UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

**TRABAJO FIN DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES**

# DISEÑO DE CÁMARA TÉRMICA IR DE BAJO COSTE BASADA EN MICROCONTROLADOR.


AUTOR: FÉLIX NIETO DEL AMOR

TUTOR: ALBERTO JOSÉ PÉREZ JIMÉNEZ


**Curso Académico: 2017-18**

# Resumen.

En el presente Proyecto Final de Carrera se ha desarrollado la implementación, integración y soporte para una cámara térmica, que puede ser de utilidad en áreas donde se requiera la inspección térmica de superficies.

Para su desarrollo, en primer lugar, se ha desarrollado el código de programación para conseguir comunicar un sensor IR, con una pantalla LCD a través de una tarjeta de programación Arduino Mega 2650 rev3. La finalidad ha sido lograr proporcionar al usuario de la misma una interfaz interactiva, así como el desarrollo de aplicaciones que puedan ser útiles durante el uso de la misma. Así, se ha desarrollado una barra térmica, un analizador de imagen y la capacidad de almacenamiento de datos en una tarjeta microSD.

En segundo lugar, se ha implementado un sistema de alimentación autónomo con la finalidad de dar soporte energético integrado a la cámara térmica. Para su realización se ha hecho uso de una batería de Ion-Litio, un convertidor Boost y un sistema BMS. Los objetivos han sido dotar de autonomía al proyecto mediante un sistema recargable de energía.

En tercer lugar, se ha desarrollado el modelado de la carcasa que servirá de soporte a todos los componentes, logrando formar un único sistema mediante el cual el usuario no tenga interacción con los componentes que la conforman, a la vez de dar protección al conjunto. Para su modelado físico se ha realizado su impresión 3D.

Finalmente, notar que pretende ser una alternativa económica ante las diferentes ofertas que oferta el mercado.



**Palabras clave**: Sensor IR, Cámara Térmica, Microcontrolador, Diseño 3D, Impresión 3D, Programación en C.

# Resum.

En el present Projecte Final de Carrera s'ha desenrotllat la implementació, integració i suport per a una càmera tèrmica, que pot ser d'utilitat en àrees on es requerisca la inspecció tèrmica de superfícies.

Per al seu desenrotllament, en primer lloc, s'ha desenrotllat el codi de programació per aconseguir comunicar el sensor IR amb una pantalla LCD a través d'una targeta de programació Arduino Mega 2650 rev3. La finalitat ha sigut aconseguir proporcionar a l'usuari de la mateixa una interfície interactiva, així com el desenrotllament d'aplicacions que puguen ser útils durant l'ús de la mateixa. Així, s'ha desenrotllat una barra tèrmica, un analitzador d'imatge i la capacitat d'emmagatzemament de dades en una targeta microSD.

En segon lloc, s'ha implementat un sistema d'alimentació autònom amb la finalitat de donar suport energètic integrat a la cambra tèrmica. Per a la seua realització s'ha fet ús d'una bateria d'Ion-Litio, un convertidor Boost i un sistema BMS. Els objectius han sigut dotar d'autonomia al projecte per mitjà d'un sistema recarregable d'energia.

En tercer lloc, s'ha desenrotllat el modelatge de la carcassa que servirà de suport a tots els components, aconseguint formar un únic sistema per mitjà del qual l'usuari no tinga interacció amb els components que la conformen, al mateix temps de donar protecció al conjunt. Per al seu modelatge físic s'ha realitzat la seua impressió 3D.

Finalment, notar que pretén ser una alternativa econòmica davant de les diferents ofertes que oferta el mercat.



**Paraules clau:** Sensor ANAR, Càmera Tèrmica, Microcontrolador, Disseny 3D, Impressió 3D, Programació en C.

# Abstract.

In this Final Career Project has been developed the implementation, integration and support for a thermal camera. This camera can use for studying thermal surfaces.

Firstly, programming code has been developed aiming to communicate the thermal camera with LCD screen, through Arduino Mega 2650 rev3 programming board. The purpose of this is to get an interactive interface and to create applications which may be useful for the camera management. For that, has been created a thermal bar, a picture analyser and skill to save information in a microSD card.

Secondly, it has been implemented an autonomous feed system to give energetic support to the camera. It needed an ion-lithium battery, a Boost converter and a BMS system. In fact, the target of this goal has been given autonomy to the project thought a recharging energy system.

Thirdly, a model of case has been created which can use to support all components, getting an only and isolated system. The thermal camera has been made with 3D system.

Finally, this thermic camera can be an economy alternative among different offers which It's in the market.



Keywords: IR Sensor, Thermal Camera, Microcontroller, 3D Design, 3D Printing, C Programming,

Este escrito consta de los siguientes documentos:

# MEMORIA

# PRESUPUESTO

# PLANOS

# ANEXOS

# MEMORIA

# Índice de contenidos

# Índice de ilustraciones.

## Indice de tablas.

# 1. Introducción.

## 1.1. Objetivos.

El principal objetivo de este Trabajo Final de Grado es el desarrollo de una cámara térmica, la cual permita identificar focos de calor dentro de un rango térmico, permitiendo además conocer sobre que temperaturas se encuentra la zona analizada.

Para llevarlo a cabo, se van a plantear una serie de objetivos a cumplir, los cuales, tras alcanzarlos, culminaran con el desarrollo del proyecto y por tanto del objetivo principal. Estos objetivos serán:

- Interconectar por medio de software las distintas partes de hardware que se integran en el proyecto.
- Necesidad de dotar al modelo propuesto de un sistema de alimentación autónoma de energía.
- Dar soporte mecánico al conjunto de componentes del sistema, con el fin de lograr un sistema integrado.

Se plantea la opción de ser un proyecto competitivo como alternativa a nivel de mercado, puesto que otras opciones dentro de este sector pueden llegar a suponer elevados costes.

