuelta para confirmar que se ha conectado y a continuación se indicará cuál será la función que se activará al recibir un mensaje. En caso de que se reciba un mensaje de error, el sistema entrará en el modo configuración Bluetooth.

```
      //Configurar el puente
      Serial.print("SSID;" + ssid + ";");
      Serial.print("PASS;" + passwordWiFi + ";");
      Serial.print("CONNECT;");
      message = Serial.readStringUntil(';');
      if(message.equalsIgnoreCase("OK")){
        //Servidor ESP-NOW iniciado con éxito
        esp_now_register_recv_cb(receiveCallback);
      }else{
        //Error al iniciar la conexión WiFi
        bt.begin(nameBT);
        while(true){
          readBt();
          delay(1);
        }
      }
```

Si se tratase de un nodo esclavo, el comportamiento sería diferente. El primer paso será configurar los sensores. En cada placa hay dos sensores colocados con una diferencia de rotación de 90º. El tratamiento de la lectura de estos sensores dependerá de cómo sean las plazas que vamos a cubrir. De forma predeterminada está configurado que los dos sensores están apuntando a la misma plaza; por tanto, el sitio se determinará como libre si los dos sensores devuelven una distancia mayor a la de activación. De todos modos, con un pequeño cambio en el código se podría tratar como si cada sensor apuntase a una plaza y publicase un mensaje por cada sitio que haya disponible.

Como habíamos comentado anteriormente, los dos sensores están conectados al bus I2C; por lo tanto, todo dispositivo que esté en el bus será reconocido por su dirección. Al tratarse de sensores iguales, vienen con la misma dirección de fábrica, por lo que será necesario cambiarle la dirección, al menos, a un sensor para que no coincidan. Esto se consigue apagando uno con el pin correspondiente y configurando el otro. Una vez que ya está configurado uno con una dirección diferente, ya sería suficiente. En nuestro caso hemos cambiado la dirección a los dos sensores.

```
    //Iniciar sensores TOF
    pinMode(pinShutL, OUTPUT);
    pinMode(pinShutR, OUTPUT);
    digitalWrite(pinShutL, HIGH);
    digitalWrite(pinShutR, LOW);
    delay(50);
    sensorL.init(true);
    sensorL.setAddress((uint8_t)24);
    digitalWrite(pinShutR, HIGH);
    delay(50);
    sensorR.init(true);
    sensorR.setAddress((uint8_t)26);
```

El paso siguiente será iniciar el ESP-NOW. Igual que el nodo máster, en caso de poder iniciarse, el nodo entrará en el modo de configuración Bluetooth. El máster, a la hora de crear la red, lo único que tiene que hacer es iniciar ESP-NOW, mientras que el nodo esclavo lo que debe de hacer es un poco más complejo, aunque no demasiado complicado. Una vez se haya iniciado correctamente, el nodo creará la información de la conexión. En este caso, indicaremos cuál es la dirección MAC del servidor y si queremos encriptar la conexión o no. Aunque este protocolo permite muchas más opciones para configurar el tipo de conexión, para nosotros con esto es suficiente. A continuación, del mismo modo que con el máster, se le deberá asignar una función callback que será activada cada vez que se envíe un mensaje. Esta función también nos avisará si el mensaje se ha recibido correctamente o no.

```
// Creamos la información del peer que vamos a conectar
    esp_now_peer_info_t peer;
    memcpy(peer.peer_addr, broadcastAddress, 6);
    peer.encrypt = false;

    // Añadimos el peer
    if (esp_now_add_peer(&peer) != ESP_OK){
      Serial.println("Failed to add peer");
      return;
    }else{
      Serial.println("Añadido el peer");
    }

    //Registramos la función de callback
    esp_now_register_send_cb(sendCallback);
```

Por último, configuraremos el mensaje que se enviará desde este dispositivo con su propio ID, para que después se pueda localizar en el mapa. Además, bajaremos la frecuencia de la cpu para reducir el consumo de energía.

```
// Configuramos el mensaje que se envía al máster con el id del sensor
    packet.id = idSensor;

    //Bajamos la frecuencia de la cpu para bajar el consumo de energía
    setCpuFrequencyMhz(10);
```

Una vez que la función setup ha finalizado, el sistema lanzará la función loop. Esta función se irá llamando durante toda la vida del programa repetidamente.

