UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

SISTEMA DE CÁMARA TÉRMICA PARA BOMBEROS

Víctor Ibáñez Molina

Tutor: José Manuel Mossi García

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2018-19

Valencia, 1 de julio de 2019

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universitat Politècnica de València
Edificio 4D. Camino de Vera, s/n, 46022 Valencia
Tel. +34 96 387 71 90, ext. 77190
www.etsit.upv.es

VLC/ CAMPUS VALENCIA, INTERNATIONAL CAMPUS OF EXCELLENCE

EMAS Gestión ambiental verificada REG.NO. ES-CV- 000030

Resumen

A lo largo de este trabajo final de grado se quiere desarrollar un sistema de cámara térmica para bomberos, centrándose en la compatibilidad del microcontrolador ESP32 con la cámara térmica Flir Lepton y una pantalla de reducido tamaño. El objetivo principal es obtener en tiempo real una imagen térmica que el usuario pueda visualizar sin necesidad de tener las manos ocupadas, permitiendo que el bombero puede desarrollar su labor de forma más independiente. Los elementos elegidos para formar parte del sistema permiten que el mismo se pueda utilizar de forma autónoma acompañándolo de una batería. Además, en posteriores desarrollos, se pretende poder aprovechar las características multi-núcleo y la conectividad inalámbrica wifi y bluetooth para transmitir en tiempo real la imagen obtenida.

Resum

A aquest treball fi de grau es vol desenvolupar un sistema de càmera tèrmica per a bombers, centrant-se en la compatibilitat del microcontrolador ESP32 amb la càmera tèrmica Flir Lepton i una pantalla de mida reduïda. L'objectiu principal es obtindré a temps real una imatge tèrmica que l'usuari puga visualitzar sense necessitat de tindre les mans ocupades, permetent que el bomber duga a terme la seua labor de forma més independent. Els elements elegits per formar part del sistema permeten que aquest s'utilitze d'una manera autònoma acompanyant-lo d'una bateria. A més a més, en posteriors desenvolupaments, es pretén aprofitar les característiques multi-nucli i la connectivitat sense fils wifi i bluetooth per a transmetre en temps real la imatge obtinguda.

Abstract

Throughout this final degree project, we want to develop a thermal camera system for firefighters, focusing on the compatibility of the ESP32 microcontroller with the Flir Lepton thermal camera and a small size screen. The main objective is to obtain in real time a thermal image that the user can visualize without having their hands occupied, allowing the firefighter to develop his work more independently. The elements chosen to be part of the system allow it to be used autonomously accompanied by a battery. In addition, in subsequent developments, it is intended to take advantage of the multi-core features and wireless connectivity wifi and bluetooth to transmit in real time the image

Índice

Índice de tablas

Índice de figuras

Capítulo 1. Preámbulo

1.1 Introducción

La miniaturización constante que sufre la tecnología actualmente ha propiciado la aparición de productos comerciales cómo las Google Glass. Este trabajo final de grado nace de un planteamiento similar, pero con un objetivo más específico, el hacer la labor de los bomberos en entornos adversos más cómoda.

El objetivo de este desarrollo es conseguir la capacidad de visualizar una imagen térmica directamente en el casco del bombero liberando así las manos de este.

Pese a que ya existen sistemas similares basados en Arduino se pretende realizar una mejora significativa adaptando desarrollos ya existentes a un entorno mucho más potente, de tamaño más reducido y mayor conectividad como es el ESP32.

1.2 Motivación

Actualmente los cuerpos de bomberos utilizan pesadas cámaras térmicas que requieren la utilización de las dos manos del operario que la manipula por lo que es necesario que los bomberos se muevan en grupos de dos, limitando las acciones que se pueden realizar.

Al ver la mejora que podría suponer el poder visualizar una imagen térmica directamente en el casco se propuso este trabajo que implicaría el avance en el objetivo final de dotar de autonomía a los bomberos en su función.

1.3 Definiciones, acrónimos y siglas

- **SPI:** Serial Peripheral Interface, protocolo síncrono que hace uso de cuatro señales MISO, MOSI, CS y CLK.
- **MISO:** Master In Slave Out, Puerto que envía información del elemento esclavo al maestro.
- **MOSI:** Master Out Slave In, puerto que envía información del maestro al elemento esclavo.
- **CS:** Chip-Select, puerto que permite indicar al elemento que recibe la señal si la información enviada es para él.
- **DC:** Data/Comand, puerto que permite indicar si la información transmitida son datos o comandos.
- **VIN:** Tensión de entrada.
- **GND:** Tierra, al ser corriente continua corresponde a 0V.
- **CLK:** Señal de reloj que permite la sincronización entre elementos.
- **SDA:** Bus serie de datos.
- **SCL:** Bus serie de reloj.
- **SoC:** System on Chip, sistema que contiene todos los elementos necesarios para su funcionamiento en una única placa.
- **Fps:** frames per second, imágenes por Segundo.

Capítulo 2. Elementos del Sistema

2.1 ESP32

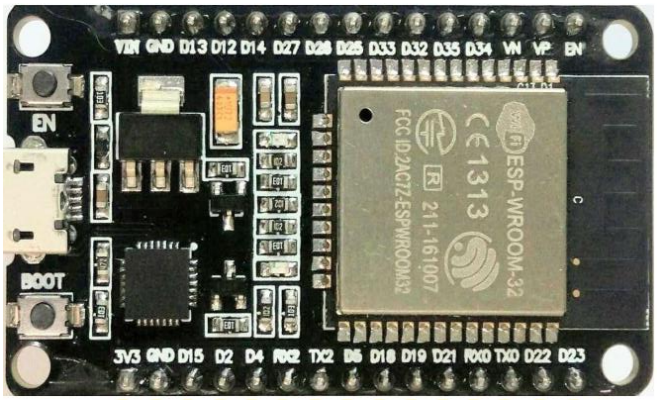


Figura 1. ESP-WROOM-32

2.1.1 Datasheet

Ítem	Especificaciones
Frecuencia	40 MHz
Núcleos	2
Memoria flash	4MB
Conexiones físicas	3xSPI
Conexiones inalámbricas	WiFi 2.4GHz y Bluetooth v4.2
Voltaje de salida	2.7V – 3.6V
Corriente de salida	500 mA
Tamaño	18mm x 25.5mm x 3.1 mm

Tabla 1. Datasheet del ESP32

Anexo a este desarrollo se encuentra el pinout correspondiente a la placa utilizada para llevar a cabo este proyecto.

2.1.2 Motivo de la elección

La elección del SoC ESP32 es debida principalmente a sus capacidades técnicas frente a su reducido tamaño, que permiten que se pueda incorporar a un casco sin modificar en exceso su volumen.

Es importante remarcar que es un sistema que utiliza poca energía para su funcionamiento, idóneo para un funcionamiento con baterías. Además, su compatibilidad con el entorno de desarrollo elegido permite aprovechar las innumerables librerías desarrolladas para Arduino.

2.2 FLIR Lepton + Breakout v1.3

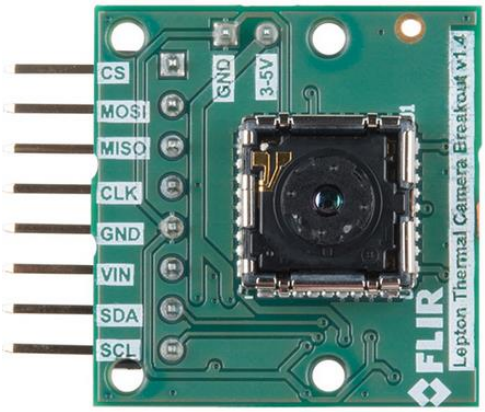


Figura 2. FLIR Lepton + Breakout v1.4

2.2.1 Datasheet

Ítem	Especificaciones
Resolución	80x60
Apertura	51º
Conexiones físicas	SPI, I2C
Voltaje de funcionamiento	3V
Tamaño	10.6mm x 11.7mm x 5.9mm

Tabla 2. Datasheet de FLIR Lepton

2.2.2 Motivo de la elección

Pese a su baja resolución el contenido coste, en torno a 200€, hace que la cámara elegida sea perfecta para la realización de un prototipo de estas características. Además, la existencia de librerías compatibles con Arduino facilita la conexión con el ESP32.

Como en el resto de los componentes elegidos su tamaño es importante y ayuda a la miniaturización del sistema.

2.3 Panel OLED SSD1331

Figura 3. SSD1331

2.3.1 Datasheet

Ítem	Especificaciones
Resolución	96x64
Profundidad de color	16 bit
Tamaño de la pantalla	0.95”
Conexiones físicas	SPI
Voltaje de funcionamiento	3.3V-5V

Tabla 3.Datasheet SSD1331

2.3.2 Motivo de la elección

La pantalla elegida para la realización del prototipo es completamente compatible con Arduino y su resolución es mayor a la de la cámara, por lo que es suficiente para la realización de este trabajo.