A nivel personal, mediante este proyecto se han conseguido desarrollar destrezas importantes en el ámbito de la ingeniería, tales como programación de software, integración de sistemas, sistemas energéticos o modelaje 3D a la vez que se ha obtenido una herramienta que puede resultar útil en campos donde se requiera la inspección y análisis de áreas donde se desprenda o se pierda calor, como puede ser el de instalaciones con aislamiento térmico.

## 1.2. Antecedentes.

Este proyecto surge para plantear una alternativa económica en los campos donde se requiera la detección y análisis de focos calientes o fríos en superficies.

Actualmente, como alternativa a la detección gráfica de estas áreas existen termómetros laser, los cuales permiten conocer con precisión la temperatura del área a la que estemos apuntando con el puntero. Sin embargo, para superficies donde existe gran cantidad de zonas calientes, resulta dificultoso la detección de la zona crítica.

De este modo, la cámara térmica resulta una alternativa a este sistema, proporcionando desde una pantalla cual será el foco crítico, sin necesidad de analizar el área punto por punto.

# 2. Conceptos básicos.

En este apartado se van a describir aquellos conceptos básicos, los cuales, mirados con la profundidad necesaria, proporcionan al lector de ésta el conocimiento para entender el desarrollo e implementación de las diferentes partes del proyecto. De esta forma no será objetivo de este capitulo desarrollar aspectos que carezcan de importancia para el posterior desarrollo del proyecto.

Puesto que el proyecto abarca el desarrollo de una cámara térmica, éste trata el desarrollo del software necesario, integración del hardware y alimentación energética autónoma, se dividirá esta introducción atendiendo a un orden funcional.

## 2.1. Sobre hardware y software.

Se introducirá a la placa Arduino Mega 2650 Rev3, TFT Touch Shield, y su entorno de desarrollo.

### 2.1.1. Arduino Mega 2650 Rev3 (1).

Arduino Mega es una placa de desarrollo de código abierto que incorpora una serie de entradas y salidas, analógicas y digitales, la cuales mediante su procesador Atmega2560 permite el desarrollo de sistemas embebidos, mediante un entorno de desarrollo basado en C++.

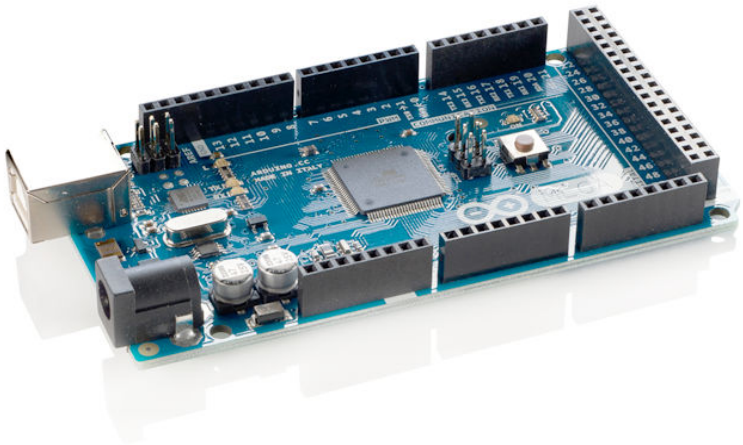


Ilustración 1. Arduino Mega 2650 Rev3.

Algunas de sus características se describen a continuación:

**Alimentación (2):** la tarjeta permite ser alimentada de varias formas. Mediante un conector macho (Power Jack) a una tensión de 7 a 20 V, aunque es aconsejable que sea entre 9 y 12 V para evitar inestabilidades o sobre calentamientos, o mediante una entrada USB, la cual proporciona un voltaje de entrada de 5V. Además, también permite la alimentación mediante los pines los pines de la propia tarjeta, conectando al pin VIn y aun GRN una tensión de entre 6 y 12 V, o por otro lado conectado a un pin de 5V y GRN una tensión de 5V. En cualquier caso, solo la entrada Power Jack nos

proporciona protección contra polaridad inversa y sobrecorriente, el pin Vin nos proporciona la misma protección que el Power Jack salvo contra polaridad inversa, mientras que la alimentación directa por pines de 5V y GRN (tierra) debe de ser totalmente regulada a 5V, esta tensión debe de ser estable ya que alimenta directamente los componentes de la placa Arduino, tampoco ofrece protección contra polaridad inversa.

**Capacidad (3):** este modelo de Arduino nos proporciona unas prestaciones altas en comparación con otros modelos de la misma entidad. Así se adjunta la siguiente tabla a modo comparativo e informativo.

| Característica de Arduino | UNO | Mega 2560 | Leonardo | DUE |
|---|---|---|---|---|
| **Memoria de programa (Flash)** | 32 Kb | 256 Kb | 32 Kb | 512 Kb |
| **Memoria de datos (SRAM)** | 2 Kb | 8 Kb | 2.5 Kb | 96 Kb |
| **Memoria auxiliar (EEPROM)** | 1 Kb | 4 Kb | 1 Kb | 0 Kb |

Tabla 1. Comparativa de Arduino.

La memoria flash nos proporcionará almacenamiento para el código de programa. La memoria SRAM es un tipo de memoria RAM en el que se almacenaran las variables dinámicas de nuestro programa. La memoria EEPROM es un tipo de almacenamiento estático, su uso en este tipo de placas se basa en el almacenamiento de constantes durante el tipo de ejecución del programa, que no queremos que se pierdan al cortar la alimentación.

**Procesador:** el procesador de la tarjeta Arduino Mega es el microcontrolador ATmega2560, el cual su reloj interno proporciona una velocidad de procesado de 16MHz. Dispone de 6 temporizadores, TimerX (donde X se numera del 0 al 5), necesarios para la sincronización en la comunicación con periféricos y para rutinas internas de procesamiento de datos.

En cuanto a los pines de entrada y salida, describiremos aquellos que son de interés para nuestro proyecto:

- **SPI (4):** es un estándar de comunicaciones para conectar la placa con el periférico. Como su nombre indica (*Serial Peripheral Interface)* es basado en un sistema de comunicación serial. Los pines que implementan esta función son 50, 51, 52 y 53.
- **I2C (5):** soporte para el protocolo de comunicaciones I2C (TWI) usando la librería Wire. En la tarjeta Arduino vienen integrados en los pines 20 (SDA) y 21 (SCL).

