UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

TRABAJO FIN DE GRADO

DISPOSITIVO DE INFORMACIÓN SOBRE PARADAS DE TRANSPORTE PÚBLICO PARA USUARIOS CON DISCAPACIDAD VISUAL BASADO EN IoT

Titulación: **Grado en Ingeniería en Tecnologías de la Telecomunicación**

Mención: **Sistemas Electrónicos**

Autor: **Dª. Sara León Gutiérrez**

Tutores: **D. Valentín De Armas Sosa**

D. Félix B. Tobajas Guerrero

Fecha: **Diciembre de 2017**

UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

TRABAJO FIN DE GRADO

DISPOSITIVO DE INFORMACIÓN SOBRE PARADAS DE TRANSPORTE PÚBLICO PARA USUARIOS CON DISCAPACIDAD VISUAL BASADO EN IoT

HOJA DE FIRMAS

Tutor 1 **Tutor 2**

Fdo. D. Valentín de Armas Sosa **Fdo. D. Félix B. Tobajas Guerrero**

Alumno/a

Fdo. Dª. Sara León Gutiérrez

Diciembre de 2017

UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

TRABAJO FIN DE GRADO

DISPOSITIVO DE INFORMACIÓN SOBRE PARADAS DE TRANSPORTE PÚBLICO PARA USUARIOS CON DISCAPACIDAD VISUAL BASADO EN IoT

HOJA DE EVALUACIÓN

Calificación:_____

Presidente

Fdo. : _____

Vocal **Secretario/a**

Fdo.: _____ Fdo.: _____

Diciembre de 2017

Agradecimientos

A mi madre, mi hermana y mis abuelos, gracias por su amor incondicional, por el apoyo que me han proporcionado, tanto moral como económico, y por estar siempre ahí en cada etapa del camino. Sin ustedes este sueño no habría sido posible.

A mis compañeros, que han hecho más llevadera esta etapa y con los que he compartido grandes experiencias a lo largo de estos años de carrera.

A María, por estar siempre ahí y creer más en mí que yo misma. Gracias por todo, hermana.

Y a mis tutores, gracias por todo el tiempo que me han dedicado durante la elaboración de este trabajo.

Índice General

Índice General I

Índice de Figuras III

Índice de Tablas VII

Acrónimos IX

Capítulo 1. Introducción 1

1.1. Antecedentes 1

1.2. Planteamiento del problema 3

1.3. Objetivos 5

1.4. Peticionario 6

1.5. Estructura del documento 6

Capítulo 2. Elementos de la Plataforma HW/SW 9

2.1. Plataforma HW/SW inicial 9

2.2. Módulos Hardware y Herramientas Software 10

2.2.1. Módulos Hardware 10

2.2.1.1. Dispositivo IoT Photon 10

2.2.1.2. Dispositivo IoT Electron 14

2.2.1.3. Módulo Emic 2 Text-to-Speech 16

2.2.1.4. Módulo GPS 18

2.2.2. Herramientas software 18

2.2.2.1. Particle Cloud 19

2.2.2.2. Particle Build 19

2.2.2.3. Particle Console 22

2.2.2.4. Particle Tinker 23

2.2.2.5. CLI de Particle 24

Capítulo 3. Desarrollo de la funcionalidad de los dispositivos 25

3.1. Acceso al servidor de la empresa Guaguas Municipales S.A. 25

3.1.1. Alternativas para recibir información en el dispositivo 25

3.1.1.1. Webhooks 25

3.1.1.2. Llamadas http 37

3.1.2. Conclusiones 41

3.2. Conversión de la información recibida a audio 41
3.2.1. Librería para shield Text-to-Speech 41
3.2.2. Integración del *shield Text-to-Speech* 42
3.3. Localización por GPS 46
3.3.1. Librería para *shield* GPS 46
3.3.2. Integración del *shield* GPS 48
3.3.3. Cálculo de distancia 51
Capítulo 4. Integración de la plataforma final 55
4.1. Plataforma final con el dispositivo *Photon* 55
4.2. Verificación y validación de la plataforma final 61
4.2.1. Integración del dispositivo *Electron* en la plataforma final 63
Capítulo 5. Conclusiones 69
5.1. Conclusiones 69
5.2. Líneas futuras de trabajo 71
Referencias 73
Pliego de condiciones 77
PL. 1 Condiciones Hardware 77
PL.2 Condiciones Software 77
PL.3 Condiciones Firmware 78
Presupuesto 79
P.1 Trabajo tarifado por tiempo empleado 79
P.2 Amortización del inmovilizado material 81
P.2.1 Amortización del material *hardware* 81
P.2.2 Amortización del software 82
P.3 Redacción del trabajo 83
P.4 Derechos de visado del COITT 84
P.5 Gastos de tramitación y envío 85
P.6 Material fungible 85
P.7 Aplicación de impuestos y coste total 86
Anexos 89
Anexo A. Contenido del CD-ROM 89

Índice de Figuras

Figura 1. Personas con discapacidad en España a diciembre de 2015. 2
Figura 2. Diseño inicial de la plataforma HW/SW. 10
Figura 3.Dispositivos *Spark Core* a la izquierda y *Photon* a la derecha. 11
Figura 4. Componentes del dispositivo *Photon*. 12
Figura 5. Componentes del dispositivo *Electron*. 15
Figura 6. Módulo *Emic 2 Text-to-Speech*. 17
Figura 7. Módulo GPS. 18
Figura 8. Interfaz de *Particle Build*. 20
Figura 9. Pestaña *Devices* de la consola de *Particle*. 22
Figura 10. Pestaña *Logs* de la consola de *Particle*. 23
Figura 11. Interfaz de la aplicación Tinker. 24
Figura 12. Integración de *webhooks* y *Particle Cloud*. 26
Figura 13. Interfaz de la página de Guaguas Municipales S.A. 27
Figura 14. Fichero *.json* para realizar la petición a la página de Guaguas Municipales S.A. 28
Figura 15. *Webhook* creado a través de CLI. 28
Figura 16. Respuesta del *webhook* en la Consola de *Particle*. 29
Figura 17. Petición desde el dispositivo *Photon*. 29
Figura 18. Petición cada 30 segundos. 30
Figura 19. Suscripción a un evento concreto de la respuesta del *webhook*. 31
Figura 20. Problema encontrado con las guaguas que llegan a una parada. 32
Figura 21. Función para realizar el parse. 32
Figura 22. Buffer para almacenar la respuesta del *webhook*. 33
Figura 23. Datos obtenidos por el *webhook*. 33
Figura 24. Primera prueba para manejar la respuesta de un *webhook*. 33
Figura 25. Primera prueba para obtener el tiempo. 34
Figura 26. Función *tryExtractString* con índices. 35
Figura 27. Función para obtener los números de las líneas. 35
Figura 28. Función para obtener los tiempos de las líneas. 36

Figura 29. Finalización de gestión de la respuesta del *webhook*. 37
Figura 30. Estructuras de *request* y *response*. 38
Figura 31. Aumento de tamaño del *buffer* de la librería *HTTPClient*. 39
Figura 32. Llamada HTTP con la librería *HTTPClient*. 39
Figura 33. Información de las líneas con la librería *HTTPClient*. 40
Figura 34. Solución para las guaguas que están llegando o han llegado a una parada. 40
Figura 35. Configuración de los ajustes del módulo *Emic 2*. 41
Figura 36. Función *say()* de la librería *Emic2TTS*. 42
Figura 37. Conexionado *Photon-Emic*. 42
Figura 38. Primera prueba de tratamiento de nombres de paradas con tildes. 43
Figura 39. Resultado del tratamiento de tildes en el caso de la vocal U. 44
Figura 40. Resultado del tratamiento de tildes en el caso de vocales distintas de U. 44
Figura 41. Error al reconocer el índice de vocales distintas de U y aparición del carácter no imprimible. 44
Figura 42. Error con nombres de paradas que contienen más de una vocal con tilde. 45
Figura 43. Tratamiento final de los nombres de paradas con tildes. 46
Figura 44. Distancia entre el Edificio de Electrónica y Telecomunicación y la carretera en *Google Maps*. 47
Figura 45. Función *distanceBetween* de la librería *AssetTrackerRK*. 48
Figura 46. Comprobaciones necesarias para el uso correcto del *shield GPS*. 48
Figura 47. Ejemplo de la librería *TinyGPS* para obtener la localización actual del GPS. 49
Figura 48. Primera prueba de localización del GPS. 50
Figura 49. Latitudes y longitudes de las paradas de guaguas predefinidas. 51
Figura 50. Código para obtener la parada de guagua más cercana. 51
Figura 51. Obtención de la parada de guagua más cercana. 52
Figura 52. Habilitación de *System Thread*. 52
Figura 53. Vista superior del *shield Internet Button*. 56
Figura 54. Vista trasera del *shield Internet Button*. 56
Figura 55. Esquema de la plataforma final con el dispositivo *Photon*. 58
Figura 56. Modificación del dispositivo *Photon* para añadir una UART adicional. 60
Figura 57. Plataforma final con el dispositivo *Photon*. 60
Figura 58. Función *loop()* de la plataforma final. 61

Figura 59. Validación de la plataforma final con cambio en caliente. 62
Figura 60. Esquema de la plataforma final con el dispositivo *Electron*. 63
Figura 61. Carga del código en el dispositivo *Electron* mediante la herramienta CLI de *Particle*. 63
Figura 62. Pulsadores para sustituir el *shield Internet Button*. 64
Figura 63. Plataforma final con el dispositivo *Electron*. 65
Figura 64. Verificación del modo normal de la plataforma final. 66
Figura 65. Comprobación de que la información recibida en la plataforma final es correcta. 66
Figura 66. Verificación del modo GPS de la plataforma final. 67
Figura 67. Verificación del cambio de modo de funcionamiento en caliente de la plataforma final. 67

Índice de Tablas

Tabla 1. Periféricos del dispositivo *Photon*. 12
Tabla 2. Periféricos del dispositivo *Electron*. 15
Tabla 3. Ubicación de las paradas de guaguas del Campus Universitario de Tafira. 50
Tabla 4. Correspondencia entre los pines del dispositivo *Photon* y los botones del *shield Internet Button*. 57
Tabla 5. Relación de equipos hardware. 77
Tabla 6. Relación de herramientas software. 78
Tabla 7. Relación de firmware. 78
Tabla 8. Coeficientes reductores para trabajo tarifado (COITT). 80
Tabla 9. Costes y amortización del hardware (I). 81
Tabla 10. Costes y amortización del hardware (II). 82
Tabla 11. Costes y amortizaciones totales del hardware. 82
Tabla 12. Costes y amortización del software. 83
Tabla 13. Presupuesto, incluyendo trabajo tarifado y amortización del inmovilizado material. 84
Tabla 14. Presupuesto, incluyendo trabajo tarifado, amortización y redacción del trabajo. 85
Tabla 15. Costes de material fungible. 86
Tabla 16. Presupuesto total del Trabajo Fin de Grado. 86

Acrónimos

A

ADC. *Analog-to-Digital Converter*

API. *Application Programming Interface*

ASCII. *American Standard Code for Information Interchange*

B

bps. *Bits por segundo*

C

CAN. *Controller Area Network*

CDMA. *Acceso Múltiple por División de Código*

CD-ROM. *Compact Disc Read-Only Memory*

CLI. *Command Line Interface*

COITT. *Colegio Oficial de Ingenieros Técnicos de Telecomunicación.*

D

DAC. *Digital-to-Analog Converter*

E

EITE. *Escuela de Ingeniería de Telecomunicación y Electrónica*

G

GPIO. *General Purpose Input/Output*

GPS. *Global Positioning System*

GSM. *Global System for Mobile communications*

H

HTML. *HyperText Markup Language*

HW. Hardware

I

I2C. *Inter-Integrated Circuit*

I2S. *Integrated Interchip Sound*

IDE. *Integrated Development Environment*

IGIC. *Impuesto General Indirecto Canario*

IoT. *Internet of Things*

J

JST. *Japan Solderless Terminal*

L

LED. *Light-Emitting Diode*

LiPo. *Lithium-Polymer*

M

MB. *Megabyte*

P

PDF. *Portable Document Format*

PWM. *Pulse-Width Modulation*

R

RAM. *Random Access Memory*

RGB. *Red, Green, Blue*

RTOS. *Real-Time Operating System*

S

S.A.. *Sociedad Anónima*

SIM. *Subscriber Identity Module*

SMPS. *Switched-Mode Power Supply*

SPI. *Serial Peripheral Interface*

T

TFG. *Trabajo Fin de Grado*

U

UART. *Universal Asynchronous Receiver-Transmitter*

ULPGC. *Universidad de Las Palmas de Gran Canaria*

URL. *Uniform Resource Locator*

USB. *Universal Serial Bus*

V

VDC. *Voltios de Corriente Continua*

Capítulo 1. Introducción

En este primer capítulo se presentan los antecedentes y el planteamiento del problema que ha llevado a la realización de este Trabajo Fin de Grado (TFG). También se presentan los objetivos establecidos y la estructura del presente documento.

1.1. Antecedentes

El concepto de *Internet of Things* (IoT) está relacionado con que los objetos físicos vean, escuchen, piensen y realicen trabajos, logrando que compartan información entre ellos y coordinen decisiones. IoT consigue que objetos comunes pasen de ser tradicionales a inteligentes [1]. En la actualidad existen multitud de ámbitos de aplicación de IoT, entre los que se incluyen el transporte, la salud, la agricultura, las casas inteligentes y los vehículos, entre otros [2].

Según estudios realizados por la empresa de consultoría Gartner en febrero de este año [3], se prevé que 2017 finalice con 8.400 millones de dispositivos conectados a nivel mundial, lo que supone un crecimiento del 31% en comparación con el año 2016, y se prevé que en 2020 esta cifra aumente hasta 20.400 millones. Además, la consultora señala que las aplicaciones más utilizadas por los consumidores serán los televisores inteligentes y los decodificadores digitales. A pesar de esto, Gartner destaca también el papel que van a jugar las aplicaciones adaptadas a sectores verticales, como la fabricación de dispositivos de campo, sensores de proceso para plantas de generación eléctrica, o dispositivos de localización en tiempo real para la asistencia sanitaria.

Las tecnologías que dan soporte a IoT están teniendo un avance significativo, siendo cada día más accesibles, disponibles y versátiles, permitiendo un crecimiento más rápido de los dispositivos interconectados a través de Internet. Estos avances técnicos están abriendo un nuevo mundo de diferentes maneras de comunicarse, potenciando las capacidades de comunicación de datos humanos y de máquinas. Un ejemplo de estos avances son las

innovaciones técnicas para personas con necesidades especiales, con el fin de ayudarles en sus actividades cotidianas [4].

Según la *Base estatal de datos de personas con discapacidad,* publicada por el Imserso [5], y tal como se observa en la Figura 1, en diciembre de 2015 había en España alrededor de 3 millones de personas con un grado de discapacidad igual o superior al 33%.

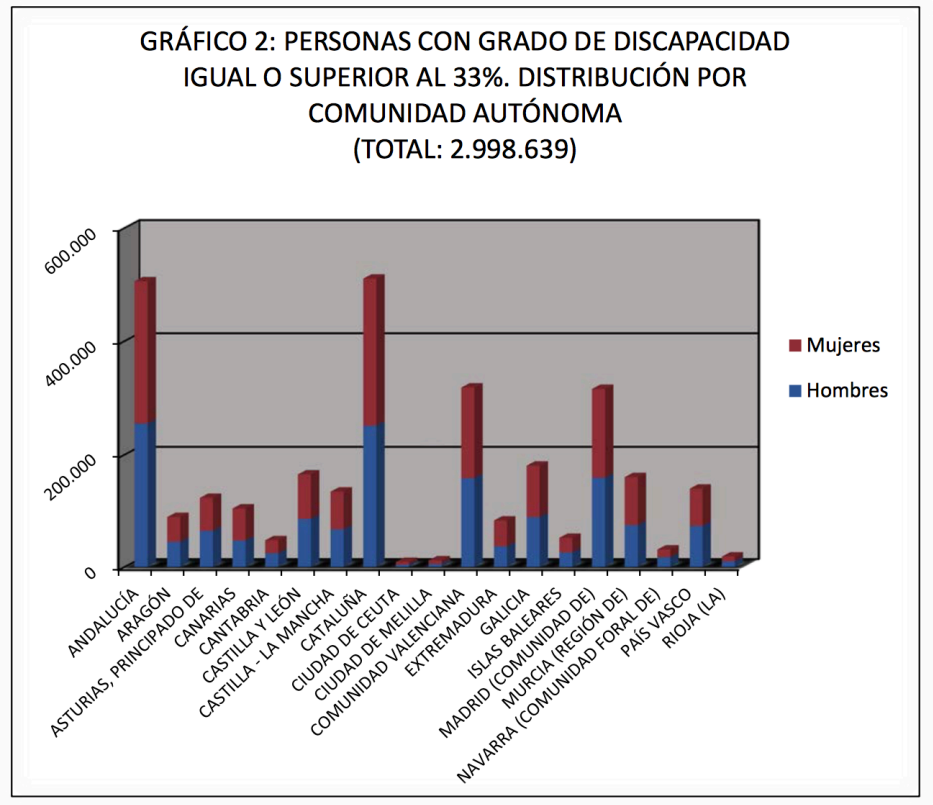


Figura 1. Personas con discapacidad en España a diciembre de 2015.

Las imperfecciones y deficiencias de las capacidades cognitivas son parte de la vida humana. Pueden darse por razones genéticas, enfermedades, traumas, lesiones o incluso debido a la edad. Estas deficiencias sensoriales o cognitivas pueden afectar las experiencias humanas y la calidad de vida, ya que numerosas personas con discapacidades sensoriales, cognitivas o motoras a menudo encuentran numerosas barreras en su vida diaria. La tecnología en general, e IoT en particular, pueden ser herramientas valiosas para mejorar las experiencias humanas [6]. Con el soporte de una tecnología IoT bien adaptada y específica para este tipo de personas, se podría mejorar de manera significativa su calidad de vida [4].

Este tipo de mejoras se pueden observar, por ejemplo, en el área de la automatización del hogar, donde una persona con discapacidad visual puede controlar el termostato a través de una interfaz de teléfono inteligente accesible, o una persona con una discapacidad relacionada con la movilidad puede desbloquear su puerta automáticamente cuando se acerca [7].

En base a todo lo expuesto en este apartado, cada vez es más indiscutible que IoT presenta un gran potencial para mejorar la vida de las personas que presentan algún tipo de discapacidad [7]. Debido a esto, se plantea la plataforma desarrollada en este Trabajo Fin de Grado, basada en dispositivos IoT.

1.2. Planteamiento del problema

La percepción del mundo y de lo que sucede a nuestro alrededor se basa en la información que se obtiene de los sentidos, y la movilidad depende de dicha percepción. La capacidad de moverse o viajar de forma independiente es un deseo humano. Este deseo es válido para todas las personas, incluidas las personas ciegas o con discapacidad visual [6].

La discapacidad visual y la ceguera ocupan el primer y segundo tipo de discapacidad humana con mayor prevalencia mundial, respectivamente. Actualmente, existen aproximadamente 285 millones de personas con discapacidad visual en el mundo, de las cuales 39 millones son ciegas [8]. Concretamente en España hay 70.775 personas ciegas y se prevé que en el futuro este número aumente como consecuencia del incremento de distintos factores de riesgo como el envejecimiento de la población o el aumento de la diabetes [9].

La ceguera, la forma más grave de discapacidad visual, puede reducir la capacidad de las personas para realizar tareas cotidianas y caminar sin ayuda [10]. Muchos sistemas de transporte y de información no son accesibles a todas las personas. El hecho de no poder acceder al transporte es un motivo habitual que desalienta a las personas con discapacidad a buscar trabajo, o que les impide acceder a la atención de salud. La eliminación de

obstáculos en los espacios públicos, transporte, información y comunicación supondría que las personas con discapacidad fueran capaces de participar en la educación, el empleo y la vida social, reduciendo así su aislamiento y dependencia [11].

Incluso un deterioro moderado de la visión puede afectar significativamente sus actividades cotidianas, y a menudo conduce a la exclusión social. La incapacidad para percibir el ambiente circundante, la mala orientación y las capacidades de navegación, o las dificultades para acceder a la información textual, dan como resultado una movilidad limitada de los ciegos y los discapacitados visuales [12].

Viajar se convierte en un desafío especial en las zonas urbanas. La falta de una buena orientación espacial hace difícil encontrar un camino seguro entre los obstáculos, e identificar puntos de interés como paradas de transporte público o pasos peatonales. La imposibilidad de acceder a la información textual como el nombre de la calle, los horarios de transporte público o el número de vehículos, puede dar lugar a dificultades adicionales [12].

El uso del sistema de transporte público requiere el conocimiento de las rutas y los horarios, así como la capacidad de leer el número de guaguas que llegan a las paradas. En la mayoría de los casos, estos datos se presentan sólo en forma visual, por lo que no son accesibles para las personas con ceguera. Aunque después de algún entrenamiento, los discapacitados visuales pueden viajar independientemente a lugares conocidos y a lo largo de rutas familiares, a menudo en tales escenarios deben confiar en un bastón blanco o un perro guía solamente [13].

Los sistemas de guiado electrónico ayudan a los ciegos en la movilidad y facilitan el uso de diversos servicios públicos. La obtención de información precisa sobre la ubicación de los usuarios puede facilitar el acceso a los servicios públicos y utilizarse para acceder a datos relacionados con la posición relativos a los servicios de transporte público, como por ejemplo los horarios de los vehículos, la información sobre las rutas, etc. [13].

1.3. Objetivos

El objetivo principal de este TFG es el diseño e implementación de un dispositivo de orientación e información de las líneas de transporte público de la empresa Guaguas Municipales S.A. [14] para personas que transitan por las paradas cercanas, por lo que contará con una funcionalidad de localización GPS. El dispositivo incluirá la posibilidad de proporcionar información de una parada determinada, o de un conjunto de paradas predefinidas. Esta plataforma estará destinada a personas con discapacidad visual, por lo que la información será recibida por el usuario mediante audio.