Esta parte es bastante sencilla. Si se trata del máster, no tiene que hacer absolutamente nada, ya que toda la lógica para gestionar los mensajes recibidos está en la función callback. Si es un nodo normal tendrá que revisar los sensores cada tiempo; una vez revisados, se pondrá en modo de espera para ahorrar energía.

```
void loop() {

  if(!master){
    checkSensors();
    delay(waitTime);
  }else{
    delay(10);
  }

}
```

Para comprobar si hay un coche en nuestra plaza se llevará a cabo una lectura de los sensores. Si ambos leen más distancia de la distancia de activación, significa que está libre. Después de hacer la lectura de los sensores, se comprobará si fuera necesario enviar información al máster. Solo hay dos situaciones en las que sería necesario enviar el mensaje:

- Cuando la lectura de los sensores sea diferente a la lectura anterior.
- Cuando esté activado el flag de error al enviar.

```
//Comprueba la lectura de sensores y gestiona el envío de mensajes
void checkSensors(){
  bool free;
  if(sensorL.readRangeSingleMillimeters() > triggerDistance){
    if(sensorR.readRangeSingleMillimeters() > triggerDistance){
      free = true;
    }else{
      free = false;
    }
  }else{
    free = false;
```

```
  }

  if(errorSending || free != packet.free){
    //Reseteamos el flag de error al enviar
    errorSending = false;
    //Actualizamos el packet
    packet.free = free;
    //Enviamos el mensaje
    esp_now_send(broadcastAddress, (uint8_t *) &packet, sizeof(packet));
  }
}
```

5.3. Diseño de la aplicación Android

Como última parte del diseño, tenemos la parte de la aplicación para smartphones con Android como sistema operativo. En este caso la aplicación es bastante simple en cuanto a diseño. Solo tiene dos pantallas: la primera que hace de pantalla y la segunda que es directamente la aplicación.

Uno de los puntos más importantes que se deben tener en cuenta a la hora de diseñar una aplicación de este tipo es la eficacia. Es fundamental que, con las mínimas interacciones posibles, el usuario sea capaz de estar rápidamente navegando hasta el aparcamiento más cercano. La pantalla de uso principal de la aplicación sería la siguiente:

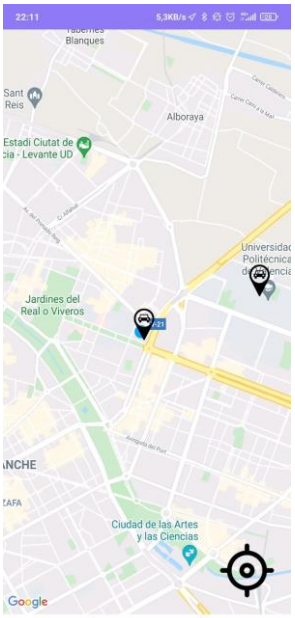


Ilustración 15 Pantalla principal de la aplicación Android

Se trata de una interfaz muy simple e intuitiva. Se pueden observar tres elementos diferentes. El punto azul indica la posición del usuario en tiempo real. Cada puntero con forma de coche identifica cada sitio libre disponible. Estos punteros se actualizarán automáticamente cada vez que cambie un dato en la base de datos. Por último, el botón de abajo a la derecha con forma de diana tiene la función de indicar al usuario cuál es el aparcamiento más cercano en relación a su ubicación.

Además, al hacer click sobre cualquier aparcamiento libre, aparecerá un pop up que preguntará al usuario si quiere empezar la navegación hacia ese punto. Si la respuesta es afirmativa, se ejecutará la aplicación de Google Maps con la ruta ya determinada hacia la dirección de la plaza en cuestión.