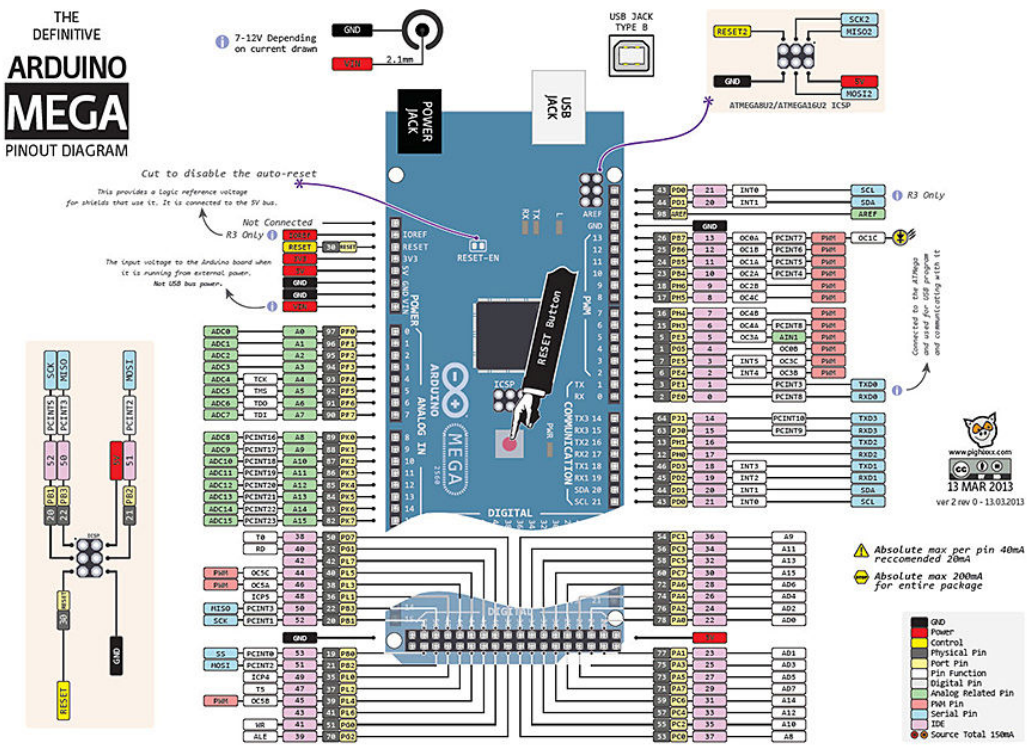



Ilustración 2. Esquema Arduino Mega.

### 2.1.2. TFT Touch Shield (6).

Se trata de una pantalla táctil la cual permite la comunicación con la placa Arduino, haciendo de soporte para la interfaz de usuario. Su resolución es de 320 x 240 pixeles, proporcionando unas dimensiones de 57.6mm*43.3mm, 2.8''. Su formato de color es de 65K RGB, con una interfaz de comunicación de un byte de datos en paralelo. Esto quiere decir que para transferir un color referido a un píxel necesitará 2 bytes, esto es $2^{16}$ = 65536, lo cual se traduce en los anteriores 65K de colores posibles. El controlador de nuestra placa es el LGDP4535. Se debe de tener en cuenta que este nos permite la actualización selectiva de píxeles en la pantalla, esto es, no deberemos de actualizar todos los píxeles de la pantalla para mostrar simplemente una ventana, podremos lanzar directamente esa ventana actualizando solo los pixeles que en ella intervienen.

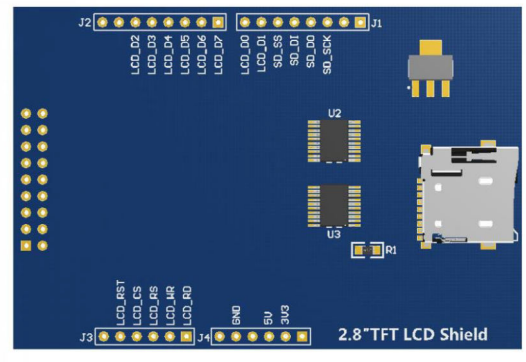



Ilustración 3. Esquema TFT Touch Shield.

Bajo el monitor, se integra un sistema táctil por presión, logrando una gran precisión en su uso a lo largo de la matriz de píxeles y una detección de la presión aplicada en la pantalla, que es transmitido a la tarjeta Arduino. Además, la pantalla también integra un lector de tarjetas microSD.

En cuanto al patillaje de la pantalla, para la transferencia de datos desde Arduino a la pantalla, se dispone de un bus de datos de un byte (LCD_D0 hasta LCD_D7), un bus de control ( LCD_RST, LCD_CS, LCD_RS, LCD_WR, LCD_RD) y unos pines para la alimentación (GRN, 5V y 3V3). En lo referido a la comunicación con la faceta táctil, decir que los pines son compartidos con los anteriores nombrados, en la imagen anterior se correspondería con LCD_CS, LCD_RS, LCD_D0 y LCD_D1. Finalmente, con respecto a la tarjeta SD, sus pines corresponderían en la imagen anterior a SD_SS, SD_DI, SD_D0 y SD_SCK.

Con la pantalla, se proporcionan una serie de librerías que dan soporte a la misma en el sistema Arduino, sin no más que para nuestro modelo tener que hacer una serie de modificaciones que se comentarán en futuros apartados.

