UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE UPV INGENIEROS DE TELECOMUNICACIÓN

INTERFACE DE REALIDAD AUMENTADA PARA EL CONTROL DE UN SISTEMA INTELIGENTE INCORPORADO EN EL TRAJE DEL CUERPO DE BOMBEROS

Jorge García Vega

Tutor: José Manuel Mossi García

Trabajo Fin de Máster presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Máster en Ingeniería de Telecomunicación

Curso 2016-17

Valencia, 24 de julio de 2017

Agradecimientos

Quisiera agradecer principalmente a mi tutor por el apoyo otorgado durante la realización del trabajo, por sus consejos, metodología aplicada y por la ambición que le caracteriza. Agradecer también a Carmen Bachiller por dejarnos hacer uso de su impresora 3D en numerosas ocasiones. Agradecer a mis padres por otorgarme la posibilidad de cursar esta carrera y por la educación que me dieron, y a mi familia en general y compañeros por hacer que el camino fuera mucho más fácil de recorrer.

Muchas gracias a todos vosotros.

Resumen

Este proyecto consiste en la realización de un prototipo de sistema de realidad aumentada para dotar a los agentes de bomberos de una cámara térmica y ver la imagen con un dispositivo integrado en el interior de la máscara de oxígeno de modo que puedan ver la visión normal junto con la imagen térmica superpuesta. Este avance les permite una mejor operatividad puesto que en estos días la forma de trabajar es llevar en la mano la cámara con display y eso les limita ya que tienen las manos ocupadas y tienen que elegir entre actuar o ver. La imagen térmica es imprescindible en las actuaciones donde hay mucho humo porque la visibilidad normal se pierde porque, si el humo es denso, la luz no lo atraviesa, mientras que la radiación térmica sí.

El sistema consta de dos partes. La incorporada en el casco consiste en la cámara térmica, la CPU, la transmisión WiFi y el sistema de alimentación con batería. La segunda parte está en el interior de la máscara y consiste en el display, cristal de presentación para superponer la imagen térmica a la luz del campo de visión normal, el microcontrolador con su receptor WiFi y la batería. Para el demostrador se ha desarrollado también el modelado 3D, tanto de la caja contenedora que se anexa al casco conteniendo la cámara y el resto de elementos, como del soporte para el interior de la máscara de oxígeno que alberga la parte receptora y el display.

Resum

Aquest projecte consisteix en la realització d'un prototip de sistema de realitat augmentada per dotar els agents de bombers d'una càmera tèrmica i veure la imatge amb un dispositiu integrat a l'interior de la màscara d'oxigen de manera que puguen vore la visió normal juntament amb la imatge tèrmica superposada. Aquest avanç els permet una millor operativatat ja que en aquests dies la forma de treballar és portar a la mà la càmera amb display i això els limita doncs tenen les mans ocupades i han de triar entre actuar o vore. La imatge tèrmica és imprescindible en les actuacions on hi ha molt de fum doncs la visibilitat normal es perd perquè, si el fum és dens, la llum no el travessa, mentre que la radiació tèrmica si.

El sistema consta de dues parts. La incorporada al casc consisteix en la càmera tèrmica, la cpu, la transmissió wifi i el sistema d'alimentació amb bateria. La segona part és a l'interior de la màscara i consisteix en la pantalla, vidre de presentació per superposar la imatge tèrmica a la llum del camp de visió normal, el microcontrolador amb el seu receptor wifi i la bateria.

Per al demostrador s'ha desenvolupat també el modelatge 3D, tant de la caixa contenidora que s'annexa al casc contenint la càmera i la resta d'elements, com del suport per a l'interior de la màscara d'oxigen que alberga la part receptora i la pantalla.

Abstract

This project consists of the realization of a prototype of augmented reality system to equip firemen with a thermal camera and see the image with an integrated device inside the oxygen mask so that they can see normal vision together with the superimposed thermal image. This advance allows them a better operation since in these days the way of working is to carry the camera with display in hand and that limits them since they have their hands occupied and have to choose between acting or seeing. The thermal image is essential in the performances where there is a lot of smoke because the normal visibility is lost because, if the smoke is dense, the light does not cross it, whereas the thermal radiation yes.

The system consists of two parts. The built-in helmet contains the thermal camera, the CPU, WiFi transmission and the battery powered system. The second part is inside the mask and consists of the display, crystal display to superimpose the thermal image in the light of the normal field of view, the microcontroller with its WiFi receiver and battery.

For the demo a 3D modeling has also been developed, both the container box that is attached to the helmet containing the camera and other elements, and the support for the interior of the oxygen mask that houses the receiving part and the display.

Índice

Capítulo 1. Introducción

Todos los bomberos están expuestos a numerosos peligros a la hora de realizar sus maniobras de rescate. Uno de esos peligros es la falta de visibilidad a la hora de entrar en algún recinto en llamas con poca ventilación, dónde el humo de la combustión se extiende rápidamente y dificulta enormemente la visibilidad tanto en el proceso de entrada como en el proceso de salida. Para ambos procesos, el cuerpo de bomberos utiliza un método basado en una cuerda que van desplegando a medida que van avanzando durante su intervención y que posteriormente les permite deshacer el trayecto recorrido y encontrar con seguridad la salida. Un agente es el que maneja la cuerda y los demás siguen al agente en primera fila. Este agente a su vez sujeta en su mano una cámara térmica que es capaz de obtener una imagen a través del humo y va informando y dando indicaciones a sus compañeros[1]. Es un protocolo aceptable con la tecnología disponible hoy en día pero es mejorable en dos sentidos. El agente que lleva la cámara no tiene las manos libres para realizar otras funciones y por otro lado sus compañeros tienen una autonomía muy limitada porque no ven y dependen de las indicaciones de la persona que lleva la cámara térmica. El presente proyecto pretende avanzar en la realización de un sistema donde el casco y la máscara de oxígeno incorporen un cámara térmica y un sistema de visualización de realidad aumentada de forma que todos y cada uno de los agentes puedan tener visión térmica sin necesidad de sostener la cámara y por tanto tener las manos libres para las labores de salvamento.

Figura 1 Cuerpo de bomberos entrenando en el parking de la central

[1] Protocolo de actuación de los bomberos en Navarra, http://www.bomberosdenavarra.com/documentos/ficheros_documentos/intervencion.pdf

A su vez, paralelamente se pensó en dotar a dicho sistema de un control para poder cambiar lo que se visualiza en el sistema de realidad aumentada, gracias a la inserción de sensores en el guante del traje de bomberos, pero dicha línea de trabajo no entra dentro de este trabajo de fin de máster pero sí en la de un compañero de la escuela.

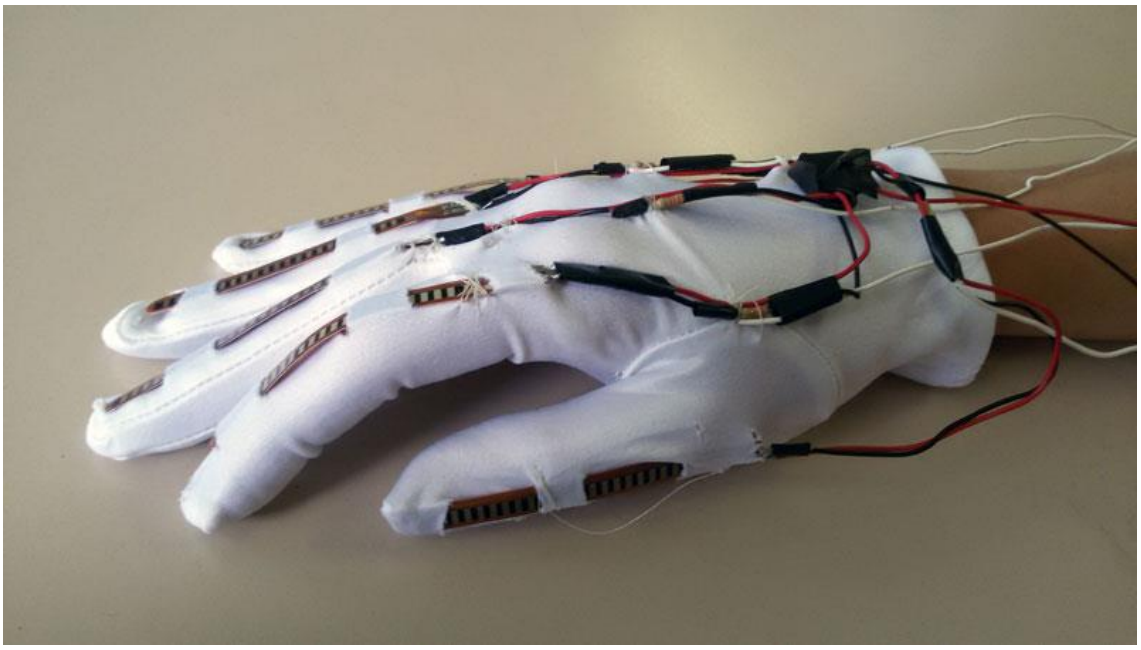

Figura 2 Ejemplo de guante para el control del sistema de realidad aumentada

La idea principal del producto en sí no es únicamente ofrecerles una visión térmica del entorno, si no poder dotarles de un sistema de realidad aumentada completo, el cuál puedan controlar mediante los gestos de una mano que tengan en ese momento desocupada, y que permita mostrarles todo tipo de información suplementaria, como pueda ser las constantes vitales suya y de sus compañeros, la cantidad de oxígeno en la bombona, temperatura, humedad, una simulación del mapeado del edificio, etc. Es lo que se denomina más comúnmente en el mundo de los videojuegos y realidad aumentada como Head-Up Display (HUD, visualización cabeza arriba)[2]. Permite al usuario obtener información adicional sin dejar de prestar atención a la tarea principal. Es enormemente usada en algunos coches de alta gama, vehículos militares, o simplemente en los videojuegos.

[2] Head-Up Displays, https://en.wikipedia.org/wiki/Head-up_display

Figura 3 Ejemplo de HUD en videojuegos

Finalmente, decir que se busca dotarles de un protocolo de actuación más avanzado y más seguro mediante el trabajo en paralelo en diferentes áreas del producto para poder así hacer un sistema completo final futuro, comercial, competitivo, y con un muy bajo coste debido al precio de la electrónica usada.