La herramienta principal de trabajo será la plataforma de desarrollo hardware *Photon* de la empresa *Particle*, para aplicaciones de IoT, que cuenta con un chip Wi-Fi Broadcom [15] con el que poder realizar el prototipo, para finalmente realizar la implementación final sobre el dispositivo *Electron*, de la misma compañía [16], que proporciona conectividad GSM 3G. La razón por la que se desarrollará el prototipo en el dispositivo *Photon* recae en motivos económicos, ya que la implementación en ambos dispositivos es similar, pero el uso del dispositivo *Electron* conlleva el uso de tarifa de datos móviles.

Con motivo de cumplir con el objetivo principal, se plantean los siguientes objetivos parciales:

- Estudiar y comprender los fundamentos de IoT, el funcionamiento básico de las aplicaciones basadas en el mismo, y su uso actual con fines dedicados a personas con discapacidad visual.
- Profundizar en el aprendizaje de las características de los dispositivos *Photon* y *Electron*, así como en las distintas herramientas que proporcionan.
- Familiarizarse con el software necesario para el desarrollo de aplicaciones en los dispositivos *Photon* y *Electron*.
- Estudiar las características y el funcionamiento del resto de dispositivos que conformarán la plataforma final.

- Desarrollar el *firmware* que permita a un usuario con discapacidad visual recibir información sobre las paradas de transporte público de la empresa Guaguas Municipales S.A.
- Verificar y validar que el sistema funciona de manera adecuada.

1.4. Peticionario

Actúa como peticionario del presente Trabajo Fin de Grado la Escuela de Ingeniería de Telecomunicación y Electrónica (EITE) de la Universidad de Las Palmas de Gran Canaria (ULPGC) como requisito indispensable para la obtención del título de Graduado/a en Ingeniería en Tecnologías de la Telecomunicación, tras haber superado con éxito las asignaturas especificadas en el Plan de Estudios

1.5. Estructura del documento

El presente documento correspondiente a la memoria del TFG, está compuesto por un total de cinco capítulos:

En el Capítulo 1 se realiza una introducción que proporcionará al lector una visión básica acerca de los conceptos que se tratarán en el presente TFG, así como los objetivos del proyecto y otros aspectos formales a tener en cuenta durante el desarrollo del mismo.

El Capítulo 2 contempla el planteamiento inicial de la plataforma HW/SW, a partir de la cual se desarrollará la localización de las paradas de transporte público, describiendo las herramientas utilizadas para la implementación de la plataforma.

En el Capítulo 3 se describe el proceso llevado a cabo para implementar el dispositivo de localización. Se analizarán las herramientas, funciones y librerías que se han empleado en la realización del presente TFG.

El Capítulo 4 detalla la interconexión llevada a cabo para diseñar la versión final de la plataforma HW/SW. Por otra parte, se explican las modificaciones necesarias para asegurar el funcionamiento conjunto de todos los elementos de la plataforma y se valida la misma.

Finalmente, en el Capítulo 5 se exponen las conclusiones obtenidas y las líneas de trabajo futuro del presente TFG.

Capítulo 2. Elementos de la Plataforma HW/SW

Con el fin de comprender su funcionamiento, y de definir los dispositivos que formarán parte de ella, en este capítulo se presenta una descripción detallada de la plataforma HW/SW inicialmente propuesta.

2.1. Plataforma HW/SW inicial

La funcionalidad de la plataforma desarrollada inicialmente en este TFG constará, en principio, de cuatro tareas básicas diferenciadas:

1. Realizar llamadas de tipo *http* al servidor de la empresa Guaguas Municipales S.A. para acceder a los datos relativos a las paradas.
2. Procesamiento de los datos adquiridos del servidor para obtener la información relativa a las guaguas que transitan por una parada concreta, tales como número de línea y tiempo restante para llegar a la parada.
3. Conversión de la información recibida, y procesada previamente, a audio.
4. Localización por GPS con el fin de encontrar la parada más cercana a la ubicación actual del GPS.

El elemento básico para el desarrollo la plataforma HW/SW es el dispositivo *Photon/Electron*. Sin embargo, para llevar a cabo la realización de todas las tareas anteriormente indicadas, es necesario incorporar una serie de componentes adicionales a este dispositivo. En la Figura 2 se muestra el planteamiento inicial para el desarrollo de la plataforma HW/SW.

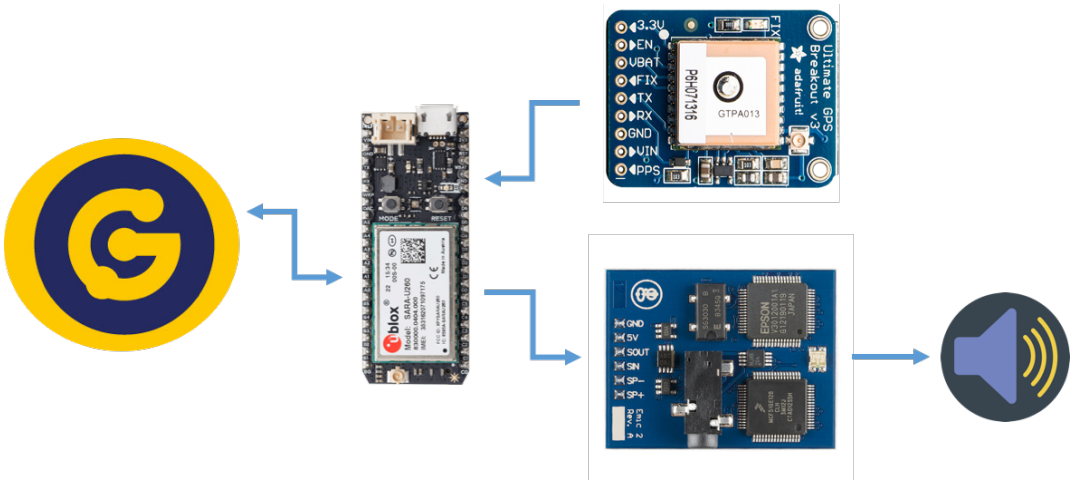


Figura 2. Diseño inicial de la plataforma HW/SW.

Así, de izquierda a derecha se muestra el servidor de la empresa Guaguas Municipales S.A. al cual se accede para obtener la información de las paradas, el dispositivo *Electron* sobre el que se implementará el desarrollo final de la plataforma HW/SW y que será el encargado de realizar la llamada al servidor, los dispositivos de conversión de texto a audio (abajo) y GPS (arriba), y la salida de audio necesaria para que el usuario final pueda utilizar el dispositivo.

2.2. Módulos Hardware y Herramientas Software

Una vez definidas las tareas básicas que determinan la funcionalidad de la plataforma HW/SW a desarrollar, se mostrará una descripción detallada de las especificaciones de los dispositivos que la componen.

2.2.1. Módulos Hardware

A continuación, se presentan los dispositivos y módulos HW que conformarán la plataforma HW/SW inicialmente propuesta.

2.2.1.1. Dispositivo IoT Photon

El kit de desarrollo hardware de IoT *Photon* de la empresa *Particle* [15], ofrece todo lo necesario para construir un producto conectado a Internet. Se ha escogido específicamente el dispositivo *Photon* para el desarrollo de este Trabajo Fin de Grado (TFG) dado su bajo

coste, su programabilidad, su utilización de código abierto, y por integrar un módulo WiFi habilitado para la creación de proyectos y prototipos conectados a Internet, además de por la experiencia previa del grupo de investigación en trabajos anteriores utilizando este dispositivo.

El *Photon* es el segundo dispositivo fabricado por la empresa *Particle* para aplicaciones IoT, ya que antes de éste creó el dispositivo denominado *Spark Core*. La principal diferencia entre ambos radica en que usan diferentes chips WiFi, siendo el del *Spark Core* de la empresa *Texas Instruments* y el del *Photon* de la empresa *Broadcom*. Además, el dispositivo *Photon* utiliza un procesador más rápido, con más memoria RAM [17]. En cuanto a los pines, los de ambos dispositivos son prácticamente iguales, a excepción de que el *Spark Core* contaba con dos pines analógicos (*A6* y *A7*) donde ahora el dispositivo *Photon* presenta dos nuevos pines, que se explicarán más adelante en este apartado. La Figura 3 muestra ambos dispositivos.

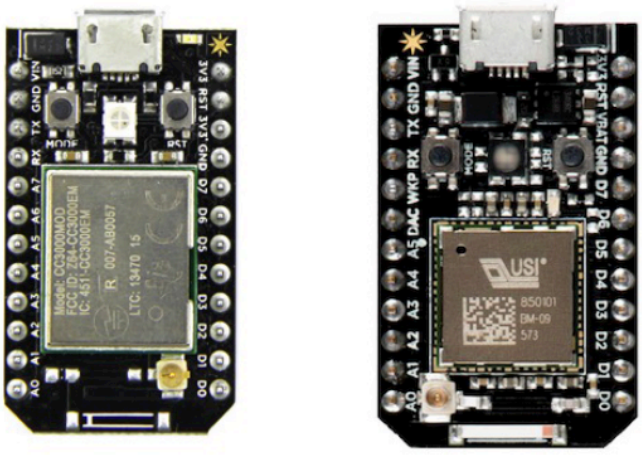


Figura 3.Dispositivos *Spark Core* a la izquierda y *Photon* a la derecha.

El dispositivo *Photon* combina un potente microcontrolador ARM Cortex M3 con un chip WiFi *Broadcom* en un pequeño módulo llamado PØ (P-cero). Además, se añaden un suministro de energía SMPS (*Switched-Mode Power Supply*) de 3,3VDC y componentes de radiofrecuencia y de interfaz de usuario al módulo PØ.

Entre otras características, el dispositivo *Photon* dispone de 1MB de memoria *Flash* y 128KB de RAM, además de varios LED (*Light-Emitting Diode*) integrados, 18 entradas/salidas de

propósito general (GPIO), periféricos avanzados y un sistema operativo en tiempo real (*FreeRTOS*). Este dispositivo cuenta con una gran cantidad de interfaces analógicas, digitales y de comunicación, tal y como se muestra en la Tabla 1 [15].

Tabla 1. Periféricos del dispositivo *Photon*.

Tipo de periférico	Cantidad	Entrada/Salida
Digital	18	Entrada/Salida
Analógico(ADC)	8	Entrada
Analógico(DAC)	2	Salida
SPI	2	Entrada/Salida
I2S	1	Entrada/Salida
I2C	1	Entrada/Salida
CAN	1	Entrada/Salida
USB	1	Entrada/Salida
PWM	9	Salida

La Figura 4 muestra un dispositivo *Photon* en el que se destacan sus principales componentes.

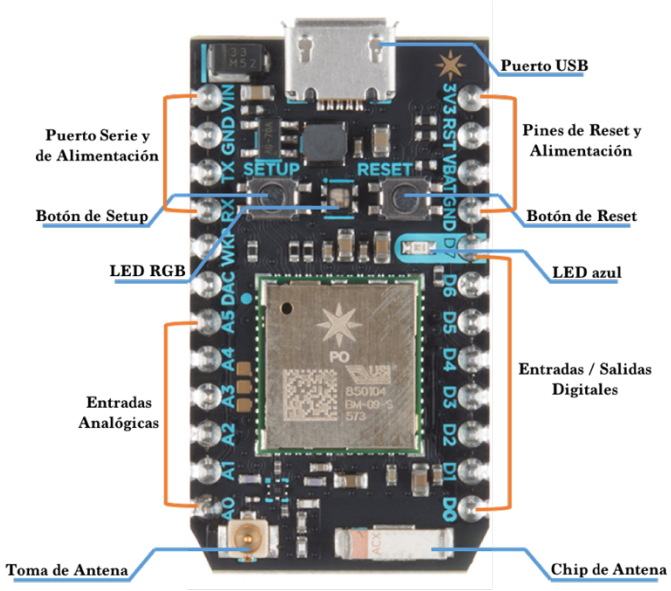


Figura 4. Componentes del dispositivo *Photon*.

Los dos botones disponibles en la plataforma, denominados *Setup* y *Reset*, permiten configurar nuevas credenciales WiFi y reiniciar el dispositivo, respectivamente, además de poder utilizarse de manera conjunta para provocar el restablecimiento de los valores de fábrica por defecto.

En la parte superior se encuentra el puerto micro-USB. Aunque el objetivo principal de éste es proporcionar alimentación al dispositivo *Photon*, también se puede utilizar para la programación del dispositivo vía USB y para realizar comunicaciones con un ordenador. A la derecha del conector micro-USB están situados los pines de *reset* y alimentación (*3V3*, *RST*, *VBAT* y *GND*). El dispositivo *Photon* convierte la energía de entrada proporcionada a través de la alimentación del conector micro-USB, o del pin *VIN* en un suministro de 3,3 Voltios, ya que toda la lógica interna del dispositivo opera con 3,3V. El pin *RST* se puede utilizar, al igual que el botón *Reset*, para reiniciar el sistema.

El pin *VBAT* permite conectar una pequeña batería de reserva a la del dispositivo *Photon*, proporcionándole así alimentación mientras éste se encuentra en modo *deep sleep*, conservando el contenido de su memoria, de forma que cuando vuelva a funcionar en modo normal, pueda continuar en el estado en el que se encontraba previamente.

Los pines *D0* a *D7* son pines de propósito general que pueden actuar como entradas o salidas digitales. Además, los pines *D0* a *D3* también pueden actuar como salidas analógicas utilizando técnicas *PWM* (*Pulse-Width Modulation*). Tal y como se aprecia en la Figura 4, existe también un LED azul situado junto al pin *D7* que se encuentra conectado directamente al terminal *D7*.

Este dispositivo posee un chip de antena que permite que éste funcione correctamente cuando utiliza conectividad WiFi. Además, dispone también de una pequeña toma a través de la cual se puede conectar una antena externa. Con esto se conseguiría ampliar el rango de cobertura WiFi, añadiendo una antena más sensible o direccional. Por defecto, el dispositivo *Photon* intentará elegir la mejor antena, pero también se puede controlar qué antena utilizar mediante *firmware*.

Los pines *A0* a *A5* constituyen entradas analógicas que trabajan con tensiones de entre 0 y 3,3V. Los pines analógicos también se pueden utilizar como entradas o salidas digitales, como los pines *D0* a *D7* y, al igual que los pines digitales, algunos pines analógicos (*A4*, *A5*) también se pueden utilizar como salidas analógicas PWM. A continuación, se encuentran los dos nuevos pines, *DAC* y *WKP*. El pin del conversor analógico-digital (*Digital Analog*

Converter, DAC), se trata de un pin de salida analógica especial, capaz de proporcionar voltajes comprendidos entre 0 y 3,3V. A su lado está el pin *WKP*, utilizado para activar al dispositivo *Photon* después de que se ha puesto en modo *sleep* o *standby*.

Los pines *TX* y *RX* (Transmisión y Recepción, respectivamente) se utilizan para la comunicación serie. Por último, situados por encima de estos pines se encuentran un segundo pin *GND,* y el pin *VIN*. Tal y como se comentó anteriormente, se puede alimentar el dispositivo *Photon* suministrando entre 3,6V y 5,5V al pin *VIN*, como alternativa al uso del puerto micro-USB.

Por último, una de las características más útiles que posee este dispositivo, es que posee un LED de tipo RGB (*Red*, *Green*, *Blue*) que proporciona información acerca del estado del dispositivo empleando distintos colores y tasas de parpadeo para indicar lo que está pasando en el dispositivo. Por ejemplo, si se encuentra conectado a una red WiFi estará parpadeando muy despacio en color celeste o, si se está cargando un programa en memoria Flash, este LED parpadeará rápidamente en color violeta [17].

2.2.1.2. Dispositivo IoT Electron

El kit de desarrollo *Electron 3G*, también de la empresa *Particle*, sirve para crear productos y proyectos conectados a través de redes de telefonía móvil GSM. Viene con una tarjeta SIM (Nano 4FF) y un plan de datos económico para bajo ancho de banda disponible en más de 100 países a través de operadoras de telefonía como *Telefónica*, *AT&T*, *T-Mobile* y muchas más. De la misma manera que el dispositivo *Photon*, el dispositivo *Electron* incorpora además herramientas de desarrollo y una plataforma en la nube para la gestión y la interactividad con el dispositivo. Puede alimentarse a través del pin *VIN* (3,9V–12VDC), mediante el conector micro-USB, o por batería de polímetro de Litio (LiPo). El módulo celular de este dispositivo es sólo para GSM, no soportando redes CDMA [18][16].

La Figura 5 muestra un dispositivo *Electron*, destacando sus principales componentes.

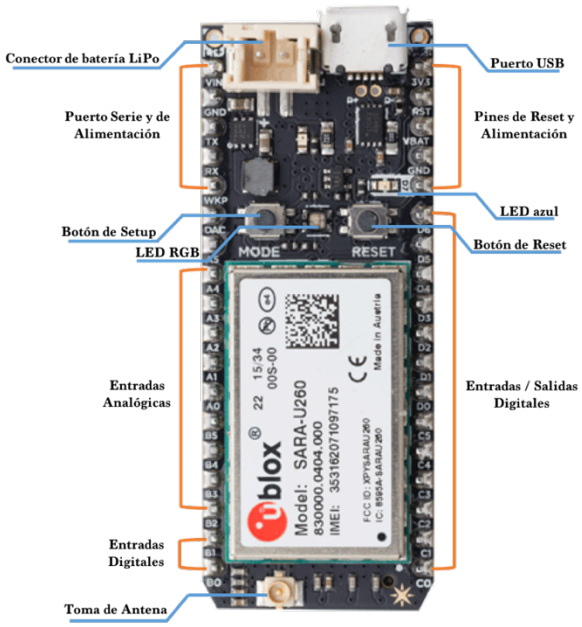


Figura 5. Componentes del dispositivo *Electron*.

El dispositivo *Electron* combina un microcontrolador ARM Cortex M3 con un módulo celular U-blox de la serie SARA-U2 para conexión con redes GSM 3G y 2G, además de tener otras características destacables como son su alimentación a 3,9VDC, 30 GPIO, periféricos avanzados, y un diseño *open source* que facilita su integración en un producto.

Entre otras características, el dispositivo *Electron* dispone de 1MB de memoria *Flash* y 128KB de RAM, además de un sistema operativo en tiempo real (*FreeRTOS*). Este dispositivo cuenta con una gran cantidad de interfaces analógicas, digitales y de comunicación, tal y como se muestra en la Tabla 2.

Tabla 2. Periféricos del dispositivo *Electron*.

Tipo de periférico	Cantidad	Entrada/Salida
Digital	30	Entrada/Salida
Analógico(ADC)	12	Entrada
Analógico(DAC)	2	Salida
UART	3	Entrada/Salida
SPI	2	Entrada/Salida
I2S	1	Entrada/Salida
I2C	1	Entrada/Salida
CAN	2	Entrada/Salida
USB	1	Entrada/Salida
PWM	13	Salida

Tal y como ocurre con el dispositivo *Photon*, el dispositivo *Electron* cuenta también con los botones de *Setup* y *Reset*, el LED de tipo RGB y el puerto micro-USB. A la derecha del conector micro-USB están situados también los pines de *reset* y alimentación.

Los pines *D0* a *D7* y *C0* a *C5* son pines de propósito general que pueden actuar como entradas o salidas digitales. Además, los pines *D0* a *D3* y los pines *C4* y *C5* también pueden actuar como salidas analógicas utilizando técnicas PWM (*Pulse-Width Modulation*) y existe también un LED azul situado junto al pin *D7* que se encuentra conectado directamente al terminal *D7*.

A los pines desde *A0* a *A5* se le suman los pines de *B2* a *B5* como entradas analógicas. Los pines analógicos también se pueden utilizar como entradas o salidas digitales y algunos pines analógicos también se pueden utilizar como salidas analógicas PWM. A continuación, se encuentran los pines *DAC* y *WKP*, tal como están situados en el dispositivo *Photon,* y con la misma funcionalidad.

Los pines *TX* y *RX* se utilizan para la comunicación serie, y situados por encima de estos pines se encuentran un segundo pin *GND* y el pin *VIN* [16].

Por tanto, entre las diferencias más significativas entre el dispositivo *Photon* y el dispositivo *Electron* se puede destacar que el *Electron* incorpora un módulo GSM en vez del módulo WiFi y que añade 12 GPIO. Por otra parte, mientras que el dispositivo *Photon* solamente cuenta con una UART, el dispositivo *Electron* incluye 2 UART adicionales, que serán necesarias para realizar la implementación de la plataforma HW/SW final propuesta en este TFG.

2.2.1.3. Módulo Emic 2 Text-to-Speech

El módulo *Emic 2 Text-to-Speech* de la empresa *Parallax* [19] constituye uno de los principales elementos de la interfaz con el usuario en el presente TFG. Este sintetizador de voz multilingüe convierte texto digital en habla de sonido natural.

Entre otras características, el módulo *Emic 2* requiere de una alimentación de 5V, tensión que es capaz de suministrar tanto el dispositivo *Photon* como el dispositivo *Electron*, y una comunicación por puerto serie de 9600 bps asíncrono serie (8N1), a través de las conexiones *SOUT* y *SIN*.

Tal como se observa en la Figura 6, para la salida de audio, el módulo *Emic 2* ofrece la posibilidad de añadir un altavoz con una impedancia de 8 Ω a través de las conexiones *SP-* y *SP+*, o de conectar unos auriculares, altavoces amplificadores o cualquier otro equipamiento de audio a través de su puerto *Jack* de 3.5mm.

Por otra parte, el módulo *Emic 2* cuenta con un LED que indica el estado de operación en el que se encuentra el dispositivo, siendo, por ejemplo, verde cuando está alimentado y esperando a recibir un comando, o rojo cuando está realizando la conversión de texto a audio.

El conjunto de comandos para el módulo se compone únicamente de caracteres imprimibles basados en *ASCII* y permite seleccionar diferentes idiomas (Inglés o Español), el cambio entre 9 voces diferentes predefinidas y configurables, e incluso controlar los parámetros de voz tales como la velocidad del habla o el tono.