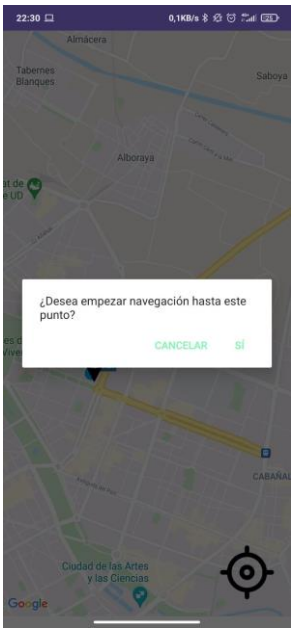


Ilustración 16 Pantalla de navegación

A continuación, haremos un pequeño repaso al código más importante de toda la aplicación Android. Todas las aplicaciones Android se separan en dos tipos de archivos diferentes. Por una parte, tendremos los layouts, que son la forma de diseñar las pantallas, es decir, indicaremos qué elementos se van a mostrar en pantalla mediante un fichero XML, qué diseño tendrán y qué relación que tendrán entre ellos. El otro archivo es un archivo Java que se encargará de todo el funcionamiento de la aplicación y la gestión de los layouts.

Para entender mejor el funcionamiento de nuestro código es necesario explicar el ciclo de vida de las aplicaciones Android. Como hemos visto antes, la estructura de la aplicación en el microcontrolador era bastante sencilla, ya que simplemente consistía en dos funciones diferentes que gestionaban todos los estados que podría tener el sistema. El ciclo de vida sería algo así:

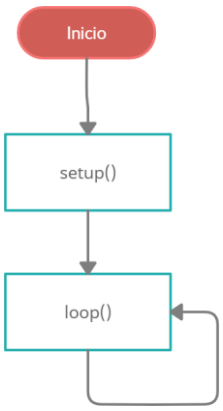


Ilustración 17 Diagrama de estado del microcontrolador

Por el contrario, el ciclo de vida de las aplicaciones Android es más complejo. Este es el diagrama de su ciclo de vida:

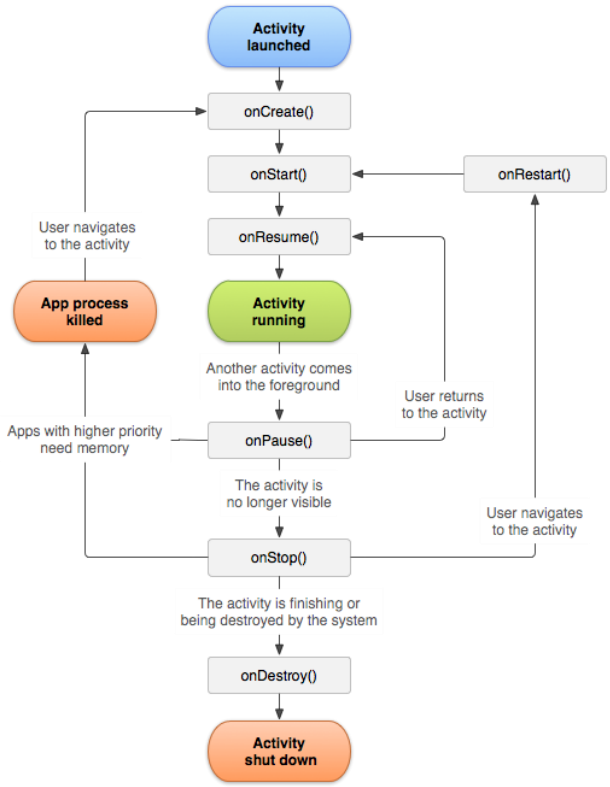


Ilustración 18 Ciclo de vida de la aplicación Android

Como podemos comprobar en el diagrama, las aplicaciones Android contienen muchos más estados de los que nos encontramos en el microcontrolador. Estos estados dependen sobre todo de si la aplicación está en pantalla o no y de si sigue activa. En nuestra aplicación solo utilizaremos uno, que será el onCreate, mientras que el resto de estados serán gestionados por el sistema. Se podrían modificar en caso de necesitar alguna funcionalidad específica, pero en nuestro caso es el sistema el que vuelve a cargar pantalla.

Una vez explicado el ciclo de vida de las aplicaciones Android, es momento de definir el funcionamiento de nuestra aplicación acompañado del código en Java.