Capítulo 2. Visión global del sistema

En este capítulo se mostrará una visión global del sistema diseñado capaz de dotar el traje del cuerpo de bomberos de realidad aumentada para facilitar las labores de rescate o extinción de fuegos. El objetivo principal del sistema es el de abaratar costes en la medida de lo posible, consiguiendo así un producto final que pueda ser incorporado en el equipamiento estándar de todos los agentes. Para ello, se ha usado un conjunto de tecnologías punteras y a la vez baratas, formando el sistema final que se muestra a continuación:

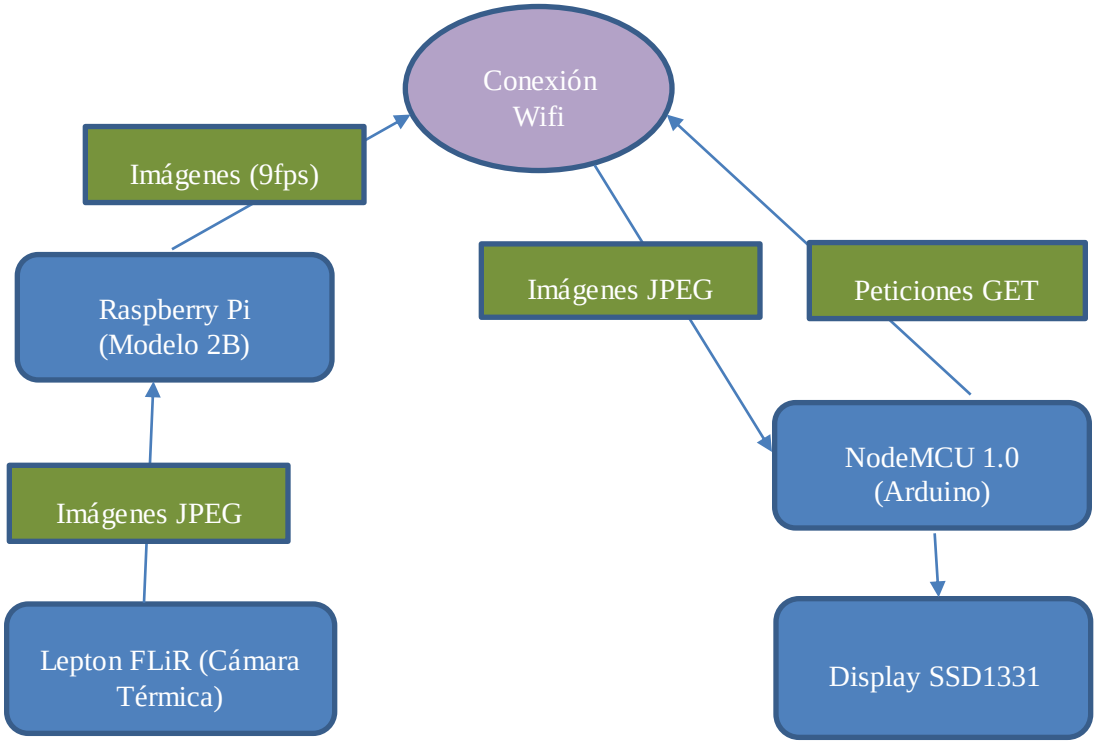

Figura 4 Vista en conjunto del sistema

Como se ha comentado en el capítulo de introducción, el sistema incorporará una imagen de visión térmica a tiempo real del entorno que está visualizando el agente. Para ello, el sistema consta de dos partes bien diferenciadas:

- La parte back-end: esta parte está compuesta de una computadora de placa reducida llamada Raspberry Pi (Modelo Zero) y de una cámara infrarroja Lepton FLiR que capta imágenes en la banda de 8 a 14 µm de longitud de onda, ofreciendo así una visión térmica del entorno.
  Dentro de la Raspberry Pi se lanza un script que permite obtener directamente las imágenes de la cámara térmica y se ofrecen en una página web accesible mediante el protocolo HTTP y el estándar de comunicación WiFi. El programa, llamado MJPG-Streamer[3], toma imágenes JPG de cámaras compatibles con Linux o de cámaras UVC (USB Video Device Class) y las ofrece a tiempo real en un servidor web. Se puede acceder a ellas directamente conectándose a la dirección IP del servidor, o haciendo peticiones GET de la imagen (como se explicará más detalladamente en el capítulo referente a la Raspberry Pi).

[3] MJPG-Streamer, https://sourceforge.net/projects/mjpg-streamer/ , https://github.com/jacksonliam/mjpg-streamer

Figura 5 Parte back-end: Raspberry Pi Zero y cámara Lepton FLiR

- La parte front-end: esta parte consta de un hardware basado en el socket WiFi ESP8266, el NodeMCU 1.0, totalmente programable en Arduino (basado en C y C++), que a su vez está conectado al display SSD1331 (80x60 pixeles de resolución).
  El firmware es el encargado de hacer peticiones al servidor web montado en la parte de back-end, recibir las imágenes captadas por la cámara térmica, y representarlas en el display. Para ello, se usa un conjunto de librerías para representar en el display y para descomprimir la imagen recibida en formato JPEG (se detallará más precisamente en el capítulo referente al firmware NodeMCU basado en Arduino). La visualización se realiza gracias a la reflexión de la imagen amplificada por la lente gracias a un cristal que se sitúa encima con el ángulo necesario para alcanzar el ojo del bombero.

Figura 6 Parte front-end: NodeMCU Firmware y display SSD1331

La parte back-end se sitúa en el frontal del casco homologado del traje del cuerpo de bomberos para captar el entorno. La parte de back-end se sitúa dentro de la máscara de gas homologada del traje del cuerpo de bomberos. Como se puede apreciar en las figuras anteriores, el resultado que se obtiene es un sistema ligero, totalmente accesible y funcional que permitirá realizar las labores del cuerpo de bomberos de manera más sencilla y eficaz dejando las manos libres.

Capítulo 3. El sistema de realidad aumentada

Los últimos años se ha observado un progreso enorme en la tecnología en todos sus ámbitos. Uno de los frentes en los que más se ha avanzado, pero que sin embargo es de difícil aceptación por el público[4], es el de la realidad virtual y realidad aumentada. La realidad virtual consiste en reemplazar la realidad por una diseñada por ordenador, y hacer sentir al usuario que el mundo en el que está desarrollándose la acción es totalmente nuevo. La realidad aumentada sin embargo, se aprovecha del entorno de la vida real para acentuarlo incorporando información adicional de gran utilidad en la visión del usuario. El primer tipo de realidad se usa enormemente en entornos de simulación, medicina, psicológica y en el ámbito de los videojuegos. No obstante, la realidad aumentada tiene un objetivo mucho más funcional y ambicioso. Se usa en proyectos educativos, arquitectura, diseño, navegación, o como en este caso, en servicios militares o de emergencia. En éste último ámbito las aplicaciones son extremadamente eficientes y útiles puesto que permiten obtener una cantidad de información adicional enorme a los usuarios y marcan la diferencia en términos de tiempo de reacción, rapidez y eficacia.

El sistema completo se ha visto altamente influenciado en su diseño por el estado del arte de la realidad virtual y realidad aumentada, más precisamente por el dispositivo llamado Gafas Moverio[5]. En estas gafas se controla con la mano una interfaz parecida a la de un smartphone y se visualiza en el cristal frontal de las gafas, otorgando así información adicional que cumpla con los requisitos del usuario.

Figura 7 Gafas de realidad aumentada EPSON Moverio

En la realidad aumentada existen tres tipos de visualización:

Realidad aumentada en gafas o cristales: se hace uso de reflexiones ópticas mediante cristales para hacer llegar la información al ojo del usuario. Un ejemplo claro de este tipo de visualización es el utilizado en la gafas Moverio o en este proyecto. En el lateral de las gafas se incluye toda la electrónica necesaria para recibir la imagen modificada por el sistema de control del usuario y proyectarla hacia el cristal semitransparente frontal de las gafas gracias a unos mini proyectores. El cristal realiza la combinación de la luz proveniente del entorno que incide en la cara externa del cristal (y pasa a través de él por "transmisión") con la luz proveniente del display que incide en su cara interna (y sale del cristal por "reflexión). La imagen pasa por un conjunto de lentes que la amplifican de manera que el ojo no fuerce el músculo a la hora de enfocar.

[4] Dificil aceptación de los dispositivos de Realidad Virtual y Aumentada, http://www.elpublicista.es/mundo-online/aceptacion-social-realidad-virtual-aumentada-dependera-autonomia

[5] Gafas Moverio, https://www.epson.es/products/see-through-mobile-viewer/moverio-bt-200

- Realidad aumentada en soporte de mano: la visualización se produce en una pantalla que cabe en la mano, ya sea en un móvil u otro tipo de interfaz. Como ejemplo tenemos la realidad aumentada utilizada en algunos videojuegos como Pokemon Go o simplemente, la utilizada en algunos filtros de aplicaciones fotográficas:

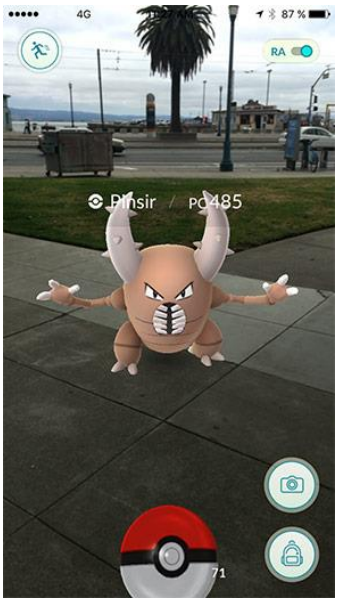

Figura 8 Realidad aumentada de tipo soporte en mano en Pokemon Go

- Realidad aumentada en proyección espacial (SAR): en éste último caso se utiliza una proyección de la imagen desde el suelo o techo de la sala o entorno para visualizar el contenido dando la sensación de que realmente está presente.

Figura 9 Realidad aumentada de tipo SAR

En este proyecto, se quiso mantener la idea principal de las gafas Moverio pero abaratando costes y logrando un sistema totalmente modificable y programable gracias al uso de firmware de software libre. La parte óptica, el principio por el cuál funcionan casi todos los sistemas de realidad aumentada, se mantiene parecida. En el caso del proyecto, se tiene un display que muestra

la información requerida dentro de una rosca circular hembra. En la rosca circular macho se encuentra una lente convergente que adapta la imagen. Debido a que cada usuario puede tener un número diferente de dioptrías, se usa este sistema de roscas para modificar la distancia focal[6] y así mejorar la nitidez de la imagen vista por el usuario.

Figura 10 Sistema de amplificación de la imagen

Tras amplificar la imagen, se coloca un cristal con una inclinación de 45º respecto a la horizontal, con un cociente de reflexión deseado, y a una distancia tal que la reflexión alcance el iris del ojo del usuario.

[6] La distancia focal de una lente se define como la distancia entre el centro óptico de la lente y el punto focal. A dicha distancia la nitidez de la imagen es máxima. https://en.wikipedia.org/wiki/Focal_length

Capítulo 4. Subsistema back-end con la Raspberry Pi

Para lograr que todo el sistema funcione, el subsistema back-end que ofrece toda la información al subsistema front-end es una de las partes fundamentales del proyecto. En ella se realizan todas las labores necesarias para que la comunicación entre ambas partes sea posible y la información totalmente accesible. En este capítulo se detallarán todas las características de la Raspberry Pi, las características de la cámara térmica utilizada, y cómo se logra la funcionabilidad de ambas.

4.1 CPU del subsistema back-end

Los requisitos que debía tener este subsistema eran los siguientes:

- Modificable: siempre que se quiera modificar le información que se ofrece o la forma en la que se ofrece debe ser una tarea sencilla a realizar.
- Autoabastecido: debe poder alimentarse sin la necesidad de estar conectado a la red eléctrica.
- Bajo consumo: debido a que las tareas de rescate o extinción de incendios pueden alargarse, debe tener una duración en funcionamiento suficientemente elevada.
- Pequeño: se debe disminuir en la medida de lo posible el tamaño del subsistema para no entorpecer las labores de los bomberos.
- Capacidad de procesado alta: en la versión del proyecto no se exige una gran carga de trabajo, pero conforme se vayan añadiendo funcionalidades el subsistema debe responder correctamente y ofrecer la menos latencia posible puesto que el tiempo es oro en las labores de rescate y extinción de incendios.

Para cumplir con todas estas características, se optó por usar una computadora de tamaño reducido llamada Raspberry Pi (Modelo Zero). Las características de la placa son las siguientes[7]:

- 1GHz, Single-core CPU
- 512MB RAM
- Puerto Mini-HDMI
- Puerto Micro-USB OTG
- Micro-USB
- 40 entradas pin compatible con HAT
- Conector de cámara CSI
- Programable en numerosos lenguajes como JavaScript, Python, C, etc.

Figura 11 Raspberry Pi Zero

El software de la Raspberry Pi es *open source* y desarrolla una versión adaptada de Debian llamada Raspbian. A pesar de ser el sistema operativo estándar y avalado por los desarrolladores de la marca, permite instalar otros sistemas operativos como una versión adaptada de Windows 10 o versiones de Linux para hacer pruebas de seguridad informática de sistema (Kali Linux). En este caso, se utilizará Raspbian para la implementación del proyecto.