Capítulo 3. Desarrollo

3.1 Entorno elegido

Para llevar a cabo el desarrollo del trabajo se optó por el entorno de Arduino (Arduino IDE) que ofrece compatibilidad con todos los componentes del sistema mediante la instalación de librerías.

La instalación y configuración del entorno se realizará en Windows 10, por lo que podría variar ligeramente si se realizase en MACOS o Linux.

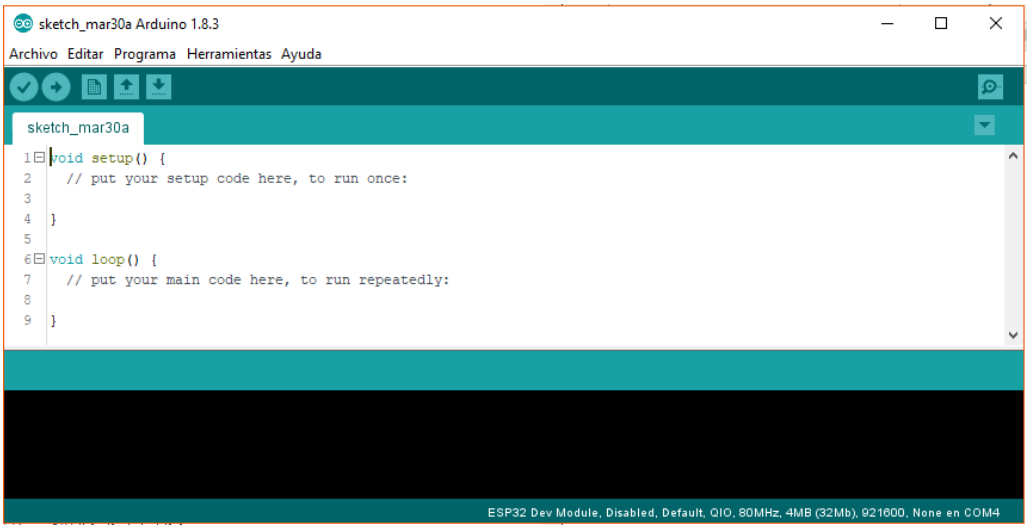


Figura 4. Arduino IDE

La programación del proyecto se realiza en C y, en general, se programa de igual forma a como se haría para un Arduino.

3.1.1 Compatibilidad con la tarjeta ESP32

De forma nativa el entorno de desarrollo no es compatible con la placa ESP32 por lo que es necesario instalar un paquete compatible. En nuestro caso hemos optado por la librería ESP32 de Espressif Systems, fabricante del chip, que se puede instalar directamente desde el Gestor de Tarjetas accesible en el menú de Herramientas. Para ello es necesario agregar en el menú Preferencias la siguiente URL "https://dl.espressif.com/dl/package_esp32_index.json" dentro del espacio reservado para el **Gestor de URLs Adicionales de Tarjetas**.

Una vez instalada la librería ya se puede configurar el entorno para llevar a cabo el proyecto.

3.1.2 Configuración del entorno

Desde el menú de Herramientas se accede a la configuración de la placa que permite tanto compilar el programa para una placa en concreto como subir el programa ya compilado a la misma.

Configuración	Selección
Placa	ESP32 Dev Module
Flash Mode	QIO
Flash Frequency	80MHz
Flash Size	4MB (32Mb)

PSRAM	Disabled
Upload Speed	921600
Core Debug Lebel	Ninguno

Tabla 4. Configuración del IDE

En cuanto al puerto es necesario seleccionar aquél del ordenador al que esté conectada la placa, en caso de desconocerlo se puede obtener desde el Administrador de dispositivos de Windows.

Una vez realizada esta configuración ya se puede proceder a iniciar el desarrollo.

3.2 Metodología seguida

Para conseguir un correcto funcionamiento de todos los componentes del sistema se considera mejor dividir el proyecto en distintas subtareas. Dichas subtareas tienen como objetivo simplificar el trabajo para asegurar un resultado final.

Las subtareas en las que se dividió el trabajo son las siguientes:

- Conexionado del panel oled con la placa ESP32: Modificación del código facilitado por el fabricante para adaptarlo a la placa y obtener imágenes personalizadas.
- Conexionado de la cámara térmica con la placa ESP32: Adaptar las librerías existentes realizadas por el fabricante y conseguir la obtención de datos que se puedan procesar para obtener imágenes.
- Unión de los dos sistemas: Conexionado simultáneo del panel y la cámara con la placa de tal forma que se pueda visualizar en tiempo real las imágenes obtenidas de la cámara en el panel oled.

3.3 Conexión del ESP32 con el Panel OLED SSD1331

3.3.1 Conexionado

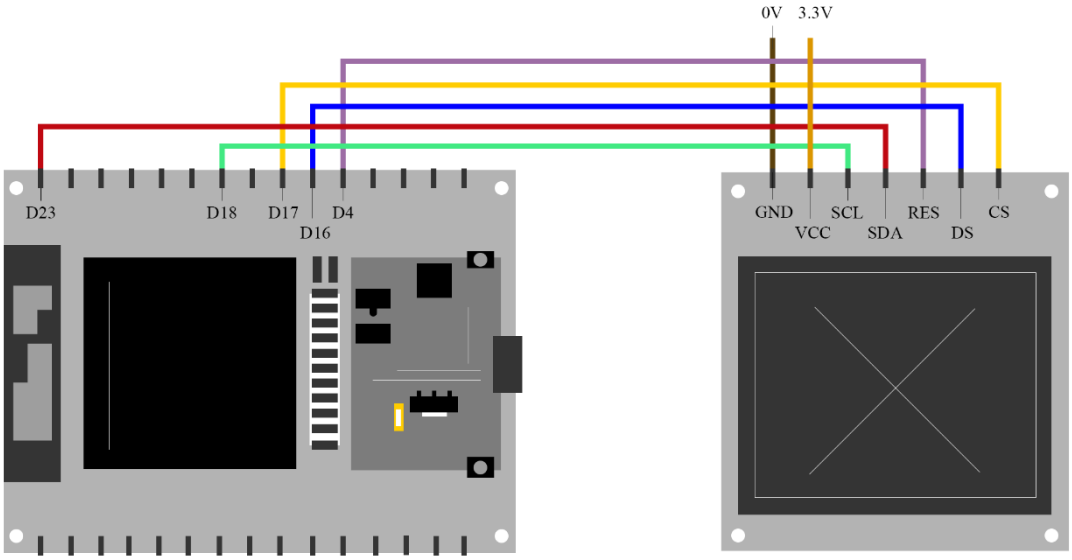


Figura 5. Esquema de conexionado ESP32-SSD1331

Se hace uso de la conectividad SPI que comparten ambos elementos. La pantalla requiere únicamente el puerto MOSI de la placa, al no transmitir información hacia el ESP32. Es por ello por lo que el conexionado es el mostrado en la Figura 5.

Puerto ESP32	Puerto SSD1331	Función
D23	SDA	MOSI
D18	SCL	CLK
D17	CS	Chip-Select
D16	DC	Data-Comand
D4	RES	Reset

Tabla 5. Conexionado ESP32 y SSD1331

3.3.2 Desarrollo

El código utilizado *[Código 1]* es una modificación del que incorpora la librería de la pantalla en la que se han cambiado los puertos utilizados para adecuarlos a nuestra placa.

3.3.3 Resultados

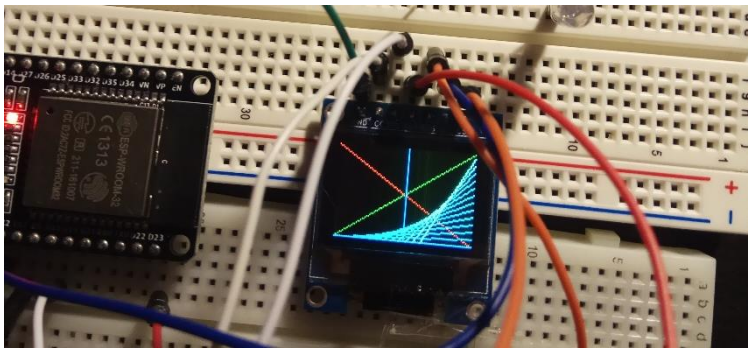


Figura 6. Resultados ESP32 y SSD1331

Tras la conexión y modificación de la librería se pueden obtener imágenes personalizadas píxel a píxel.