Lo primero que vamos a repasar es la función onCreate() de nuestra aplicación. Este sería su diagrama de flujo:

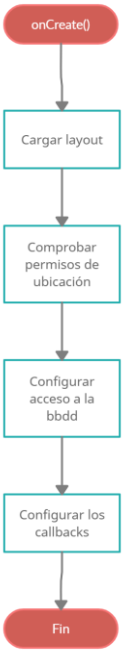


Ilustración 19 Diagrama de flujo de la aplicación Android

Esta función se encarga de cargar el layout. Una vez se haya cargado, se deberán gestionar las interacciones del usuario con los elementos. Para poder controlarlas, lo primero que hay que hacer es identificar todos los elementos que hay en la pantalla y relacionarlos con un objeto de su tipo para que, posteriormente, se puedan crear funciones que sean activadas por estos. Como se ha explicado en la aplicación del microcontrolador, las funciones que son activadas por un evento externo son llamadas funciones callback.

```
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_maps);
SupportMapFragment mapFragment = (SupportMapFragment)
getSupportFragmentManager().findFragmentById(R.id.map);
mapFragment.getMapAsync(this);

ivFind = findViewById(R.id.ivFind);
ivFind.setOnClickListener(this);
```

Lo siguiente será comprobar que tenemos todos los permisos necesarios para que la aplicación funcione correctamente. En nuestro caso, necesitaremos solo dos permisos. Por una parte, necesitaremos permisos de internet, ya que la aplicación tiene que conectarse con la base de datos para actualizar los datos. Por otra, , necesitaremos permisos para acceder a la información del GPS, puesto que es necesario saber la ubicación del usuario para poder calcular cuál es la plaza libre más cercana.

```
LocationManager locationManager = (LocationManager)
getSystemService(Context.LOCATION_SERVICE);
if (checkSelfPermission(Manifest.permission.ACCESS_FINE_LOCATION) !=
```

```
PackageManager.PERMISSION_GRANTED &&
checkSelfPermission(Manifest.permission.ACCESS_COARSE_LOCATION) !=
PackageManager.PERMISSION_GRANTED) {
    ActivityCompat.requestPermissions(this, new
String[]{Manifest.permission.ACCESS_FINE_LOCATION,
Manifest.permission.ACCESS_COARSE_LOCATION}, 0);
}else{
    locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
0, 0, this);

locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER,
0, 0, this);
}
```

Una vez obtenidos los permisos necesarios, seguiremos con la conexión a la base de datos. Este es un paso que, a nivel de código, es bastante simple, pero lo más complejo ha sido toda la configuración previa para poder conectar nuestra aplicación a la base de datos Firebase. Aun así, la gran compatibilidad de los dos sistemas y la gran cantidad de documentación que hay en la red hacen que no sea una tarea demasiado complicada. A la hora de crear la base de datos en la página web de Firebase se debe especificar qué tipo de dispositivos van a acceder a la base de datos; es aquí cuando hay que indicar el id de la aplicación Android. Al configurar el acceso desde una aplicación Android se generará un archivo con las credenciales necesarias para acceder la base de datos que tendremos que copiar en el directorio donde se encuentre nuestro proyecto.

```
sensors = new HashMap<Integer, Sensor>();
markers = new HashMap<Integer, Marker>();
db = FirebaseDatabase.getInstance().getReference().child("sensors");
db.addChildEventListener(this);
```

Para gestionar toda la información sobre las plazas libres, se ha creado una clase con toda la información necesaria de cada sensor. Esta clase nos hará tratar cada nodo como un objeto con sus propias funciones, lo cual nos hará el trabajo mucho más fácil al tratar gran cantidad de datos. Este es el constructor del objeto en cuestión:

```
public Sensor(int id, float lat, float lon, boolean free) {
    this.lat = lat;
    this.lon = lon;
    this.free = free;
    this.id = id;
}
```

A diferencia de la aplicación del microcontrolador, aquí también necesitamos conocer la latitud y la longitud del dispositivo, ya que es en esta aplicación donde se va a mostrar el punto disponible en el mapa.

Hasta aquí nuestra función onCreate(). Como es de esperar, la mayor parte del código se encuentra en las funciones callback. A partir de aquí, haremos un pequeño repaso a todas

estas funciones que forman parte de nuestro programa para entender un poco mejor el funcionamiento de la aplicación.