[7] Características de la Raspberry Pi Zero, https://www.raspberrypi.org/products/raspberry-pi-zero/

4.2 Módulo de captación de imágenes por infrarrojos

Por otro lado, era necesario encontrar el dispositivo de captación de imágenes térmicas. Se decidió trabajar con el módulo de cámara Lepton FLiR puesto que aparte de cumplir con los requisitos que se habían marcado, la comunidad había trabajado enormemente con este módulo junto con la Raspberry. Las características de la cámara son las siguientes:

- Banda de trabajo de 8 a 14 µm
- 51 grados de FOV horizontal, 63.5 grados para la diagonal
- Imágenes de 80x60 píxeles
- Interfaces de video SPI y MIPI
- Potencia nominal requerida de 150 mW
- Tasa de frames por segundo de 9 fps (limitado por normativa militar estadounidense)

Figura 12 Módulo Lepton FLiR

4.3 Conexión de la cámara a la Raspberry y configuración

Para conectar la cámara Lepton FLiR a la Raspberry es necesario seguir el esquema de conexiones siguiente:

- CS → Pin 24, CE0
- MOSI → Pin 19, MOSI
- MISO → Pin 21, MISO
- CLK → Pin 23, CLK
- GND → Pin 6, GND
- VIN → Pin 1, 3V3
- SDA → Pin 3, SDA
- SCL → Pin 5, SCL

Las conexiones quedarían de la siguiente forma:

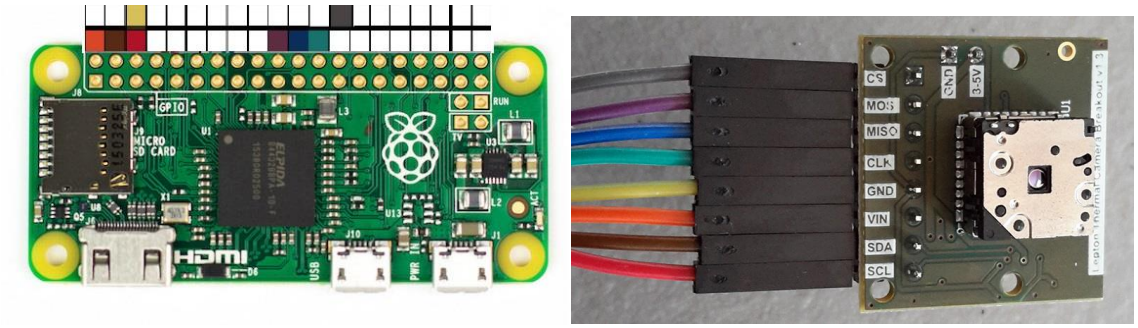

Figura 13 Conexiones de la cámara a la Raspberry Pi

Normalmente, el paso previo a conectar ambos dispositivos es el de activar las opciones de interfaz I2C así como las de soporte de Raspberry Pi Camera en el arranque de la computadora[8], pero las versiones más actualizadas de Raspbian tienen una interfaz para el control de estas opciones en el propio escritorio.

A continuación, será necesario descargar las librerías de OpenCV-Python para poder ejecutar códigos de C/C++ bajo un envoltorio de Python (puesto que C/C++ son lenguajes más rápidos que Python[9]) y los *packages* necesarios para hacer funcionar la cámara Lepton correctamente. Para ello, ejecutamos en la terminal de la Raspberry Pi los siguientes comandos:

$ sudo apt-get install python-opencv (descarga e instala las librerías de OpenCV-Python)

$ git clone https://github.com/kekiefer/pylepton.git (clona el repositorio de github en el directorio de trabajo)

$ cd pylepton (abre la carpeta pylepton)

$ sudo python setup.py install (ejecuta el código de python *setup.py*)

$ pylepton_overlay -a 155 (ejecuta una muestra de la captación de la cámara térmica con una transparencia de valor 155, siendo 255 una imagen térmica pura, y 0 una imagen normal)

$ pylepton_capture output.png (captura la imagen y la guarda en el fichero *output.png*)

4.4 Subsistema en funcionamiento

Una vez conectado el módulo de la cámara con la Raspberry, y funcionando el programa que guarda las imágenes captadas en ficheros JPEG, la siguiente tarea a realizar era la de servir dichas imágenes y hacerlas accesibles para el subsistema de front-end. MJPG-Streamer, la aplicación utilizada, es un proyecto realizado originalmente por *Tom Stöveken* que ejecuta un conjunto de líneas de comando para obtener imágenes JPEG de un directorio, de una cámara USB conectada, de un servidor web, o de la cámara oficial de Raspberry para copiarlas o redirigirlas hacia otro tipo de salida como puede ser un servidor web por HTTP, una simple ventana de visualización dentro del sistema de la Raspberry, u ofrecer las imágenes por protocolo UDP. Lo más lógico dado el tipo de aplicación sería ofrecer las imágenes por UDP en vez de TCP puesto que es un protocolo pensado para este tipo de aplicaciones, pero debido al estado de las librerías para el firmware utilizado en el front-end, y por más motivos que se explicarán en el capítulo siguiente referente al subsistema front-end, se optó por la solución del servidor web y trabajar con el protocolo HTTP y TCP.

MJPG-Streamer obtendrá las imágenes captadas por la cámara térmica y montará un servidor web local en la Raspberry accesible gracias a cualquier dispositivo que se conecte a su IP. En el caso de la imagen siguiente, se tiene una computadora conectada al punto de acceso WiFi creado por la Raspberry, y accediendo a su dirección IP y servicio con *10.0.2.1:8080/?action=stream.*

[8] Tutorial de conexiones y configuración de la Raspberry Pi y la cámara Lepton FLiR:
https://groupgets.com/blog/posts/8-installation-guide-for-pure-breakout-board-on-raspberry-pi-2
[9] Computer Language Benchmarks Game:
http://benchmarksgame.alioth.debian.org/u64q/compare.php?lang=python3&lang2=gpp

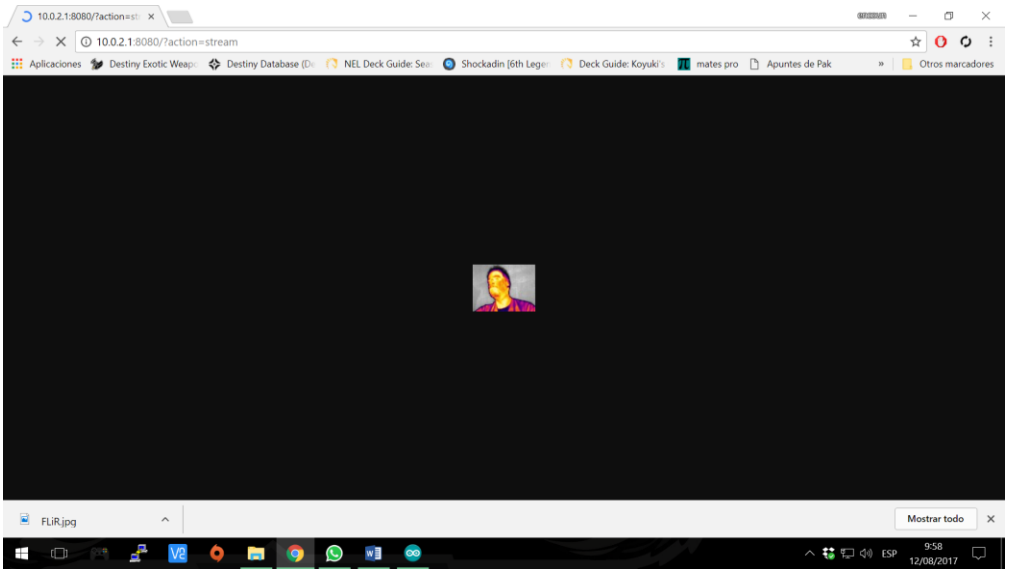

Figura 14 Representación de las imágenes capturadas por MJPG-Streamer en un browser

De esta forma es como se accederá a las imágenes desde la parte del front-end con el firmware NodeMCU gracias a un programa en Arduino. Se detallará el proceso en el capítulo referente al front-end.

Capítulo 5. Estándares y protocolos usados para la comunicación

Tal y como se ha mencionado en el capítulo anterior, el subsistema de back-end con la Raspberry Pi lanza un servidor web al que será posible acceder utilizando el protocolo HTTP junto con TCP. Dado que la parte de front-end se sitúa dentro de la máscara del traje de bomberos y la parte de back-end se sitúa en la parte frontal del casco, se descartó totalmente utilizar un medio de comunicación cableado, por lo que la comunicación entre ambas partes se hace vía WiFi. En este capítulo se detallarán tanto las características del estándar WiFi utilizado como las del protocolo HTTP y se justificará el uso de éste.

5.1 Estándar de comunicaciones WiFi

WiFi es el nombre comercial dado a la tecnología de comunicación inalámbrica de dispositivos basados en el estándar IEEE 802.11. Casi la gran mayoría de dispositivos electrónicos hoy en día tienen acceso a esta tecnología debido al gran crecimiento del mundo de Internet de las Cosas (IoT) en el que el principio fundamental es la interconexión de dispositivos. De hecho, tanto la Raspberry Pi como el firmware NodeMCU utilizado para la parte de front-end son dispositivos que tienen acceso a este tipo de tecnología en parte porque nacieron del mundo de IoT. La tecnología WiFi permite interconectar dispositivos entre sí gracias a la creación de redes de diferente tamaño gestionadas por uno o varios routers que son los encargados de redirigir la información recibida por parte de algunos dispositivos hacia el destino correcto. En el caso de este proyecto, la Raspberry Pi es el router central de la red WiFi creada a la que se pueden conectar uno o varios dispositivos. Por ejemplo, en el caso de esta muestra del proyecto es un dispositivo el que se conecta a la Raspberry Pi y obtiene la información ofrecida por ésta, pero varios dispositivos podrían conectarse para obtener la misma información o incluso una información específica para cada dispositivo si se desea. Por lo tanto, este tipo de tecnología da acceso a cualquier dispositivo a la información que éste requiera. Se descartó en un principio el uso de tecnologías como Bluetooth puesto que el ancho de banda de éste es limitado, la cobertura también es limitada, y no se tiene tanto control a nivel de las diferentes capas de protocolos como se puede tener con WiFi. Tal y como se comentará en el capítulo referente al subsistema de front-end, la facilidad para acceder a los paquetes transmitidos entre ambos subsistemas permitió solventar una numerosa cantidad de problemas para la recepción de los paquetes conteniendo la imagen[10].

5.2 HTTP y TCP, frente a UDP

5.2.1 HTTP en capa de aplicación

HTTP (HyperText Transfer Protocol) define la semántica y sintaxis que se utilizan para la comunicación entre elementos de software de la arquitectura web, es decir, clientes, servidores, proxies, etc. El funcionamiento es simple: el cliente hace peticiones al servidor y éste le contesta con el contenido demandado en caso de que lo tenga, o con confirmaciones de que la acción solicitada se ha realizado o no. Básicamente sigue el esquema de petición-respuesta, en el que no se mantiene una conexión entre cliente y servidor, su relación es momentánea y temporalmente restringida al tiempo entre petición y respuesta. Sí que es cierto que en ciertas versiones avanzadas del protocolo HTTP se permitió el uso de conexiones persistentes (de hecho son usadas en este proyecto para limitar las peticiones de la imagen a una única petición) pero el protocolo en sí es

[10] WiFi, http://ftp1.digi.com/support/documentation/0190170_b.pdf

un protocolo que sigue el esquema de petición y respuesta. Entre las peticiones existentes en el protocolo, cabe destacar las siguientes:

Petición	Descripción	Ejemplo
GET	Pide un recurso contenido en el servidor: pide la imagen logo.png utilizando el protocolo HTTP/1.1	GET /images/logo.png HTTP/1.1
HEAD	Idéntico a GET solo que se recibe el metadata pero no el cuerpo del recurso: en este caso recibiría toda la información acerca de la fecha, versión, marca temporal, etc.	HEAD /images/logo.png HTTP/1.1
POST	Envía datos contenidos en el cuerpo de la petición. Si el servidor acepta la recepción de esos datos, los tratará como es debido.	POST /path/script.cgi HTTP/1.0

Tabla 1 Peticiones HTTP más utilizadas

En el caso de nuestro proyecto, sólo se utiliza el método GET junto con numerosas opciones para aumentar la eficacia de la transmisión de datos así como para facilitar la recepción de éstos por parte del cliente. Se detallará en el capítulo siguiente en dónde se explica todo el funcionamiento de la solicitud de las imágenes por parte del front-end.