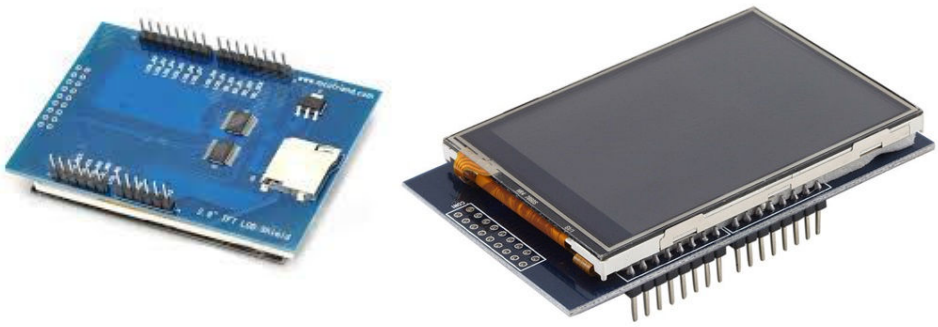
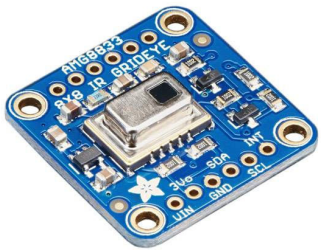


Ilustración 4. TFT Arduino.

### 2.1.3. Adafruit AMG8833 IR Thermal Camera Breakout (7).

Se trata de un sensor térmico de una resolución de 8x8 pixeles, que nos proporciona un total de 64 medidas individuales de temperatura mediante una lectura infrarroja del medio.



Su rango de medida es de entre 0 y 80 ºC, con una precisión de +-2.5 ºC, pudiendo, por ejemplo, poder detectar a un humano a una distancia de 7 m. Tiene una frecuencia de muestreo de 10 Hz, 0.1 segundos, lo cual resulta suficiente para una experiencia con el usuario adecuada.

Ilustración 5. Adafruit AMG8833 IR Thermal Camera Breakout.

Se comunica con la tarjeta Arduino mediante un bus I2C, el cual ya ha sido descrito en apartados anteriores. Sus dimensiones son de: 25.8mm x 25.5mm x 6.0mm.

En cuanto a su patillaje, podemos encontrar:

- Una entrada de 5V para la alimentación, que se corresponde con VIN.
- Una entrada de 3.3V, como alternativa a la alimentación de 5V.
- GND, tierra para la diferencia de potencial y la logica.
- SDA y SCL, conectados a la entradas del I2C de la tarjeta Arduin. SCL para el Clock, proporcionando sincronización y SDA para la transmisión de datos.
- INT, esta salida se activa cuando detecta una variación en alguno de sus datos de lectura cambia, es decir, detectaría si algo se mueve cerca de su campo de visión.

Se proporcionan también una serie librerías que dan soporte e integran la cámara con la placa Arduino, teniendo que hacer unas modificaciones para adaptarla a nuestro modelo de tarjeta.

### 2.1.4. Arduino IDE (8).

Arduino IDE versión 1.8.5 es el entorno de desarrollo que se usará para desarrollar el software necesario para lograr el propósito. Permite el uso de las librerías ofrecidas para el control del hardware implementado en la tarjeta Arduino Mega, de las de la cámara y de la pantalla.

Se trata de un entorno de desarrollo basado en C++, pero incorpora algunas características propias. Los programas desarrollados en este entorno conforman el llamado Sketh IDE de Arduino. El código escrito en este Sketch será lo que posteriormente se suba a la tarjeta Arduino.


Ilustración 6. Sketh IDE de Arduino.

En dicho lenguaje de programación característico de Arduino, se pueden distinguir dos funciones principales nada más iniciar un nuevo proyecto, void setup(){}, dentro de estas comillas irá toda la parte correspondiente a la inicialización de los componentes de hardware y su configuración. Es la parte análoga al lenguaje C++, dentro de la función void main(){}, a la parte que hay

antes del superloop. Mientras que la parte de loop(){}, se corresponde al programa que está continuamente ejecutándose en bucle, y se corresponde con el superloop de C ya mencionado.

## 2.2. Sobre sistema de alimentación.

Los conceptos básicos a comprender en este apartado es el sistema de gestión de batería (BMS), imprescindible si queremos aportar al proyecto un sistema de carga y descarga de energía basado en tecnología ion-litio. Como más tarde se verá se hará necesario el uso de un convertidor Boost, por lo tanto también se darán unas nociones sobre este.

### 2.2.1. Sistema de gestión de batería (BMS).

Un BMS (9), Battery Management System, es un sistema que se encarga de gestionar la carga y descarga de la batería. Entre algunas de las funciones integradas de estos sistemas, se encuentran sistemas de protección contra sobretensiones, sobrecorrientes, protección contra temperatura excesiva, sobrecarga de la unidad de almacenaje o balanceo entre celdas.

Este tipo de sistemas es más común encontrárselo en montajes de baterías en paralelo y en serie, puesto que como se ha comentado antes es necesario un balanceo de cargas para evitar que unas trabajen a más intensidad que otras, pero en las baterías de ion-litio, este sistema es indispensable, puesto que todos los puntos anteriores deben de ser controlados para evitar un posible sobrecalentamiento de la celda, que puede llevarla a arder (10).

### 2.2.2. Convertidor Boost.

Un convertidor Boost (11) es un convertidor DC-DC cuya función es aumentar la tensión de salida con respecto a la entrada. El sistema consiste básicamente en el circuito mostrado a continuación:

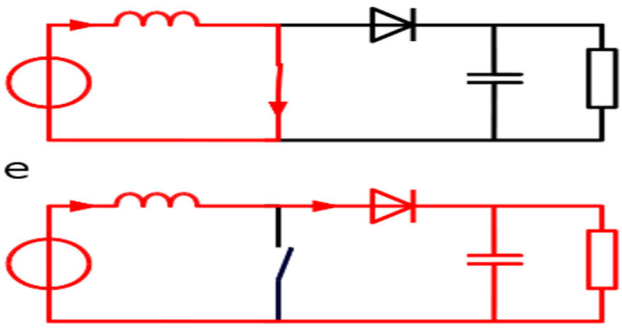



Ilustración 7. Convertidor Boost.

Su funcionamiento consiste básicamente en la conmutación de un interruptor, suele ser un transistor, bajo cierto rango de frecuencias que logra cortocircuitar el sistema, haciendo que la bobina almacene energía cuando este está cerrado y la carga se sigue alimentando por medio del condensador.