En relación con el mapa existe una función que se activa cuando este está listo. Dentro de esta función se guardará el mapa en una variable para después poder acceder a él. Se hará un zoom a la ciudad de Valencia, ya que de momento todas las pruebas se están haciendo en dicha ciudad, y después, se indicará que este es el elemento que tendrá que estar pendiente de si el usuario haga click en algún marcador.

```
@Override
public void onMapReady(GoogleMap googleMap) {
    mMap = googleMap;
    mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(new
LatLng(39.479070, -0.368334), 13));
    mMap.setOnMarkerClickListener(this);
}
```

De la localización depende simplemente una función que es llamada cada vez que la ubicación del dispositivo cambia; por lo tanto, también será llamada la primera vez que se conozca la ubicación. Esta función colocará nuestro marcador en el lugar donde estemos. Si, además, es la primera vez que se coloca, la aplicación hará un zoom a la posición en la que el usuario se encuentre en el mapa.

```
@Override
public void onLocationChanged(Location location) {
    position = location;
    //If the marker was not created, we create it
    if(positionMarker == null) {
        if (mMap != null) {
            positionMarker = mMap.addMarker(new
MarkerOptions().position(new LatLng(position.getLatitude(),
position.getLongitude()))

.icon(BitmapDescriptorFactory.fromBitmap(resizeMapIcons(R.drawable.blue_c
ircle, 50, 50))));
            //Make zoom to the place we are
            mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(new
LatLng(position.getLatitude(), position.getLongitude()), 15));
        }
    }else{
        positionMarker.setPosition(new LatLng(position.getLatitude(),
position.getLongitude()));
    }
}
```

Los callbacks relacionados con la base de datos son los que se encargan de añadir a la lista de la aplicación todos los nodos disponibles. En este caso, se pueden distinguir dos funciones: una función que se activa al añadirse un nuevo dato a la base de datos, y otra

cada vez que se modifica un valor. La primera solamente se activará al lanzar la aplicación, momento en el que se comprobará si ese elemento ya ha sido previamente agregado a la lista; de no ser así, se añadirá. Después de incorporarlo a la lista, se añadirá como marcador al mapa. También se comprobará si la plaza está disponible o no, ya que si la plaza no estuviera libre se añadiría, pero esta vez como un marcador invisible.

```
@Override
public void onChildAdded(@NonNull DataSnapshot snapshot, @Nullable String
previousChildName) {
    Sensor sensor = snapshot.getValue(Sensor.class);

    //Add sensor to the sensor map
    if(!sensors.containsKey(sensor.getId())){
        sensors.put(sensor.getId(), sensor);
    }

    //Add marker to the marker map. If it isn't free set visible to false
    if(mMap != null && !markers.containsKey(sensor.getId())){
        MarkerOptions markerOptions = new MarkerOptions().position(new
LatLng(snapshot.getValue(Sensor.class).getLat(),
snapshot.getValue(Sensor.class).getLon()))

.icon(BitmapDescriptorFactory.fromBitmap(resizeMapIcons(R.drawable.car_ma
rker, 150, 150)));
        if(!sensor.isFree()){
            markerOptions.visible(false);
        }
        markers.put(sensor.getId(), mMap.addMarker(markerOptions));
    }
}
```

La otra función se activará cada vez que se cambie un valor de cualquier nodo en la base de datos. En realidad, solo se contempla que pueda cambiar un dato en concreto, el que indica si está libre o no. En caso de que cambie a libre se hará el marcador visible, si está ocupado se hará invisible.

```
@Override
public void onChildChanged(@NonNull DataSnapshot snapshot, @Nullable
String previousChildName) {
    Sensor sensor = snapshot.getValue(Sensor.class);
    Marker marker = markers.get(sensor.getId());
    if(sensor.isFree()){
        marker.setVisible(true);
    }else{
        marker.setVisible(false);
    }
}
```