5.2.2 TCP frente a UDP

Transfer Control Protocol (TCP) es un protocolo que actúa en la capa de transporte según el modelo OSI. Permite crear conexiones entre cliente y servidor de cara a garantizar que los paquetes entregados son recibidos correctamente gracias al modelo de envío de ACKs (Acknowledges). Sin entrar en mucho detalle, éste protocolo es el encargado de realizar la conexión en un primer paso entre cliente y servidor, de gestionar los números de secuencias de los paquetes para una ordenación si es necesaria, de detectar paquetes perdidos, detectar errores, y para controlar el flujo de datos. Por otro lado, User Datagram Protocol (UDP) no realiza conexiones previas, no realiza control de flujo, y es más ligero que TCP. En un principio se puede pensar (y sería correcto) que para una aplicación de recepción de vídeo lo más recomendable es utilizar UDP. De hecho, UDP se utiliza para aplicaciones de trasmisión de vídeo y audio en tiempo real, en dónde no se debe perder tiempo en mandar de nuevo paquetes perdidos, realizar tareas de control de llegada de paquetes, ordenación, etc. A parte de usarse en aplicaciones de *streaming* de vídeo, también se utiliza en resolución de nombres de dominio por Domain Name System (DNS) y otas aplicaciones de administración de redes o enrutamiento.

Sin embargo, cabe recordar que el subsistema de back-end pone en marcha un servidor web, al que se accede mediante el protocolo de capa de aplicación HTTP, el cuál es normalmente utilizado sobre TCP/IP[11]. Es el subsistema de front-end el que solicita un recurso que se encuentra en la página web, por lo tanto dicha petición se realiza por TCP para garantizar la llegada de la información. Además el servidor web está mostrando las imágenes que se captan desde la cámara en la página y éstas tienen un tamaño realmente pequeño, por lo que utilizar el protocolo TCP no resulta problemático.

[11] RFC 2616, http://www.ietf.org/rfc/rfc2616.txt

5.2.3 Estructura de la trama transmitida

Con todas las consideraciones anteriores, se puede obtener una estructura general de la trama que se transmitirá desde el subsistema back-end hacia el subsistema front-end. Dicho paquete variará en longitud dependiendo de la cantidad de bytes de datos de imagen que contenga.

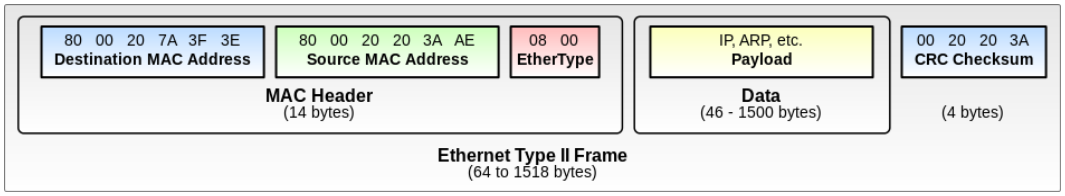

Figura 15 Estructura de la trama transmitida [12]

Como se puede comprobar en la imagen, obviando toda la información de cabeceras las cuáles no son importantes para el proyecto, el contenedor de datos que manejará el subsistema front-end es el de color amarillo, de 1500 bytes de longitud máxima. A ese tamaño, hay que restarle las cabeceras TCP de 40 bytes, teniendo así 1460 bytes de datos puros por paquete (tamaño que se utilizará en el front-end para determinar la longitud del buffer de recepción).

[12] Trama Ethernet, https://en.wikipedia.org/wiki/Ethernet_frame

Capítulo 6. Subsistema front-end basado en Arduino

Una vez están las imágenes disponibles y accesibles mediante un servidor web, será necesario programar en el subsistema front-end el firmware utilizado, NodeMCU, programable desde el entorno estándar de Arduino, para que sea capaz de recibir dichas imágenes y representarlas en el display SSD1331 al que está conectado. En este capítulo se explicará tanto la parte de recepción de los datos como la representación de éstos en el display. Para ello se introducirá en primer lugar el protocolo de compresión de imágenes JPEG, para entender posteriormente cómo se gestiona la recepción de los datos que únicamente contienen información acerca de la imagen, para finalmente explicar cómo se representa cada uno de los píxeles en el display.

6.1 Conceptos básicos del protocolo JPEG

Joint Photographic Experts Group (JPEG) es un estándar de compresión con pérdidas de imágenes ampliamente utilizado. El algoritmo de compresión utilizado se aprovecha de las características del ojo humano: una sensibilidad más alta a los cambios de luminancia que de crominancia, es decir, una sensibilidad más alta a cambios de brillo que de color.

Se aplica en primer lugar una transformación del espacio en el que se trabaja. Las imágenes tendrán por cada píxel un número de bits asignados para definir el color según el esquema RGB (Red, Green y Blue). Por ejemplo, una imagen con 24 bits por píxel tendrá 8 bits para cada color, generando el color final con la combinación de los tres. Tal y como se ha mencionado anteriormente, al tener el ojo humano mayor sensibilidad por los cambios de brillo, se aplica una transformación del espacio RGB al espacio YUV (Luminancia, Crominancia Azul y Crominancia Rojo). Las ecuaciones utilizadas para la transformación de cada píxel al nuevo espacio son las siguientes:

$$Y = 0.257 * R + 0.504 * G + 0.098 * B + 16 \ (1)$$

$$Cb = U = \ -0.148 * R - 0.291 * G + 0.439 * B + 128 \ (2)$$

$$Cr = V = \ 0.439 * R - 0.368 * G - 0.071 * B + 128 \ (3)$$

Tras ello, se puede aplicar un submuestreo de cara a reducir la cantidad de información referente al color. El submuestreo denominado como 4:4:4 no modifica la información con la que se trabaja. El submuestreo 4:2:2 reduce a la mitad la información sobre crominancia (se puede entender como de cada 4 píxeles, se escoge 4 valores de luminancia, 2 de crominancia azul, y 2 de crominancia rojo).

Finalmente, se aplica una Transformación Discreta del Coseno (DCT) en la cual en primer lugar se divide la imagen en bloques de 8x8 píxeles, se le resta 128 a cada valor, y cada bloque se convierte al dominio de la frecuencia aplicando una DCT y un redondeo. Se obtiene entonces matrices de 8x8 representando los valores obtenidos tras la DCT, que serán posteriormente cuantificados (si se desea) y codificados mediante una codificación entrópica tomando elementos de la matriz en zig-zag.

El proceso de decodificación es el mismo pero viceversa: se decodifica, se deshace la cuantificación si se ha realizado ,se deshace la transformación DCT, se suma 128 a cada valor y se obtiene un conjuntos de bloques de 8x8 que difieren de los obtenidos previamente debido a los redondeos y la cuantificación.

Al codificar la imagen, para que un software de edición de imágenes o de representación pueda decodificar la imagen JPEG correctamente, el estándar JPEG genera un conjunto de

metadatos al principio y al final de la imagen comprimida de manera que se pueda descomprimir correctamente. Al haber un gran número de variantes en las que realizar la compresión JPEG, toda esa información acerca de la cuantificación realizada, tablas de Huffman, etc. debe ir incluida en el propio fichero[13]. Tal y como se verá en el apartado siguiente, parte de los metadatos servirán para delimitar el comienzo y el final de la información de la imagen dentro de la inmensa cantidad de cabeceras que se utilizan para los diferentes protocolos a la hora de transmitir datos desde un servidor web a un cliente.

6.2 Recepción de datos desde el firmware basado en Arduino

Una vez confirmada la estructura de los datos a recibir, será necesario encontrar la forma de delimitar el contenido de la información útil y descartar todo aquello que no contiene datos sobre la imagen en sí a representar. Para ello se utilizará la estructura del fichero JPEG. El formato de fichero JPEG utiliza una serie de marcadores para delimitar las diferentes zonas de los ficheros indicando si se trata de la tabla de Huffman para decodificar, si hay algún comentario, etc. En particular, utiliza dos marcadores para indicar el inicio y el fin del fichero. Dichos marcadores tienen como valores en hexadecimal 0xFFD8 y 0xFFD9 respectivamente. Así pues, a la hora de recibir los datos por parte del servidor web, los marcadores indicarán el principio y el final de la información útil.

El firmware utilizado integra el socket WiFi ESP8266 para gestionar las conexiones inalámbricas vía WiFi. Además, dada su inmensa comunidad de programadores, dispone de una enorme cantidad de tutoriales para desempeñar la función que el usuario desea, entre ellas, conectarse a una red WiFi, almacenar los datos recibidos, etc. Por otro lado, cabe destacar que la cámara toma 9 imágenes por segundo, lo que significa que para que la representación de la imagen sea lo más fidedigna posible, se dispone de 111 ms para recibir el paquete de una imagen, decodificarlo, y representarlo.

La recepción de los datos se realiza de manera sencilla. Básicamente hay que incluir la librería correspondiente para conectarse a una red WiFi, instanciar el objeto e iniciar la conexión a dicha red. Una vez conectado a la red, se deberá conectar a la dirección IP (la dirección IP de la Raspberry Pi) y puerto dónde está lanzado el servidor web, y una vez conectado se realiza la petición GET por HTTP. Dentro del bucle propio de la programación en Arduino, se instancia un buffer dónde se almacenarán los datos recibidos siempre y cuando éstos se reciban y sólo si vienen a continuación de un valor en hexadecimal igual a 0xFFD8 y se finalizará la recepción cuando se reciba un valor en hexadecimal igual a 0xFFD9 (correspondientes al inicio y fin de la imagen respectivamente). Dicho buffer de datos se tratarán gracias al uso de librerías para decodificación de ficheros JPEG tal y como se especifica en la sección siguiente. Para una explicación detallada del código en cuestión, véase en la sección de anexos el anexo I.

6.3 Representación de la información en el display SSD1331

Una vez se obtiene el buffer con toda la información referente a la imagen, estará listo para usarse junto con las librerías de decodificación de imágenes JPEG. La decodificación de las imágenes en formato JPEG se realiza tal y como se ha explicado en el apartado anterior, haciendo el proceso inverso a la codificación. Para ello, por cada imagen recibida, se llama a la función de decodificación de ficheros JPEG con el array con los datos y el tamaño del array. Hecho esto, si la decodificación ha sido realizada correctamente, se llama a la función de renderizado de la imagen que básicamente coge uno a uno los bloques de la imagen y los representa en el display. Para una explicación detallada del código en cuestión, véase en la sección de anexos el anexo II.

[13] JPEG, http://ieeexplore.ieee.org/document/7924246/?reload=true

Estas librerías se han obtenido de un repositorio de GitHub del usuario *Bodmer*[14] que se basa en el código de *Makoto Kurauchi*[15].

6.4 Gestión del tiempo entre imágenes

Tal y como se explicó anteriormente, se disponía de un tiempo de procesado de la imagen de 111 ms, en los cuales se debía hacer el proceso de recepción de datos así como la decodificación y posteriormente la representación en el display. El proceso más costoso temporalmente hablando es el de decodificación de la imagen debido a la enorme cantidad de operaciones matemáticas que se realizan. A su vez, se le suman tiempos en la comunicación entre cliente y servidor, los cuales provocaron numerosos problemas de latencia en las versiones iniciales del proyecto. En este apartado, se explicará qué problemas surgieron a la hora de gestionar el tiempo disponible y cómo se solventaron gracias a la herramienta de monitorización de tráfico Wireshark.

En las versiones iniciales del proyecto, utilizando las librerías estándar, la recepción de datos se hacía byte a byte lo que aumentaba considerablemente el tiempo de recepción de datos. Además, tal y como funciona la librería interna que gestiona las transmisiones TCP entre cliente y servidor, el ACK del paquete sólo se manda cuando se recoge el último byte del paquete incluido en el buffer de recepción. Esto provocaba en ocasiones que el ACK del paquete número uno se mandara después de que el paquete número 2 estuviera disponible para recepción. En versiones posteriores se solucionó este problema con un código más eficiente.