Cuando el conmutador se abre, la bobina, que ha almacenado energía se comienza a descargar, haciendo que el condensador se cargue y que le llegue más tensión a la carga. El diodo situado en medio evita que la corriente vaya desde el condensador hasta la batería, cuando el voltaje de este es mayor.

## 2.3. Sobre software de modelado.

La carcasa de la cámara térmica es una parte central del proyecto. Para su modelado y posterior impresión se va a utilizar como entorno de trabajo AutoDesk Inventor. Este software ofrece al usuario un conjunto de herramientas para el modelado 3D de la maqueta. Desde esta herramienta es posible hacer un examen preliminar sobre el futuro montaje de proyecto, con el fin de conseguir que todas las piezas encajen correctamente y lograr un buen grado de compactación. Además, a partir del modelo 3D, será posible obtener los planos.

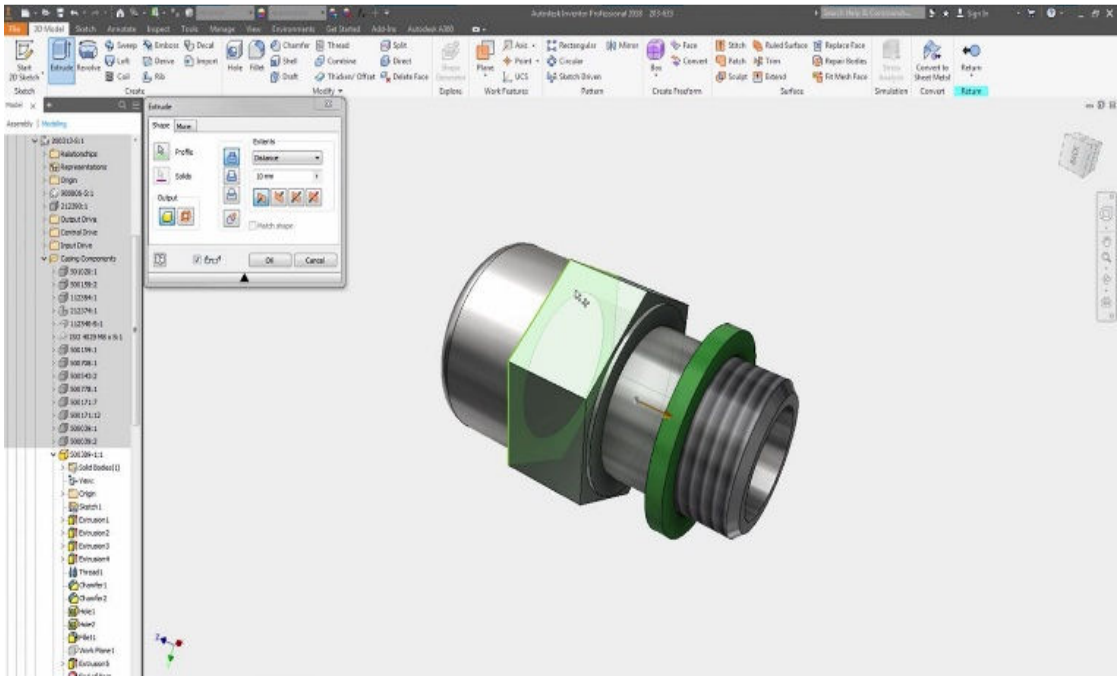


Ilustración 8 Software de modelado.

# 3. Desarrollo.

Este apartado abarca el desarrollo del proyecto. Se seguirá un orden cronológico, entendiendo que lo posterior no puede ser posible sin haber finalizado el paso anterior.

## 3.1. Ensamblaje de tarjeta, cámara y pantalla.

Antes de comenzar con el desarrollo del software, es necesario hacer un breve inciso sobre el montaje de estos tres componentes. Así, se comenzará con el ensamblaje de la pantalla sobre la tarjeta Arduino Mega.

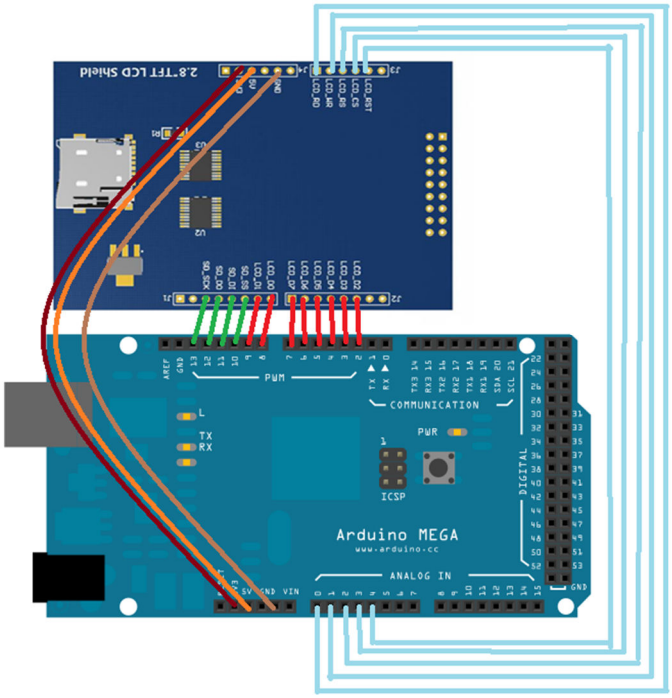



Ilustración 9. Unión TFT Arduino

El montaje se realiza mediante la inserción de un patillaje sólido como el que se muestra en la ilustración 10, puesto que la distribución de los pines de la pantalla está orientada a que encaje directamente en una tarjeta Arduino Uno.