3.4 Conexión del ESP32 con la cámara FLIR Lepton

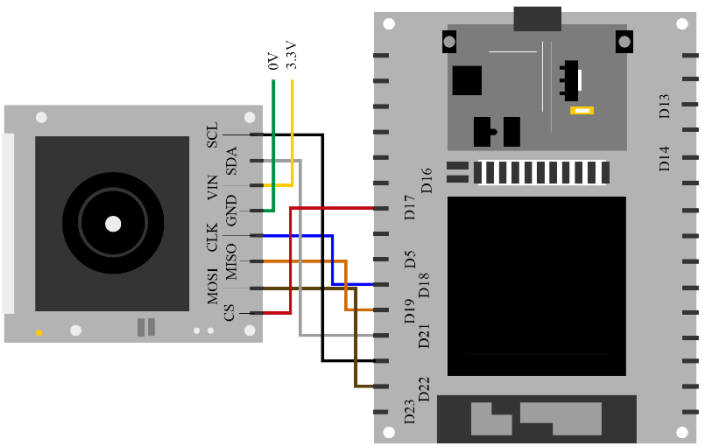


Figura 7. Esquema de conexionado ESP32-cámara FLIR Lepton

3.4.1 Conexionado

La conexión del módulo de la cámara con el ESP32 es más compleja que la realizada con la pantalla ya que hace uso tanto de una conexión SPI completa (MOSI, MISO, CLK y CS) como de una conexión bus serie (SDA y SCL).

En la Figura 7 se puede observar el conexionado realizado.

Puerto ESP32	Puerto FlirLepton	Función
D5	CS	Chip-Select
D23	MOSI	MOSI
D19	MISO	MOSO
D18	CLK	Reloj síncrono
D21	SDA	Bus serie, datos
D22	SCL	Bus serie, reloj

Tabla 6. Conexionado Flir Lepton y ESP32

3.4.2 Desarrollo

El código utilizado [Código 2] es una variación del código facilitado por el fabricante para el conexionado de la cámara con una placa Arduino. Para adaptar dicho código a nuestra placa se ha realizado una modificación de los puertos, así como la sustitución de ciertas funciones no compatibles con nuestro microcontrolador.

Para su correcto funcionamiento es esencial alimentar la cámara antes de iniciar el controlador, puesto que la cámara debe recibir las instrucciones iniciales cuando ya esté funcionando.

La obtención de los datos capturados se ha realizado mediante el envío por el puerto de comunicaciones cada uno de los píxeles de la imagen.

3.4.3 Resultados

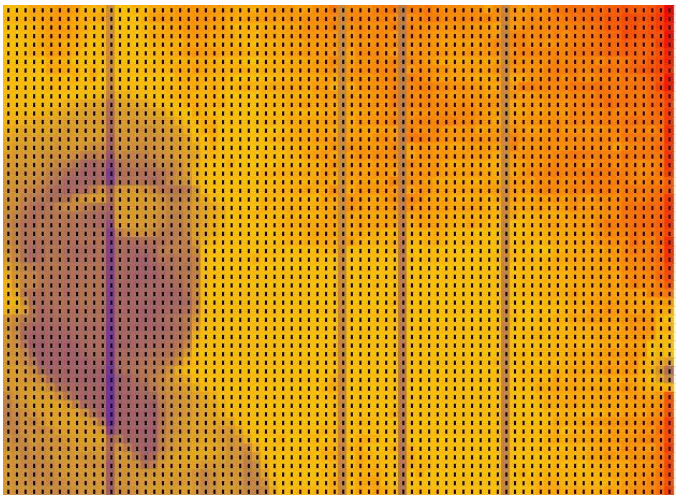


Figura 8. Resultados tras procesar los datos

Para poder observar de forma correcta los datos extraídos de la cámara se han trasladado a una hoja de cálculos y se le ha asignado a cada valor numérico un color, de forma que se pueda visualizar la imagen resultante.

En la Figura 8 se observa como el rostro es de un color más azulado (representa mayor temperatura) y las gafas, el pelo y la ropa tienen un color más amarillento (más frío).

3.5 Sistema en conjunto

3.5.1 Conexionado

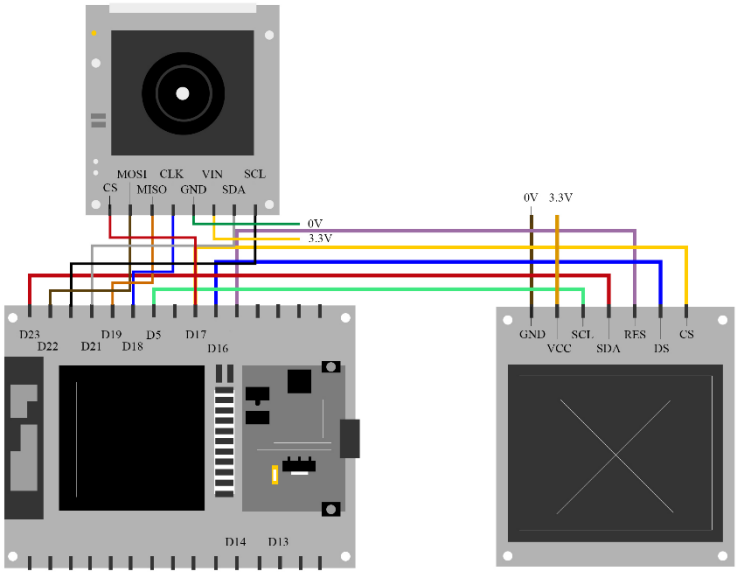


Figura 9. Conexionado final del sistema

El conexionado del sistema en conjunto es más complejo puesto que en los dos casos anteriores se hace uso de los mismos puertos SPI, en este caso haremos uso de dos puertos SPI distintos, así como de un BUS serie.

La alimentación de los distintos elementos se ha realizado mediante una batería, puesto que el ESP32 no es capaz de aportar suficiente energía para alimentar a la cámara y a la pantalla.

Puerto ESP32	Puerto FlirLepton	SSD1331	Función
D5	CS		Chip-Select
D23	MOSI		MOSI
D19	MISO		MISO
D18	CLK		Reloj síncrono
D21	SDA		Bus serie, datos
D22	SCL		Bus serie, reloj
D13		SDA	MOSI
D14		SCL	CLK
D17		CS	Chip-Select
D16		DC	Data-Comand
D4		RES	Reset

Tabla 7. Conexionado final

3.5.2 Desarrollo

Al realizar uso tanto la cámara térmica cómo la pantalla del protocolo SPI se ha optado por que la cámara haga uso del bus principal y la cámara del bus hSPI del módulo ESP-32. Además, para la visualización de la imagen en la pantalla se ha trasladado cada píxel obtenido de la cámara a uno de la pantalla asignándole un color en función de su valor numérico.

En un primer momento se ha optado por utilizar la escala de grises ya que es más sencillo el trasladar un valor numérico a uno cromático, para eso los componentes R, G y B tienen el mismo valor. Para eliminar los valores extremos que nunca se van a representar se han escalado de forma experimental los resultados obtenidos. Dicho escalado debe ser adaptado al iniciar el sistema para que se represente de forma correcta.

3.5.3 Resultados

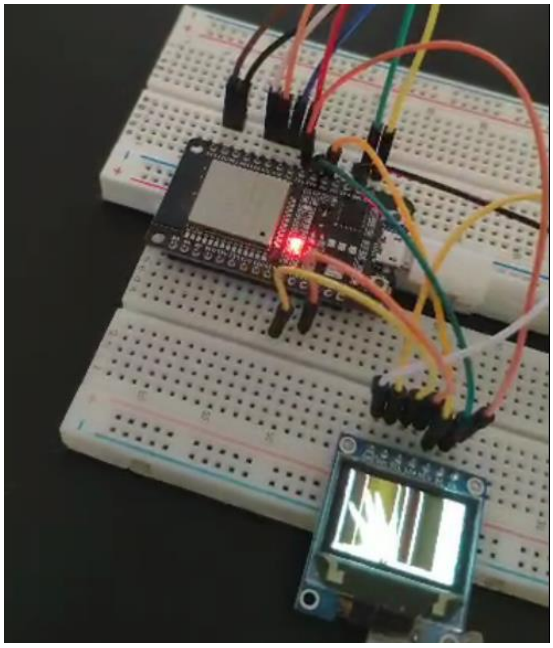


Figura 10. Resultado obtenido.

Con la configuración realizada se ha conseguido obtener, en tiempo real, la imagen térmica capturada en la cámara sobre la pantalla oled. Como se puede observar en la Figura 10 se aprecian 4 líneas verticales en la imagen obtenida, dichas líneas también se pueden observar en la Figura 8, por lo que se considera un fallo de la cámara a la hora de transmitir o procesar la imagen.

El funcionamiento final es estable, y se ha conseguido una tasa de refresco de unos 2fps.

3.6 Sistema en conjunto: Código final.

Tras comprobar el funcionamiento de todos los elementos se propuso realizar las siguientes mejoras para conseguir un resultado óptimo:

-Eliminación del offset de la cámara

-Obtención de una mayor fluidez, llegando a mostrar por pantalla los fps máximos que permite la cámara.