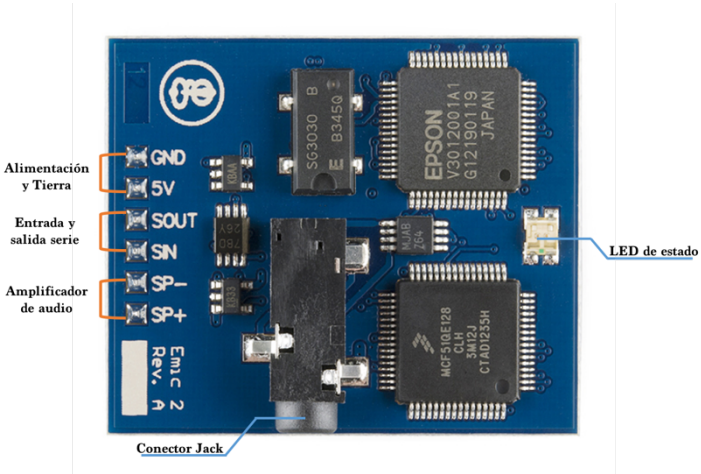


Figura 6. Módulo *Emic 2 Text-to-Speech*.

2.2.1.4. Módulo GPS

El módulo *Ultimate GPS Breakout* de la empresa *Adafruit*, mostrado en la Figura 7, se alimenta con tan solo 5V, tensión que es capaz de suministrar tanto el dispositivo *Photon* como el dispositivo *Electron*.

En cuanto a la comunicación, este módulo se comunica a través de puerto serie de 9600 bps mediante los pines *TX* y *RX*. Además tiene un LED indicador del estado de conexión, que parpadea cada segundo mientras está buscando satélites, y cada 15 segundos cuando se ha conectado a alguno de ellos.

Por otra parte, cuenta con una antena interna, además de ofrecer la posibilidad de conectar una antena externa a través de su conector *u.FL* [20].

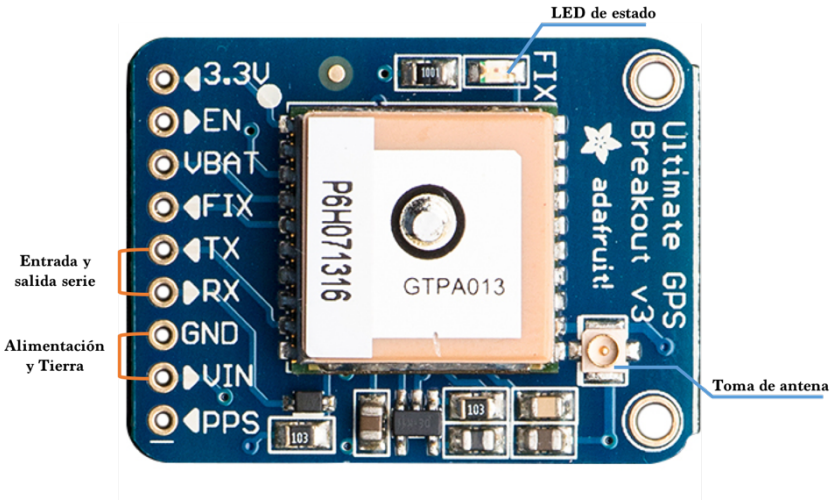


Figura 7. Módulo GPS.

2.2.2. Herramientas software

Para llevar a cabo el desarrollo del *firmware* es importante conocer cuáles son aquellas herramientas que posee el dispositivo *Photon* para llevar a cabo su programación, así como el lenguaje en el que se basa.

El dispositivo *Photon* se puede programar utilizando lenguaje C, C++ o, incluso, ensamblador. *Particle* utiliza *Wiring*, un *framework* de código abierto para microcontroladores, cuyas funciones son compatibles con la plataforma *Arduino*. Esto implica que buena parte de código de *Arduino* podrá ejecutarse sin modificaciones. Además, *Particle* cuenta con una amplia colección de librerías creadas a partir de la contribución de terceros, lo cual simplifica las tareas de programación [17].

2.2.2.1. *Particle Cloud*

Particle Cloud es un servicio gratuito en la nube, que tiene algunas características excelentes para el desarrollo de proyectos conectados, incluyendo actualizaciones de *firmware on the air*, una API REST fácil de usar, y desarrollo de *firmware* compatible con IDE (*Integrated Development Environment*) web y locales.

Particle Cloud presenta una seguridad robusta y una infraestructura confiable. Es un sistema altamente distribuido capaz de soportar millones de conexiones simultáneas de dispositivos, con escalamiento automático e infraestructura de nube redundante dirigida a mantener los dispositivos en línea en todo momento.

Una de las claves de *Particle Cloud* para este proyecto en concreto, es que admite integraciones para permitir la conexión perfecta de datos de dispositivos con herramientas tales como *webhooks* [21].

2.2.2.2. *Particle Build*

Particle ofrece la posibilidad de realizar el desarrollo de software de usuario en una aplicación fácil de usar y que se ejecuta directamente desde un navegador web. El IDE *Particle Build* sirve para desarrollar código, compilarlo y cargarlo en la memoria *Flash* de cualquier dispositivo *Photon* o *Electron* asociado a la cuenta de usuario, de forma inalámbrica. Este método es el que se utilizará para el desarrollo de la parte *firmware* de este TFG. En la Figura 8 se muestra la interfaz de la plataforma *Particle Build* [22].

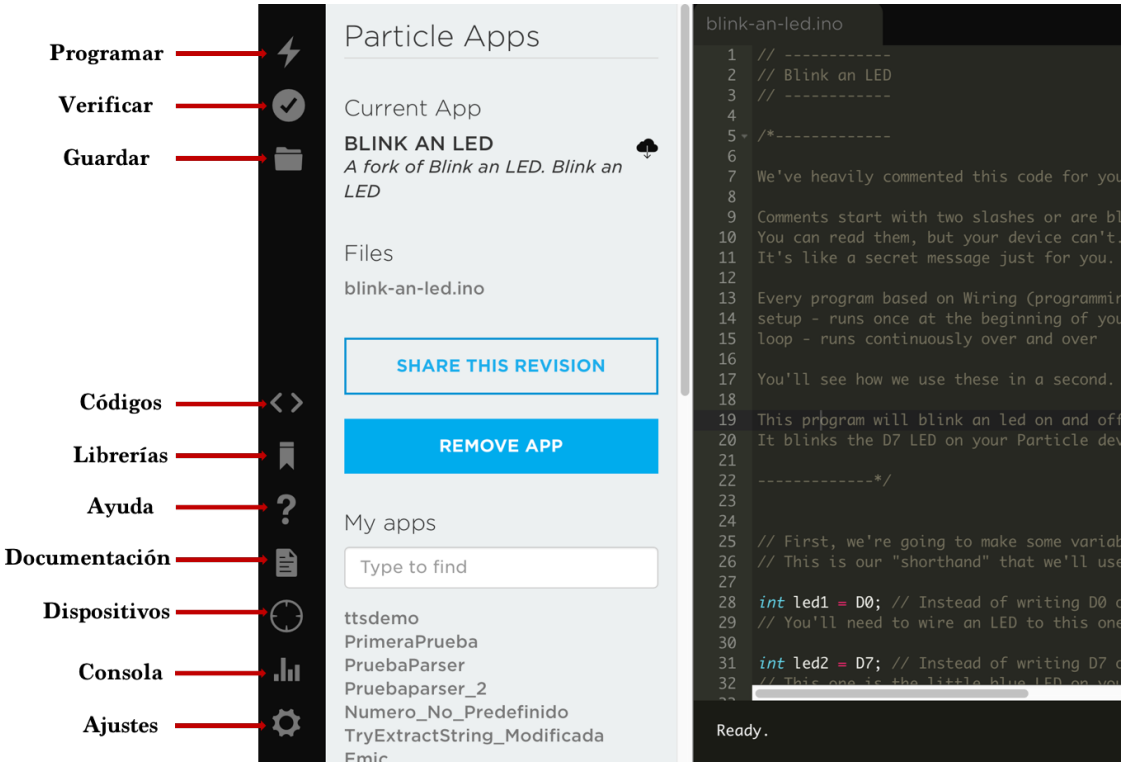


Figura 8. Interfaz de *Particle Build*.

La interfaz de usuario de *Particle Build* muestra una barra de navegación a la izquierda. En la parte superior, hay tres botones que realizan funciones importantes:

- *Programar*: Programa el dispositivo con el código actual. Esto inicia una actualización de *firmware* y carga el nuevo software en el dispositivo seleccionado.
- *Verificar*: Compila el código, sin cargar el código en el dispositivo. Si hay algún error en el código, se muestra en la consola de depuración, en la parte inferior de la pantalla.
- *Guardar*: Almacena los cambios que se han realizado en el código actual.

En la parte inferior, hay siete botones más para navegar por el IDE *Particle Build*:

- *Códigos*: Muestra una lista de las aplicaciones de usuario que se han desarrollado y permite seleccionar cuál editar/programar/compilar.
- *Librerías*: Permite explorar las librerías disponibles y que pueden incluirse fácilmente en los códigos desarrollados.

- *Ayuda*: Muestra la lista de colores que puede tomar el LED RGB del dispositivo *Photon* o *Electron*, junto con el significado de cada color.
- *Documentación*: Redirige a la página de documentación de *Particle*.
- *Dispositivos*: Muestra una lista de los dispositivos asociados a la cuenta del usuario, para elegir sobre cuál trabajar y obtener más información sobre cada uno de ellos.
- *Consola*: Redirige a la página de la consola de *Particle* (*Particle Console*), en la que se pueden visualizar los eventos enviados por los dispositivos.
- *Ajustes*: Permite modificar la contraseña, cerrar la sesión actual, y además muestra el *token* de acceso que contiene las credenciales de identificación del usuario que utiliza el entorno *IDE* de *Particle*.

Por último, en la parte derecha del entorno de desarrollo se encuentra localizado el panel en el que se escribirá todo el código necesario para implementar las aplicaciones de usuario. En la parte inferior de dicho panel se halla una sección que determina el estado del código, mostrando los errores, el tamaño de memoria utilizado, o si se ha realizado la programación el dispositivo de forma satisfactoria, entre otros.

Todos los procedimientos se realizan en la nube, desde el proceso de compilación del código hasta la programación del dispositivo. Además, se puede optar por editar desde el navegador, utilizando la página web de *Particle*, o descargar su *IDE* y editar el código de forma local [17].

Por otra parte, es importante tener en cuenta que la estructura que siguen todos los programas desarrollados para el dispositivo *Photon* (formato *.ino*) consta de dos funciones básicas: *setup*() y *loop*(). La función *setup*() solamente se ejecuta una vez, cuando se enciende el dispositivo o cuando se realiza un *reset*, y es utilizada normalmente para inicializar pines y objetos. Dentro de la función *loop*() se desarrolla el código que define el funcionamiento de la aplicación [23].

2.2.2.3. *Particle Console*

Particle Console es el centro de comandos IoT centralizado de la empresa *Particle*. Proporciona interfaces para facilitar la interacción y la gestión de los dispositivos de *Particle*.

La consola ofrece muchas características útiles para el usuario mientras se desarrolla un producto o proyecto IoT. Mediante la consola se puede ver la última vez que un dispositivo ha estado conectado, se puede depurar un problema de *firmware* observando los registros de eventos, configurar un *webhook* para enviar datos a un servicio externo, etc.

Desde la pestaña *Devices*, mostrada en la Figura 9, se puede ver la lista de los dispositivos asociados con la cuenta del usuario y acceder a información específica acerca de cada dispositivo, incluyendo su ID único, su nombre, el tipo de dispositivo (es decir, *Photon* o *Electron*) la última vez que se conectó a la nube, y si el dispositivo está actualmente en línea.

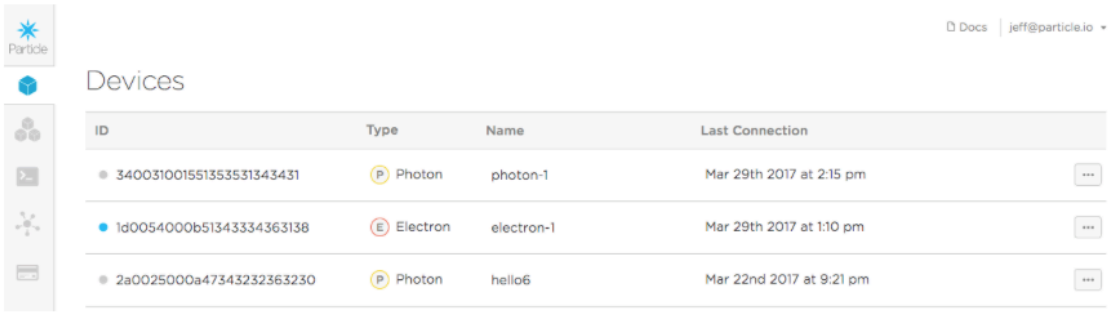


Figura 9. Pestaña *Devices* de la consola de *Particle*.

Por otra parte, desde la pestaña *Logs*, representada en la Figura 10, se pueden visualizar los eventos publicados desde los dispositivos mediante la sentencia *Particle.publish*(), en tiempo real. Esta es una herramienta extremadamente útil a la hora de trabajar con *webhooks* [24].

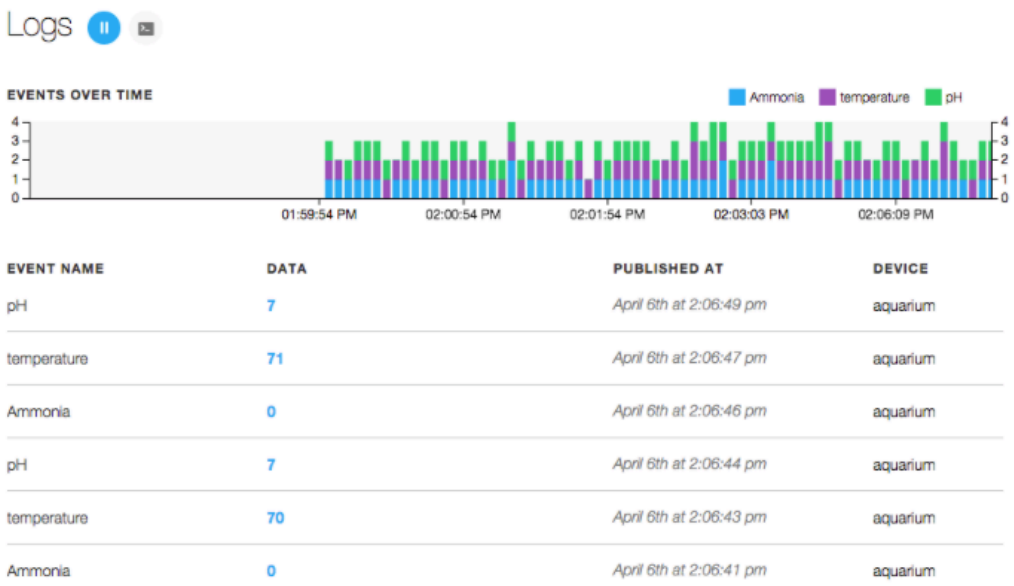


Figura 10. Pestaña *Logs* de la consola de *Particle*.

2.2.2.4. *Particle Tinker*

El dispositivo *Photon* viene pre-programado con un gestor de arranque (o *bootloader*) y una aplicación de usuario denominada *Tinker*. Se trata de una librería de *firmware* capaz de ejecutar funciones básicas de GPIO a través de una API (*Application Programming Interface*). Funciona con una aplicación móvil para *iOS* y *Android* también llamada *Tinker*, que permite conmutar fácilmente pines digitales y realizar lecturas analógicas y digitales sin necesidad de escribir ni una línea de código.

En la Figura 11 se muestra la interfaz de la aplicación móvil *Tinker*. A la izquierda se encuentra la pantalla con la lista de los dispositivos asociados a la cuenta de usuario y a la derecha se muestran los 16 pines GPIO del dispositivo *Photon* seleccionado. Mediante la pantalla de pines se puede controlar y leer el estado de los pines del dispositivo directamente desde el móvil [25].

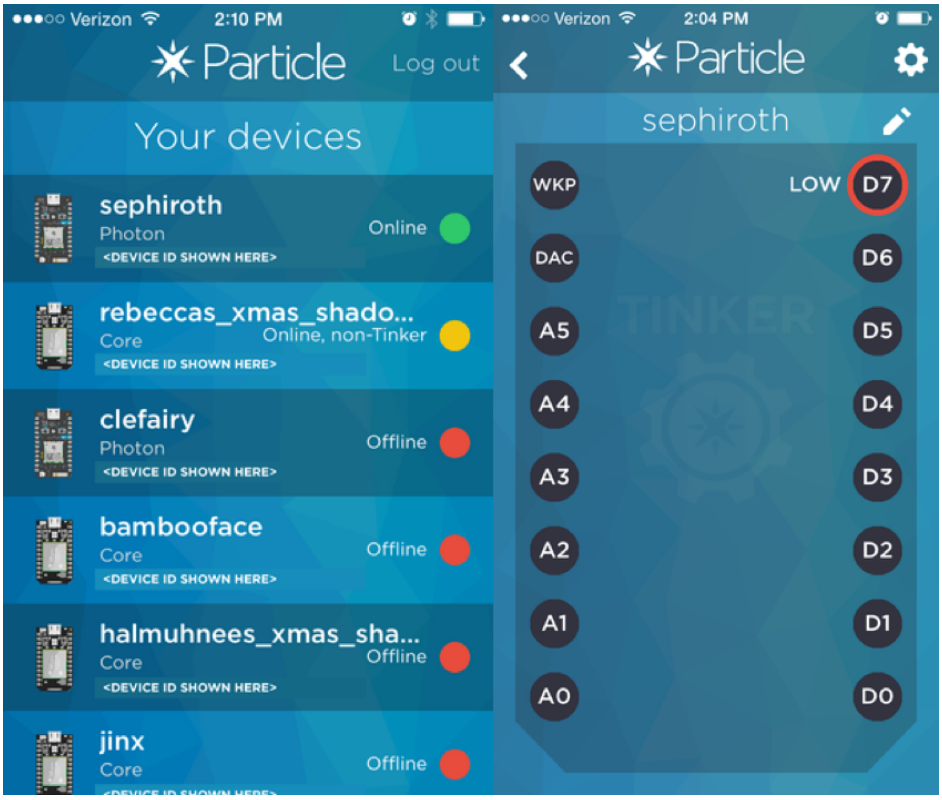


Figura 11. Interfaz de la aplicación Tinker.

Por otra parte, para empezar a trabajar con los dispositivos de *Particle*, es necesario asociarlos antes a la cuenta de usuario. Esta aplicación permite realizar esta operación de forma sencilla.

2.2.2.5. *CLI de Particle*

La herramienta CLI (*Command Line Interface*) permite realizar la configuración de los dispositivos, la monitorización de datos, la configuración de integraciones, etc., mediante el tecleo de comandos en la ventana de terminal del ordenador. Esta herramienta es clave para realizar el proceso de depuración del *firmware* del presente TFG [26].

Capítulo 3. Desarrollo de la funcionalidad de los dispositivos

En este capítulo se describe con detalle la implementación de las diferentes funciones definidas en la especificación de la plataforma HW/SW a desarrollar en este TFG, a partir de los módulos HW y de las herramientas SW presentadas en el capítulo anterior.

Por cuestiones de coste, relativos al pago de tarifas de datos, el desarrollo inicial de las funciones definidas se realiza sobre el dispositivo *Photon.* Este dispositivo es compatible con el dispositivo *Electron,* sobre el que se integrará la plataforma final.

3.1. Acceso al servidor de la empresa Guaguas Municipales S.A.

La primera tarea del presente TFG consiste en acceder al servidor de la empresa de Guaguas Municipales S.A. con el fin de obtener los datos relativos a las paradas. Por tanto, el primer paso es definir la manera en la que se accede a dicho servidor desde el dispositivo *Photon/Electron*.

3.1.1. Alternativas para recibir información en el dispositivo

Para obtener la información del servidor de la empresa de Guaguas Municipales S.A., se consideraron dos posibles opciones con el fin de realizar las consultas necesarias desde el dispositivo *Photon/Electron*: la implementación de *webhooks* y la realización de llamadas *http* directamente desde el dispositivo.

3.1.1.1. Webhooks

Los *webhooks* son una forma sencilla y flexible de que los dispositivos de la empresa *Particle* realicen peticiones a otras aplicaciones y servicios en Internet. Constituyen un puente entre el mundo físico y el digital, ayudando a obtener datos donde sea necesario. Se pueden utilizar para guardar información valiosa en una base de datos, visualizar los datos que se leen de un sensor, enviar un mensaje de texto, y otras funciones [27].

Un *webhook* escucha un evento específico publicado por dispositivos como el *Photon* o el *Electron*. Una vez que se publica el evento en la nube, el *webhook* activa una petición (*request*) a una URL (*Uniform Resource Locator*) en la web. Los *webhooks* permanecen a la espera de recibir eventos desde los dispositivos, de forma que cuando se envía un evento determinado, el *hook* (*callback*) envía una petición a una aplicación web definida, incluyendo toda la información necesaria.

Para crear un *webhook* es necesario definir su comportamiento (ante qué tipo de eventos debe reaccionar) y configurar el servidor para recibir y gestionar la información resultante.

Los *webhooks* se pueden configurar para crear diferentes tipos de solicitudes web. El tipo más común de solicitud de *webhook* es POST, que es un método de envío de datos a otro servidor web. En el caso de los *webhooks* de la empresa *Particle*, esto significaría enviar datos desde los dispositivos a un servicio web de terceros. Otros tipos de solicitudes web, como GET y PUT también se pueden realizar con *webhooks*.

Una vez que se realiza la petición a un servidor web, éste devolverá los datos como resultado de la solicitud realizada. Cuando esto sucede, los dispositivos pueden suscribirse a un nombre de evento específico para recibir la respuesta del servidor web y utilizarla en la lógica del *firmware*.

La combinación de *webhooks* con el sistema de eventos de *Particle Cloud* crea una forma muy eficiente para aprovechar herramientas y servicios en línea e integrarlos en el producto conectado, como se muestra en la Figura 12 [27].