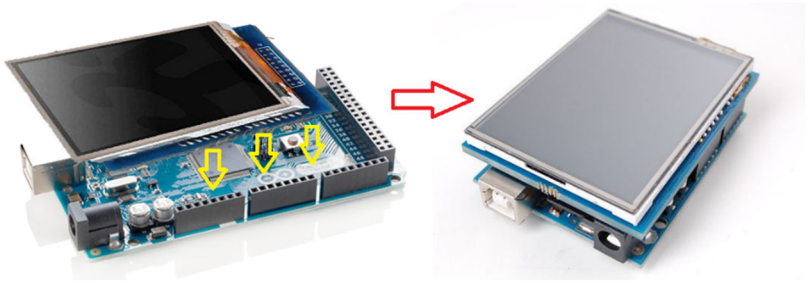


Ilustración 10. Montaje TFT Arduino.

Ahora se avanza al montaje de la cámara térmica con la placa Arduino.

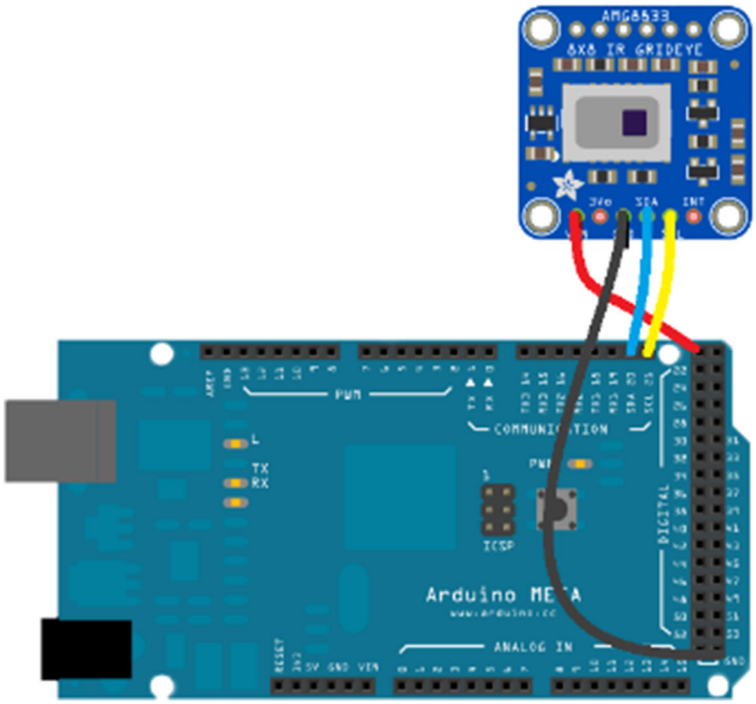



Ilustración 11. Montaje cámara Arduina

La comunicación se hace mediante un bus I2C, la alimentación mediante la entrada de 5V de tensión (cable rojo) y la tierra común (cable marrón).

### 3.2. Desarrollo del software.

Este apartado, en primer lugar, se centrará en el tratamiento y manipulación de los datos obtenidos por la cámara para, posteriormente, mostrarlos por la pantalla, de forma que el usuario pueda obtener una representación, lo más intuitivamente posible, de lo que se muestra por pantalla, a través de una interfaz atractiva para el mismo.

Además, se abordará el desarrollo de diferentes utilidades con el fin de mejorar las prestaciones que la cámara en sí puede aportar al usuario.

Así, se va a proceder de forma secuencial con el desarrollo de lo planteado.

#### 3.2.1. Comunicación cámara-pantalla.

Para comenzar, pantalla y cámara no se conectan directamente, entre ellos es necesario que intervenga un mediador, en este caso la placa Arduino, que se encarga, a partir de los datos obtenidos por la cámara, procesarlos, y mandarlos a la pantalla, para mostrar al usuario una representación de lo que la cámara está captando.

#### 3.2.1.1.Comunicación cámara-Arduino.

La cámara térmica que se está usando para el proyecto, tiene una resolución de 8x8 píxeles, es decir, es capaz de entregar a la placa un total de 64 valores diferentes, cada uno representante de la temperatura que rodea a la visión de dicha cámara. Estos valores que recibe la cámara Arduino, tras su procesado a través de las librerías que nos ofrece el fabricante, llega en forma de un vector de temperaturas, tal como se muestra en la siguiente imagen. Para que esto sea posible, primero se debe configurar Arduino para que pueda detectar la transmisión de datos por el puerto I2C, esto puede consultarse en el Anexos A.2.

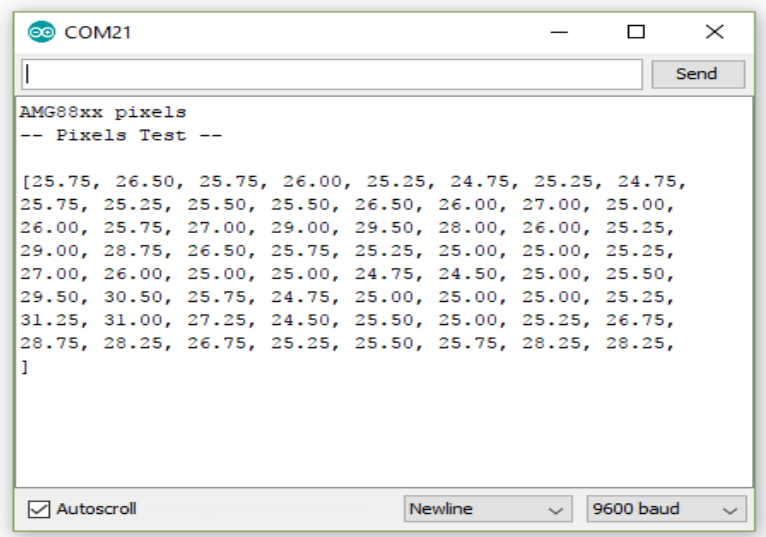



Ilustración 12. Matriz de temperaturas.

Antes de comenzar, tener en cuenta que los pixeles se organizan en filas de 8 en la cámara, formando un total de 8 filas. La imagen anterior, por lo tanto, en orden, cada grupo de 8 representa una fila de dicha matriz, apilándolas hasta un total de 8, formando así la matriz de 8x8 píxeles.