-Modificar la escala de grises por una imagen a color en la que los colores fríos se muestren azules y los cálidos rojizos.

Los cambios realizados se aprecian en el Código 4 que representa el código definitivo en el que se consiguen las mejoras esperadas. A continuación, se va a realizar una explicación de dichos cambios:

3.6.1 Eliminación del offset

En la función ***void setOffset(void)*** a la que se llama al arranque del sistema se capturan 10 fotogramas de la cámara térmica y se obtiene una imagen de control del sistema. Para esto es necesario que se inicie el sistema en un lugar controlado y sin fuentes de calor cercanas (bombillas incandescentes, radiadores, etc).

Una vez obtenido el offset este se eliminará de cada píxel. Con esto conseguimos eliminar las franjas verticales que aparecían en los puntos iniciales y a su vez obtener unos valores similares en todas las mediciones, eliminando así las posibles variaciones de la cámara y de la temperatura ambiente.

3.6.2 Mejora de la fluidez

Para conseguir una mejora significativa de la fluidez en el envío de las imágenes a la pantalla ha sido necesario modificar de forma significativa el código encargado de ello.

Anteriormente se enviaban a la pantalla aquellos píxeles que se habían visto modificados suponiendo así un envío de instrucciones y datos por cada píxel en el que había variado. Esto creaba un cuello de botella que era el principal inconveniente a la hora de conseguir un buen frame-rate.

Al hacer uso de la instrucción ***ssd1331.drawImage(0,0,imagen,82,60);*** se puede enviar a la pantalla una imagen de forma directa, sin necesidad de indicar píxel a píxel que color debe mostrar. Este cambio ha permitido obtener un frame-rate superior al fijado por la cámara, por lo que ya no es una limitación del sistema.

Hay que tener en cuenta que existen librerías que permiten el envío de una imagen de forma similar a como se realiza en el código descrito pero dichas librerías no permiten personalizar la interfaz SPI a utilizar lo cual no las hace útiles en nuestro caso ya que el sistema requiere que se haga uso de la interfaz hSPI ya que la SPI se utiliza para la obtención de imágenes desde la cámara.

Se han realizado medidas de los fotogramas por segundo. En los datos obtenidos se muestra que se tarda una media de 38609 milisegundos en mostrar 1000 fotogramas, por lo que se están obteniendo 25-26 fps dato que supera los máximos teóricos de 8.6 fps que puede obtener la cámara, asegurándonos de esta forma la obtención de todas las imágenes posibles. Esto repercute en la fluidez apreciable del sistema que no sólo muestra las imágenes de una forma fluida, sino que lo hace de forma inmediata, sin retardos apreciables.

3.6.3 Modificación del color

El paso de una escala de grises a una imagen a color en la que los colores más fríos tengan un tono azul y los cálidos un tono rojo se realiza mediante un array de valores de 2 bytes (**camColors[]**) en el que los primeros elementos corresponden a tonos fríos/azulados y los últimos, hasta llegar a 255, corresponden a tonos cálidos/rojizos.

Para seleccionar qué color corresponde a cada temperatura se elimina al valor obtenido para cada píxel el offset y se selecciona un valor del array en función del valor obtenido (de 0 a 255).

Dicho valor es el que se envía a la instrucción encargada de enviar la imagen a la pantalla.

3.6.4 Resultado

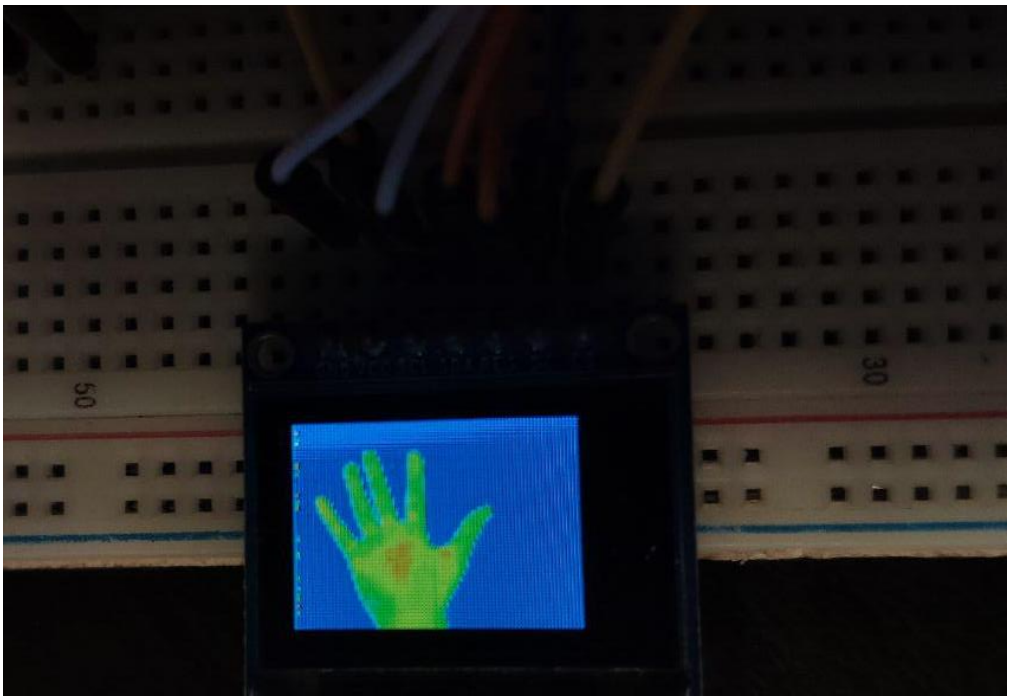


Figura 11. Resultado final

Siguiendo el conexionado utilizado en el apartado 3.5 y mediante las modificaciones realizadas en el 3.6 se ha conseguido obtener una imagen térmica fluida, con una gama cromática que permiten aprovechar al máximo todos los componentes del sistema.

Capítulo 4. Conclusiones

El desarrollo de este trabajo fin de grado ha requerido un esfuerzo importante de búsqueda de información para conseguir la integración total de la cámara térmica con el micro-controlador ESP32.

Se ha conseguido captar imágenes completamente estables de la cámara en tiempo real, así como su visualización en el panel oled. Es por eso que se considera que se han conseguido los objetivos principales de este trabajo final de grado marcando las bases para un futuro sistema autónomo que dote de libertad a el bombero que lo utilice.

4.1 Principales problemas encontrados

Durante la realización del proyecto se han superado distintas problemáticas. A continuación se explicarán tanto las problemáticas como las decisiones tomadas para poder superarlas

4.1.1 Compatibilidad de los distintos elementos

La mayor problemática encontrada es la compatibilidad de la cámara y la pantalla con el microcontrolador elegido ya que, pese a que se ha utilizado un entorno de desarrollo soportado por los fabricantes de todos los elementos, no existía compatibilidad directa entre ellos.

No se han podido utilizar librerías proporcionadas por los fabricantes. En el caso de la cámara la librería proporcionada por FLIR contenía instrucciones que no eran compatibles con el ESP-32 por lo que no permitía la compilación del código por lo que se optó por utilizar una variación simplificada que sí tenía compatibilidad con el microcontrolador utilizado. La librería de la pantalla sí es plenamente compatible con el sistema, pero no permite el uso de un SPI distinto al por defecto y, puesto que el conjunto requería de la utilización del hSPI, se ha optado por utilizar una modificación que sí permite utilizar el protocolo SPI con un bus distinto al principal.

4.1.2 Alimentación

Al inicio del desarrollo se utilizaba como única fuente de alimentación la incorporada en la placa de desarrollo que permitía utilizar sin problema la pantalla elegida. Tras esto se procedió al conexionado del sensor térmico que, tras distintas pruebas, se comprobó que requería de una alimentación superior a la que podía aportar la placa y era éste el motivo por el que no se podía conseguir una imagen válida. Por esto se decidió utilizar una fuente externa para alimentar la cámara.

4.2 Puntos de mejora

Los aspectos más importantes por mejorar para conseguir un resultado óptimo son los siguientes:

4.2.1 Mejora en el calibrado

Actualmente se requiere de un entorno controlado para poder calibrar inicialmente el sistema y obtener las imágenes lo más exactas posibles. En las próximas versiones del sistema se hará uso de la versión de esta cámara que incorpora un shutter para hacer el calibrado de la cámara en cualquier entorno.

4.2.2 Alimentación del sistema

Para la alimentación del sistema es necesario utilizar una fuente independiente a la que aporta el ESP-32 por lo que el prototipo tiene un tamaño considerable que no permite su uso en un casco, sería conveniente encontrar una forma de alimentar el sistema sin depender de fuentes externas de un tamaño considerable.