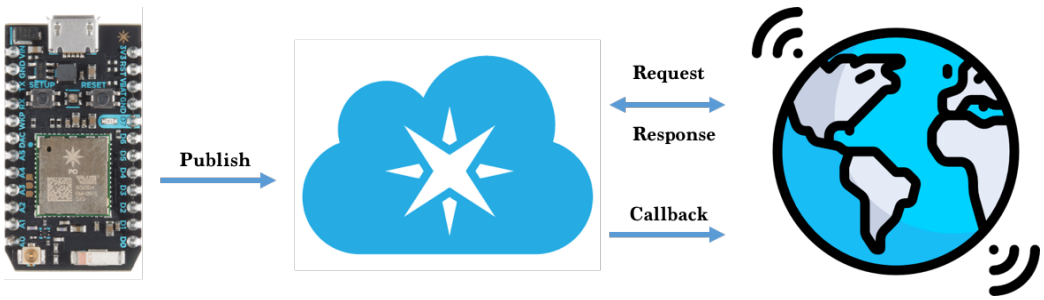


Figura 12. Integración de *webhooks* y *Particle Cloud*.

Para crear un *webhook* es necesario especificar ciertas propiedades en un fichero *.json*. Algunas de las propiedades que pueden ser incluidas en la configuración de un *webhook* son "*eventName*", con el que todos los eventos que empiecen con el nombre definido activan el *webhook*; "*url*", la dirección de la página web a la que se accede cuando se activa el *webhook*; o "*requestType*" con el que se define si se quiere hacer una petición de POST o de GET, entre otras [28].

Una vez estudiado el funcionamiento de los *webhooks*, se realizaron pruebas enviando peticiones a la página web http://paradas.guaguas.com/ de la empresa Guaguas Municipales S.A [17], obteniendo en el navegador la respuesta mostrada en la Figura 13 .

Figura 13. Interfaz de la página de Guaguas Municipales S.A.

Esta página ofrece la posibilidad de realizar la consulta de los horarios de guaguas que transitan por una parada concreta, mediante la introducción del número de dicha parada, como se puede observar en la Figura 13 para el caso concreto de la parada 387, asociada a la página web http://paradas.guaguas.com/387.

Así, para implementar el *webhook* asociado al acceso a esta página web, en primer lugar, es necesaria la creación de un fichero *.json* que incluya el nombre del evento, la dirección URL, y el tipo de petición que se va a realizar, entre otros parámetros. El fichero creado se muestra en la Figura 14.

```
{
  "eventName": "parada-guagua",
  "url": "http://paradas.guaguas.com/387",
  "requestType": "GET",
  "query: undefined",
  "headers":
  {
    "Authorization": undefined"
  },
  "mydevices": true
}
```

Figura 14. Fichero *.json* para realizar la petición a la página de Guaguas Municipales S.A.

A continuación, se registra el *webhook* en *Particle Cloud* a través del terminal, obteniéndose su identificación propia, tal como se muestra en la Figura 15. Para realizar este registro, se emplea el comando "*particle webhok create*" seguido del nombre del evento establecido en el fichero *.json* y de la dirección URL también asociada al *webhook* en el fichero. En este caso, se realiza la petición para obtener los datos de la parada número 387 con el comando *"particle webhook create parada-guagua http://paradas.guaguas.com/387"* .

```
Sending webhook request  { uri: '/v1/webhooks',
  method: 'POST',
  json:
   { event: 'parada-guagua',
     url: 'http://paradas.guaguas.com/387',
     deviceid: undefined,
     requestType: undefined,
     mydevices: true },
  headers: { Authorization: 'Bearer da56cc8f68a61271eaa76bccdb2d36194589e774' } }
Successfully created webhook with ID 58a5e4daf37a58177a9fe2bf
```

Figura 15. *Webhook* creado a través de CLI.

Una vez creado el *webhook*, se realiza la publicación del evento a través del terminal y se visualiza el resultado a través de la consola de *Particle*. El código desarrollado inicialmente es el que se muestra en la Figura 16.

Figura 16. Respuesta del *webhook* en la Consola de *Particle*.

Como resultado, se reciben los datos de las líneas que pasan por la parada número 387 a la hora en la que se realiza la publicación del evento a través del CLI. Una vez hecho esto, se pasa a crear el código necesario para que sea el dispositivo *Photon* el que realice la petición automáticamente, mostrado en la Figura 17.

```
1     #include "application.h"
2
3     void setup() {
4
5     }
6
7     void loop() {
8
9     delay(30000); //espera 30 segundos antes de volver a publicar
10          Particle.publish("parada-guagua");
11
12    }
13
```

Figura 17. Petición desde el dispositivo *Photon*.

Mediante esas líneas de código se obtiene, cada 30 segundos, la información relativa a la parada número 387, tal como se muestra en la Figura 18. Observando los datos de los eventos, se observa que la información relevante, correspondiente al número de línea y tiempo que falta para que pase por la parada, se encuentra siempre en el evento con el

nombre *hook-response/parada-guagua/2*. Por tanto, el siguiente paso sería suscribirse solamente a ese evento concreto.

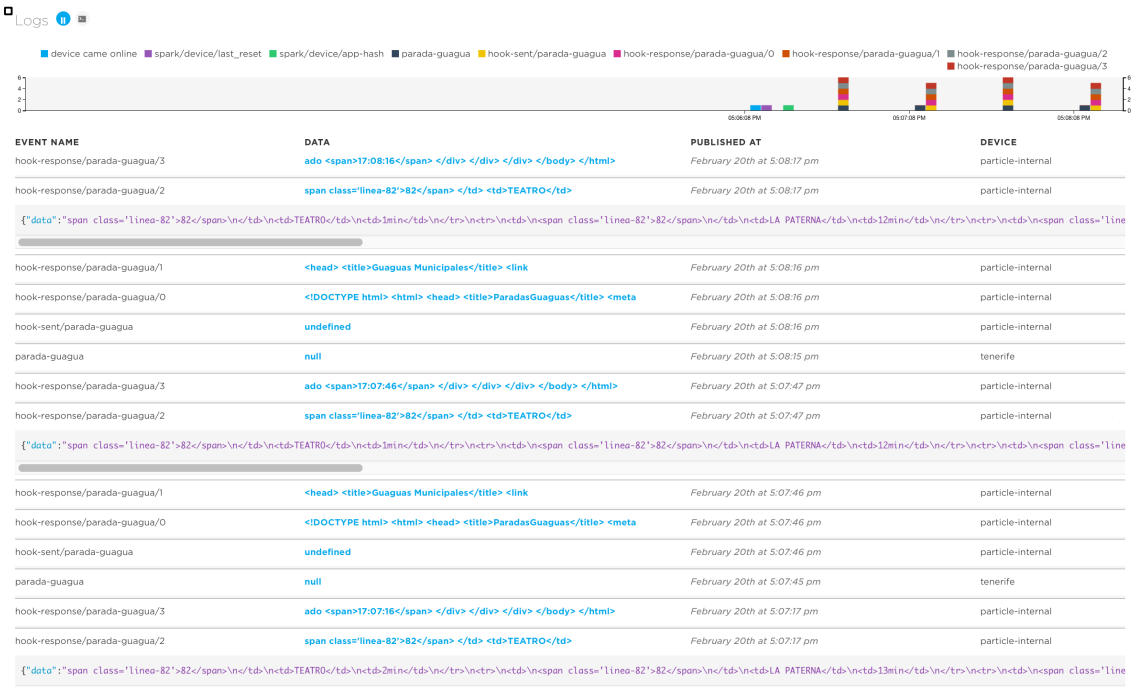


Figura 18. Petición cada 30 segundos.

Para suscribirse a esa respuesta concreta, se añade en la función *setup*() del *firmware* del dispositivo *Photon* la suscripción mediante la sentencia *Particle.suscribe*(), tal como se muestra en la Figura 19. Dentro de la función *obtenerHorario*() se realiza, como se mostrará más adelante, la gestión de la respuesta obtenida mediante el *webhook* registrado.

```
#include "application.h"

//función para manejar la respuesta
void obtenerHorario(const char *nombre, const char *datos) {

      Serial.println("Running obtenerHorario function");
   //1 vez cada un "hook-response/parada-guagua/X"

}


void setup() {

   Serial.begin(9600);
   #define HOOK_RESP "hook-response/parada-guagua/2"

   //Con MY_DEVICES escucha eventos publicados por nuestros
dispositivos
   Particle.subscribe(HOOK_RESP, obtenerHorario, MY_DEVICES);
}

void loop() {

      Serial.println("Running loop");
      delay(30000); //espera 30 segundos antes de publicar
      Particle.publish("parada-guagua");

}
```

Figura 19. Suscripción a un evento concreto de la respuesta del *webhook*.

Tal como se observa en la Figura 20 , la función *obtenerHorario*() sólo se ejecutará una vez al realizarse una petición. Uno de los problemas iniciales encontrados a la hora de tratar la información obtenida se observa también en la imagen. Cuando una guagua está llegando o ha llegado a la parada, en la página web se muestra el mensaje “>>” donde se debería recibir el tiempo en minutos. Este mensaje corresponde a los caracteres *>>* que se observan en la imagen. Por tanto, será necesario gestionar este problema, indicando que la guagua está llegando cuando se obtienen dichos caracteres.

Figura 20. Problema encontrado con las guaguas que llegan a una parada.

Una vez realizada la suscripción a un evento concreto, se pasa a realizar la gestión de la respuesta recibida en el *firmware* del dispositivo *Photon*. En primer lugar, es necesario añadir una función para realizar la función *parse*, es decir, para ir dividiendo la respuesta por partes y así poder tratarla adecuadamente. Esta nueva función se muestra en la Figura 21.

```
1     String tryExtractString(String str, const char* start, const
char* end) {
2           if (str == NULL) {
3                 return NULL;
4           }
5
6           int idx = str.indexOf(start);
7           if (idx < 0) {
8                 return NULL;
9           }
10
11          int endIdx = str.indexOf(end);
12          if (endIdx < 0) {
13                return NULL;
14          }
15
16          return str.substring(idx + strlen(start), endIdx);
17    }
18
```

Figura 21. Función para realizar el parse.

Por otra parte, será necesario crear un *buffer*, mostrado en la Figura 22, en el que almacenar la respuesta que se obtiene, y que contendrá todos los caracteres HTML que se observan en la consola de *Particle*.

```
String str = String(datos);
char strBuffer[1000] = "";
str.toCharArray(strBuffer, 1000);
Serial.println(strBuffer);
```

Figura 22. Buffer para almacenar la respuesta del *webhook*.

Una vez que se obtiene la información mostrada en la Figura 23, se puede empezar a tratar la respuesta realizando la función *parser* correspondiente.

Figura 23. Datos obtenidos por el *webhook*.

Tras realizar una primera prueba con la función *tryExtractString*(), y creando el código de la función encargada de gestionar la respuesta recibida, tal como se ve en la Figura 24, se obtiene correctamente por el terminal el número de la primera línea que pasa por la parada.

```
//Parse para obtener la información de número de línea

String linea = tryExtractString(strBuffer, "linea-", "'>");
Serial.println("Número de línea:");
Serial.println(linea);
```

Figura 24. Primera prueba para manejar la respuesta de un *webhook*.

Además, con el código mostrado en la Figura 25 se consiguió obtener también el tiempo que quedaba para que pasara la primera línea.

```
    ////Se hace parse para que nos de solamente el tiempo////

    String tiempo = tryExtractString(strBuffer, "</td>\n<td>",
"min"); //Nos da más caracteres aparte de los números
    char bufferTiempo[100]=""; //Creamos el bufferTiempo para
almacenar el String tiempo
    tiempo.toCharArray(bufferTiempo,100); //Lo guardamos en el
buffer como una cadena de char

    char prueba [4]=""; //Creamos un buffer prueba para almacenar
los numeros
    int j =0;
    for(int i=0;i<bufferTiempo[100];i++){ //Recorremos el
bufferTiempo para encontrar los números
        if(bufferTiempo[i] >= '0' && bufferTiempo[i] <= '9') {
            prueba[j]=bufferTiempo[i]; //Se guarda en el buffer
prueba en caso de que sean números
            j++;
        }
    }

    Serial.println("Tiempo para la línea en minutos:");
    Serial.println(prueba); //Imprimimos el buffer que contiene
solamente números
```

Figura 25. Primera prueba para obtener el tiempo.

Sin embargo, para obtener correctamente los números de línea y tiempos de todas las líneas que pasan por la parada, fue necesario modificar la función *tryExtractString*() con el fin de añadirle un índice al principio y al final. Así, se le añadieron límites a la función *indexOf*(), de forma que una vez que se llame a la función *tryExtractString*(), la siguiente vez la búsqueda de los valores no empiece de nuevo en cero, sino a partir del último carácter que se extrajo del *buffer*. La función quedó tal como se puede observar en la Figura 26.

```
String tryExtractString(String str, const char* start, const char* end) {

      if (str == NULL) {
            return "";
      }

      idx = str.indexOf(start,endIdx);
      if (idx < 0) {
            return "";
      }

      endIdx = str.indexOf(end,idx);
      if (endIdx < 0) {
            return "";
      }

      return str.substring(idx + strlen(start), endIdx);
}
```

Figura 26. Función *tryExtractString* con índices.

Una vez modificada la función de *parse*, y debido a que no se obtenían correctamente los tiempos de las líneas, se dividió la función *obtenerHorario*() en dos funciones: una para obtener los números de línea, y otra para obtener los tiempos.

En la Figura 27 se muestra la función correspondiente a la obtención de los números de líneas, en la que no fue necesario realizar ningún tratamiento especial de la información, una vez realizada la función *parse*.

```
void obtenerHorario(const char *nombre, const char *datos) {

      Serial.println("Running obtenerHorario");

      String str = String(datos);
      str.toCharArray(strBuffer, 1500);

      for(int i=0;i<strBuffer[1500];i++){

            String linea = tryExtractString(strBuffer, "linea-", "'>");
            Serial.println("Número línea: " + linea);
            linea="";
      }
      idx=0;
      endIdx=0;
}

```

Figura 27. Función para obtener los números de las líneas.

Por otro lado, en la Figura 28 se muestra la función correspondiente a la obtención de los tiempos de las líneas. Esta vez sí fue necesario realizar un tratamiento especial, ya que una vez que se realizaba la función *parse*, además de los números se obtenía más información, por lo que fue necesario extraer solamente los números una vez realizada la función *parse*. Además, para evitar que se obtuvieran tiempos vacíos o errores por el terminal, se añadió una condición para que solamente fueran válidos los tiempos que tenían al menos una cifra, y nunca más de dos, ya que la página de Guaguas Municipales S.A. nunca muestra el tiempo de líneas más allá de 99 minutos.

```
1     void obtenerTiempo(const char *nombre, const char *datos) {
2
3           Serial.println("Running obtenerTiempo");
4
5           String str = String(datos);
6           str.toCharArray(strBuffer, 1500);
7
8           for(int i=0;i<strBuffer[1500];i++){
9
10                String tiempo = tryExtractString(strBuffer,
"</td>\n<td>", "min"); //Nos da más caracteres aparte de los números
11                char bufferTiempo[100]=""; //Creamos el bufferTiempo
para almacenar el String tiempo
12                tiempo.toCharArray(bufferTiempo,100); //Lo guardamos
en el buffer como una cadena de char
13                char prueba [4]=""; //Creamos un buffer prueba para
almacenar los numeros
14                int j =0;
15
16                for(int i=0;i<bufferTiempo[100];i++){ //Recorremos
el bufferTiempo para encontrar los números
17                      if(bufferTiempo[i] >= '0' && bufferTiempo[i] <=
'9') {
18                      prueba[j]=bufferTiempo[i]; //Se guarda en el
buffer prueba en caso de que sean números
19                      j++;
20                      }
21                }
22
23                if(prueba[0]!=NULL & prueba[2]==NULL){ //Condición
para que saque el tiempo sólo cuando es válido el valor del tiempo y
evitar errores
24                Serial.print("Tiempo para la línea en minutos: ");
25                Serial.println(prueba); //Imprimimos el buffer que
contiene solamente números
26                }
27          }
28          idx=0;
29          endIdx=0;
30    }
```

Figura 28. Función para obtener los tiempos de las líneas.

Una vez que el tratamiento de la respuesta del *webhook* se dividió en dos funciones separadas, se recibió correctamente toda la información de números de líneas y de tiempos, tal como se puede observar en el ejemplo mostrado en la Figura 29. De este modo, se dio por concluida la realización de pruebas con *webhooks*.

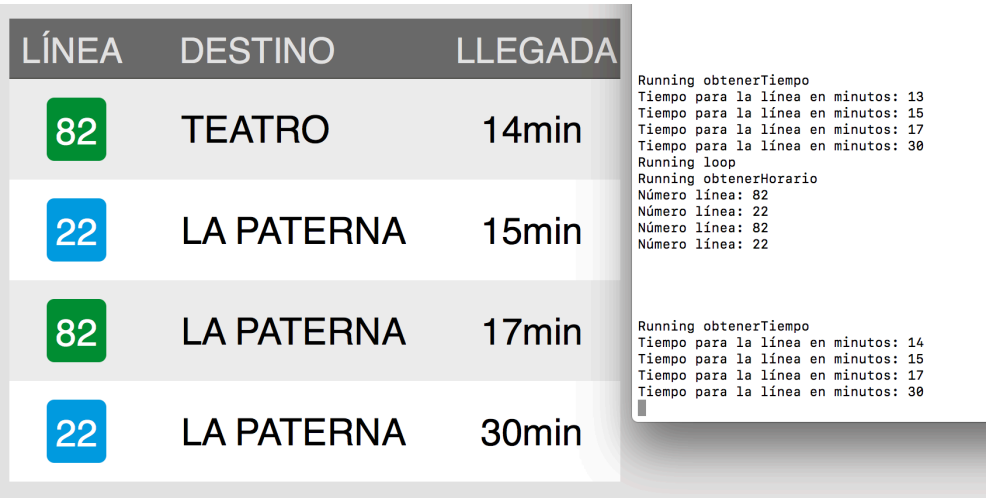


Figura 29. Finalización de gestión de la respuesta del *webhook*.

3.1.1.2. Llamadas http

Una vez finalizadas las pruebas con el uso de *webhooks* para acceder a la información de las paradas en la página web de la empresa Guaguas municipales S.A., se pasó a realizar las llamadas *http* directamente desde el dispositivo *Photon,* utilizando la librería *HTTPClient* [30], incluida en el entorno de desarrollo de *Particle*, para obtener y gestionar los datos recibidos desde la página web.

Dentro del fichero de cabecera de la librería se definen dos estructuras que corresponden a la petición al servidor (*request*) y a la respuesta obtenida del mismo (*response*), ambas mostradas en la Figura 30. En cuanto a la primera, indica los parámetros que se han de añadir para realizar una llamada *http*. La segunda cuenta con el *string* "*body*" que contendrá los datos obtenidos del servidor al que se realice la llamada *http*.

```
1   //HTTP Request struct.
2
3   typedef struct
4   {
5     String hostname;
6     IPAddress ip;
7     String path;
8     // TODO: Look at setting the port by default.
9     //int port = 80;
10    int port;
11    String body;
12  } http_request_t;
13
14  //HTTP Response struct.
15
16  typedef struct
17  {
18    int status;
19    String body;
20  } http_response_t;
```

Figura 30. Estructuras de *request* y *response*.

Una vez estudiadas las posibilidades que brindaba esta librería, el siguiente paso consistió en la realización de distintas pruebas utilizando los ejemplos incluidos en la misma, modificando el código necesario para adaptar su funcionamiento a las características de la comunicación entre el dispositivo *Photon* y la página web de la empresa Guaguas Municipales S.A.

Al realizar la primera prueba se encontró un problema con esta librería, y es que siempre se obtenía la misma información, perteneciente a la cabecera, en la llamada a la página de Guaguas Municipales S.A. Esto es debido a que en el fichero *.h* de la librería *HttpClient* se encuentra un *buffer* de tamaño 1024, en el que se almacena la información recibida desde la página web. Como era necesario obtener una gran cantidad de información de la página web de la empresa Guaguas Municipales S.A., era fundamental aumentar el tamaño de dicho *buffer*. Se aumentó su tamaño hasta 4096, tal como se muestra en la Figura 31, para evitar problemas en caso de que transitara un elevado número de guaguas por una determinada parada.

```
class HttpClient {
public:
    //Public references to variables.
    TCPClient client;
    char buffer[4096];
```

Figura 31. Aumento de tamaño del *buffer* de la librería *HTTPClient*.

Una vez realizado este cambio, se recibió correctamente toda la información de las paradas de guaguas en las pruebas iniciales realizadas. Además, se consiguió añadir la función *parse* para el número de línea y para el tiempo en una sola función, denominada *getLinea*(), al contrario que con los *webhooks*, donde había sido necesario dividirla en dos funciones.

A continuación, se suprimió de la función *setup*() el código relativo a la petición mediante *webhooks* y se incluyó en la función de tratamiento de los datos el código que se muestra en la Figura 32. A la función ya no era necesario proporcionarle ningún parámetro, por tanto, lo que ahora se almacenaba en el *buffer* denominado *strBuffer* correspondía a la respuesta (*response*) que se recibía desde la llamada *http*, con el *hostname*, puerto y ruta especificadas.

```
    request.hostname = "paradas.guaguas.com";
    request.port = 80;
    request.path = "/387";

    http.get(request, response);
    String str = String(response.body);
    str.toCharArray(strBuffer, 1500);
```

Figura 32. Llamada HTTP con la librería *HTTPClient*.

Con el método *get*() incluido en la librería *HTTPClient* se obtiene la respuesta del servidor (response) de la cual se obtendrá el *string* "*body*", mencionado anteriormente, que será el que se almacenará en el *buffer*. El resto de código se reutilizó de las pruebas realizadas con *webhooks* y, tal como se muestra en el resultado de la Figura 33, el código se ejecutaba correctamente.