#### 3.2.1.2. Comunicación pantalla-Arduino.

Pará lograr comunicar la pantalla con Arduino, es necesario configurar este último, estableciendo entre otras cosas los pines listos para la comunicación con la pantalla. Esto viene implementado en las librerías que nos proporciona el fabricante de la pantalla, teniendo el usuario no más escribir unas líneas de código. Esta configuración podemos verlas en los Anexos A.2 y E.3. Ahora, se continuará por donde se había dejado.

### 3.2.2. Representación por pantalla.

El primer paso podría ser mostrar por pantalla, usando las librerías que proporciona el fabricante de la pantalla, cada una de las temperaturas, asignándole a cada temperatura un color distinto. El rango de colores irá desde un tono violeta hasta rojo, pasando por azul, verde, amarillo y naranja,

de temperaturas más frías a más cálidas respectivamente (se puede ver el vector asociado a estos colores, codificados en 16 bits, en el Anexo A.6).
Sin embargo, lo que se obtiene en este caso es un cuadrado de 8 x 8 píxeles en una pantalla de 320 x 240 píxeles. Esto es prácticamente inapreciable para el usuario, luego el siguiente paso será asignar a cada pixel tomado por la cámara, un tamaño tal que entre todos ellos se ajuste lo máximo posible al tamaño de la pantalla. La solución está en asignar a cada pixel, un cuadrado de 30x30 pixeles, obteniendo algo parecido a lo que se muestra a continuación:

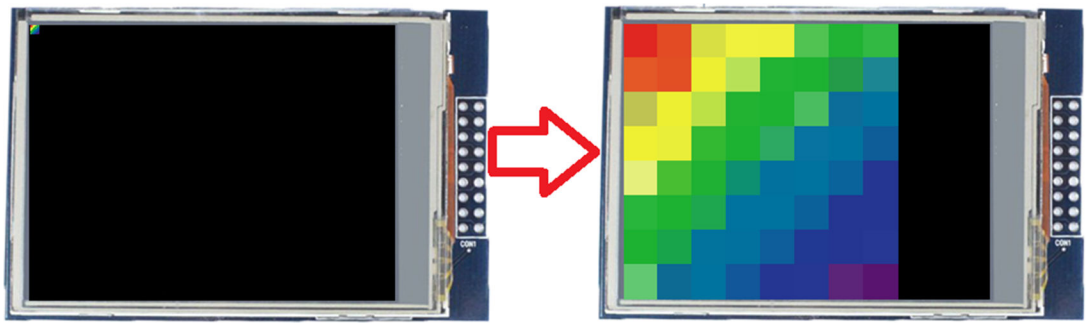


Ilustración 13. Adaptación a pantalla.

En este ejemplo, se puede ver lo inapreciable que resulta la imagen de la izquierda, frente a una imagen completamente visualizable a la derecha. En la parte superior derecha, tendríamos la temperatura más alta, descendiendo así hasta la temperatura más baja, en la parte inferior izquierda.

Sin embargo, llegados a este punto, se aprecia que la imagen sigue manteniendo un alto grado de pixelado, pues se encuentra con bloques de 30x30 pixeles, monocromáticos. La solución para conseguir una transición más suave entre los píxeles contiguos, será la interpolación. Es decir, con el objetivo de conseguir transiciones más suaves de pixel a pixel, se puede encontrar una solución si se parte de un vector de temperaturas superior a las 8x8 que proporciona la cámara. Así pues, para llevarlo a cabo, se comenzará realizando interpolaciones entre dichas temperaturas, teniendo en cuenta los cuadrados contiguos y las condiciones de contorno. Realizar una interpolación para pasar de 8x8 a 240x240, conllevará un gran gasto computacional, alargando los tiempos de computo a márgenes que no permiten llevar al usuario a la obtención de transiciones entre imágenes fluidas (recordar que el tiempo de refresco de la adquisición del vector de temperaturas está en torno a los 0.1s). Realizando pruebas con esta interpolación, lleva a tiempos de entre refresco y refresco de cerca de 3s. Para evitar esto, se interpolará de 8x8 a 24x24, teniendo que poner para completar los 240x240 de la pantalla bloques de 10x10 pixeles. Así, conseguimos una transición bastante rápida, al tiempo que se mejora notablemente la imagen mostrada al usuario. En la ilustración 14 se pueden observar los resultados obtenidos:

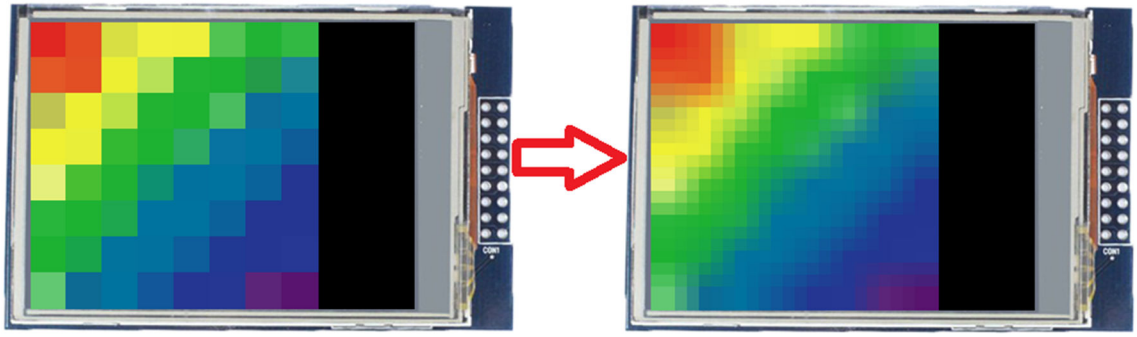


Ilustración 14. Interpolado.

En este caso, se debe hacer notar que el software que proporcionaba la interpolación iba incluido en un sketch de un ejemplo del fabricante de la cámara. En este punto, el trabajo realizado ha sido de adaptación del código para su encapsulación y consecución de un código más modular. Puede consultarse parte del código en el Anexo B.