4.3 Siguientes pasos

Para continuar con el desarrollo y, dejando de lado los puntos de mejora, sería conveniente aprovechar las capacidades multi-núcleo del controlador para llevar a cabo las tareas de una forma más eficiente. Dicho desarrollo se ha iniciado y está presente en el [Código 5] anexado que permite visualizar una imagen cambiante en el panel oled y a su vez encender y apagar un led conectado a una de las salidas, de forma que los tiempos de espera del led no influyen en el funcionamiento de la pantalla.

Además, la utilización de la conectividad disponible para transmitir la imagen obtenida a un receptor que permita visualizar la información obtenida desde un lugar alejado del punto de acción del bombero. También se podrían obtener datos del estado del bombero tales como el pulso, la ubicación o posibles caídas mediante el uso de otros sensores y del resto de entradas disponibles.

Sería conveniente además realizar un estudio del consumo en conjunto del sistema pues la batería necesaria para alimentar el sistema en conjunto podría marcar los límites de tamaño y/o tiempo de utilización.

Capítulo 5. Bibliografía

[1] Espressif Systems, “Arduino core for the ESP32” https://github.com/espressif/arduino-esp32. [Online].

[2] MGO-TEC, “OLED SSD1331 SPI library” https://github.com/mgo-tec/ESP32_SSD1331. [Online].

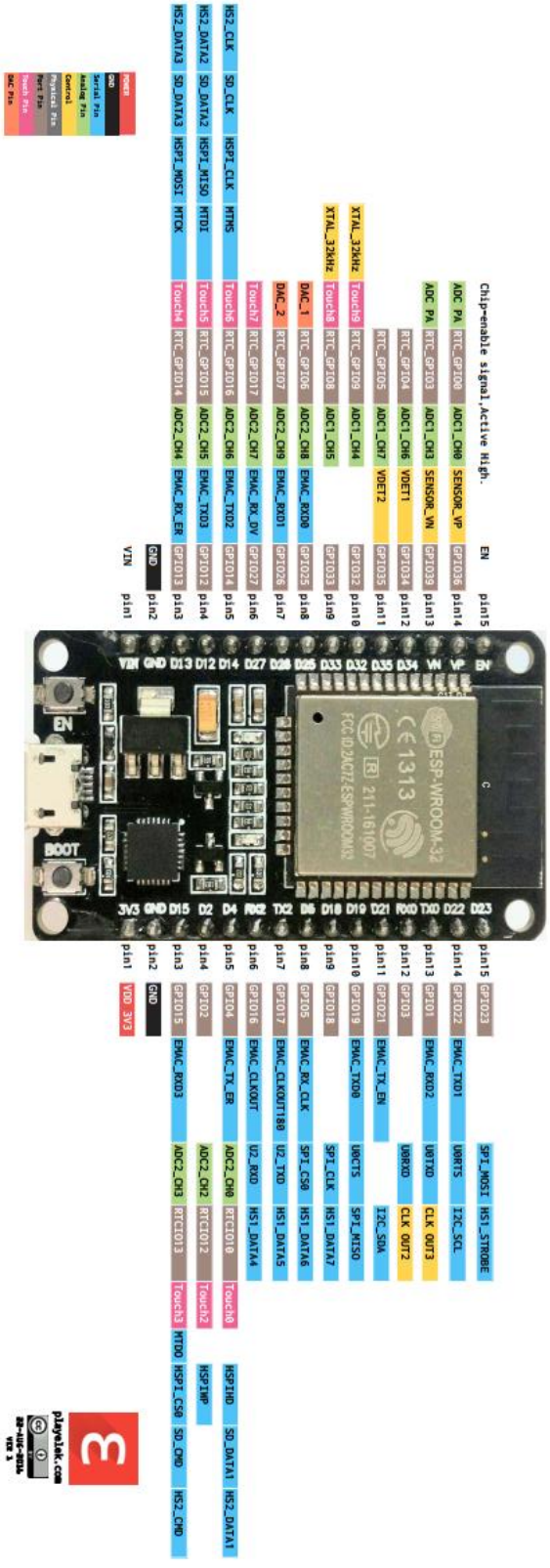
[3] TAKE-SAN, “Termografía 3D con la cámara termográfica ESP32 + LEPTON” http://takesan.hatenablog.com/entry/2017/02/21/152240. [Online].

[4] QUADMEUP, “Arduino, ESP32 and 3 hardware serial ports” https://quadmeup.com/arduino-esp32-and-3-hardware-serial-ports/. [Online].

[5] COCOLOG-NIFTY, “Controlador de visualización de mapa de bits para OLED-SSD1331” http://k-hiura.cocolog-nifty.com/blog/2019/04/post-2f9ffe.html. [Online].

Capítulo 6. Anexos

6.1 Pinout ESP32



6.2 Código 1: ESP32 y SSD1331

```
#include <Wire.h>
#include <SPI.h>
 //SCLK=18 MOSI=23
const uint8_t cs_OLED=17; //CS
const uint8_t DCpin=16; // DC
const uint8_t RSTpin=4; // Reset

void setup() {
 SSD1331_Init(cs_OLED,DCpin,RSTpin);

 Wire.begin();
 Serial.begin(115200);
 digitalWrite (cs_OLED, LOW);
 digitalWrite (cs_OLED, HIGH);
}

void loop() {
  CommandWrite(0xA0);
  CommandWrite(0b00110010);
  Display_Clear(0, 0, 95, 63);
  CommandWrite(0xA0);
  CommandWrite(0b01110010);

  Drawing_Circle_Line_65kColor(31, 31, 31, 0, 63, 31);

digitalWrite (cs_OLED, LOW);
digitalWrite (cs_OLED, HIGH);
}

void SSD1331_Init(uint8_t CS, uint8_t DC, uint8_t RST){

  pinMode(RST,OUTPUT);
  pinMode(DC,OUTPUT);
  pinMode(CS,OUTPUT);

  digitalWrite(RST, HIGH);
  digitalWrite(RST,LOW);
  delay(1);
  digitalWrite(RST,HIGH);

  digitalWrite(CS,HIGH);
  digitalWrite(DC,HIGH);

 SPISettings(SPISettings(360000, LSBFIRST, SPI_MODE3));
  SPI.setClockDivider(SPI_CLOCK_DIV2);
  SPI.begin();

  CommandWrite(0xAE);
  CommandWrite(0xA0);
  CommandWrite(0b00110010);
  CommandWrite(0xA1);
  CommandWrite(0);
  CommandWrite(0xA2);
```

```
  CommandWrite(0);
  CommandWrite(0xA4);
  CommandWrite(0xA8);
  CommandWrite(63);
  CommandWrite(0xAD);
  CommandWrite(0b10001110);
  CommandWrite(0xB0);
  CommandWrite(0x1A);
  CommandWrite(0xB1);
  CommandWrite(0x74);
  CommandWrite(0xB3);
  CommandWrite(0xF0);
  CommandWrite(0x8A);
  CommandWrite(0x81);
  CommandWrite(0x8B);
  CommandWrite(0x82);
  CommandWrite(0x8C);
  CommandWrite(0x83);
  CommandWrite(0xBB);
  CommandWrite(0x3A);
  CommandWrite(0xBE);
  CommandWrite(0x87);
  CommandWrite(0x06);
  CommandWrite(0x15);
  CommandWrite(0);
  CommandWrite(95);
  CommandWrite(0x75);
  CommandWrite(0);
  CommandWrite(63);
  CommandWrite(0x81);
  CommandWrite(255);
  CommandWrite(0x82);
  CommandWrite(255);
  CommandWrite(0x83);
  CommandWrite(255);
  CommandWrite(0xAF);
  delay(110);
  }

void Display_Clear(uint8_t x0, uint8_t y0, uint8_t x1, uint8_t y1){
  delayMicroseconds(500);
  CommandWrite(0x25);
  CommandWrite(x0);
  CommandWrite(y0);
  CommandWrite(x1);
  CommandWrite(y1);
  delayMicroseconds(800);
}


void Drawing_Pixel_65kColor(uint8_t x0, uint8_t y0, uint8_t R,
uint8_t G, uint8_t B){
  uint8_t Dot1, Dot2;
  CommandWrite(0x15);
  CommandWrite(x0);
  CommandWrite(x0);
  CommandWrite(0x75);
```

```
  CommandWrite(y0);
  CommandWrite(y0);

  Dot1 = (R << 3) | (G >> 3);
  Dot2 = (G << 5) | B;

  DataWrite(Dot1);
  DataWrite(Dot2);
}

void Drawing_Circle_Line_65kColor(uint8_t x0, uint8_t y0, uint16_t
r, uint8_t Line_R, uint8_t Line_G, uint8_t Line_B){
  uint8_t x1, y1;

  for(int i=0; i<360; i++){
    x1 = round((float)(x0 + (r * cos(radians(i)))));
    y1 = round((float)(y0 + (r * sin(radians(i)))));
    Drawing_Pixel_65kColor(x1, y1, Line_R, Line_G, Line_B);
  }
  delay(1);
  CommandWrite(0x15);
  CommandWrite(0);
  CommandWrite(95);
  CommandWrite(0x75);
  CommandWrite(0);
  CommandWrite(63);
}

void CommandWrite(uint8_t b){
  digitalWrite(cs_OLED, LOW);
  digitalWrite(DCpin, LOW);//DC
  SPI.write(b);
  digitalWrite(cs_OLED, HIGH);
}

void DataWrite(uint8_t b){
  digitalWrite(cs_OLED, LOW);
  digitalWrite(DCpin, HIGH);
  SPI.write(b);
  digitalWrite(cs_OLED, HIGH);
}
```