El segundo problema que surgió estaba relacionado con la cantidad de tiempo que se utilizaba para la recepción de los datos. Aun realizando la recepción byte a byte, se consumía demasiado tiempo en la recepción de los datos y se disponía de muy poco tiempo para el procesado de la imagen, lo que provocaba que para imágenes con poca tasa de compresión (por lo tanto, de mayor tamaño) el dispositivo funcionara con mucho retraso. Tal comportamiento se puede observar gracias a la siguiente captura de tráfico:

No.	Time	Source	Destination	Protocol	Length	Delta Time	Info
237	8.4240…	192.168.1.35	192.168.1.33	TCP	127	0.002158	[TCP segment of a reassembled PDU]
238	8.4243…	192.168.1.35	192.168.1.33	TCP	1514	0.000248	[TCP segment of a reassembled PDU]
239	8.4243…	192.168.1.33	192.168.1.35	TCP	54	0.000057	51269 → 8080 [ACK] Seq=392 Ack=1869 Win=261…
240	8.4245…	192.168.1.35	192.168.1.33	TCP	1514	0.000193	[TCP segment of a reassembled PDU]
241	8.4245…	192.168.1.35	192.168.1.33	TCP	1514	0.000001	[TCP segment of a reassembled PDU]
242	8.4246…	192.168.1.33	192.168.1.35	TCP	54	0.000043	51269 → 8080 [ACK] Seq=392 Ack=4789 Win=261…
243	8.4247…	192.168.1.35	192.168.1.33	TCP	1514	0.000187	[TCP segment of a reassembled PDU]
244	8.4250…	192.168.1.35	192.168.1.33	TCP	1003	0.000204	[TCP segment of a reassembled PDU]
245	8.4250…	192.168.1.33	192.168.1.35	TCP	54	0.000043	51269 → 8080 [ACK] Seq=392 Ack=7198 Win=261…
246	8.5362…	192.168.1.35	192.168.1.33	TCP	127	0.111236	[TCP segment of a reassembled PDU]

Figura 16 Tráfico entre cliente y servidor para una imagen sin compresión

```
========================================
start new image detected
*******image COMPLETED. Go to decode it.
*******DECODE. Go to Render it.
Total render time was    : 85 ms
```

Figura 17 Tiempo de decodificación de la imagen sin compresión

[14] https://github.com/Bodmer/JPEGDecoder
[15] https://github.com/MakotoKurauchi/JPEGDecoder

Tal y como se observa en la **Figura 14**, el Delta Time (tiempo de diferencia entre el paquete transmitido o recibido anterior y éste) del primer paquete TCP recibido de 127 bytes es de 0.002158 segundos, es decir, 2 ms. Este paquete de 127 bytes es la respuesta del servidor tras la solicitud de la imagen por parte del cliente. El paquete siguiente de 1514 bytes contiene los primeros bytes del fichero JPEG (imagen *x*). Esto significa que el ACK correspondiente a la imagen *x* se envía muy poco antes de recibir la siguiente imagen *x+1*, lo que significa que se empezó a procesar la imagen *x* con mucho retraso. A esto, se le suma un tiempo de decodificación de 85 ms tal y como muestra la **Figura 15**. El resultado final es un procesado por parte del subsistema de back-end muy retrasado temporalmente hablando en comparación con la imagen que debería estar procesando. Para imágenes con mucha tasa de compresión, este fenómeno desaparecía puesto que la cantidad de bytes a recibir era prácticamente la tercera parte y se reducía en 20 ms el tiempo de decodificación. En este caso, el subsistema front-end disponía de tiempo suficiente para todas las tareas necesarias antes de la representación en el display, tal y como se muestra en las figuras siguientes:

No.	Time	Source	Destination	Protocol	Length	Delta Time	Info
212	4.2277…	192.168.1.33	192.168.1.35	TCP	54	0.000071	51463 → 8080 [ACK] Seq=392 Ack=87469 Win=26…
213	4.3377…	192.168.1.35	192.168.1.33	TCP	127	0.109968	[TCP segment of a reassembled PDU]
214	4.3380…	192.168.1.35	192.168.1.33	TCP	1514	0.000260	[TCP segment of a reassembled PDU]
215	4.3381…	192.168.1.33	192.168.1.35	TCP	54	0.000097	51463 → 8080 [ACK] Seq=392 Ack=89002 Win=26…
216	4.3383…	192.168.1.35	192.168.1.33	TCP	1514	0.000214	[TCP segment of a reassembled PDU]
217	4.3386…	192.168.1.35	192.168.1.33	TCP	859	0.000283	[TCP segment of a reassembled PDU]
218	4.3386…	192.168.1.33	192.168.1.35	TCP	54	0.000072	51463 → 8080 [ACK] Seq=392 Ack=91267 Win=26…
219	4.4488…	192.168.1.35	192.168.1.33	TCP	127	0.110136	[TCP segment of a reassembled PDU]
220	4.4490…	192.168.1.35	192.168.1.33	TCP	1514	0.000261	[TCP segment of a reassembled PDU]
221	4.4491…	192.168.1.33	192.168.1.35	TCP	54	0.000059	51463 → 8080 [ACK] Seq=392 Ack=92800 Win=26…

Figura 18 Tráfico entre cliente y servidor para una imagen con mucha compresión

```
=======================================
start new image detected
*******image COMPLETED. Go to decode it.
*******DECODE. Go to Render it.
Total render time was    : 65 ms
```

Figura 19 Tiempo de decodificación de la imagen con mucha compresión

Para poder representar imágenes con poca tasa de compresión y así tener una mejor calidad de imagen, se utilizó una lectura del buffer de recepción por bloques de 1460 bytes. 1460 bytes es el tamaño máximo de *payload* (datos) de un paquete TCP (Maximum Segment Size de un Maximum Tranfer Unit de Ethernet v2). Por lo tanto, se realizaba una lectura paquete a paquete. Este método reducía enormemente los tiempos de recepción de datos, por lo tanto mandaba mucho antes los ACKs de los primeros paquetes de la imagen, lo que se traduce en una recepción más rápida de los paquetes siguientes, y por lo tanto de toda la imagen. Se disponía por lo tanto de más tiempo para decodificar la imagen y representarla en el display.

Capítulo 7. Diseño de los soportes mediante FreeCAD

Una vez realizada toda la programación de los subsistemas de front-end y back-end, se procederá a construir un conjunto de soportes para ambas partes. Para ello, se diseñará un conjunto de piezas para dar soporte físico a la Raspberry Pi, su batería y la cámara Lepton por un lado y para el microcontrolador NodeMCU, su batería, y el display SSD1331 por otro lado. Para ello, se utilizará un programa de diseño de formas 3D. En este capítulo se explicará el proceso de diseño.

7.1 Entorno de diseño

El programa utilizado es un programa llamado FreeCAD[16] que ofrece la posibilidad de exportar los bocetos 3D en formato *.stl* capaz de ser interpretado por una impresora 3D. Standard Triangle Language (STL) es un formato de archivo de diseño asistido por ordenador que define la geometría de objetos 3D gracias a un conjunto de triángulos. Las formas están incluidas en ficheros con este formato que son leídos por una impresora de extrusión de plástico termofusible que modela las formas previamente dibujadas. De esta forma, se puede construir de manera rápida y sencilla los soportes necesarios para el proyecto. A continuación se va a explicar cuáles han sido de todas las opciones que el programa ofrece aquellas que se han utilizado para el diseño de los soportes.

7.1.1 Opciones de Sketcher y Draft

Gracias a esta utilidad se puede diseñar mediante un conjunto de puntos o gracias al uso de formas ya predefinidas la forma que se va a utilizar. Esta herramienta es extremadamente útil para definir restricciones geométricas o dimensionales (mantener perpendicularidad entre dos rectas, mantener una distancia entre dos rectas igual a un valor, restringir el movimiento de una forma, etc.). Se utilizará enormemente en el diseño del subsistema back-end puesto que se ha utilizado un moldeador de formas y papel milimetrado para extraer el contorno del casco y así poder adaptarse lo máximo posible a su geometría.

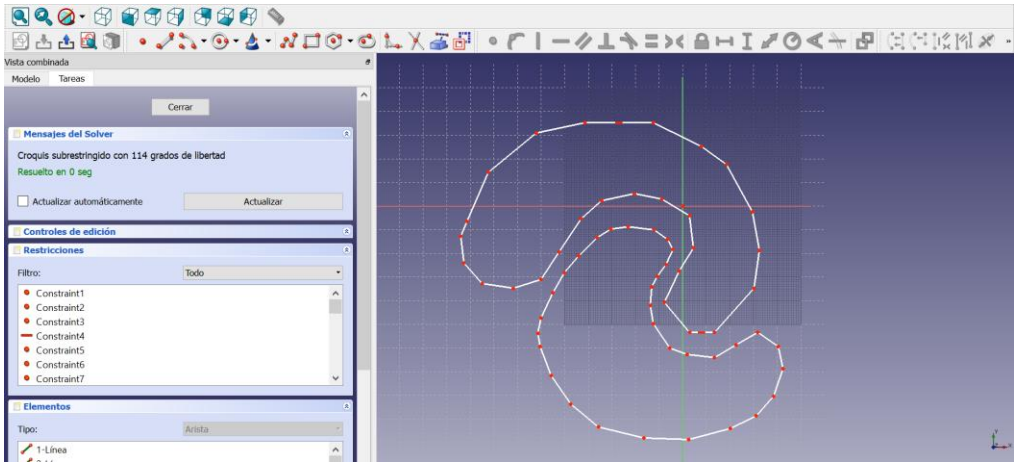

Figura 20 Sketcher de FreeCAD

7.1.2 Opción de Part

Con esta herramienta se realiza todo el modelado de las piezas 3D. Se puede construir rectángulos, cilindros, esferas, toroides, etc. para luego realizar operaciones entre ellos. Las operaciones más utilizadas y útiles son las operaciones booleanas entre formas. Permite realizar cortes, uniones o intersecciones de dos formas, lo que resulta indispensable a la hora de generar

[16] FreeCAD, https://www.freecadweb.org/?lang=es_ES

la forma final para el fichero de impresión. La función de *Extruir* será también fundamental en el diseño de los soportes puesto que permite, utilizando una forma diseñada en el *Sketcher* como "raíl", extender una forma diseñada en *Part* a lo largo de ese "raíl" modificando así su estructura. Se utilizará para el diseño de las roscas y los contornos del casco.

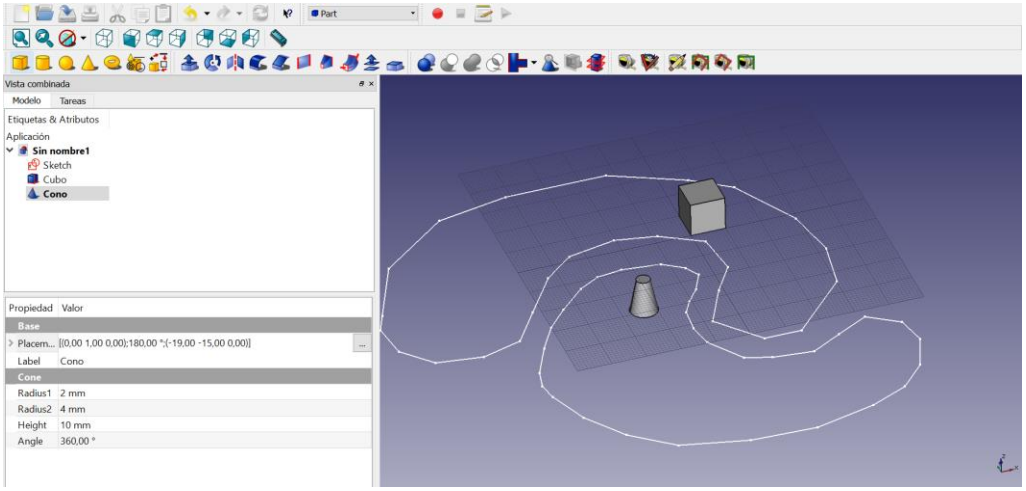

Figura 21 Part de FreeCAD

7.2 Soporte para el subsistema front-end

Tal y como se introdujo en este documento, el soporte para este subsistema consta de dos partes fundamentales: el soporte o brazo general que realiza la sujeción al frontal del casco y la pareja tuerca macho y tuerca hembra que permite la variación de la distancia focal en función de la altura de la lente para dotar al proyecto de un cierto grado de personalización en función de las dioptrías que requiera el usuario. Para realizar todo el soporte, en primer lugar se construyó un soporte a base de arcilla polimérica. Es un tipo de arcilla totalmente moldeable que al someterse a altas temperaturas (se suele hornear) mantiene su forma y se convierte en un objeto totalmente sólido. A partir de ese prototipo muy básico y observando que se adaptada a las necesidades del soporte, se comenzó a diseñar la pieza.