Por último, la función que se activa cuando el usuario pulse el botón de la diana, se encargará de buscar el sitio libre más cercano y centrar el mapa en ese punto. Se controlará también en el caso de desconocer la posición del teléfono o si no hay ningún sitio libre.

```
@Override
public void onClick(View v) {
    Marker marker;
    if(v.getId() == R.id.ivFind){
        if(position == null){
            Toast.makeText(getApplicationContext(), "Esperando por
posición GPS", Toast.LENGTH_SHORT).show();
        }else{
            marker = getCloser(new LatLng(position.getLatitude(),
position.getLongitude()));
            if(marker == null){
                Toast.makeText(this, "No hay sitios libres",
Toast.LENGTH_SHORT).show();
            }else{

mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(marker.getPosition(),
17));
            }
        }
    }
}
```

6. Pruebas del sistema

A lo largo del desarrollo del proyecto se han llevado a cabo numerosas pruebas para validar conceptos y comprobar que el sistema se estaba desarrollando correctamente. Las pruebas realizadas se dividen en tres tipos:

- Pruebas de comunicación
- Pruebas de autonomía
- Pruebas de rango

6.1. Pruebas de comunicación

Las pruebas de comunicación se realizaron con el fin de llevar al máximo la red de sensores. En este caso, estábamos limitados por el número de placas fabricadas, ya que solo disponíamos de 5 de ellas.

En la primera prueba de estrés hemos conectado cuatro de las placas a una placa maestra que se encargará de recibir todos los mensajes del resto. Se instalará un programa diferente para esta ocasión. Los nodos normales estarán enviando un mensaje como el que harían en la aplicación real con una espera especifica que se irá disminuyendo para buscar el máximo flujo de datos que puede soportar la red. En cuanto al nodo máster, lo único que hará será recibir la información y escribir el id del nodo por el monitor serie, ya que lo que se busca es simplemente buscar el máximo de la red por lo tanto cuantas menos cosas influyan a las pruebas, más certera será la información.

Hemos empezado el experimento con una espera de un segundo entre cada mensaje. Se irá reduciendo este número hasta que se haya un número considerable de mensajes perdidos.

En las pruebas realizadas con esperas de entre 1000 ms y 150 ms han sido un éxito total, a partir de ahí hemos notado que el rendimiento de la red había bajado. Con una espera de 100 ms se estaban perdiendo aproximadamente un 10% de todos los mensajes enviados, y si reducimos la espera a 50 ms se estaban perdiendo más de la mitad de los mensajes. Por lo tanto, determinaremos que la espera mínima segura para la comunicación de toda la red será de 150 ms.

Teniendo en cuenta que estaban conectados a la vez 4 dispositivos a un mismo se puede calcular cuál es el máximo número de mensajes que puede soportar la red.

$$Ratio\left(\frac{mensajes}{s}\right) = \frac{nº\ de\ nodos\ conectados}{Tiempo\ de\ espera\ (s)}$$

En nuestro caso nos da un ratio total de 26.66 mensajes/s.

Según Espressif (fabricante del microcontrolador) no recomienda colocar más de 24 dispositivos conectados al mismo dispositivo. Esta característica resulta mucho más restrictiva que el ratio de mensaje, dado que, aunque todos los dispositivos de la red estuvieran conectados y enviarán un mensaje cada segundo seguiríamos por debajo del ratio máximo del sistema.

6.2. Pruebas de autonomía

A la hora de hacer nuestro análisis energético teórico el resultado fue lo que estábamos buscando, un sistema que funcionase durante el tiempo deseado sin necesidad de cambiar la batería.

Pero la realidad se aleja bastante de lo que habíamos previsto al inicio del proyecto. Se han detectado dos problemas al hacer el análisis que hacen que no sea preciso. El primero de ellos es que no se ha contado con condiciones adversas en cuanto al clima, ya que, a la hora de trabajar con placas fotovoltaicas, estas no tienen el mismo rendimiento si hay un cielo totalmente despejado o está nublado. Además, nos hemos encontrado con que la placa que hemos comprado para este proyecto no da las prestaciones que indicaba en su hoja de datos. Se esperaba tener una corriente de 100 mA cuando el máximo que se ha conseguido a la hora de hacer las pruebas ha sido de 65 mA.

En cuanto al consumo de la placa hemos estado realmente acertados con nuestra predicción. Hemos estimado un consumo de 33.937 mA y se ha medido una corriente de 35 mA aproximadamente.