6.3 Código 2: ESP32 y FLIR Lepton

```
#include <SPI.h>
#include <Wire.h>

#define ADDRESS  (0x2A)
#define SYS (0x02)
#define GET (0x00)

void setup()
{
  Wire.begin();
  Serial.begin(115200);

  pinMode(5, OUTPUT);
```

```
  SPISettings(SPISettings(360000, LSBFIRST, SPI_MODE3));
  SPI.setClockDivider(SPI_CLOCK_DIV2);
  SPI.begin();
}

static uint16_t image[63][82];
static uint8_t image1[82*2];

void read_lepton_frame(void) {
  uint16_t data = 0x000f;
  delay(5);
  while ((data & 0x000f) == 0x000f)
  {
    delay(0.02);
    data = SPI.transfer16(0x0000);
    image[0][0] =data;
    SPI.transferBytes(0x0000,image1,162);
    for (int i = 1; i < 82; i++)
    {
      image[0][i]= (image1[2*i]<< 8  | image1[2*i+1]);
    }
  }
  for (int frame = 1; frame < 60; frame++){
    SPI.transferBytes(0x0000,image1,164);
    for (int i = 0; i < 82; i++)
    {
        image[frame][i] =  (image1[2*i]<< 8  | image1[2*i+1]);
     }
   }
}

void print_lepton_frame(void)
{
  Serial.println("");
  for (int frame = 0; frame < 60; frame++){
    for (int i = 0; i < 80; i++)
    {
        Serial.print(image[frame][i] );
        Serial.print(",");
    }
        Serial.println("");
  }

}

void lepton_command(unsigned int mID, unsigned int cID, unsigned
int com)  {
  Wire.beginTransmission(ADDRESS);

  Wire.write(0x00);
  Wire.write(0x04);

  if (mID == 0x08){
    Wire.write(0x48);
  }
  else{
    Wire.write(mID & 0x0f);
```

```
  }
  Wire.write( ((cID << 2 ) & 0xfc) | (com & 0x3));

}

void set_reg(unsigned int r){
  Wire.beginTransmission(ADDRESS);
  Wire.write(r >> 8 & 0xff);
  Wire.write(r & 0xff);
}

int read_reg(unsigned int r){
  int reading = 0;
  set_reg(r);
  Wire.requestFrom(ADDRESS, 2);
  reading = Wire.read();
  reading = reading << 8;
  reading |= Wire.read();
  return reading;
}

int read_data(){
  int p_length;
  int data;

  while (read_reg(0x2) & 0x01){
    Serial.println("busy");
  }
  p_length = read_reg(0x6);
  Wire.requestFrom(ADDRESS, p_length);
  set_reg(0x08);
  for (int i = 0; i < (p_length / 2); i++){
    data = Wire.read() << 8;
    data |= Wire.read();
  }
  return data;
}

void loop()
{
  lepton_command(SYS, 0x19>>2 ,GET);
  read_data();
    while(1){
error1:
     read_lepton_frame();
     print_lepton_frame();
   }
}
```

6.4 Código 3: Sistema Completo

```
#include <SPI.h>
#include <Wire.h>

const uint8_t cs_OLED=17; //CS
const uint8_t DCpin=16; //DC
const uint8_t RSTpin=4; //Reset
```

```
#define ADDRESS  (0x2A)
#define SYS (0x02)
#define GET (0x00)
SPIClass spiRFID(HSPI);

void setup()
{
  Wire.begin();
  Serial.begin(115200);
  SSD1331_Init(cs_OLED,DCpin,RSTpin);

  pinMode(5, OUTPUT);
  SPISettings(SPISettings(360000, LSBFIRST, SPI_MODE3));
  SPI.setClockDivider(SPI_CLOCK_DIV2);
  SPI.begin();
  digitalWrite (cs_OLED, LOW);
  lepton_command(SYS, 0x19>>2 ,GET);
  read_data();
  digitalWrite (cs_OLED, HIGH);

}

void SSD1331_Init(uint8_t CS, uint8_t DC, uint8_t RST){

  pinMode(RST,OUTPUT);
  pinMode(CS,OUTPUT);
  pinMode(DC,OUTPUT);

  digitalWrite(RST, HIGH);
  digitalWrite(RST,LOW);
  delay(1);
  digitalWrite(RST,HIGH);

  digitalWrite(CS,HIGH);
  digitalWrite(DC,HIGH);

 SPISettings(SPISettings(360000, LSBFIRST, SPI_MODE3));
 spiRFID.setClockDivider(SPI_CLOCK_DIV2);
 spiRFID.begin(14,12,13,15);

  CommandWrite(0xAE);
  CommandWrite(0xA0);
  CommandWrite(0b00110010);
  CommandWrite(0xA1);
  CommandWrite(0);
  CommandWrite(0xA2);
  CommandWrite(0);
  CommandWrite(0xA4);
  CommandWrite(0xA8);
  CommandWrite(63);
  CommandWrite(0xAD);
  CommandWrite(0b10001110);
  CommandWrite(0xB0);
  CommandWrite(0x1A);
  CommandWrite(0xB1);
  CommandWrite(0x74);
```

```
  CommandWrite(0xB3);
  CommandWrite(0xF0);
  CommandWrite(0x8A);
  CommandWrite(0x81);
  CommandWrite(0x8B);
  CommandWrite(0x82);
  CommandWrite(0x8C);
  CommandWrite(0x83);
  CommandWrite(0xBB);
  CommandWrite(0x3A);
  CommandWrite(0xBE);
  CommandWrite(0x87);
  CommandWrite(0x06);
  CommandWrite(0x15);
  CommandWrite(0);
  CommandWrite(95);
  CommandWrite(0x75);
  CommandWrite(0);
  CommandWrite(63);
  CommandWrite(0x81);
  CommandWrite(255);
  CommandWrite(0x82);
  CommandWrite(255);
  CommandWrite(0x83);
  CommandWrite(255);
  CommandWrite(0xAF);
  delay(110);
  }

static uint16_t image[63][82];
static uint8_t image1[82*2];
void read_lepton_frame(void) {
  uint16_t data = 0x000f;
  delay(5);
  while ((data & 0x000f) == 0x000f)
  {
    delay(0.02);
    data = SPI.transfer16(0x0000);
    image[0][0] =data;
    SPI.transferBytes(0x0000,image1,162);
    for (int i = 1; i < 82; i++){
      image[0][i]= (image1[2*i]<< 8  | image1[2*i+1]);
    }
  }
  for (int frame = 1; frame < 60; frame++){
    SPI.transferBytes(0x0000,image1,164);
    for (int i = 0; i < 82; i++){
        image[frame][i] =  (image1[2*i]<< 8  | image1[2*i+1]);
     }
   }
}

void print_lepton_frame(void) {
  int MAX=5000;
  int MIN=10000;
  int DIFF=MAX-MIN;
  int color;
```

```
  for (int frame = 0; frame < 60; frame++){
    for (int i = 0; i < 80; i++)
    {
      color = (image[frame][i]);
      //Serial.print(color);
      if(color>MAX){
        color=31;
      }else if(color<MIN){
        color =0;
      }else{
        color = 30*(color-MIN)/(DIFF);
      }
      Drawing_Pixel_65kColor(i, frame, color, 2*color, color);
    }
  }
}

void lepton_command(unsigned int mID, unsigned int cID, unsigned
int com) {
  Wire.beginTransmission(ADDRESS);

  Wire.write(0x00);
  Wire.write(0x04);

  if (mID == 0x08){
    Wire.write(0x48);
  }
  else {
    Wire.write(mID & 0x0f);
  }
  Wire.write( ((cID << 2 ) & 0xfc) | (com & 0x3));
}

void set_reg(unsigned int r){
  Wire.beginTransmission(ADDRESS);
  Wire.write(r >> 8 & 0xff);
  Wire.write(r & 0xff);
}

int read_reg(unsigned int r){
  int reading = 0;
  set_reg(r);
  Wire.requestFrom(ADDRESS, 2);
  reading = Wire.read();
  reading = reading << 8;
  reading |= Wire.read();
  return reading;
}

int read_data(){
  int data;
  int p_length;

  while (read_reg(0x2) & 0x01){
    Serial.println("busy");
  }
  p_length = read_reg(0x6);
```

```
  Wire.requestFrom(ADDRESS, p_length);
  set_reg(0x08);
  for (int i = 0; i < (p_length / 2); i++){
    data = Wire.read() << 8;
    data |= Wire.read();
  }
  return data;
}

void loop()
{
  lepton_command(SYS, 0x19>>2 ,GET);
  read_data();

    while(1){
error1:
  CommandWrite(0xA0);
  CommandWrite(0b00110010);
  CommandWrite(0xA0);
  CommandWrite(0b01110010);
  digitalWrite (cs_OLED, LOW);
  digitalWrite (cs_OLED, HIGH);
  read_lepton_frame();
  print_lepton_frame();
   }
}


void CommandWrite(uint8_t b){
  digitalWrite(cs_OLED, LOW);
  digitalWrite(DCpin, LOW);//DC
  spiRFID.write(b);
  digitalWrite(cs_OLED, HIGH);
}

void DataWrite(uint8_t b){
  digitalWrite(cs_OLED, LOW);
  digitalWrite(DCpin, HIGH);
  spiRFID.write(b);
  digitalWrite(cs_OLED, HIGH);
}

void Drawing_Pixel_65kColor(uint8_t x0, uint8_t y0, uint8_t R,
uint8_t G, uint8_t B){
  uint8_t Dot1, Dot2;
  CommandWrite(0x15);
  CommandWrite(x0);
  CommandWrite(x0);
  CommandWrite(0x75);
  CommandWrite(y0);
  CommandWrite(y0);

  Dot1 = (R << 3) | (G >> 3);
  Dot2 = (G << 5) | B;

  DataWrite(Dot1);
  DataWrite(Dot2);
```

```
}
```