Las dos partes del soporte son las que se muestran en la siguiente figura:

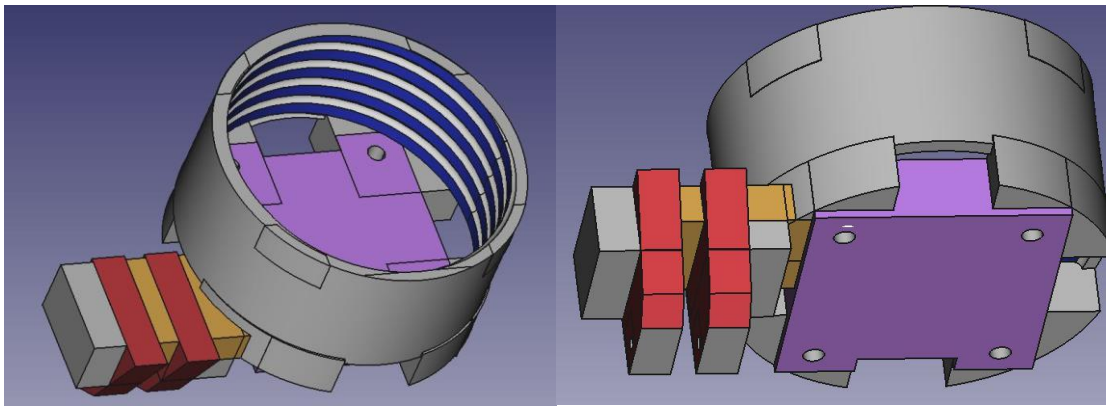

Figura 22 Brazo front-end

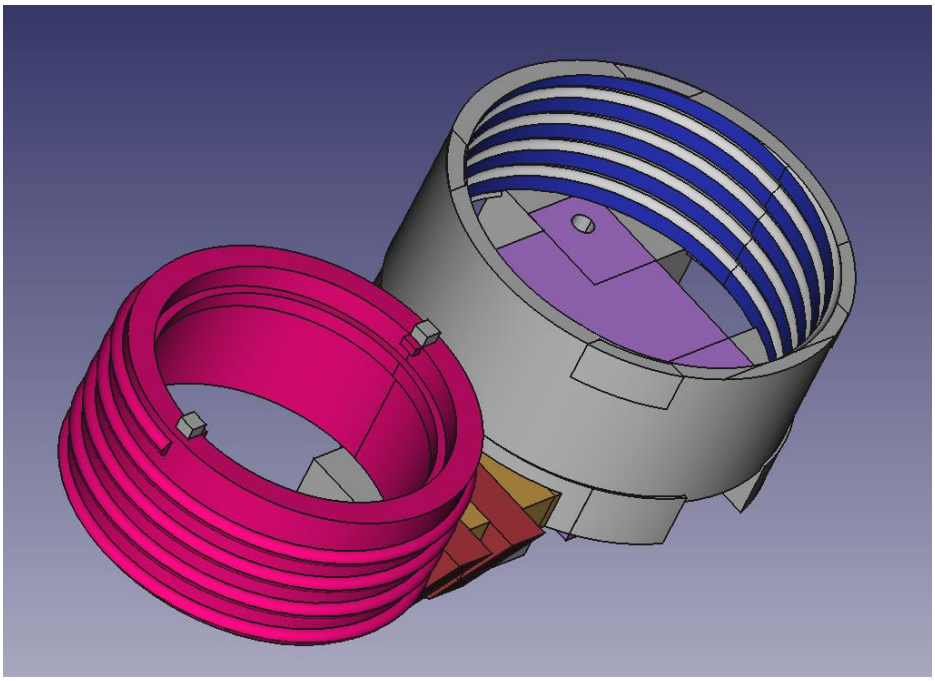

Figura 23 Juego de tuercas

Para realizar el brazo, se han usado las opciones contenidas en *Part* para la unión y resta de formas, creando así un solo brazo con orificios para tornillos incluyendo el soporte dónde reposara el display con sus correspondientes agujeros para la fijación de éste. El diseño de los dos brazos rojos se ha realizado usando la opción de *Sketcher* puesto que requería que la forma se adaptara totalmente a la geometría de la superficie de la máscara para que el display no estuviera torcido respecto a la horizontal. Para la realización del juego de tuercas se dibujó con *Draft* una hélice por la cual se extruyó un disco sólido. Uniéndose con un cilindro, generaba la forma de un tornillo. Bastaba con vaciar en su interior formando así la forma de color rosa. Su correspondiente forma hembra se diseñó gracias a la resta entre un cilindro de radio mayor y la tuerca previa.

7.3 Soporte para el subsistema back-end

El diseño de este soporte es bastante complejo. No sólo es la parte más visual del proyecto, sino que además se tiene que adaptar al contorno del casco e incluir en su interior cuatro piezas fundamentales conectadas físicamente entre sí. Este soporte contiene en su interior la Raspberry Pi Zero, su batería, el chip encargado de la gestión de la carga de ésta y la cámara Lepton. Había que asegurarse que todo estuviera sujeto en su interior, ocupando el mínimo tamaño posible, permitiendo que las conexiones eléctricas se realizaran con comodidad, otorgando a la cámara una visión totalmente horizontal y por último que se adaptara bien a la superficie del casco. Para ello, se realizó un scanner 3D del casco de bomberos gracias a un dispositivo llamado *Structure Sensor 3D*, para iPad[17]. Permite escanear objetos y obtener un fichero aceptado por programas de diseño y modelado 3D, como lo es FreeCAD. Una vez el fichero dentro de FreeCAD, hay que realizar una serie de acciones para limpiar la malla del objeto, convertirla en objeto solido 3D y luego realizarle un escalado a la dimensión real para poder usar el molde del objeto en la opción *Part* que se ha comentado anteriormente. El resultado es el siguiente:

[17] Structure 3D para iPad, https://structure.io/

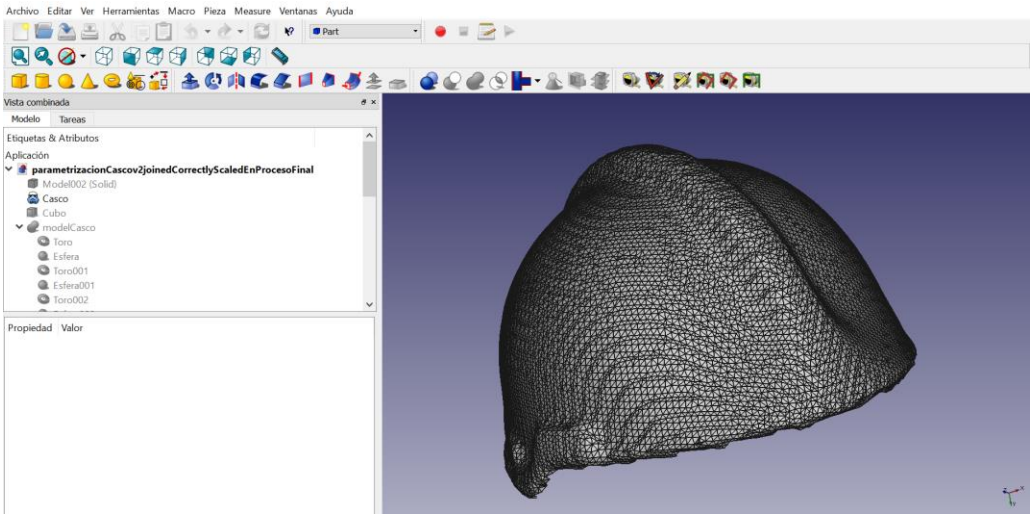

Figura 24 Scanner 3D del casco de bomberos en FreeCAD

La idea principal era utilizar este moldeado del casco para restárselo al soporte realizado de manera que los contornos de la caja se amoldaran a la perfección a la superficie del casco. Sin embargo, el escaneo del casco no se lograba hacer correctamente, por lo que el programa se cerraba a la hora de hacer operaciones booleanas con el molde. Esto es debido a que si no se escanea correctamente la superficie del objeto, pueden quedar ciertas zonas muy pequeñas que no están conectadas con el resto de la superficie, y esta condición es indispensable para que el programa pueda hacer operaciones booleanas. Tras numerosos intentos fallidos escaneando el casco, se utilizó el molde de éste únicamente para realizar comprobaciones de tamaño del soporte respecto al casco.

Aún quedaba por obtener los contornos del casco para hacer que el soporte se adaptara lo máximo posible a su contorno. Para ello, se utilizó una plantilla de formas: es una herramienta utilizada sobre todo en diseño y en albañilería para obtener contornos arbitrarios y poder cortar los azulejos en función de estos y hacer que se adapten lo máximo posible.

Figura 25 Plantilla de formas

Obteniendo los contornos necesarios del casco de bomberos, se traspasaban a papel milimetrado, y con precisión se dibujaba el contorno obtenido en la opción *Draft* de FreeCAD. Una vez dibujada la forma, se le otorgaba grosor y se obtenía un objeto sólido 3D capaz de interactuar en la opción *Part* para poder realizar operaciones booleanas con él.

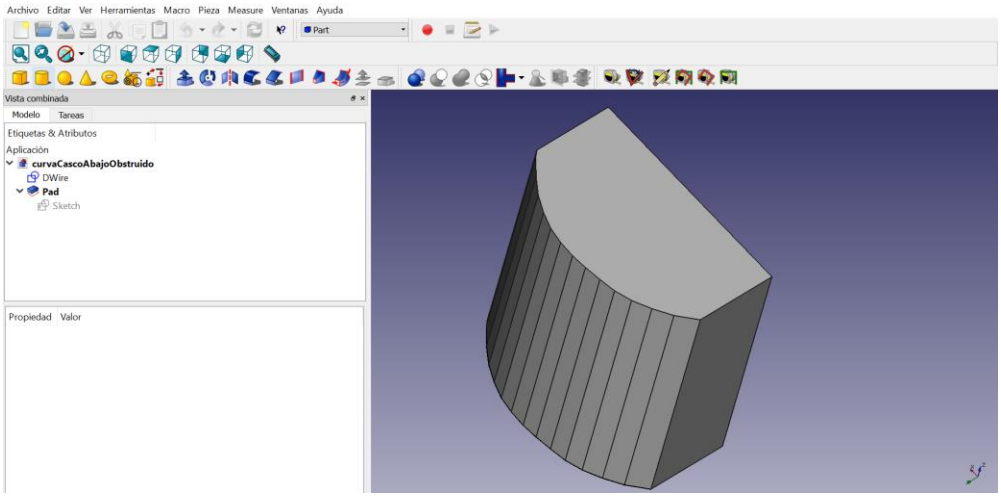

Figura 26 Contorno de la parte inferior del casco de bomberos

Realizando las mismas operaciones con el contorno superior y los laterales, y habiendo diseñado la caja anteriormente, se obtiene el soporte adaptándose a la estructura del casco perfectamente:

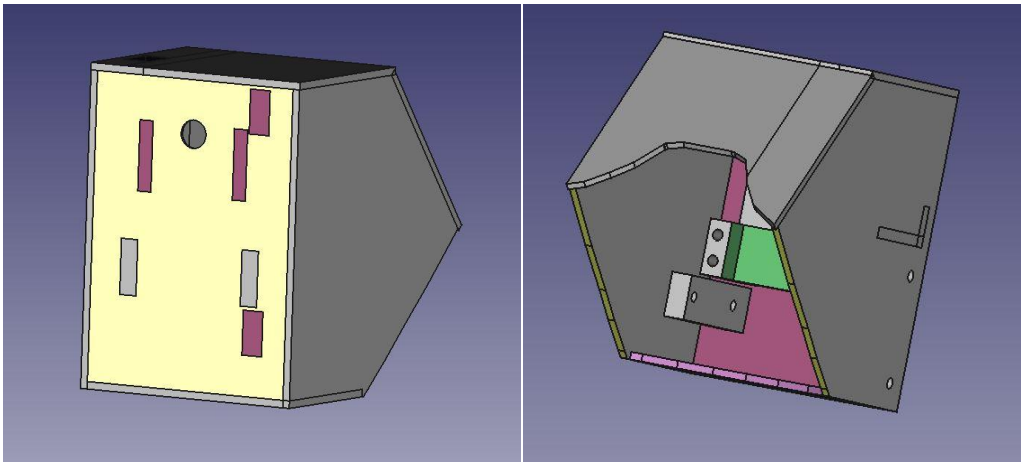

Figura 27 Visión general del soporte

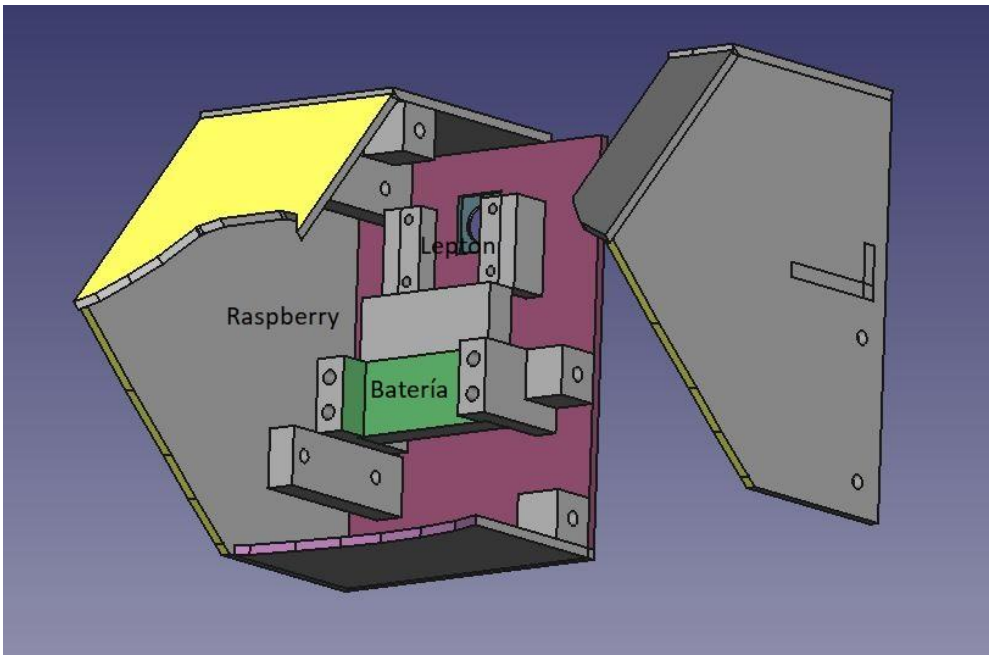

Figura 28 Interior del soporte abierto

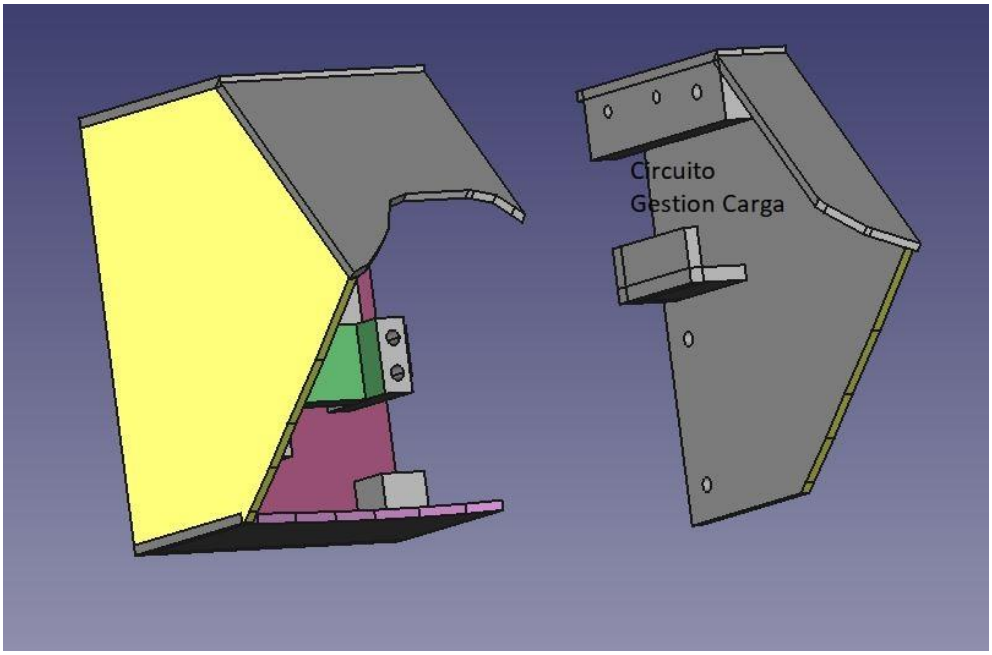

Figura 29 Tapa lateral del soporte

Capítulo 8. Coste temporal y económico del proyecto

En este capítulo se recogerá el tiempo destinado a cada una de las partes del proyecto, incluyendo en éstas tareas de investigación, realización, testeo, y programación. A su vez, debido a que se trata de un dispositivo que como objetivo tiene solventar las dificultades de las labores de rescate del cuerpo de bomberos a un precio asequible y competitivo, se realizará un resumen de presupuestos indicando el coste final del producto desglosando en cada uno de los subsistemas.

8.1 Desglose temporal del proyecto

El tiempo destinado a cada una de las tareas se muestra a continuación:

Tarea	**Mes**							
	02/17	**03/17**	**04/17**	**05/17**	**06/17**	**07/17**	**08/17**	**09/17**
Búsqueda de información acerca del estado del arte en lo referente al proyecto	20 horas	5 horas						
Introducción al entorno Arduino y recepción de datos por WiFi	30 horas	20 horas						
Búsqueda de soporte para NodeMCU y librerías para comunicación con el display SSD1331	40 horas	20 horas						
Realización del prototipo del soporte mediante arcilla polimérica	4 horas	4 horas						
Testeo de recepción de datos y funcionamiento de MJPG-Streamer		15 horas	15 horas	15 horas				
Programación de la primera versión de front-end			30 horas	30 horas				

Diseño 3D del prototipo del soporte		25 horas	10 horas					
Investigación y testeo sobre protocolos de comunicación			10 horas	5 horas	10 horas			
Programación de la versión final de front-end				30 horas	15 horas			
Diseño 3D final del soporte					25 horas		20 horas	
Redacción de la memoria							50 horas	10 horas
Preparación de la presentación							15 horas	20 horas

Tabla 2 Organización temporal del proyecto

8.2 Coste económico del proyecto

El coste económico del proyecto queda reflejado en la siguiente tabla:

Dispositivo	Precio (€, $)	Proveedor
Raspberry Pi Zero	5.50 €	Kubii, https://www.kubii.fr/es/raspberry-pi-3-2-b/1401-raspberry-pi-zero--3272496003408.html
Cámara Lepton FLiR	260 €	8.2.1.1.1.1.1.1.1 Sparkfun, https://www.sparkfun.com/products/13233
NodeMCU	8.46 €	Mouser, http://www.mouser.es/ProductDetail/Adafruit/2471/?qs=GURawfaeGuAoxPWz4nSJYg%3d%3d
SSD1331 Display	13 €	8.2.1.1.1.1.1.1.2 Bangood, https://www.banggood.com/es/0_95-Inch-7pin-Full-Color-65K-Color-SSD1331-SPI-OLED-Display-For-Arduino-p-1068167.html
Batería front-end	9.50 €	Amazon, https://www.amazon.es/Bateria-conector-Avion-Helicoptero-4138/dp/B0187HVOOI/ref=sr_1_5?s=electronics&ie=UTF8&qid=1504017085&sr=1-5&keywords=3.7V+1200mAh+25C+Lipo+Battery
Batería back-end	2.97 €	DX, http://www.dx.com/es/p/hubsan-h107c-a24-3-7v-380mah-li-po-battery-for-h301c-r-c-quadcopter-white-

		243226?tc=EUR&gclid=Cj0KCQjwoZTNBRCWARIsAOMZHmFZfIWiXL0F8-XzgCVK7hY-oEeO4MxaIGyZDy00mU7-r-nXg8EGAYUaAreeEALw_wcB#.WaV6b8irQ2w
Cables	5 €	/
Total	304.43 €	

Tabla 3 Coste económico del proyecto

Asumiendo que la mano de obra puede costar unos 40€ contando con costes de mobiliario de la empresa, servicios generales, etc. el coste del diseño del producto asciende a aproximadamente 19.720 €. Dicho coste se dividirá por el número de dispositivos creados en fabricación para añadirse al coste económico material. Por ejemplo, una producción de 1000 dispositivos (suponiendo que el coste del material es el coste por unidad) hará que el coste por producto sea de:

$$Coste = \frac{19720}{1000} + 304.43 = 324.15\ € \quad (4)$$

Conclusiones

Se ha conseguido realizar un prototipo demostrador para dotar a los agentes de bomberos que desempeñan su labor de extinción y rescate en entornos saturados de humo donde no hay visibilidad de un sistema de realidad aumentada que les permite tener disponible en su campo de visión la imagen que les proporciona una cámara térmica. La radiación térmica sí que se propaga a través del humo y esto les permite ver si hay personas tras el humo y donde están los focos de calor que es donde deben centrar sus labores de extinción.

La realización del prototipo ha requerido la utilización tanto de conocimientos de electrónica, telemática y comunicaciones para poder implementar el dispositivo con la cámara térmica, la compresión y transmisión de la imagen con los protocolos adecuados y la realización del equipo receptor con su microcontrolador y display. Un punto especialmente crítico ha sido la optimización del código en el microcontrolador para que pueda decodificar la secuencia de imágenes comprimidas en tiempo real.

También se ha realizado con un programa CAD el soporte suplementario al casco de bomberos para albergar la cámara, la CPU, la batería y su gestor de carga. Para la visualización se ha realizado el soporte a integrar en el interior de la máscara de oxígeno que alberga el microcontrolador, el display y el sistema de visualización.

Líneas futuras

Las líneas futuras van a ser mejorar el sistema a medida que el mercado vaya ofreciendo precios comedidos en las versiones de mejores prestaciones de los componentes de forma que el precio final sea asumible por los cuerpos de bomberos. La cámara que estamos utilizando cumple las condiciones de un precio moderado y un tamaño miniatura pero su resolución es de 80x60. Como punto de partida supone una aportación muy grande porque supone tener visión con manos libres, pero en cuanto haya posibilidad de aumentar la resolución se hará porque es una prestación que aporta mucha calidad al sistema.

Lo mismo ocurrirá con el display y junto con ambos dispositivos se incorporarán las siguientes generaciones de procesadores para poder comprimir, descomprimir y gestionar la visualización de imágenes de más resolución.