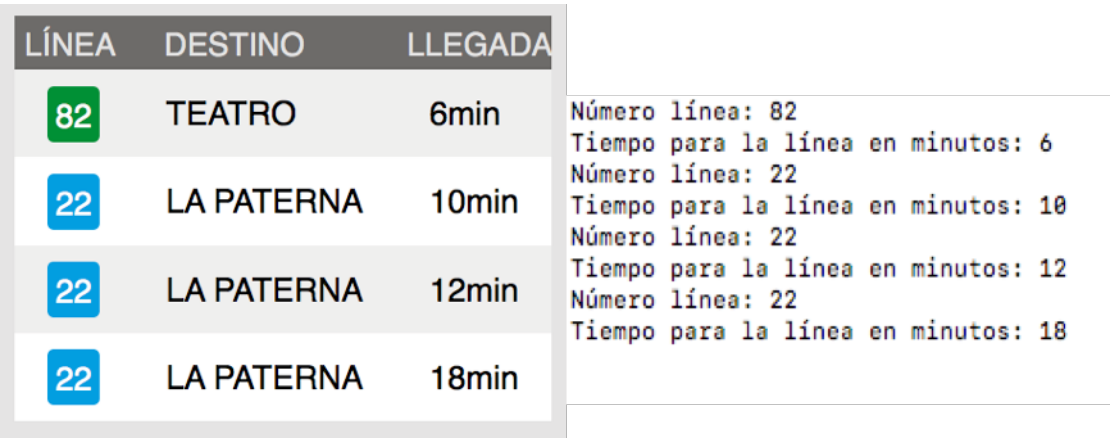


Figura 33. Información de las líneas con la librería *HTTPClient*.

Por otra parte, se consiguió solventar el problema que se produce cuando una guagua está llegando, o ya ha llegado a la parada, con el código de la Figura 34. De modo que, si no se encuentra el texto *"min"* en la información recibida con la función *parse*, y se encuentran los caracteres *"&gt"*, significa que la guagua está llegando a la parada y se muestra dicho mensaje por pantalla, y en caso de que se encuentre el texto *"min"*, se muestra por pantalla el número que va antes, correspondiente al tiempo que falta para que la guagua llegue.

```
1     String info = tryExtractString(strBuffer, "</td>\n<td>",
"</td>\n</tr>"); //Nos da más caracteres aparte de los números
2     int idx_td = info.indexOf("<td>");
3     int idx_min = info.indexOf("min");
4     int idx_tmp = info.indexOf("&gt");
5
6     if (idx_min == -1) { // no está el string "min"
7           if (idx_tmp != -1) { // si está el string "&gt"
8                 Serial.println("Está en la parada");
9           }
10    }
11    else {
12          String prueba = info.substring(idx_td + strlen("<td>"),
idx_min);
13          Serial.print("Tiempo para la línea en minutos: ");
14          Serial.println(prueba);
15    }
```

Figura 34. Solución para las guaguas que están llegando o han llegado a una parada.

3.1.2. Conclusiones

Una vez realizadas las pruebas con los *webhooks* y la librería *HTTPClient*, se decidió continuar trabajando en el desarrollo del dispositivo con las llamadas *http*, debido a la simplicidad de uso, la latencia adicional que representa el uso de los webhooks, y la dependencia de estos con *Particle Cloud*.

3.2. Conversión de la información recibida a audio

Debido a la finalidad a la que está destinada el presente TFG, se hace necesario incluir un dispositivo que realice la conversión a audio del texto obtenido después de realizar el procesamiento de los datos del servidor de Guaguas Municipales S.A. El dispositivo empleado para tal fin es el *Emic 2 Text-to-Speech*. A continuación, se explica la librería empleada para incluirlo en la plataforma y cómo se realizó su integración.

3.2.1. Librería para shield Text-to-Speech

Para integrar en la plataforma HW/SW el módulo *Emic 2*, se emplea la librería *Emic2TTS* [31]. Con esta librería se pueden configurar los ajustes del audio de salida, tal como se muestra en la Figura 35. Esta librería cuenta con funciones tales como *setVolume*() o *setVoice*(), entre otras, para establecer el valor deseado de los ajustes que permite configurar el fabricante del módulo. Para ello, basta con proporcionarle por parámetros el valor que se desea a dichas funciones. Así, en este caso se establece el volumen en 10, el rango de volumen está en decibelios, y va desde -48 a 18. Se selecciona la voz y el idioma deseados, en este caso *Perfect Paul* y español castellano, respectivamente, y, por último, las palabras por minuto, en este caso 220.

```
1     emic.setVolume(10);
2     emic.setVoice(PerfectPaul);
3     emic.setLanguage(CastilianSpanish);
4     emic.setWordsPerMinute(220);
```

Figura 35. Configuración de los ajustes del módulo *Emic 2*.

La manera en la que se indica al módulo *Emic 2* lo que tiene que reproducir, se basa en el uso de la función *say*() de la librería, tal como se muestra en la Figura 36.

```
1    String linea = tryExtractString(strBuffer, "linea-", "'>");
2    emic.say("La guagua");
3    emic.say(linea);
4    Serial.println("Número línea: " + linea);
5    linea="";
```

Figura 36. Función *say()* de la librería *Emic2TTS*.

3.2.2. Integración del *shield Text-to-Speech*

Una vez estudiadas las posibilidades de la librería *Emic2TTS*, se pasó a incluir el módulo *Emic 2* en el código *firmware* del dispositivo *Photon*. El conexionado entre el dispositivo *Photon* y el módulo *Emic 2* se realizó tal como se muestra en el esquema de la Figura 37.

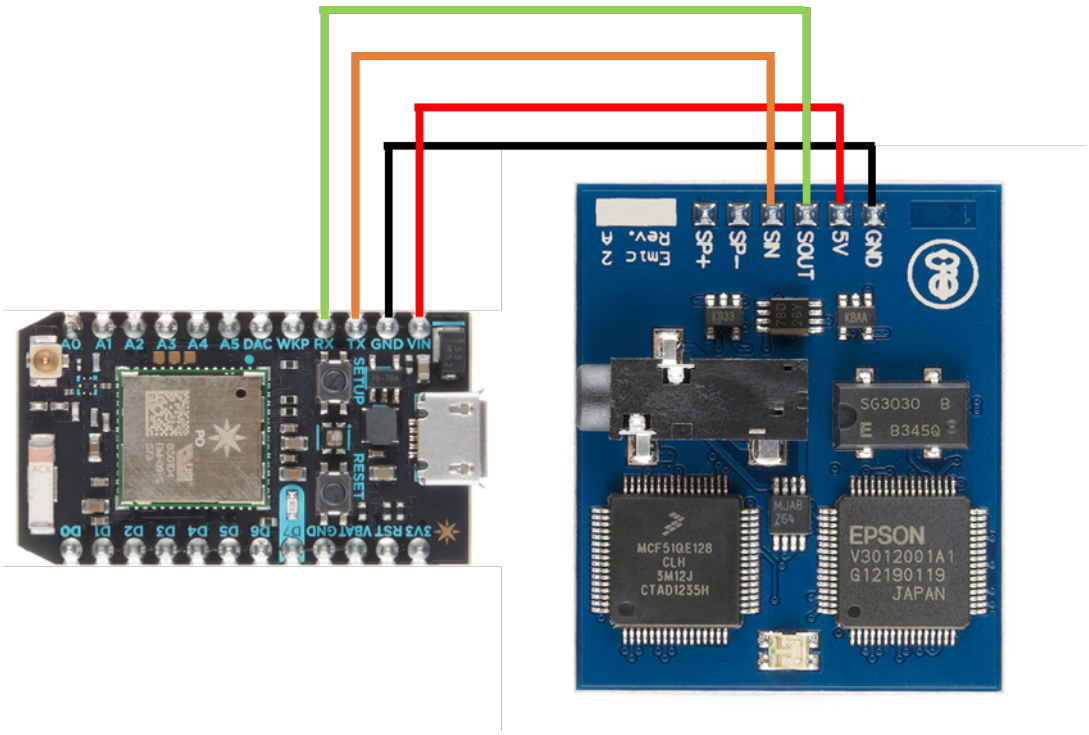


Figura 37. Conexionado *Photon-Emic*.

Uno de los problemas que surgieron a la hora de integrar el módulo *Emic 2* fue en el caso de las paradas con tildes en su nombre. El módulo *Emic 2* no es capaz de reproducir las letras mayúsculas que contienen tildes. Por tanto, si hay alguna parada que contiene una tilde en alguna de sus palabras, el módulo *Emic 2* no reproduce el nombre de dicha parada.

Debido a esta circunstancia, fue necesario tratar los nombres de las paradas, para lo cual se realizó la modificación de la Figura 38 en el código de la función *getLinea*().

```
1     for(int i=0;i<strBuffer[1500];i++){
2                   String nombreParada =
tryExtractString(strBuffer,"<h1>", "</h1>");
3
4           if(nombreParada!=NULL){
5
6                   String reemplazo = nombreParada;
7
8                   posicionTilde = reemplazo.indexOf('Á');
9                   reemplazo.replace('Á','A');
10                  reemplazo.setCharAt(posicionTilde-1, ' ');
11
12                  posicionTilde = reemplazo.indexOf('É');
13                  reemplazo.replace('É', 'E');
14                  reemplazo.setCharAt(posicionTilde-1, ' ');
15
16                  posicionTilde = reemplazo.indexOf('Í');
17                  reemplazo.replace('Í', 'I');
18                  reemplazo.setCharAt(posicionTilde-1, ' ');
19
20                  posicionTilde = reemplazo.indexOf('Ó');
21                  reemplazo.replace('Ó', 'O');
22                  reemplazo.setCharAt(posicionTilde-1, ' ');
23
24                  posicionTilde = reemplazo.indexOf('Ú');
25                  reemplazo.replace('Ú', 'U');
26                  reemplazo.setCharAt(posicionTilde-1, ' ');
27
28                  String str2 = String(reemplazo);
29                  str2.toCharArray(parada, 80);
30
31                  for(int j = posicionTilde;j<sizeof(parada)-1;j++){
32                        parada[j-1] = parada[j];
33                  }
```

Figura 38. Primera prueba de tratamiento de nombres de paradas con tildes.

En primer lugar, empleando la función *indexOf*(), se busca la posición en la que se encuentra una vocal con tilde. A continuación, mediante la función *replace*(), se le quita la tilde a la vocal. Por último, mediante la función *setCharAt*() se añade un espacio en la posición anterior a la vocal, ya que en el lugar de ese espacio se obtenía un símbolo que no se podía tratar.

Una vez hecho esto, se convierte el *string* "reemplazo" en un array de caracteres (*parada*) para poder reconstruir la palabra que contiene ahora un espacio. El resultado, para el caso de la vocal U, se muestra en la Figura 39.

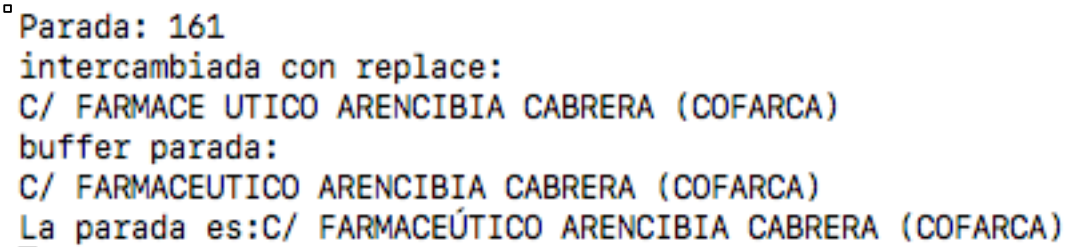


Figura 39. Resultado del tratamiento de tildes en el caso de la vocal U.

Sin embargo, esto solo funciona correctamente para el caso de esa vocal, ya que con las demás pruebas realizadas con paradas que contienen vocales con tilde distintas de U, se obtiene el valor -1 en "*posicionTilde*" y, por tanto, no se puede reconstruir la palabra al no reconocer la posición de la tilde. El nombre de la parada queda tal y como se muestra en la Figura 40.

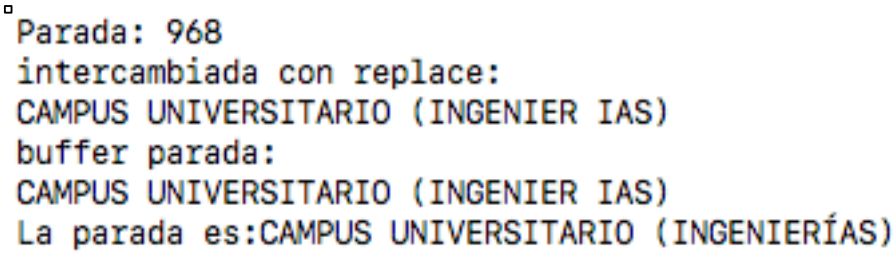


Figura 40. Resultado del tratamiento de tildes en el caso de vocales distintas de U.

En la Figura 41 se observa el índice en el que está la letra U (11) y que no se reconoce el índice de la vocal O de la segunda parada. Además, se observa el carácter que hay antes de la U en el caso concreto de la palabra FARMACEUTICO, razón por la que se añadió un espacio en blanco en medio de la palabra.

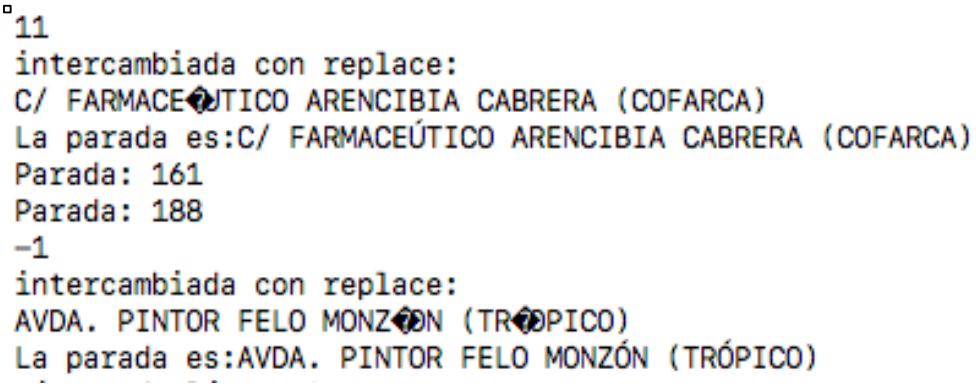


Figura 41. Error al reconocer el índice de vocales distintas de U y aparición del carácter no imprimible.

Por otra parte, en el caso de encontrarse más de una vocal con tilde en distintas palabras del nombre de la parada, solamente se realizaba la sustitución de manera correcta en la primera, tal como se observa en la Figura 42.

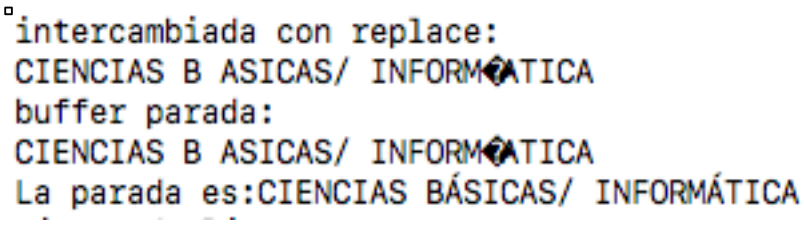


Figura 42. Error con nombres de paradas que contienen más de una vocal con tilde.

Por todo esto, fue necesario recurrir al código ASCII para solucionar los diferentes problemas encontrados. Se consiguió solventar el problema editando la función de la siguiente manera: primero, con la función *replace* de la clase *String* se buscan las vocales con tilde y se sustituyen por las vocales sin tilde, quedando un carácter desconocido delante de la vocal intercambiada. Para tratar ese carácter se convierte el *String* (*nombreParada*) en un *array* de caracteres (*parada*). Se recorre ese nuevo *array* buscando los caracteres que estén dentro del rango ASCII de 32 a 127 (caracteres imprimibles), exceptuando el carácter "/" que aparece en algunos nombres de paradas y hace que el dispositivo *Emic 2* no diga correctamente el nombre de la misma. En el caso de que los caracteres pertenezcan a ese rango, se van incluyendo en el array "*paradaLimpia*", que ya no contendrá ese carácter desconocido. Dicho *array* será el que proporcionará finalmente al módulo *Emic 2* para ser reproducido. El resultado de las modificaciones realizadas se muestra en la Figura 43.

```
    if(nombreParada!=NULL){

          nombreParada.replace('Á', 'A');
          nombreParada.replace('É', 'E');
          nombreParada.replace('Í', 'I');
          nombreParada.replace('Ó', 'O');
          nombreParada.replace('Ú', 'U');

          String str2 = String(nombreParada);
          str2.toCharArray(parada, 80);

          for (int k = 0; k < strlen(parada); k++){
               if (parada[k] >= 32 && parada[k] < 127 &&
parada[k]!=47){ //Comprobación con el código ASCII
                   paradaLimpia[m++] = parada[k];
               }
          }
          emic.say("La parada es");
          emic.say(paradaLimpia);

          Serial.println("La parada es: ");
          Serial.println(paradaLimpia);
          nombreParada="";
          m=0;
    }
```

Figura 43. Tratamiento final de los nombres de paradas con tildes.

Con estas modificaciones, el módulo *Emic 2* es capaz de reproducir correctamente el nombre de todas las paradas de guagua.

3.3. Localización por GPS

Debido a que uno de los modos de funcionamiento del dispositivo planteado inicialmente consiste en localizar la parada de guagua más cercana a la ubicación del usuario, se hace necesario incluir un módulo GPS. A continuación, se explica cómo se integró en la plataforma.

3.3.1. Librería para *shield* GPS

Para realizar la integración del módulo GPS, se barajaron tres librerías distintas, todas ellas incluidas en el entorno de desarrollo de *Particle*. Dichas librerías fueron *TinyGPS* [32], *TinyGPS++* [33] y *AssetTrackerRK* [34]. Aunque las dos primeras funcionaban correctamente a la hora de proporcionar la posición actual del módulo GPS, cuando se empleaba la función para calcular la distancia entre dos puntos, se producían errores de cálculo. Por ejemplo, en la Figura 44 se observa una distancia de 97.39 metros, medida con

Google Maps, entre el Edificio de Electrónica y Telecomunicación ubicadao en el Campus Universitario de Tafira, y la carretera. Al intentar calcular dicha distancia utilizando las librerías *TinyGPS*, *TinyGPS++* y *AssetTrackerRK, se obtuvieron unos valores de 13004598.00, 13004597.27 y 91.65 metros, respectivamente.* Por tanto, se realizó el proyecto con la librería *AssetTrackerRK*, que además está basada en la librería *TinyGPS++*.

Figura 44. Distancia entre el Edificio de Electrónica y Telecomunicación y la carretera en *Google Maps*.

Las funciones que finalmente se emplearon de dicha librería son las referentes a la obtención de la posición actual del módulo GPS y a la realización del cálculo de la distancia entre dos puntos. Para obtener la posición actual del módulo GPS se obtiene por un lado la latitud y por otro la longitud mediante las funciones *location.lat*() y *location.lon*(), respectivamente. En cuanto al cálculo de distancia entre dos puntos, se realiza con la función *distanceBetween*(), mostrada en la Figura 45, a la que basta con proporcionarle los parámetros correspondientes a las longitudes y latitudes de los dos puntos para que realice el cálculo.

```
1   double TinyGPSPlus::distanceBetween(double lat1, double long1,
double lat2, double long2)
2   {
3     double delta = radians(long1-long2);
4     double sdlong = sin(delta);
5     double cdlong = cos(delta);
6     lat1 = radians(lat1);
7     lat2 = radians(lat2);
8     double slat1 = sin(lat1);
9     double clat1 = cos(lat1);
10    double slat2 = sin(lat2);
11    double clat2 = cos(lat2);
12    delta = (clat1 * slat2) - (slat1 * clat2 * cdlong);
13    delta = sq(delta);
14    delta += sq(clat2 * sdlong);
15    delta = sqrt(delta);
16    double denom = (slat1 * slat2) + (clat1 * clat2 * cdlong);
17    delta = atan2(delta, denom);
18    return delta * 6372795;
19  }
```

Figura 45. Función *distanceBetween* de la librería *AssetTrackerRK*.

Por otra parte, a la hora de trabajar con el módulo GPS es necesario comprobar tres condiciones, que se muestran en la Figura 46. Dichas comprobaciones son: si el puerto serie al que se conecta está disponible, si al módulo GPS le llegan datos carácter a carácter por dicho puerto y, por último, si los datos de localización que llegan son válidos.

```
1     while (Serial1.available() > 0) {
2           if (gps.encode(Serial1.read())) {
3                 if (gps.location.isValid()) {
```

Figura 46. Comprobaciones necesarias para el uso correcto del *shield GPS*.

Una vez que se llega a la última comprobación se puede operar correctamente con los datos obtenidos.

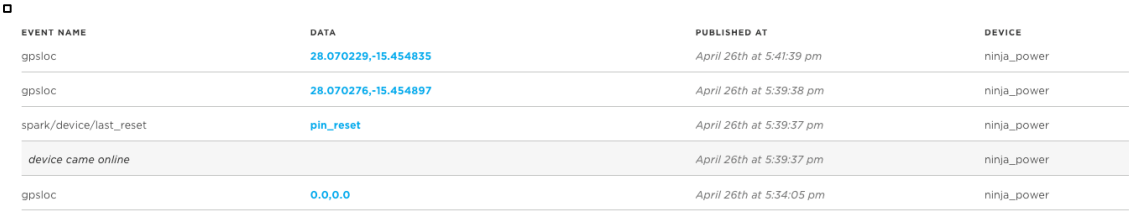
3.3.2. Integración del *shield* GPS

La primera prueba realizada fue la de obtener la localización actual del módulo GPS y así comprobar cuál es el formato con el que se reciben las coordenadas. Para ello se empleó el ejemplo disponible en la librería *TinyGPS,* mostrado en la Figura 47, que emplea la función *get_position*().

```
1   // This #include statement was automatically added by the Spark
IDE.
2   #include "TinyGPS.h"
3
4   TinyGPS gps;
5   char szInfo[64];
6   // Every 15 minutes
7   int sleep = 15 * 60 * 1000;
8
9   void setup(){
10      Serial1.begin(9600);
11  }
12
13  void loop(){
14      bool isValidGPS = false;
15
16      for (unsigned long start = millis(); millis() - start <
1000;){
17          // Check GPS data is available
18          while (Serial1.available()){
19              char c = Serial1.read();
20
21              // parse GPS data
22              if (gps.encode(c))
23                  isValidGPS = true;
24          }
25      }
26
27      // If we have a valid GPS location then publish it
28      if (isValidGPS){
29          float lat, lon;
30          unsigned long age;
31
32          gps.f_get_position(&lat, &lon, &age);
33
34          sprintf(szInfo, "%.6f,%.6f", (lat ==
TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : lat), (lon ==
TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : lon));
35      }
36      else{
37          // Not a valid GPS location, just pass 0.0,0.0
38          sprintf(szInfo, "0.0,0.0");
39      }
40
41      Spark.publish("gpsloc", szInfo);
42
43      // Sleep for some time
44      delay(sleep);
45  }
```

Figura 47. Ejemplo de la librería *TinyGPS* para obtener la localización actual del GPS.