6.5 Código 4: Código final

```
#include <SPI.h>
#include <Wire.h>
SPIClass spi(HSPI);

#define ADDRESS  (0x2A)
#define SYS (0x02)
#define GET (0x00)

static uint16_t image[63][82];
static uint8_t image1[82*2];
unsigned char frameArray[9920];
int  offset_uchar[9920];
uint8_t color5;
bool isoffset;


const uint16_t camColors[] = {0x480F,
0x400F,0x400F,0x400F,0x4010,0x3810,0x3810,0x3810,0x3810,0x3010,0x30
10,
0x3010,0x2810,0x2810,0x2810,0x2810,0x2010,0x2010,0x2010,0x1810,0x18
10,
0x1811,0x1811,0x1011,0x1011,0x1011,0x0811,0x0811,0x0811,0x0011,0x00
11,
0x0011,0x0011,0x0011,0x0031,0x0031,0x0051,0x0072,0x0072,0x0092,0x00
B2,
0x00B2,0x00D2,0x00F2,0x00F2,0x0112,0x0132,0x0152,0x0152,0x0172,0x01
92,
0x0192,0x01B2,0x01D2,0x01F3,0x01F3,0x0213,0x0233,0x0253,0x0253,0x02
73,
0x0293,0x02B3,0x02D3,0x02D3,0x02F3,0x0313,0x0333,0x0333,0x0353,0x03
73,
0x0394,0x03B4,0x03D4,0x03D4,0x03F4,0x0414,0x0434,0x0454,0x0474,0x04
74,
0x0494,0x04B4,0x04D4,0x04F4,0x0514,0x0534,0x0534,0x0554,0x0554,0x05
74,
0x0574,0x0573,0x0573,0x0573,0x0572,0x0572,0x0572,0x0571,0x0591,0x05
91,
0x0590,0x0590,0x058F,0x058F,0x058F,0x058E,0x05AE,0x05AE,0x05AD,0x05
AD,
0x05AD,0x05AC,0x05AC,0x05AB,0x05CB,0x05CB,0x05CA,0x05CA,0x05CA,0x05
C9,
0x05C9,0x05C8,0x05E8,0x05E8,0x05E7,0x05E7,0x05E6,0x05E6,0x05E6,0x05
E5,
0x05E5,0x0604,0x0604,0x0604,0x0603,0x0603,0x0602,0x0602,0x0601,0x06
21,
0x0621,0x0620,0x0620,0x0620,0x0620,0x0E20,0x0E20,0x0E40,0x1640,0x16
40,
0x1E40,0x1E40,0x2640,0x2640,0x2E40,0x2E60,0x3660,0x3660,0x3E60,0x3E
60,
0x3E60,0x4660,0x4660,0x4E60,0x4E80,0x5680,0x5680,0x5E80,0x5E80,0x66
80,
0x6680,0x6E80,0x6EA0,0x76A0,0x76A0,0x7EA0,0x7EA0,0x86A0,0x86A0,0x8E
A0,
```

```
0x8EC0,0x96C0,0x96C0,0x9EC0,0x9EC0,0xA6C0,0xAEC0,0xAEC0,0xB6E0,0xB6
E0,
0xBEE0,0xBEE0,0xC6E0,0xC6E0,0xCEE0,0xCEE0,0xD6E0,0xD700,0xDF00,0xDE
E0,
0xDEC0,0xDEA0,0xDE80,0xDE80,0xE660,0xE640,0xE620,0xE600,0xE5E0,0xE5
C0,
0xE5A0,0xE580,0xE560,0xE540,0xE520,0xE500,0xE4E0,0xE4C0,0xE4A0,0xE4
80,
0xE460,0xEC40,0xEC20,0xEC00,0xEBE0,0xEBC0,0xEBA0,0xEB80,0xEB60,0xEB
40,
0xEB20,0xEB00,0xEAE0,0xEAC0,0xEAA0,0xEA80,0xEA60,0xEA40,0xF220,0xF2
00,
0xF1E0,0xF1C0,0xF1A0,0xF180,0xF160,0xF140,0xF100,0xF0E0,0xF0C0,0xF0
A0,
0xF080,0xF060,0xF040,0xF020,0xF800,};

uint8_t SSD1331_ic[]={

0xAE,0xA0,0b00110010,0xA1,0,0xA2,0,0xA4,0xA8,0b00111111,0xAD,0b1000
1110
  ,0xB0,0b00000000 ,0xB1,0x74 ,0xB3,0xF0 ,0x8A,0x81 ,0x8B,0x82
,0x8C,0x83
  ,0xBB,0x3A ,0xBE,0x3E ,0x87,0x06 ,0x15,0,95 ,0x75,0,63
  ,0x81,255 ,0x82,255, 0x83,255
   };

class hspiSSD1331 {
   uint8_t sck;
   uint8_t miso;
   uint8_t mosi;
   uint8_t cs;
   uint8_t dc;
   uint8_t rst;
public:

   hspiSSD1331(uint8_t miso_,uint8_t mosi_,uint8_t sck_,uint8_t cs_
            ,uint8_t dc_,uint8_t rst_)

:miso(miso_),mosi(mosi_),sck(sck_),cs(cs_),dc(dc_),rst(rst_){}

   void commandWriteBytes(uint8_t *a, uint16_t m){
      digitalWrite(dc, LOW);
      spi.writeBytes(a, m);
      }

   void commandWrite(uint8_t a){
      digitalWrite(dc, LOW);
      spi.write(a);
      }


    void dataWriteBytes(uint8_t *a, uint16_t m){
      digitalWrite(dc, HIGH);
      spi.writeBytes(a, m);
      }

   void init(){
```

```
      pinMode(rst,OUTPUT);
      pinMode(dc, OUTPUT);
      digitalWrite(rst, HIGH);
      digitalWrite(rst, LOW);
      delay(1);
      digitalWrite(rst, HIGH);
      digitalWrite(dc,  HIGH);
      spi.begin(sck, miso, mosi, cs);
      spi.setFrequency(7000000);
      spi.setDataMode(SPI_MODE3);
      spi.setHwCs(true);
      commandWriteBytes(SSD1331_ic,sizeof(SSD1331_ic));
      commandWrite(0xAF);
      delay(110);
      }

   void display_Clear(uint8_t x0, uint8_t y0, uint8_t x1, uint8_t
y1){
      delay(1);
      commandWrite(0x25);
        commandWrite(x0);
        commandWrite(y0);
        commandWrite(x1);
        commandWrite(y1);
      delay(1);
      }

   void drawImage(int w_,int h_,uint8_t* image_,int width_,int
height_){
      uint8_t _com[8]={0x15,0,95,0x75,0,63,0xA0,0b01110010};
      _com[1]= w_;
      _com[2]= w_+width_-1;
      _com[4]= h_;
      _com[5]= h_+height_-1;
      commandWriteBytes(_com,sizeof(_com));
      dataWriteBytes(image_,width_*height_*2);
      }
   };

void read_lepton_frame(void)
{
  uint16_t data = 0x000f;
  int colorEnd;

  delay(5);
  while ((data & 0x000f) == 0x000f)
  {
    delay(0.02);
    data = SPI.transfer16(0x0000);
    image[0][0] =data;
    SPI.transferBytes(0x0000,image1,162);
    for (int i = 1; i < 82; i++)
    {
      image[0][i]= (image1[2*i]<< 8  | image1[2*i+1]);
    }
  }
  for (int frame = 1; frame < 60; frame++){
```

```
    SPI.transferBytes(0x0000,image1,164);
    for (int i = 0; i < 82; i++)
    {
        colorEnd = (image1[2*i]<< 8  | image1[2*i+1]);
        if (isoffset){
         offset_uchar[(frame*82+i)*2] = colorEnd;
        }else{
          if((colorEnd +50)> offset_uchar[(frame*82+i)*2]){
                color5 = colorEnd-offset_uchar[(frame*82+i)*2]+50;
          }else{
            color5 = 0;
          }
        }

        uint8_t colorIndex = map(color5, 0, 255, 0, 255);
        colorIndex = constrain(colorIndex, 0, 255);
        frameArray[(frame*82+i)*2] = camColors[colorIndex]>>8;
        frameArray[(frame*82+i)*2+1] = camColors[colorIndex];

     }
   }
}

void print_lepton_frameOff(void)
{

for (int frame = 0; frame < 60; frame++){
    for (int i = 0; i < 80; i++)
    {
        Serial.print(offset_uchar[(frame*82+i)*2]);
        Serial.print(",");
    }
        Serial.println(" ");
  }
        Serial.println(" ");
}

void set_reg(unsigned int reg)
{
  Wire.beginTransmission(ADDRESS);
  Wire.write(reg >> 8 & 0xff);
  Wire.write(reg & 0xff);
}

int read_reg(unsigned int reg)
{
  int reading = 0;
  set_reg(reg);
  Wire.requestFrom(ADDRESS, 2);
  reading = Wire.read();
  reading = reading << 8;
  reading |= Wire.read();
  return reading;
}

int read_data()
{
```

```
  int data;
  int p_length;

  while (read_reg(0x2) & 0x01)
  {
    Serial.println("busy");
  }
  p_length = read_reg(0x6);
  Wire.requestFrom(ADDRESS, p_length);
  set_reg(0x08);
  for (int i = 0; i < (p_length / 2); i++)
  {
    data = Wire.read() << 8;
    data |= Wire.read();
  }
  return data;
}

void setOffset(void){
  for (int j = 0; j < 10; j++){
    read_lepton_frame();
  }
    read_lepton_frame();

  print_lepton_frameOff();
}
void lepton_command(unsigned int mID, unsigned int cID, unsigned
int com)  {
  Wire.beginTransmission(ADDRESS);

  Wire.write(0x00);
  Wire.write(0x04);

  if (mID == 0x08){
    Wire.write(0x48);
  }
  else{
    Wire.write(mID & 0x0f);
  }
  Wire.write( ((cID << 2 ) & 0xfc) | (com & 0x3));
}


//PIN_SCL  = 12
//PIN_SDA  = 13
//PIN_SCLK = 14
//PIN_CS   = 15
//PIN_DC   = 16
//PIN_RST  = 17

hspiSSD1331 ssd1331(12,13,14,15,16,17);


void setup() {
   Wire.begin();
  Serial.begin(115200);
 isoffset = true;
```

```
  pinMode(5, OUTPUT);
  SPISettings(360000, LSBFIRST, SPI_MODE3);
  SPI.setClockDivider(SPI_CLOCK_DIV2);
  SPI.begin();

  lepton_command(SYS, 0x19>>2 ,GET);
  read_data();
  setOffset();

  isoffset = false;
   ssd1331.init();
   }


void loop() {
  unsigned long tiempoInicial;
  unsigned long tiempoFinal;

  Serial.println("SYS Telemetry Enable State");
  lepton_command(SYS, 0x19>>2 ,GET);
  read_data();
  ssd1331.display_Clear(0,0,95,63);
    while(1){
error1:
    tiempoInicial = millis();
  for (int i =0 ;i<1000; i++){
    read_lepton_frame();
    ssd1331.drawImage(0,0,frameArray,82,60);
    }
    tiempoFinal = millis();
    Serial.println(tiempoFinal-tiempoInicial);
   }
}
```