Con los nuevos datos reales, la autonomía se ha visto afectada. Antes creíamos que el dispositivo podría estar trabajando durante todo el tiempo deseado sin estar perdiendo energía, pero en la realidad no es así. Ahora mismo el dispositivo estaría gastando más energía de la que está generando la placa fotovoltaica, esto significa que la autonomía no es ilimitada, aun así, estaríamos hablando de al menos un mes de funcionamiento continuo.

6.3. Pruebas de rango

Al igual que en las pruebas de comunicación, se utilizará un programa especial. En este caso la prueba es mucho más simple que las anteriores. El nodo maestro estará a la espera de un mensaje, al recibir este mensaje el led de la placa parpadeará. El nodo esclavo estará enviando un mensaje cada medio segundo.

La idea de este experimento es conectar los dos nodos entre sí y ser capaces de saber cuándo pierden conexión entre ellos para calcular el rango máximo. Se comprobará en diferentes condiciones: en el medio de una calle rodeada por edificios y en un solar vacío.

La distancia en exterior la hemos calculado con la ayuda de nuestro teléfono y la ubicación por GPS.

En el caso de una calle hemos alcanzado un máximo de 110 metros sin pérdida de datos. Es cierto que podríamos haber forzado mucho más el sistema, pero a partir de los 110 metros no llegaban el 100% de los datos.

Las pruebas en el solar vacío, como era de esperar fueron mejor, pero no conlleva una gran mejora. Hemos conseguido un máximo de 130 metros, unos 20 metros más que en la ciudad.

7. Conclusión

A la hora de empezar a hacer el proyecto teníamos miedo de que el tema escogido fuera demasiado complicado y eso pudiese llevarnos a tener que cambiarlo a mitad de proceso. Pero hemos conseguido desarrollarlo, no libres de problemas. Hemos aprendido muchas cosas de diferentes campos en los que no estábamos acostumbrados a trabajar, y lo más importante a cómo gestionar un proyecto más complejo de lo habitual.

En cuanto al resultado, has sido más que satisfactorio. Estamos ante un sistema que, a pesar de estar muy lejos de ser un producto final, cumple los requisitos que se pedían al principio. Se han encontrado diferentes problemas en algunos diseños que nos han llevado mucho más tiempo del que deberían. Pero así es el proceso de aprendizaje: prueba y error. Estos problemas han aparecido más que nada por desconocimiento del campo donde estábamos trabajando, pero esto nos ha hecho aprender de la forma más veraz que existe, de forma práctica.

De todos modos, este proyecto no se queda aquí. Durante todo este tiempo desarrollando el sistema nos hemos dado cuenta de las flaquezas que tiene nuestro producto y que deberíamos solucionar si queremos que se convierta en un producto realmente válido para el mercado.

No estamos realmente satisfechos con la eficiencia energética del sistema. Bien es cierto que con el panel fotovoltaico puede llegar a ser energéticamente autónomo esto nos limita a que la aplicación sea para escenarios exteriores. Sabemos que esto se puede mejorar mediante modos de ahorro en los microcontroladores, pero esto requeriría una programación a un nivel más bajo que no sería rentable comparado con la cantidad de horas que nos harían falta para que el sistema funcionase correctamente.

La selección de tecnología de comunicación creo que ha sido la correcta para el funcionamiento que tiene ahora mismo el sistema, pero si nuestra meta es alcanzar una mayor eficiencia energética deberíamos optar por otra opción. Lo mejor habría sido escoger BLE (Bluetooth Low Energy), pero todavía se trata de una tecnología bastante joven cuando hablamos de red de sensores.

En nuestro caso estamos utilizando una topología en estrella, cuando lo más común en una instalación de este tipo sería utilizar malla. Hemos escogido estrella porque hacía que aumentara la autonomía del sistema, pero esta selección nos ha hecho que se limitará a 20 los nodos por red. Si escogiéramos una topología en malla habría complicado extremadamente el desarrollo de la red por varios motivos. El primero es que cada nodo tiene que gestionar su conexión con al menos dos nodos distintos y además los nodos deberían estar más tiempo encendidos lo que haría que perdieran autonomía.