Mediante este ejemplo, se obtiene la latitud y la longitud con 6 decimales, tal como se muestra en la Figura 48.



EVENT NAME	DATA	PUBLISHED AT	DEVICE
gpsloc	28.070229,-15.454835	April 26th at 5:41:39 pm	ninja_power
gpsloc	28.070276,-15.454897	April 26th at 5:39:38 pm	ninja_power
spark/device/last_reset	pin_reset	April 26th at 5:39:37 pm	ninja_power
device came online		April 26th at 5:39:37 pm	ninja_power
gpsloc	0.0,0.0	April 26th at 5:34:05 pm	ninja_power

Figura 48. Primera prueba de localización del GPS.

Una vez comprobado el formato de las coordenadas, se obtuvieron, gracias a *Google Maps* [35], las localizaciones GPS de las paradas de guaguas del Campus Universitario de Tafira de la ULPGC, consideradas como referencia, y mostradas en la Tabla 3, para añadirlas al código.

Tabla 3. Ubicación de las paradas de guaguas del Campus Universitario de Tafira.

Número	Nombre	Latitud	Longitud
968	CAMPUS UNIVERSITARIO (INGENIERÍAS)	28.070950	-15.455301
898	INGENIERÍAS	28.071466	-15.455325
731	INGENIERÍAS	28.070647	-15.455137
605	CIENCIAS BÁSICAS/ INFORMÁTICA	28.072820	-15.451986
612	CIENCIAS BÁSICAS/ TEOLOGÍA	28.072523	-15.451691

La ubicación de las paradas se incluyó en dos *arrays*, uno para las latitudes y otro para las longitudes, tal como se observa en la Figura 49. De esta manera, se establecieron como predefinidas estas cinco paradas en la plataforma HW/SW desarrollada en este TFG.

```
1    float latitudes[]=
{28.070950,28.071466,28.070647,28.072820,28.072523};
2    float longitudes[]={-15.455301,-15.455325,-15.455137,-15.451986,-
15.451691};
```

Figura 49. Latitudes y longitudes de las paradas de guaguas predefinidas.

3.3.3. Cálculo de distancia

Una vez predefinidas las ubicaciones de las paradas del Campus Universitario de Tafira, se desarrolló el código para calcular la menor de las distancias a las paradas predefinidas en función de la posición actual, mostrado en la Figura 50. Para ello, se recorren las latitudes y longitudes de una en una, se realiza el cálculo de la distancia mediante la función *distanceBetween*(), y se almacena cada una de las distancias en el *buffer bufferDistancias*. Una vez que se tienen todas las distancias, se recorre el *buffer* para encontrar la menor y su posición en el mismo. Por último, como las latitudes y longitudes se encuentran almacenadas en el orden en el que están puestos los números de las paradas predefinidas en *bufferParadas*, una vez que se encuentra la posición de la menor, se sabe que ésta corresponde a la parada que está en esa misma posición en *bufferParadas*. Se obtiene la parada que está situada en la posición de la menor, y se almacena en la variable *masCercana*, obteniéndose de ese modo el número de la parada más cercana a la posición actual del módulo GPS.

```
1    for(int i=0;i<sizeof(latitudes)-1;i++){
2          latParada=latitudes[i];
3          lonParada=longitudes[i];
4          bufferDistancias[k]=
gps.distanceBetween(latParada,lonParada,lat,lon);
5          k++;
6      }
7      distancia=bufferDistancias[0];
8      posicion=0;
9
10     for(int m=1;m<sizeof(bufferDistancias)-1;m++){
11         if(bufferDistancias[m]<distancia){
12             distancia=bufferDistancias[m];
13             posicion=m;
14         }
15     }
16     int masCercana= bufferParadas [posicion];
```

Figura 50. Código para obtener la parada de guagua más cercana.

Realizando pruebas del código en movimiento se obtuvieron satisfactoriamente las distancias a las paradas de guaguas y los números de las paradas más cercanas, tal como se muestra en el ejemplo de la Figura 51.

```
27.68
66.52
33.84
364.35
371.57
La posicion es:0
La distancia menor es: 27.68
La parada más cercana es:
968
59.08
94.71
44.11
350.18
353.95
La posicion es:2
La distancia menor es: 44.11
La parada más cercana es:
731
```

Figura 51. Obtención de la parada de guagua más cercana.

Por otra parte, fue necesario añadir una sentencia adicional al código global del dispositivo para que todo el programa se ejecutara correctamente, correspondiente a *System Thread*, mostrada en la última línea de código de la Figura 52.

```
1   //GPS
2   TinyGPSPlus gps;
3   float latitudes[]=
{28.070950,28.071466,28.070647,28.072820,28.072523};
4   float longitudes[]={-15.455301,-15.455325,-15.455137,-15.451986,-
15.451691};
5   int bufferDistancias[5]= {};
6   float latParada,lonParada,lat,lon,distancia;
7   int posicion,flag,encode,high,k;
8   SYSTEM_THREAD(ENABLED);
```

Figura 52. Habilitación de *System Thread*.

System thread es una configuración del sistema que ayuda a asegurar que el bucle de la aplicación no sea interrumpido por el procesamiento de *background* del sistema ni por la administración de red. Para ello, ejecuta el bucle de la aplicación y el bucle del sistema en

subprocesos separados, por lo que se ejecutan en paralelo en lugar de secuencialmente [36].

Por último, durante las pruebas realizadas se detectó que el dispositivo GPS no funcionaba correctamente en condiciones meteorológicas adversas, siendo imposible conectarse a los satélites durante varios días, por lo que fue necesario añadir una antena externa para que el GPS funcionara en cualquier situación.

Capítulo 4. Integración de la plataforma final

Una vez comprobado el funcionamiento individual de todos los componentes que conforman el dispositivo a desarrollar en este TFG, se pasó su integración con los elementos adicionales necesarios para su manejo.

4.1. Plataforma final con el dispositivo *Photon*

En la plataforma final desarrollada se han implementado dos modos de funcionamiento, denominados *modo normal* y *modo GPS*. En el *modo normal*, el usuario podrá seleccionar una parada de guagua específica de entre una serie de paradas predefinidas, y escuchar la información de líneas y horarios relativos a la parada seleccionada. En el *modo GPS*, el usuario recibirá la información de líneas y horarios relativos a la parada más cercana a su posición actual.

Para que estos modos de funcionamiento puedan ser gestionados adecuadamente y de manera sencilla por el usuario, se ha considerado necesario introducir botones en la plataforma. En concreto, los botones utilizados se encuentran integrados en el *shield Internet Button* de la empresa *Particle* [37].

El *shield Internet Button* es un accesorio del dispositivo *Photon* que permite al usuario interactuar de forma dinámica con aplicaciones basadas en IoT. Como se muestra en la Figura 53, este *shield* posee, en su parte superior, 11 LEDs RGB que se controlan de forma individual, y un acelerómetro de 3 ejes ADXL362.

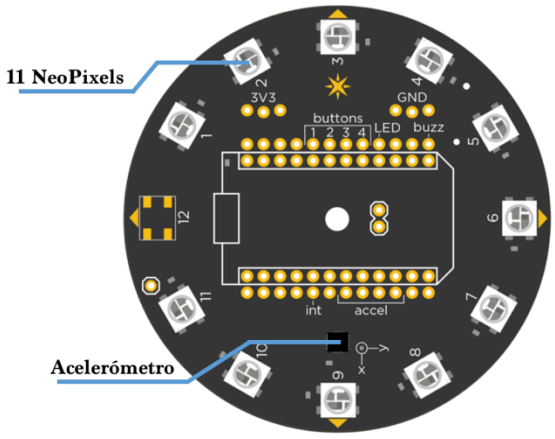


Figura 53. Vista superior del *shield Internet Button*.

En cuanto a los LED, éstos son *NeoPixels* [38]. Se trata de diodos LED con un encapsulado 5050 que integran un controlador WS2812 [39]. En el *shield Internet Button*, estos LEDs se encuentran conectados en serie y son controlados mediante una señal de pulsos.

Siguiendo con la parte inferior del *shield Internet Button*, este accesorio cuenta, tal y como se puede observar en la Figura 54, con 4 botones, un *buzzer* o zumbador y un espacio diseñado para utilizar un conector JST (*Japan Solderless Terminal*) en caso de que se prefiera alimentar con batería en lugar de usar la alimentación vía USB.

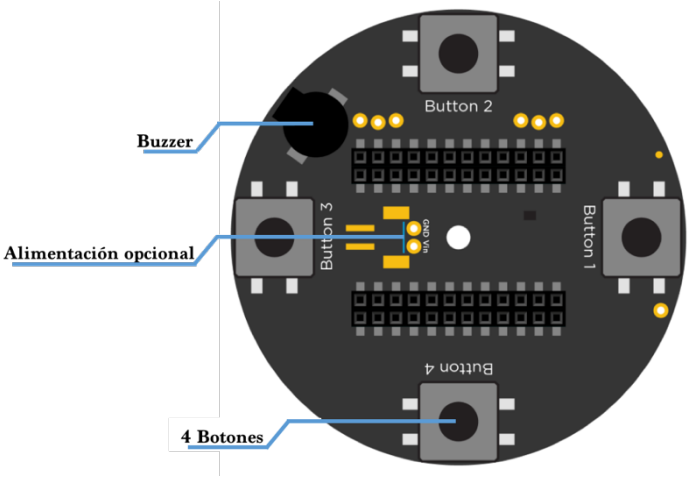


Figura 54. Vista trasera del *shield Internet Button*.

Por otra parte, este *shield* dispone también de la posibilidad de realizar conexiones adicionales. Esto es posible gracias a que posee, tanto una fila de pines para realizar la conexión con el dispositivo *Photon*, como una segunda fila de pines que permite el uso de

sensores o actuadores adicionales. Además, el dispositivo *Internet Button* está provisto de una carcasa de protección que atenúa el brillo de los LED.

La librería empleada para utilizar el *shield Internet Button* es la que lleva su mismo nombre, disponible en la web IDE de *Particle*. Las funciones a emplear de esta librería son las referentes a la inicialización del *shield*, la gestión de los botones y de los LED.

Para incluir esta librería en el código de usuario, es necesario incluir la sentencia *begin*() dentro de la función *setup()*. Para gestionar los botones se emplea la función *buttonOn*(). Por último, para gestionar los LED, se emplean las funciones *ledOn*() (para encender un led) y *ledOff*() (para apagarlo).

La librería que proporciona *Particle* para el uso del *shield Internet Button* establece las correspondencias con los pines del dispositivo *Photon* que se muestran en la Tabla 4.

Tabla 4. Correspondencia entre los pines del dispositivo *Photon* y los botones del *shield Internet Button*.

Pin *Photon*	**Internet *Button***
D4	Botón 1
D5	Botón 2
D6	Botón 3
D7	Botón 4

En este sentido, se ha establecido que el pin *D4* sea utilizado como interruptor para realizar el cambio de modo de funcionamiento del dispositivo, empleándose los botones 2 y 3 del *shield Internet Button* para integrarlo en la plataforma final. Este interruptor se implementó inicialmente mediante un cable conectado al pin *D4* y, según si está conectado a las tensiones 0V o 3,3V, correspondientes a los niveles lógicos “0” o “1”, respectivamente, se ejecutará un modo de funcionamiento u otro.

El uso del Botón 3 tendrá el mismo propósito, tanto para el *modo normal* como para el *modo GPS*. Su función será dar la orden al dispositivo *Photon/Electron* para que realice la

llamada *http* a la página de la empresa Guaguas Municipales S.A. En cuanto al Botón 2, en el *modo normal* será el encargado de recorrer el *buffer* de las paradas de guaguas predefinidas, de manera que cada vez que se pulse, recorra una posición del buffer, mientras que en el *modo GPS* será el encargado de activar el módulo GPS para que encuentre la parada más cercana a su posición actual.

Así, una vez integrado el *shield Internet Button*, la plataforma final está formada por los componentes mostrados en la Figura 55.

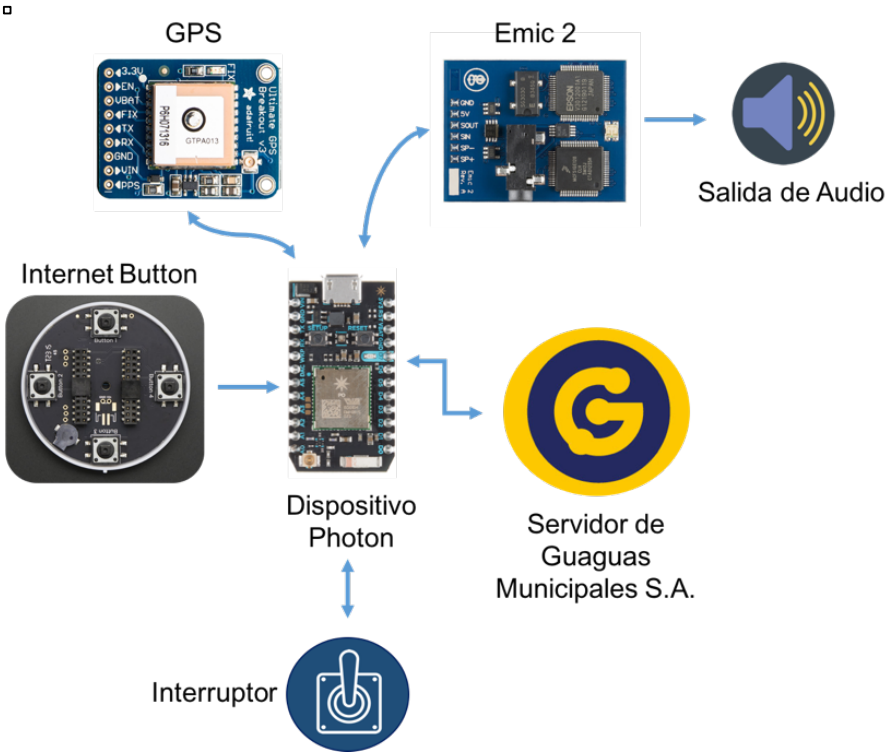


Figura 55. Esquema de la plataforma final con el dispositivo *Photon*.

A continuación se detalla el funcionamiento de cada uno de los modos definidos. En el caso de que se haya seleccionado el *modo normal*, disponiendo el interruptor en la posición correspondiente al nivel lógico "1", el funcionamiento consiste en los siguientes pasos:

1. Mediante el Botón 2 del *shield Internet Button* se navega por las distintas paradas predefinidas, gracias a la salida de audio proporcionada por el módulo *Emic 2 Text-to-Speech*.

2. Una vez que se escucha la parada deseada, se da la orden, mediante el Botón 3 del *shield Internet Button*, para que el dispositivo *Photon* realice la llamada *http* al servidor de la empresa Guaguas Municipales S.A.
3. El dispositivo *Photon* realiza el procesamiento de los datos adquiridos desde el servidor de la empresa Guaguas Municipales S.A.
4. El usuario obtiene, mediante audio, la información de la parada que ha seleccionado gracias a la conversión realizada por el módulo *Emic 2*.

Si por el contrario se ha seleccionado el *modo GPS*, disponiendo el interruptor en la posición correspondiente al nivel lógico "0", el funcionamiento consiste en los siguientes pasos:

1. Mediante el Botón 2 del *shield Internet Button* se activa el módulo GPS para que realice el cálculo de la distancia entre la posición actual del usuario, y las posiciones de las paradas predefinidas, encontrando la parada más cercana.
2. Mediante el Botón 3 del *shield Internet Button* se da la orden al dispositivo *Photon* para que realice la llamada *http* al servidor de la empresa Guaguas Municipales S.A.
3. El dispositivo *Photon* realiza el procesamiento de los datos adquiridos del servidor de la empresa Guaguas Municipales S.A.
4. El usuario obtiene, mediante audio, la información de la parada más cercana a su posición gracias a la conversión de texto a audio realizada por el módulo *Emic 2*.

Antes de añadir finalmente el dispositivo *Electron* a la plataforma final desarrollada, se realizaron una serie de pruebas con el dispositivo *Photon*, encontrándose un problema relacionado con las interfaces que éste presenta. Así, debido a que el dispositivo *Photon* solamente cuenta con una UART, y tanto el *shield* GPS como el módulo *Emic 2* se conectan mediante UART, fue necesario realizar una modificación en el dispositivo para añadirle una interfaz serie adicional. Con esta finalidad, se siguió la guía de la Comunidad de *Particle* disponible en [40].

La manera de añadir la UART adicional pasa por sacrificar el LED de estado del dispositivo *Photon*, por lo que se hace necesario añadir un LED externo conectado a los pines *D0*, *D1* y *D2*, quedando la modificación tal como se muestra en la Figura 56.

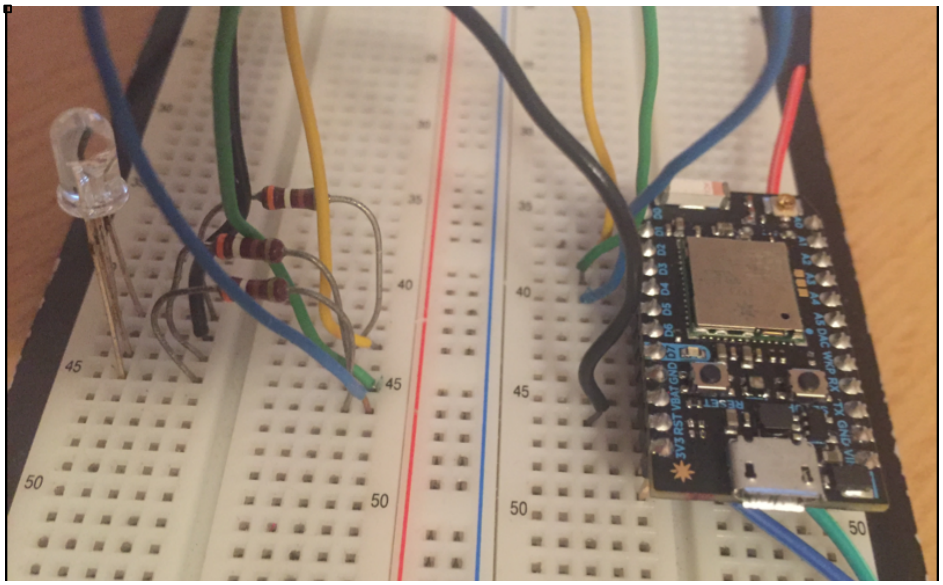


Figura 56. Modificación del dispositivo *Photon* para añadir una UART adicional.

Una vez añadido el nuevo LED, la plataforma final con el dispositivo *Photon* modificado quedó tal como se muestra en la Figura 57.

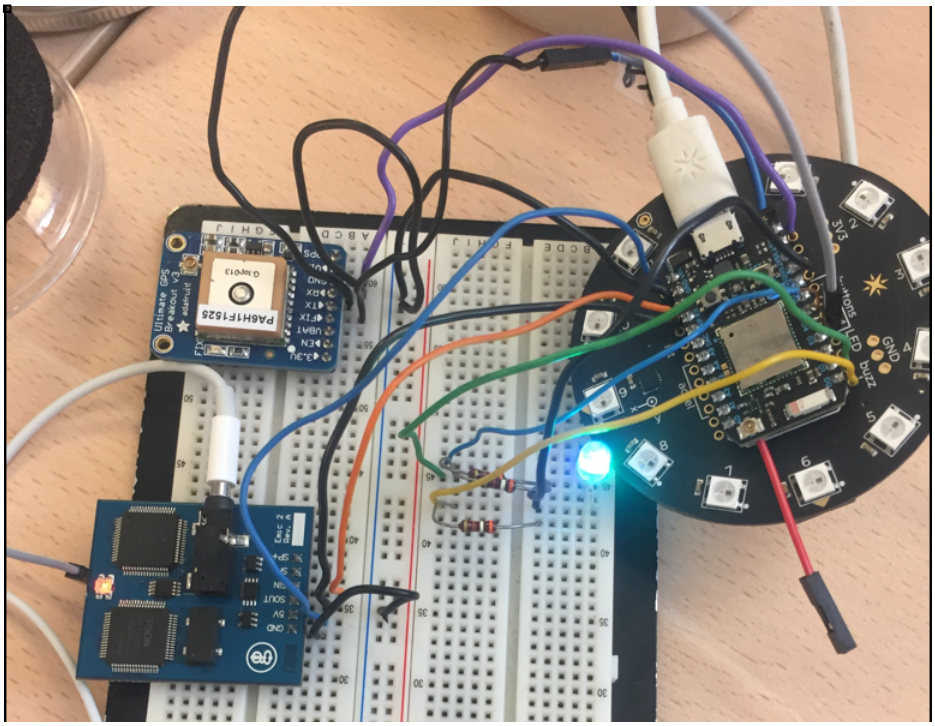


Figura 57. Plataforma final con el dispositivo *Photon*.