6.6 Código 5: Funcionamiento multi-core

```
#include <SPI.h>
 //SCLK=18 MOSI=23
const uint8_t cs_OLED=17; //CS (Chip Select)
const uint8_t DCpin=16; //OLED DC(Data/Command)
const uint8_t RSTpin=4; //OLED Reset

TaskHandle_t Task1, Task2;

void setup() {
  SSD1331_Init(cs_OLED,DCpin,RSTpin);

   Serial.begin(115200);

  xTaskCreatePinnedToCore(
    codeForTask1,
    "led1Task",
    1000,
    NULL,
    1,
    &Task1,
    0);
  delay(500);  // needed to start-up task1
```

```
}

void codeForTask1( void * parameter ){
  unsigned long i, j;
  for (;;) {
    long start = millis();


    Serial.print("Finish Task 1 which runs on Core ");
    Serial.print( xPortGetCoreID());
    Serial.print(" Time ");
    Serial.println(millis() - start);
    delay(1000);
  }
}

void loop() {

    Serial.print("Finish Task loop which runs on Core ");
    Serial.print( xPortGetCoreID());

  int i,j,k;
  uint8_t R,G,B,Dot1,Dot2;

  Display_Clear(0,0,95,63);

  CommandWrite(0xAE);
  delay(1000);

  CommandWrite(0xA0);
  CommandWrite(0x0b00110010);
  for (k=0;k<50;k=k){

  for(j=0;j<64;j++){
    for(i=0;i<96;i++){
      R=random(7);G=random(7);B=random(7);
      Dot1=(R<<5)|(G<<2) | B;
      DataWrite(Dot1);
      }
    }
  CommandWrite(0xAF);
  }
  delay(10);

}

void SSD1331_Init(uint8_t CS, uint8_t DC, uint8_t RST){

  pinMode(RST,OUTPUT);
  pinMode(DC,OUTPUT);
  pinMode(CS,OUTPUT);

  digitalWrite(RST, HIGH);
  digitalWrite(RST,LOW);
  delay(1);
  digitalWrite(RST,HIGH);
```

```
  digitalWrite(CS,HIGH);
  digitalWrite(DC,HIGH);

  SPI.begin();
  SPI.setFrequency(5000000);
  SPI.setDataMode(SPI_MODE3);

  CommandWrite(0xAE);
  CommandWrite(0xA0);
  CommandWrite(0b00110010);
  CommandWrite(0xA1);
  CommandWrite(0);
  CommandWrite(0xA2);
  CommandWrite(0);
  CommandWrite(0xA4);
  CommandWrite(0xA8);
  CommandWrite(63);
  CommandWrite(0xAD);
  CommandWrite(0b10001110);
  CommandWrite(0xB0);
  CommandWrite(0x1A);
  CommandWrite(0xB1);
  CommandWrite(0x74);
  CommandWrite(0xB3);
  CommandWrite(0xF0);
  CommandWrite(0x8A);
  CommandWrite(0x81);
  CommandWrite(0x8B);
  CommandWrite(0x82);
  CommandWrite(0x8C);
  CommandWrite(0x83);
  CommandWrite(0xBB);
  CommandWrite(0x3A);
  CommandWrite(0xBE);
  CommandWrite(0x87);
  CommandWrite(0x06);
  CommandWrite(0x15);
  CommandWrite(0);
  CommandWrite(95);
  CommandWrite(0x75);
  CommandWrite(0);
  CommandWrite(63);
  CommandWrite(0x81);
  CommandWrite(255);
  CommandWrite(0x82);
  CommandWrite(255);
  CommandWrite(0x83);
  CommandWrite(255);
  CommandWrite(0xAF);
  delay(110);
  }

void Display_Clear(uint8_t x0, uint8_t y0, uint8_t x1, uint8_t y1){
  delayMicroseconds(500);
  CommandWrite(0x25);
  CommandWrite(x0);
  CommandWrite(y0);
```

```
  CommandWrite(x1);
  CommandWrite(y1);
  delayMicroseconds(800);
}

void CommandWrite(uint8_t b){
  digitalWrite(cs_OLED, LOW);
  digitalWrite(DCpin, LOW);//DC
  SPI.write(b);
  digitalWrite(cs_OLED, HIGH);
}

void DataWrite(uint8_t b){
  digitalWrite(cs_OLED, LOW);
  digitalWrite(DCpin, HIGH);//DC
  SPI.write(b);
  digitalWrite(cs_OLED, HIGH);
}
```