Como última mejora podría incluir algún tipo de conexión 4G para evitar que dependiera de una conexión WiFi para conectarse con internet. Sabemos que esto es una solución mucho mejor que la que tenemos, pero también somos conscientes que sería más caro de la solución actual.

En conclusión, hemos realizado un producto que soluciona el problema que se ha planteado al principio del documento. Aunque hubo muchos problemas que solucionar, hemos llegado a una solución muy similar a la que teníamos en mente y en los plazos

planificados. Habría que trabajar más para convertirlo en un producto listo para el mercado, pero desde luego se ha hecho una prueba de concepto bastante importante con unos resultados más que satisfactorios.

7.1 Relación con los estudios cursados

Mi caso es un caso excepcional a la hora de hacer este apartado, ya que soy un estudiante de Sicue procedente de la Universidad de Burgos. De todos modos, he tenido la suerte de que las dos asignaturas que he cursado en la UPV me han resultado de gran ayuda a la hora de hacer el trabajo.

La asignatura de "Configuración, administración y gestión de redes" me ha ayudado a comprender como diseñar una red inalámbrica para que funcionase de la mejor manera posible. Me ha sido muy útil a la hora de escoger ciertas características como, por ejemplo, el tipo de topología que mejor se adaptase a mi aplicación.

"Gestión de proyectos" fue la otra asignatura que cursé en Valencia. Esta me ha ayudado mucho a la hora de afrontar un trabajo como este, que hasta el momento es el mayor que he hecho. He sabido gestionar todo el trabajo, dividirlo por etapas y planificarlo para alcanzar las fechas programadas.

8. Bibliografía

Hoja de datos de ESP-32 [En línea] [fecha de consulta: 18 de enero de 2021]. Disponible en: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

Repositorio ESP-32 Boot Mode Selection [En línea] [Fecha de consulta: 14 de febrero de 2021] Disponible en: https://github.com/espressif/esptool/wiki/ESP32-Boot-Mode-Selection

Hoja de datos de ESP8266 [En línea] [Fecha de consulta: 18 de enero de 2021] Disponible en: https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf

Hoja de datos de Arduino UNO [En línea] [Fecha de consulta: 18 de enero de 2021] Disponible en: https://www.farnell.com/datasheets/1682209.pdf

Guía de usuario de ESP-NOW [En línea] [Fecha de consulta: 20 de diciembre de 2020] Disponible en https://www.espressif.com/sites/default/files/documentation/esp-now_user_guide_en.pdf

Documentación de LoRa [En línea] [Fecha de consulta: 20 de diciembre de 2020] Disponible en: https://lora.readthedocs.io/en/latest/

Portal del fabricante de ZigBee [En línea] [Fecha de consulta: 20 de diciembre de 2020] Disponible en: https://zigbeealliance.org/solution/zigbee/

Especificaciones de ZigBee [En línea] [Fecha de consulta: 20 de diciembre de 2020] Disponible en: https://zigbeealliance.org/wp-content/uploads/2019/11/docs-05-3474-21-0csg-zigbee-specification.pdf

Hojas de datos del sensor Vl53lox [En línea] [Fecha de consulta: 15 de diciembre de 2020] Disponible en: https://www.st.com/resource/en/datasheet/vl53l0x.pdf

Hoja de datos del sensor HC-SR04 [En línea] [Fecha de consulta: 15 de diciembre de 2020] Disponible en: https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf

Hoja de datos del sensor GP2Y02YK0F [En línea] [Fecha de consulta: 15 de diciembre de 2020] Disponible en: https://www.sparkfun.com/datasheets/Sensors/Infrared/gp2y0a02yk_e.pdf

Documentación para desarrolladores de Android [En línea] [Fecha de consulta: 20 de abril de 2021] Disponible en: https://developer.android.com/docs

Documentación para desarrolladores de Firebase [En línea] [Fecha de consulta: 20 de abril de 2021] Disponible en: https://firebase.google.com/docs

Repositorio de la librería Firebase-ESP32 [En línea] [Fecha de consulta: 10 de marzo de 2021] Disponible en: https://github.com/mobizt/Firebase-ESP32