4.2. Verificación y validación de la plataforma final

El bucle principal del programa desarrollado para integrar los dos modos de funcionamiento anteriormente descritos se observa en la Figura 58. La operación del dispositivo comienza con la selección del modo de funcionamiento mediante el interruptor, aunque este interruptor también funciona mientras se está ejecutando el programa. Se diferencian dos modos de funcionamiento según la posición del interruptor: a nivel alto se encuentra en *modo normal*, y a nivel bajo se encuentra en *modo GPS*.

```
1   void loop() {
2
3       valor=digitalRead(interruptor);
4
5       if(valor==LOW){
6           if(b.buttonOn(2)){
7               flag=1;
8               modoGPS();
9           }
10      }
11      if(valor==HIGH){
12          modoNormal();
13      }
14      if(b.buttonOn(3)){
15          peticion();
16          getLinea();
17      }
18  }
```

Figura 58. Función *loop()* de la plataforma final.

Dentro del *modo normal*, el usuario debe presionar el Botón 2 del *shield Internet Button* para escuchar las paradas que están predefinidas en el dispositivo, una pulsación equivale a una parada. Una vez que se escuche la parada deseada, se selecciona la misma mediante el Botón 3 del *shield Internet Button*, que al pulsarse realiza la llamada *http* a la página de la empresa Guaguas Municipales S.A, al igual que en el *modo GPS*.

En el *modo GPS*, el usuario debe presionar el Botón 2 del *shield Internet Button* para que el módulo GPS localice la parada más cercana a su ubicación actual. Una vez localizada, se realiza la llamada *http* al servidor de la empresa Guaguas Municipales S.A. mediante el Botón 3.

Una vez que se realiza la llamada *http*, se procesa la información recibida y se obtiene la información convertida a audio mediante el dispositivo *Emic 2*, integrado en la plataforma final.

En la Figura 59 se observa el funcionamiento del dispositivo por el terminal del ordenador conectado al dispositivo *Photon* a través de un puerto mini-USB. Así, se observa que la plataforma HW/SW final desarrollada empieza funcionando en *modo GPS*, ya que se obtiene la distancia desde la posición del usuario hasta la parada más cercana, en este caso la parada número 605. Una vez que se obtiene el número de la parada, al pulsar el botón de llamada se obtiene la información de las líneas asociadas. A continuación, se cambia el modo de funcionamiento al *modo normal* y a medida que se pulsa el Botón 2 del *shield Internet Button*, van mostrándose los números de las paradas predefinidas. Se selecciona la parada número 731 con el botón de llamada, y se obtiene la información de las líneas.

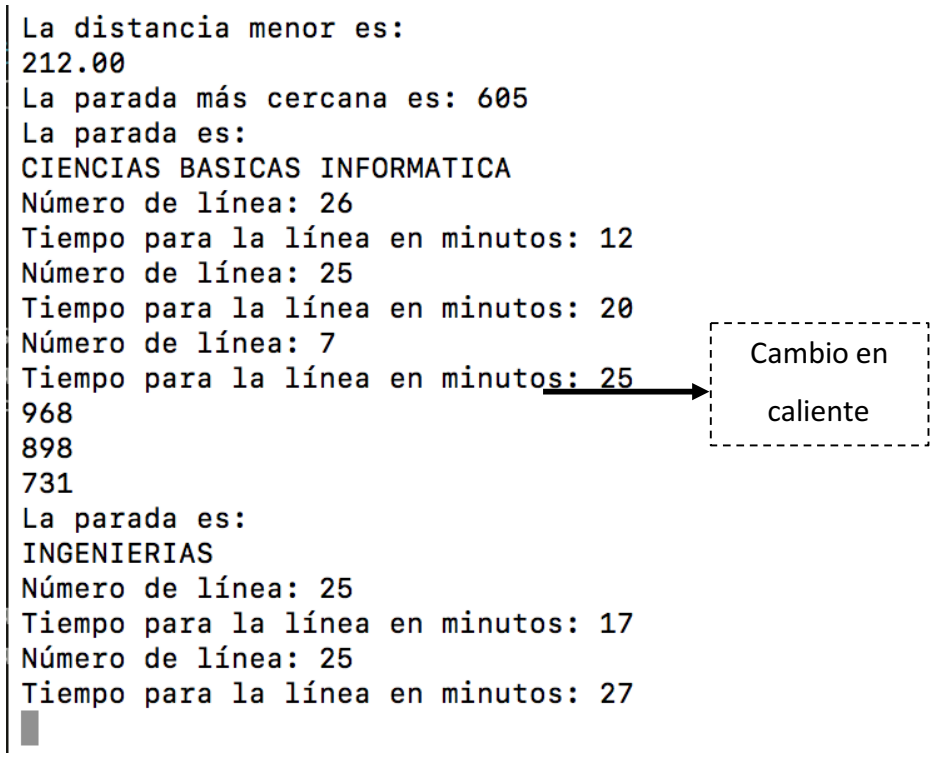


Figura 59. Validación de la plataforma final con cambio en caliente.

A partir de las diferentes pruebas realizadas se pudo dar por concluida la validación de la plataforma final con el dispositivo *Photon*, y se pasa a sustituirlo por el dispositivo *Electron*.

4.2.1. Integración del dispositivo *Electron* en la plataforma final

Al contrario que el dispositivo *Photon*, el dispositivo *Electron* presenta más de una UART, por lo que en este caso no hay que realizar ninguna modificación sobre el mismo para poder integrarlo en la plataforma final. Sin embargo, el *shield Internet Button* no es compatible con el dispositivo *Electron*, por lo que se hace necesario sustituirlo por dos pulsadores que realicen las funciones de los botones 2 y 3 del *shield Internet Button*. El esquema de la plataforma final con el dispositivo *Electron* se muestra en la Figura 60.

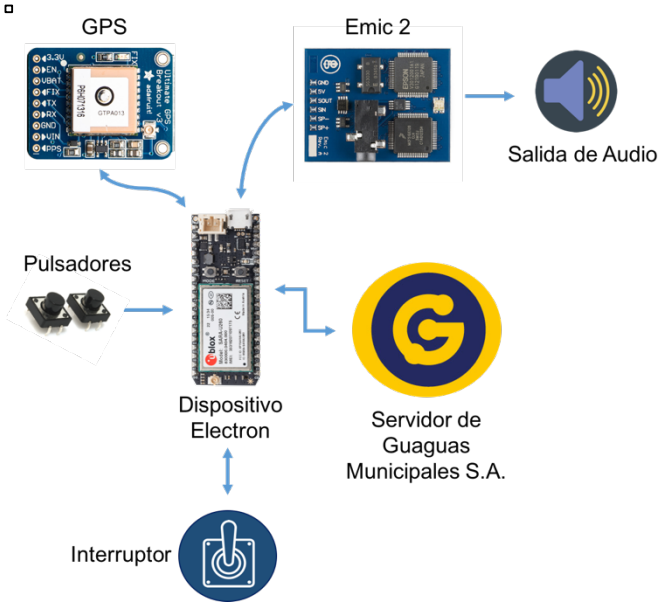


Figura 60. Esquema de la plataforma final con el dispositivo *Electron*.

Dado que el uso del dispositivo *Electron* implica el uso de datos en redes móviles 3G, una de las medidas que se toman para disminuir el coste consiste en cargar el código en el dispositivo *Electron* mediante la herramienta CLI de la empresa *Particle* [41] en vez de hacerlo directamente desde *Particle Build*. Esta forma de cargar el código se muestra en la Figura 61.

```
MacBook-Pro-de-Sara:~ saraleon$ cd ~/Desktop
MacBook-Pro-de-Sara:Desktop saraleon$ particle flash --serial firmware-2.bin
! PROTIP: Hold the SETUP button on your device until it blinks blue!
? Press ENTER when your device is blinking BLUE
sending file: firmware-2.bin

Flash success!
```

Figura 61. Carga del código en el dispositivo *Electron* mediante la herramienta CLI de *Particle*.

Por otro lado, como se mencionó anteriormente, el dispositivo *Electron* cuenta con más de una UART, con lo cual se pueden conectar tanto al módulo *Emic 2 Text-to-Speech* como al módulo GPS sin tener que realizar modificaciones en el HW. En este caso, se emplea la UART denominada *Serial5*, que se encuentra ubicada en los pines *C1(TX)* y *C0(RX)* del dispositivo *Electron* [42], para conectar el módulo GPS, y la interfaz *Serial1*, asociada a los pines *TX* y *RX*, para conectar el módulo *Emic 2 Text-to-Speech*.

Por otra parte, para realizar la conexión de los dos nuevos pulsadores, se añaden dos resistencias de 10 kΩ en configuración *pull-down* de manera que se tiene un valor *HIGH* al presionar los pulsadores, y un valor *LOW* al soltarlos. Los pulsadores integrados en la plataforma HW/SW se muestran en la Figura 62.

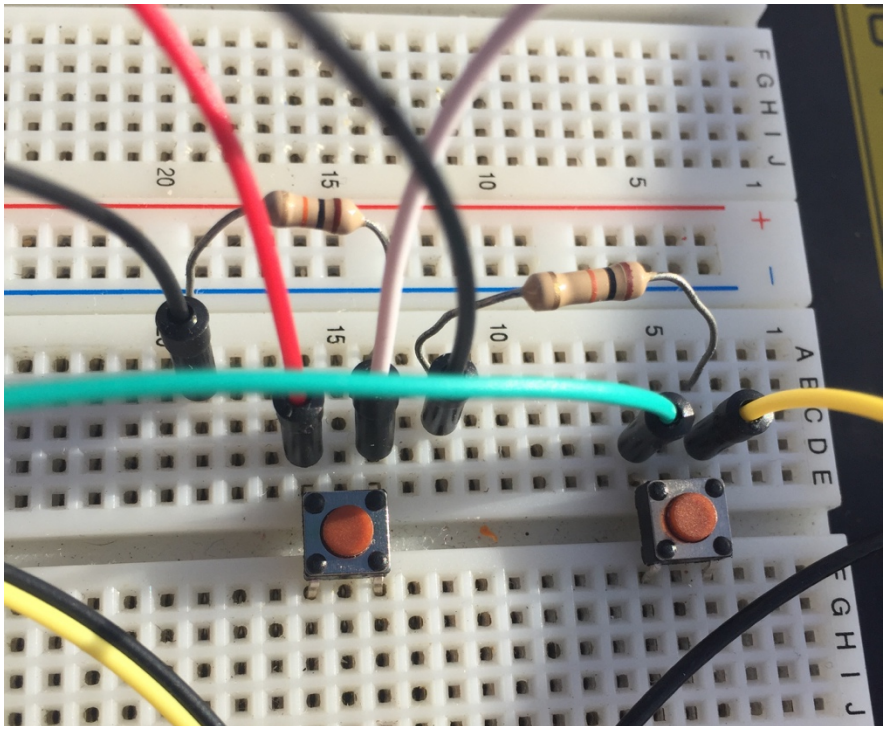


Figura 62. Pulsadores para sustituir el *shield Internet Button*.

Una vez añadidos los dos nuevos pulsadores, la plataforma HW/SW final integrando el dispositivo *Electron* quedó tal como se muestra en la Figura 63.

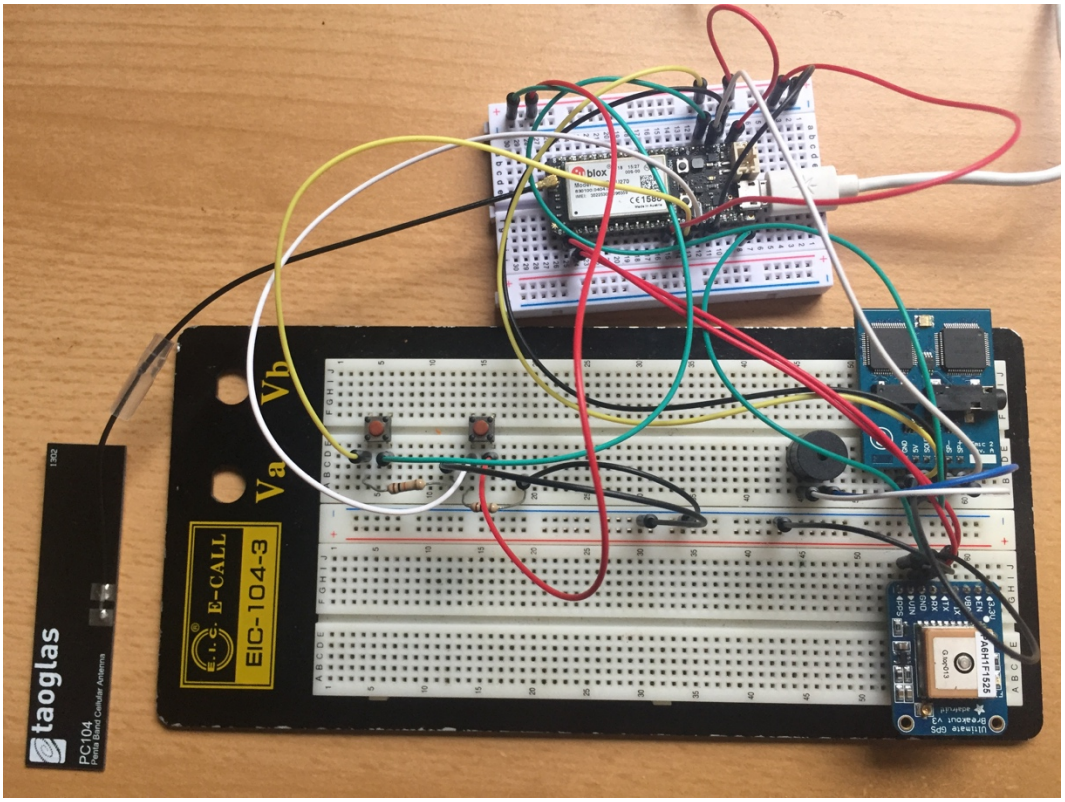


Figura 63. Plataforma final con el dispositivo *Electron*.

El código desarrollado es compatible entre el dispositivo *Photon* y el dispositivo *Electron*, por lo que se reutiliza completamente, exceptuando la sección correspondiente a la interacción con el *shield Internet Button*, que es sustituido por el código necesario para realizar el manejo de los dos nuevos pulsadores.

Una vez cargado el código en la plataforma HW/SW final, se realizaron algunas pruebas para verificar su correcto funcionamiento. La primera prueba consistió en utilizar el *modo normal* de la plataforma. En la Figura 64 se muestra, en primer lugar, el uso del Pulsador 1 para cambiar de parada, recorriendo el *buffer* de las paradas predefinidas hasta que se selecciona la parada 968 mediante el Pulsador 2. Una vez seleccionada la parada, el dispositivo *Electron* realizó la llamada *http* al servidor de la empresa Guaguas Municipales S.A. y se obtuvo la información correspondiente a la parada 968. Después de obtener la información de las dos líneas que transitaban por la parada, se esperó un minuto y se volvió a pulsar el Pulsador 2, obteniéndose la información actualizada. Una vez obtenida esta información se hizo uso del Pulsador 1 nuevamente para volver a seleccionar otra parada, seleccionándose en este caso la parada 605 con el Pulsador 2 y obteniendo la información relativa a las 5 líneas que iban a transitar por esa parada.

```
968
898
731
605
612
968
898
La parada es:
INGENIERIAS
Número de línea: 25
Tiempo para la línea en minutos: 2
Número de línea: 25
Tiempo para la línea en minutos: 18
La parada es:
INGENIERIAS
Número de línea: 25
Tiempo para la línea en minutos: 1
Número de línea: 25
Tiempo para la línea en minutos: 17
731
605
La parada es:
CIENCIAS BASICAS INFORMATICA
Número de línea: 26
Tiempo para la línea en minutos: 2
Número de línea: 25
Tiempo para la línea en minutos: 13
Número de línea: 26
Tiempo para la línea en minutos: 19
Número de línea: 48
Tiempo para la línea en minutos: 24
Número de línea: 25
Tiempo para la línea en minutos: 27
```

Figura 64. Verificación del modo normal de la plataforma final.

En la Figura 65 se muestra que la información recibida en la plataforma final coincide con la información proporcionada por el servidor de la empresa Guaguas Municipales S.A. en el caso particular de la parada número 731.

Figura 65. Comprobación de que la información recibida en la plataforma final es correcta.

La siguiente prueba realizada consistió en hacer uso del *modo GPS* de la plataforma HW/SW final integrando el dispositivo *Electron*. En la Figura 66 se muestra la comprobación de la

distancia más cercana entre el usuario y las paradas de guaguas predefinidas, mediante el uso del Pulsador 1. Una vez obtenida la distancia más cercana, se obtuvo la parada de guagua correspondiente, mediante el uso del Pulsador 2, recibiéndose correctamente la información relativa a esta parada, obteniéndose así la información de las líneas que transitaban por la parada más cercana a la posición actual del usuario.

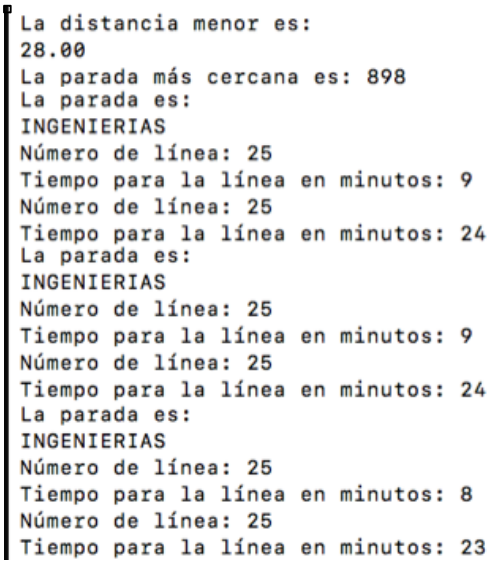


Figura 66. Verificación del modo GPS de la plataforma final.

La última prueba realizada consistió en realizar el cambio de modo de funcionamiento en caliente. En la Figura 67 se muestra la realización satisfactoria del cambio de *modo normal* a *modo GPS*.

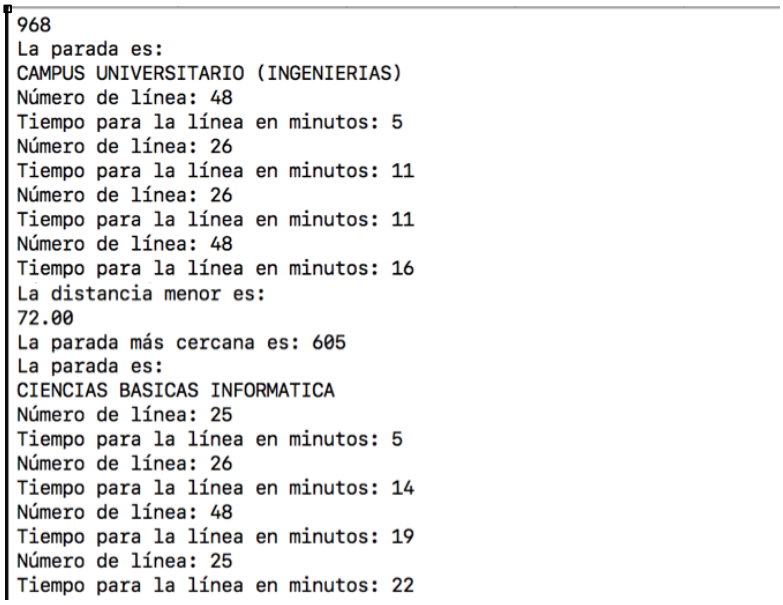


Figura 67. Verificación del cambio de modo de funcionamiento en caliente de la plataforma final.

Por último, a la hora de realizar las pruebas se observa que, en la realización de la llamada *http* al servidor de la empresa Guaguas Municipales S.A, la latencia asociada al dispositivo *Electron* es mayor que la observada con el uso del dispositivo *Photon*.

Capítulo 5. Conclusiones

5.1. Conclusiones

A partir de los resultados favorables obtenidos del proceso de validación, se puede confirmar que con el presente Trabajo Fin de Grado se ha logrado el objetivo de desarrollar un prototipo de plataforma HW/SW económica adaptada a personas con discapacidad visual, la cual es capaz de informar al usuario sobre la información de las líneas que transitan por las paradas de guaguas cercanas, así como por aquellas predefinidas por el usuario.

En primer lugar, se realizaron diferentes pruebas para acceder a los datos del servidor de la empresa Guaguas Municipales S.A., considerando como opciones para ello las llamadas *http* y el uso de *webhooks*, descartando estos últimos debido a la latencia adicional que suponía su uso, y su dependencia con *Particle Cloud*. Al terminar estas pruebas, se concluyeron las funciones asociadas al que sería el primero de los dos modos de funcionamiento definidos en la plataforma final: el *modo normal*, mediante el cual el usuario puede seleccionar una de las paradas de guaguas predefinidas para obtener, mediante audio, la información relativa a los horarios de las líneas que transitan por la parada seleccionada.

A continuación, se realizaron las primeras pruebas con el *shield Emic 2 Text-to-Speech* de la empresa *Parallax* y la librería incluida en el IDE de *Particle*. Una vez comprendido el funcionamiento del módulo *Emic 2 Text-to-Speech*, se pasó a desarrollar el *modo GPS*, definido en la plataforma HW/SW final, comenzando por las pruebas de las librerías *TinyGPS*, *TinyGPS++* y *AssetTrackerRK*, de las que se descartaron las dos primeras. Una vez seleccionada la librería *AssetTrackerRK* se desarrolló el código necesario para realizar el cálculo de distancia y obtener la parada de guagua más cercana a la posición actual del usuario.

Una vez completada la implementación de los dos modos de funcionamiento definidos en el dispositivo por separado, se pasó a integrarlo todo en la plataforma HW/SW final junto

con el *shield Internet Button,* empleado para gestionar los dos modos de funcionamiento, y se dieron por concluidas las pruebas iniciales con el dispositivo *Photon*, pasando a sustituirlo por el dispositivo *Electron*.