En otro orden de cosas el proyecto también va a seguir avanzando en la parte del interface de gestión. El sistema de realidad aumentada abre la puerta a que el agente disponga de información valiosa durante la operación, tanto de monitorización de variables como pueden ser la cantidad de aire disponible en las botellas, información de sensores de presencia de gases, etc. como de planos de la zona del edificio que pueden ser presentados a medida que se adentra en él. Para gestionar todo ello se está desarrollando un interface que permita de forma ágil interactuar con el sistema sin que le suponga un estorbo durante su operación.

Bibliografía

[1] Protocolo de actuación de los Bomberos de Navarra, http://www.bomberosdenavarra.com/documentos/ficheros_documentos/intervencion.pdf

[2] Head-Up Displays, https://en.wikipedia.org/wiki/Head-up_display

[3] MJPG-Streamer, hilos oficiales, https://sourceforge.net/projects/mjpg-streamer/ , https://github.com/jacksonliam/mjpg-streamer

[4] Dificil aceptación de los dispositivos de Realidad Virtual y Aumentada, http://www.elpublicista.es/mundo-online/aceptacion-social-realidad-virtual-aumentada-dependera-autonomia

[5] Gafas Moverio, https://www.epson.es/products/see-through-mobile-viewer/moverio-bt-200

[6] Distancia focal, https://en.wikipedia.org/wiki/Focal_length

[7] Características de la Raspberry Pi Zero, https://www.raspberrypi.org/products/raspberry-pi-zero/

[8] Tutorial de conexiones y configuración de la Raspberry Pi y la cámara Lepton FLiR: https://groupgets.com/blog/posts/8-installation-guide-for-pure-breakout-board-on-raspberry-pi-2

[9] Computer Language Benchmarks Game: http://benchmarksgame.alioth.debian.org/u64q/compare.php?lang=python3&lang2=gpp

[10] WiFi, http://ftp1.digi.com/support/documentation/0190170_b.pdf

[11] RFC 2616, http://www.ietf.org/rfc/rfc2616.txt

[12] JPEG, http://ieeexplore.ieee.org/document/7924246/?reload=true

[13][14] Librerías de decodificación JPEG, https://github.com/Bodmer/JPEGDecoder , https://github.com/MakotoKurauchi/JPEGDecoder

[15] Trama Ethernet, https://en.wikipedia.org/wiki/Ethernet_frame

[16] FreeCAD, https://www.freecadweb.org/?lang=es_ES

[17] Structure 3D para iPad, https://structure.io/

Anexos

La sección de anexos contiene todos aquellos documentos, líneas de código, imágenes, y/o aclaraciones que no se han incluido en la memoria para no entorpecer una lectura continuada y amena de la misma.

Anexo I: Recepción de datos por WiFi en el firmware NodeMCU

Conexión a red WiFi y petición del recurso

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>

const char WIFI_SSID[] = "Raspi_wifi";
const char WIFI_PSK[] = "Uniii000";
const char http_site[] = "10.0.2.1";
const int http_port = 8080;

WiFiClient client;

void connectWiFi() {

  byte led_status = 0;

  // Set WiFi mode to station (client)
  WiFi.mode(WIFI_STA);

  // Initiate connection with SSID and PSK
  WiFi.begin(WIFI_SSID, WIFI_PSK);


  // Blink LED while we wait for WiFi connection
  while ( WiFi.status() != WL_CONNECTED ) {
    digitalWrite(LED_PIN, led_status);
    led_status ^= 0x01;
    delay(100);
  }
  delay(2000);
  // Turn LED on when we are connected
  digitalWrite(LED_PIN, HIGH);
}

bool requestPage() {
  // Connect to server
  tTime = millis();
  while(!client.connect(http_site, http_port)){
    Serial.println("Trying to conncect to server");
  }
  tTime = millis() - tTime;
  Serial.print("conncected to server ");
Serial.print(http_site);Serial.print(" after "); Serial.print(tTime);
Serial.println(" ms");

  // Settings
  client.setNoDelay(true);
  client.setTimeout(50);

  tTime = millis();
  client.print(requestPageCommand);
  tTime = millis() - tTime;
  Serial.print("Page requested in "); Serial.print(tTime);
Serial.println(" ms");
  return true;
}
```

```
void setup() {
  Serial.begin(57600);

  Serial.println("===============");
  Serial.println("Hola Setup JPEG ");
  Serial.println("===============");

  tft.begin();
  tft.setRotation(0);  // 0 & 2 Portrait. 1 & 3 landscape
  tft.fillScreen(TFT_BLACK);

  // Set up LED for debugging
  pinMode(LED_PIN, OUTPUT);

  // We connect to WiFi
  tTime = millis();
  connectWiFi();
  tTime = millis() - tTime;
  Serial.print("Connected to wifi ");
Serial.println(WIFI_SSID);Serial.print("   after ");
Serial.print(tTime); Serial.println(" ms");

  //We request the page:
  requestPageCommand = String("GET /?action=stream HTTP/1.1\r\nHost:
") + http_site + "\r\n" +
               "Connection: keep-alive\r\n\r\n";
  requestPage();
  Serial.println("First page requested");

  jpegBuffer = arrayJpeg;

} // End of setup()
```

Las primeras líneas de código corresponden a la instanciación de librerías y objetos necesarios para la conexión.

`void connectWiFi()` es una función que realiza la conexión a la red WiFi creada por la Raspberry Pi Zero. En ella se configura el objeto a modo cliente, se comienza la búsqueda de la red WiFi, y se intenta conectar hasta que lo consiga.

`bool requestPage()` es una función que solicita el recurso de la imagen tal y como dice la documentación de MJPG-Streamer gracias a `requestPageCommand`. Se conecta en primer lugar a la dirección IP y puerto del servidor web para luego realizar la petición mediante `client.print(requestPageCommand)`. Como se puede observar en el campo de `requestPageCommand`, se llama a un recurso que lanza un código en el back-end para que éste ofrezca sin pausa las imágenes obtenidas por la cámara Lepton. Se añade la opción `Connection: keep-alive` para evitar tener que abrir una conexión nueva cada vez que se recibe un recurso, en este caso, imágenes, y así evitar consumir más tiempo de los 111 ms disponibles por imagen.

Recepción de datos y almacenamiento en buffer

```
uint8_t arrayTcp[MAXSIZE_ARRAYTCP];
uint8_t arrayJpeg[MAXSIZE_ARRAYJPEG];
uint32_t arrayJpeg_currentSize;
uint8_t* tcpBuffer;
uint8_t* jpegBuffer;

void loop() {
  tcpBuffer = arrayTcp;
  if (client.available() > 0) {
    bytesRead = client.read(tcpBuffer, 1460);

...
} // End loop()
```

Las primeras líneas de código instancian los buffers necesarios para recepción.

Dentro del bucle de Arduino, se inicializa la posición del buffer, se comprueba que hay datos disponibles para tratar, y en caso afirmativo se almacenan. En el bucle, a continuación, se realiza todo el proceso de verificación de marcadores JPEG y llamadas a funciones de decodificación y representación en el display. Una explicación detallada se encuentra en el **Anexo II**.

Anexo II: Decodificación y representación de la imagen en el display SSD1331

Identificación de los marcadores

```
void loop() {
  tcpBuffer = arrayTcp;
  if (client.available() > 0) {
    bytesRead = client.read(tcpBuffer, 1460);
    //Serial.println("Bytes read: " + bytesRead);
    // Searching for 0xFFD8 and 0xFFD9 to save pure JPEG content
    for(int i=0; i < bytesRead; i++){
      //Serial.print(arrayTcp[i], HEX);
      if (state == 2){
        *jpegBuffer++ = arrayTcp[i];
        arrayJpeg_currentSize++;
        if (arrayJpeg_currentSize > MAXSIZE_ARRAYJPEG){
          //Serial.println("********arrayJpeg overflow. Discarding
image");
          jpegBuffer = arrayJpeg;
          arrayJpeg_currentSize = 0;
        }
        if(arrayTcp[i] == 0xD9){
          if (arrayTcp[i-1] == 0xFF){
            //Serial.println("********Image completed. Decoding...");
            decoded = JpegDec.decodeArray(arrayJpeg,
arrayJpeg_currentSize);
            if (decoded){
              //jpegInfo();
              //Serial.println("********Image decoded. Rendering...");
              jpegRender(8,2);
            } // End (decoded)
            else {
              //Serial.println("********Error rendering");
            }
            state = 0;
            jpegBuffer = arrayJpeg;
            arrayJpeg_currentSize = 0;
          } // End (arrayTcp[i-1] == 0xFF)
        } // End (arrayTcp[i] == 0xD9)
      } // End (state == 2)
      else if (arrayTcp[i] == 0xFF && state == 0){
        state = 1; // Init marker detected
      }
      else if (arrayTcp[i] == 0xD8 && state == 1) {
        state = 2;
        jpegBuffer = arrayJpeg;
        *jpegBuffer++ = 0xFF;
        *jpegBuffer++ = 0xD8;
        arrayJpeg_currentSize = 2;
        //Serial.println("start new image detected");
      } // End (arrayTcp[i] == 0xD8 && state == 1)
    } // End (int i=0; i < 1460; i++)
  } // End (client.available() > 0)
} // End loop()
```

En el código del bucle se esperará hasta que haya datos disponibles. Una vez estén accesibles, se van leyendo los datos del buffer de recepción y comprobando si se reciben los octetos 0xFFD8 de manera consecutiva. En caso afirmativo, se cambia la variable state a 1. Se cambiará la variable a 2 en el caso en que se reciba los octetos 0xFFD9 consecutivamente, y en ese momento empezará el proceso de decodificación. Se inicializan y limpian todos los buffers utilizados para la siguiente imagen.

Renderizado de la imagen en el display

```
void jpegRender(int xpos, int ypos) {

  // Retrieve infomration about the image
  uint16_t  *pImg;
  uint16_t mcu_w = JpegDec.MCUWidth;
  uint16_t mcu_h = JpegDec.MCUHeight;
  uint32_t max_x = JpegDec.width;
  uint32_t max_y = JpegDec.height;

  // Jpeg images are draw as a set of image block (tiles) called
Minimum Coding Units (MCUs)
  // Typically these MCUs are 16x16 pixel blocks
  // Determine the width and height of the right and bottom edge image
blocks
  uint32_t min_w = minimum(mcu_w, max_x % mcu_w);
  uint32_t min_h = minimum(mcu_h, max_y % mcu_h);

  // Save the current image block size
  uint32_t win_w = mcu_w;
  uint32_t win_h = mcu_h;

  // Record the current time so we can measure how long it takes to
draw an image
  //uint32_t drawTime = millis();

  // Save the coordinate of the right and bottom edges to assist image
cropping
  // To the screen size
  max_x += xpos;
  max_y += ypos;

  // Read each MCU block until there are no more
#ifdef USE_SPI_BUFFER
  while( JpegDec.readSwappedBytes()){ // Swap byte order so the SPI
buffer can be used
#else
  while ( JpegDec.read()) { // Normal byte order read
#endif
    // Save a pointer to the image block
    pImg = JpegDec.pImage;

    // Calculate where the image block should be drawn on the screen
    int mcu_x = JpegDec.MCUx * mcu_w + xpos;
    int mcu_y = JpegDec.MCUy * mcu_h + ypos;

    // Check if the image block size needs to be changed for the right
and bottom edges
    if (mcu_x + mcu_w <= max_x) win_w = mcu_w;
    else win_w = min_w;
    if (mcu_y + mcu_h <= max_y) win_h = mcu_h;
    else win_h = min_h;

    // Calculate how many pixels must be drawn
    uint32_t mcu_pixels = win_w * win_h;

    // Draw image MCU block only if it will fit on the screen
    if ( ( mcu_x + win_w) <= tft.width() && ( mcu_y + win_h) <=
tft.height())
```

```
  {

      tft.startPushData((uint16_t)mcu_x, (uint16_t)mcu_y,
(uint16_t)(mcu_x + win_w - 1), (uint16_t)(mcu_y + win_h - 1));
      while (mcu_pixels--) tft.pushData(*pImg++);
      tft.endPushData();
    }
    else if ( ( mcu_y + win_h) >= tft.height()) JpegDec.abort();
  }

}
```