Una vez integrado el dispositivo *Electron* en la plataforma HW/SW final, se encontró el problema de que el *shield Internet Button* no era compatible con este dispositivo, siendo necesario sustituirlo por pulsadores. En cuanto al código empleado, no fue necesario realizar cambios, excepto por la sustitución de la parte del código asociada a la integración del *shield Internet Button* por la de la gestión de los dos nuevos pulsadores.

Finalizado el proceso de implementación y validación de la plataforma HW/SW, y tras la experiencia adquirida durante la realización del mismo, se pueden establecer las siguientes conclusiones:

- El entorno de desarrollo IDE de *Particle* es muy sencillo e intuitivo, lo que facilita en gran medida las tareas de programación.

- Existe una gran cantidad de librerías desarrolladas, tanto por la empresa *Particle* como por terceros, que facilita las tareas de programación y resulta un buen punto de partida para generar, a partir de ellas, aplicaciones o ideas más elaboradas.

- Debido a que el dispositivo *Photon* trabaja con *firmware open source*, se ha podido desarrollar de una manera más rápida y sencilla la programación de la aplicación propuesta, debido a la extensa cantidad de usuarios que han trabajado con el *firmware*, facilitando información sobre el mismo en Internet.

- Los foros de *Particle* constituyen una gran ayuda. Aquellas dudas específicas acerca de errores en el desarrollo de la presente aplicación se publicaron en los foros, obteniendo respuestas muy rápidas y claras por parte del equipo de *Particle*.

- El hecho de que el dispositivo *Photon* solo cuente con una UART ha ocasionado problemas a la hora de realizar las pruebas de la plataforma final, aunque gracias a

la posibilidad de modificar el dispositivo se han podido solventar. Sin embargo, esto limita la capacidad de este tipo de proyectos en cuanto a tener varios módulos, que requieran conexión mediante UART, conectados al dispositivo *Photon*.

- Debido a que el *shield Internet Button* no es compatible con el dispositivo *Electron* ha sido necesario realizar cambios en el código para añadir los nuevos pulsadores, por tanto se considera de gran utilidad que la empresa *Particle* desarrolle un *shield Internet Button* específico para el dispositivo *Electron*, o bien uno compatible con los dispositivos *Photon* y *Electron*.

- Se observan diferencias de latencia entre los dispositivos *Photon* y *Electron* en la realización de la llamada *http* al servidor de la empresa Guaguas Municipales S.A, siendo la latencia asociada al dispositivo *Electron* mayor que la observada con el uso del dispositivo *Photon*.

5.2. Líneas futuras de trabajo

A continuación se destacan las líneas futuras de trabajo más importantes que se proponen tras la realización de este TFG:

- Integración de comandos de voz que sustituyan a los pulsadores integrados en la plataforma HW/SW final.

- Creación de una base de datos con la localización de toda la red de líneas de guaguas de la empresa Guaguas Municipales S.A.

- Añadir una opción de rutas, de manera que el usuario pueda ser guiado hasta la parada seleccionada, o la parada más cercana a su ubicación actual.

- Diseño y fabricación de una placa que integre todos los componentes de la plataforma HW/SW final desarrollada.

Referencias

[1] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, y M. Ayyash, «Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications», *IEEE Commun. Surv. Tutor.*, vol. 17, n.º 4, pp. 2347-2376, Fourthquarter 2015.

[2] S. Feng, P. Setoodeh, y S. Haykin, «Smart Home: Cognitive Interactive People-Centric Internet of Things», *IEEE Commun. Mag.*, vol. 55, n.º 2, pp. 34-39, feb. 2017.

[3] «Gartner Says 8.4 Billion Connected». [Online]. Disponible en: //www.gartner.com/newsroom/id/3598917. [Último acceso: 17-oct-2017].

[4] N. V. Lopes, F. Pinto, P. Furtado, y J. Silva, «IoT architecture proposal for disabled people», en *2014 IEEE 10th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, 2014, pp. 152-158.

[5] Instituto de Mayores y Servicios Sociales, «Base Estatal de datos de personas con valoración del grado de discapacidad». 2015.

[6] S. Seyed, *Assistive IoT: Enhancing Human Experiences*. 2017.

[7] G3ict, «Internet of Things: New Promises for. Persons with Disabilities». 2015.

[8] «OMS | Ceguera y discapacidad visual», *WHO*. [Online]. Disponible en: http://www.who.int/mediacentre/factsheets/fs282/es/. [Último acceso: 09-feb-2017].

[9] RetinaPlus+ y Ernst & Young, *Informe sobre la ceguera en España*. 2012.

[10] «OMS | 10 datos sobre la ceguera y la discapacidad visual», *WHO*. [Online]. Disponible en: http://www.who.int/features/factfiles/blindness/es/. [Último acceso: 09-feb-2017].

[11] Organización Mundial de la Salud, «Informe mundial sobre la discapacidad». 2011.

[12] P. Korbel, P. Skulimowski, P. Wasilewski, y P. Wawrzyniak, «Mobile applications aiding the visually impaired in travelling with public transport», en *2013 Federated Conference on Computer Science and Information Systems*, 2013, pp. 825-828.

[13] P. Korbel, P. Skulimowski, y P. Wasilewski, «A radio network for guidance and public transport assistance of the visually impaired», en *2013 6th International Conference on Human System Interactions (HSI)*, 2013, pp. 485-488.

[14] «Guaguas Municipales S.A.» [Online]. Disponible en: http://www.guaguas.com/. [Último acceso: 09-feb-2017].

[15] Particle, «Photon datasheet». [Online]. Disponible en: https://docs.particle.io/datasheets/photon-datasheet/. [Último acceso: 02-jun-2017].

[16] Particle, «Electron datasheet». [Online]. Disponible en: https://docs.particle.io/datasheets/electron-datasheet/. [Último acceso: 02-jun-2017].

[17] S. Monk, *Make: Getting Started with the Photon*. Maker Media.

[18] A. Vasudevan, K. Ghoshdastidar, y R. Khan, *Learning IoT with Particle Photon and Electron*. Packt Publishing.

[19] Parallax, «Emic 2 Text-to-Speech Module». [Online]. Disponible en: https://www.parallax.com/sites/default/files/downloads/30016-Emic-2-Text-To-Speech-Documentation-v1.2.pdf. [Último acceso: 02-sep-2017].

[20] Adafruit, «GPS datasheet». [Online]. Disponible en: https://www.adafruit.com/product/746. [Último acceso: 02-jun-2017].

[21] Particle, «Particle Cloud - scalable, secure infrastructure for your IoT product». [Online]. Disponible en: https://www.particle.io/products/platform/particle-cloud. [Último acceso: 09-jun-2017].

[22] Particle, «Particle Build». [Online]. Disponible en: https://build.particle.io/. [Último acceso: 09-jun-2017].

[23] Particle, «Language Syntax». [Online]. Disponible en: https://docs.particle.io/reference/firmware/photon/#language-syntax. [Último acceso: 09-jun-2017].

[24] Particle, «Particle Console». [Online]. Disponible en: https://docs.particle.io/guide/tools-and-features/console/. [Último acceso: 02-jun-2017].

[25] Particle, «Tinker». [Online]. Disponible en: https://docs.particle.io/guide/getting-started/tinker/core/. [Último acceso: 02-jun-2017].

[26] Particle, «Command Line Interface». [Online]. Disponible en: https://www.particle.io/products/development-tools/particle-command-line-interface. [Último acceso: 06-jun-2017].

[27] Particle, «Webhooks». [Online]. Disponible en: https://docs.particle.io/guide/tools-and-features/webhooks/. [Último acceso: 06-jun-2017].

[28] «Webhooks Overview». [Online]. Disponible en: https://docs.particle.io/reference/webhooks/#overview. [Último acceso: 10-sep-2017].

[29] Guaguas Municipales S.A., «Búsqueda por paradas». [Online]. Disponible en:

http://paradas.guaguas.com. [Último acceso: 06-jun-2017].

[30] N. Mattisson, *HttpClient: Http Client Library for the Spark Core (also well suited for Arduino and other embedded platforms)*. 2017.

[31] K. Varma, *TTS_SparkCore: Spark Core TTS Sample Using Emic2 Text-to-Speech module*. 2015.

[32] M. Hart, *TinyGPS: A compact Arduino NMEA (GPS) parsing library*. 2017.

[33] A. Kalinchuk, *tiny_gps_plus: TinyGPS++ Library*. 2016.

[34] *AssetTracker: Library for the Electron Asset Tracker!* Particle, 2017.

[35] «Google Maps», *Google Maps*. [Online]. Disponible en: https://www.google.es/maps. [Último acceso: 17-oct-2017].

[36] Particle, «System Thread». [Online]. Disponible en: https://docs.particle.io/reference/firmware/photon/#system-thread. [Último acceso: 19-jun-2017].

[37] Particle, «Shields and accessories, Internet Button». [Online]. Disponible en: https://docs.particle.io/datasheets/particle-shields/#internet-button. [Último acceso: 29-jun-2017].

[38] Adafruit, «The Magic of Neopixels». [Online]. Disponible en: https://learn.adafruit.com/adafruit-neopixel-uberguide/overview. [Último acceso: 02-jun-2017].

[39] Worldsemi, «WS2812 Intelligent control LED integrated light source». [Online]. Disponible en: https://cdn-shop.adafruit.com/datasheets/WS2812.pdf. [Último acceso: 02-jun-2017].

[40] «Using Serial2 on the Photon», *Particle*. [Online]. Disponible en: https://community.particle.io/t/using-serial2-on-the-photon/26871. [Último acceso: 03-jul-2017].

[41] «Flashing over Serial for the Electron». [Online]. Disponible en: https://docs.particle.io/guide/tools-and-features/cli/electron/#flashing-over-serial-for-the-electron. [Último acceso: 10-nov-2017].

[42] «Electron Serial». [Online]. Disponible en: https://docs.particle.io/reference/firmware/electron/#serial. [Último acceso: 10-nov-2017].

Pliego de condiciones

El Pliego de Condiciones expone las condiciones bajo las que se ha desarrollado el presente Trabajo Fin de Grado. A continuación se muestra el conjunto de herramientas hardware, software y firmware empleadas durante su realización.

PL. 1 Condiciones Hardware

En la Tabla 5 se presentan los equipos *hardware* utilizados.

Tabla 5. Relación de equipos hardware.

Equipo	Modelo	Fabricante/Comerciante
Ordenador	▪ MacBook Pro Retina ▪ CPU Intel Core i5 a 2,7 GHz ▪ 8 GB RAM	Apple
Particle Photon	PHOTON	Particle
Particle Electron	3G-U270	Particle
Módulo Emic 2 Text-to-Speech	Emic 2 Text-to-Speech Module	Parallax
Módulo GPS	Adafruit Ultimate GPS Breakout, versión 3	Adafruit
Internet Button	Internet Button	Particle

PL.2 Condiciones Software

En la Tabla 6 se exponen las herramientas software utilizadas, especificando su versión.

Tabla 6. Relación de herramientas software.

Aplicación	Versión	Fabricante
macOS Sierra	10.12.3	Apple
Microsoft Office	2015	Microsoft
Notepad++	7.3.3	Plataforma código abierto NOTEPAD++
Particle CLI	1.25.0	-

PL.3 Condiciones Firmware

Por último, en la Tabla 7 se muestra el *firmware* utilizado y su versión.

Tabla 7. Relación de firmware.

Firmware	Versión
Dispositivo Photon	0.6.3
Dispositivo Electron	0.6.3

Presupuesto

Esta sección contiene el Presupuesto que recoge los gastos generados en la realización del presente Trabajo Fin de Grado. Dicho presupuesto se divide en las siguientes partes:

- Trabajo tarifado por tiempo empleado.
- Amortización del inmovilizado material, dividida a su vez en:
    - Amortización del material *hardware*.
    - Amortización del material *software*.
- Redacción de la documentación.
- Derechos de visado del COITT (Colegio Oficial de Ingenieros Técnicos de Telecomunicación).
- Gastos de tramitación y envío.

Una vez analizados cada uno de los criterios establecidos, se aplicarán los impuestos vigentes y se procederá a la obtención del coste total del TFG.

P.1 Trabajo tarifado por tiempo empleado

Este concepto contabiliza los gastos que corresponden a la mano de obra, según el salario correspondiente a la hora de trabajo de un Ingeniero Técnico de Telecomunicación. Se propone utilizar la siguiente fórmula:

$$H = C_t \times 74,88 \times H_n + C_t \times 96,72 \times H_e, \qquad \text{(P.1)}$$

donde:

- H representa los honorarios totales por el tiempo dedicado.
- H_n detalla las horas normales trabajadas dentro de la jornada laboral.
- H_e específica las horas especiales trabajadas.
- C_t es un factor de corrección función del número de horas trabajadas.

Para la realización del presente TFG se han invertido un total de 300 horas. Todas ellas se han realizado dentro del horario normal, por lo que el número de horas especiales es cero. Además, de acuerdo a lo establecido por el COITT, el factor de corrección C_t a aplicar para 300 horas trabajadas es de 0,60, tal y como se puede comprobar en la Tabla 8.

Tabla 8. Coeficientes reductores para trabajo tarifado (COITT).

Horas	**Factor de corrección**
Hasta 36	1,00
Exceso de 36 hasta 72	0,90
Exceso de 72 hasta 108	0,80
Exceso de 108 hasta 144	0,70
Exceso de 144 hasta 180	0,65
Exceso de 180 hasta 360	0,60
Exceso de 360 hasta 510	0,55
Exceso de 510 hasta 720	0,50
Exceso de 720 hasta 1080	0,45
Exceso de 1800	0,40

Teniendo en cuenta estos datos, el coste total de honorarios asciende a:

$$H = 0{,}6 \times 74{,}88 \times 300 + 0{,}6 \times 96{,}72 \times 0 = 13.478{,}40 \text{ €} \qquad \text{(P.2)}$$

Por lo tanto, el trabajo tarifado por tiempo empleado asciende a la cantidad de *trece mil cuatrocientos setenta y ocho euros con cuarenta céntimos.*

P.2 Amortización del inmovilizado material

En el inmovilizado material se consideran tanto los recursos *hardware* como *software* empleados para la realización de este TFG.

Para estipular el coste de amortización en un periodo de 3 años se utiliza un sistema de amortización lineal, en el que se supone que el inmovilizado material se deprecia de forma constante a lo largo de su vida útil. La cuota de amortización anual se calcula de la siguiente forma:

$$Cuota\ anual = \frac{Valor\ de\ adquisición - Valor\ residual}{Número\ de\ años\ de\ vida\ útil}\ , \qquad \text{(P.3)}$$

donde el valor residual se corresponde con el valor teórico que se supone que tendrá el elemento en cuestión después de su vida útil.

P.2.1 Amortización del material *hardware*

Debido a que la duración de este Trabajo Fin de Grado es de tan solo 4 meses, siendo este periodo muy inferior al de 3 años estipulado para el coste de amortización, los costes se calcularán en base a los derivados de los primeros 4 meses.

En la Tabla 9 se especifica el *hardware* amortizable necesario para la realización del TFG, indicando su valor de adquisición y su amortización, teniendo en cuenta un tiempo de uso de 4 meses.

Tabla 9. Costes y amortización del hardware (I).

Elemento	Valor de adquisición	Amortización
Ordenador MacBook Pro	1329 €	147,66 €
Total	**1329 €**	**147,66 €**

Por otro lado, en la Tabla 10 se muestra el resto de material *hardware* utilizado y para el que, debido a su bajo precio, su amortización coincide con su valor de adquisición.

Tabla 10. Costes y amortización del hardware (II).

Elemento	Valor de adquisición	Amortización
Particle *Photon*	16,19 €	16,19 €
Particle *Electron*	58,80 €	58,80 €
Módulo *Emic 2 Text-to-Speech*	51,13 €	51,13 €
Adafruit Ultimate GPS Breakout	34 €	34 €
Total	**160,12 €**	**160,12 €**

Finalmente, tras realizar la suma de ambos conceptos se obtiene el coste total del material *hardware*, tal y como se muestra en la Tabla 11.

Tabla 11. Costes y amortizaciones totales del hardware.

Concepto	Coste
Costes y amortización del hardware (I)	147,66 €
Costes y amortización del hardware (II)	160,12 €
Total	**307,78 €**

El coste total del material *hardware* asciende a *trescientos siete euros con setenta y ocho céntimos.*

P.2.2 Amortización del software

Para el cálculo de los costes de amortización del material *software* se considerarán, al igual que con el material *hardware*, los costes derivados de los primeros 4 meses de uso.

La Tabla 12 muestra los elementos *software* necesarios para la realización del TFG, así como su valor de adquisición y su amortización.

Tabla 12. Costes y amortización del software.

Elemento	Valor de adquisición	Amortización
macOS Sierra	0,00 €	0,00 €
Microsoft Office 2015	0,00 €(*)	0,00 €(*)
Notepad++	0,00 €	0,00 €
Particle CLI	0,00 €	0,00 €
Total	**0 €**	**0 €**

(*) Licencia de uso proporcionada por la ULPGC.

Por tanto, el coste total del material *software* es de *cero* euros.

P.3 Redacción del trabajo

Se ha utilizado la expresión (P.4) para determinar el coste asociado a la redacción de la memoria del presente Trabajo Fin de Grado.

$$R = 0,07 \times P \times C_n, \qquad \text{(P.4)}$$

donde:

- R son los honorarios por la redacción del trabajo.
- P es el presupuesto.
- C_n es el coeficiente de ponderación en función del presupuesto.

El valor del presupuesto P se calcula sumando los costes del trabajo tarifado por tiempo empleado y de la amortización del inmovilizado material, tanto *hardware* como *software*. El resultado de los costes se muestra en la Tabla 13.

Tabla 13. Presupuesto, incluyendo trabajo tarifado y amortización del inmovilizado material.

Concepto	Coste
Trabajo tarifado por tiempo empleado	13.478,40
Amortización del material *hardware*	307,78 €
Amortización del *software*	0,00 €
Total	**13.786,18 €**

Como el coeficiente de ponderación C_n para presupuestos menores de 30.050,00€ viene definido por el COITT con un valor de 1.00, el coste derivado de la redacción del Trabajo Fin de Grado es de:

$$R = 0,07 \times 13.786,18 \times 1 = 965,03 \in \quad \text{(P.5)}$$

Ascendiendo de esta forma el coste de la redacción del TFG a *novecientos sesenta y cinco euros con tres céntimos.*

P.4 Derechos de visado del COITT

El COITT establece que para proyectos técnicos de carácter general, los derechos de visado para 2017 se calculan en base a (P.6).

$$V = 0,006 \times P_1 \times C_1 + 0,003 \times P_2 \times C_2, \quad \text{(P.6)}$$

donde:

- V es el coste de visado del trabajo.
- P_1 es el presupuesto del proyecto.
- C_1 es el coeficiente reductor en función del presupuesto.
- P_2 es el presupuesto de ejecución material correspondiente a la obra civil.
- C_2 es el coeficiente reductor en función a P_2.

El valor del presupuesto P_1 se halla sumando los costes de las secciones correspondientes al trabajo tarifado por tiempo empleado, a la amortización del inmovilizado material y a la

redacción del documento. Esta suma se muestra en la Tabla 14. Al igual que en el caso anterior, el coeficiente C_1 para proyectos de presupuesto inferior a 30.050,00€ es de 1,00, asimismo el valor de P_2 es de 0,00€, ya que no se realiza ninguna obra.

Tabla 14. Presupuesto, incluyendo trabajo tarifado, amortización y redacción del trabajo.

Concepto	**Coste**
Trabajo tarifado por tiempo empleado	13.478,40
Amortización del material hardware	307,78 €
Amortización del material software	0,00 €
Redacción del trabajo	965,03 €
Total	**14.751,21 €**

De esta forma, aplicando a la expresión (P.6) los datos de la Tabla 14 y el coeficiente especificado, se obtiene:

$$V = 0,006 \times 14.751,21 \times 1 = 88,50 \text{ €} \qquad \text{(P.7)}$$

Los costes por derechos de visado del TFG ascienden a *ochenta y ocho euros con cincuenta céntimos*.

P.5 Gastos de tramitación y envío

Los gastos de tramitación y envió están estipulados en seis euros (6,00€) por cada documento visado de forma telemática.

P.6 Material fungible

Además de los recursos *hardware* y *software*, en este TFG se han empleado otros materiales, como folios y el tóner de la impresora, entre otros, que quedan englobados como material fungible. En la Tabla 15 se muestran los costes derivados de estos recursos.

Tabla 15. Costes de material fungible.

Concepto	Coste
Folios	10,00 €
Tóner de la impresora	30,00 €
Encuadernado	4,00 €
Tres CD-ROM	6,00 €
Total	**50,00 €**

Los costes de material fungible ascienden a *cincuenta euros*.

P.7 Aplicación de impuestos y coste total

La realización del presente TFG está gravada por el Impuesto General Indirecto Canario (IGIC) en un siete por ciento (7 %). En la Tabla 16 se muestra el presupuesto final con los impuestos aplicados.

Tabla 16. Presupuesto total del Trabajo Fin de Grado.

Concepto	Coste
Trabajo tarifado por tiempo empleado	13.478,40
Amortización del material hardware	307,78 €
Amortización del material software	0,00 €
Redacción del trabajo	965,03 €
Derechos de visado del COIT	88,50 €
Gastos de tramitación y envío	6,00 €
Costes de material fungible	50,00 €
Total (Sin IGIC)	**14.895,71 €**
IGIC (7%)	1.042,70 €
Total	**15.938,40 €**

El presupuesto total del Trabajo Fin de Grado *"Dispositivo de información sobre paradas de transporte público para usuarios con discapacidad visual basado en IoT"* asciende a *quince mil novecientos treinta y ocho euros con cuarenta céntimos*.

Fdo.: Dª. Sara León Gutiérrez

En Las Palmas de Gran Canaria a 1 de diciembre de 2017

Anexos

Anexo A. Contenido del CD-ROM

En este Anexo se presenta el contenido del CD-ROM adjunto a este documento, el cual está formado por las siguientes carpetas:

- En la carpeta *Memoria del TFG* se encuentra la memoria del TFG "*Dispositivo de información sobre paradas de transporte público para usuarios con discapacidad visual basado en IoT*" en formato PDF.