Por último, cabe decir, que aunque se acelera el proceso interpolando solo entre 24 en lugar de 240 píxeles, el proceso sigue tardando tiempo en generar la imagen en sí. Esto es debido a que las operaciones que tiene que realizar el procesador, tanto para interpolar como para enviarlo a la pantalla para su representación consumen mucho volumen de computo. En total, entre una imagen y la siguiente se tarda 1.05 segundos (una frecuencia de 0.95 Hz). Si se quisiera obtener una transición lo más fluida posible, tendríamos que reducir el tiempo de computo al tiempo que tarda la cámara térmica de actualizar la información, esto es 0.1s. Dado que la tarjeta Arduino Mega 2560 r3 tiene un procesado que trabaja a una velocidad de 16 MHz y deberíamos de reducir la velocidad de 1.05 s a por debajo de 0.1 s, esto lleva a tener que aumentar la velocidad de procesado en 10 veces la que ofrece la tarjeta que se está utilizando, es decir, se necesitaría un procesador que trabaje a más de 160 MHz. De esta forma, echando una ojeada a lo que actualmente nos ofrece el mercado, Arduino Due es la opción que más se acerca, aunque se queda lejos, pues su procesador trabaja a 84 MHz. Otra opción que satisfarían sobradamente esta demanda de velocidad, sería por ejemplo Raspberry Pi 3 Model B, cuya velocidad de procesado es de 1.4 GHz.

### 3.2.3. Desarrollo de utilidades.

Este apartado se centrará en el desarrollo de diferentes aplicaciones para conseguir que la cámara sea un instrumento realmente útil y manejable para el usuario.

### 3.2.3.1. Barra térmica.

Con el fin de conocer la temperatura que se muestra en cualquier punto de la pantalla, puesto que se ha asignado a la temperatura un color determinado, será de utilidad tener un indicador que nos informe en todo momento de a que temperatura se encuentra cualquier punto de la pantalla, por comparación con los colores que presenta la barra térmica.

En primer lugar, se ha tenido en cuenta su posicionamiento en la pantalla. Puesto que se ha elegido en el apartado anterior la posición de la imagen mostrada pegada al borde, la barra, ya que funciona por comparación será útil colocarla lo mas cerca posible de la imagen mostrada.

Además, se hará necesario mostrar el rango de temperaturas superior e inferior que nos proporciona la cámara térmica.

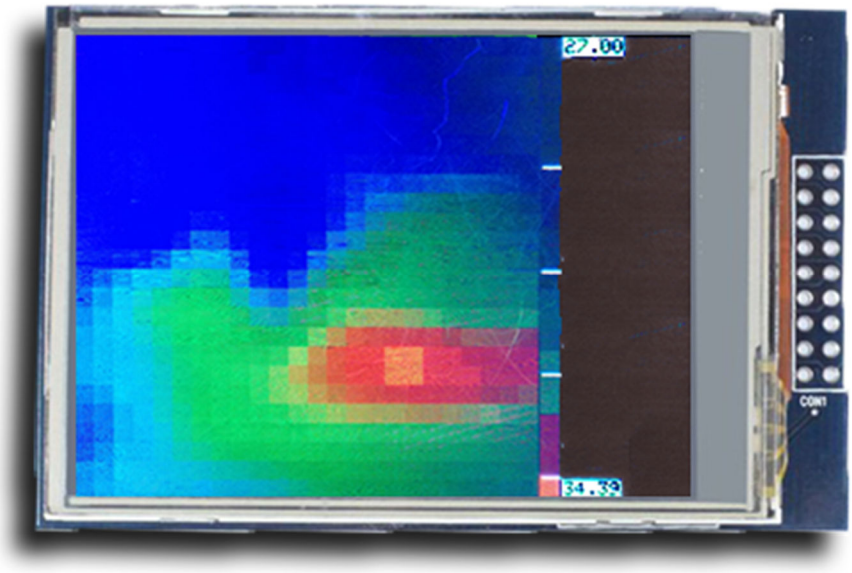



Ilustración 15. Brra térmica.

Como se puede observar, también se ha incorporado unas marcas blancas en la barra térmica, en el fin de escalar los colores de la misma.

En este punto, se debe tener en cuenta que los colores siempre se asignan del más cálido que detecta al más frio. Es decir, tanto la imagen que se muestra como la barra, se le asigna el mayor rango de colores posibles, si la temperatura mínima son 20ºC o sean 30ºC siempre se le asigna el color violeta, si la temperatura máxima es 60ºC o 50ºC, siempre se le asignará un color rojo. Tener en cuenta que el fabricante proporciona este rango de temperaturas fijo, luego resulta un cambio en el código proporcionado por el fabricante, que ya ha sido aplicado en el código de interpolación (Anexo B), y que ya se incluye en lo mostrado por pantalla de la imagen anterior. Más adelante se verá otra función aplicada a la cámara relacionada con esta distribución de colores. (ver apartado 3.3.2.3).

Puede verse la implementación de este código en el Anexo D, especialmente en D.5, donde se desarrolla la barra térmica como tal. Su lanzamiento puede verse en el Anexo C.3.

#### 3.2.3.2. Verificación de temperatura.

Dado que por comparación de colores es difícil hacerse una idea de tener una aproximación fiable de la temperatura que se muestra por pantalla, en este apartado se desarrollará una aplicación que permita, mediante la utilización de la pantalla táctil, seleccionar cualquier punto de la imagen mostrada por pantalla, y que se muestre, tanto la franja que ocupa en la barra térmica, como la temperatura exacta a la que se encuentra.

### Calibrado táctil

Para lograr este objetivo, se ha de hacer uso de las librerías que ofrece el fabricante para el uso de la función táctil de la pantalla (la inicialización del táctil puede consultarse en los Anexos A.2 y E.3). Así, calibrando las puntos que se obtienen mediante el uso de las librerías mencionadas, se puede obtener un sistema de coordenadas, donde cada píxel representa una unidad en el eje dirección x e y, tal como muestra la ilustración 16.

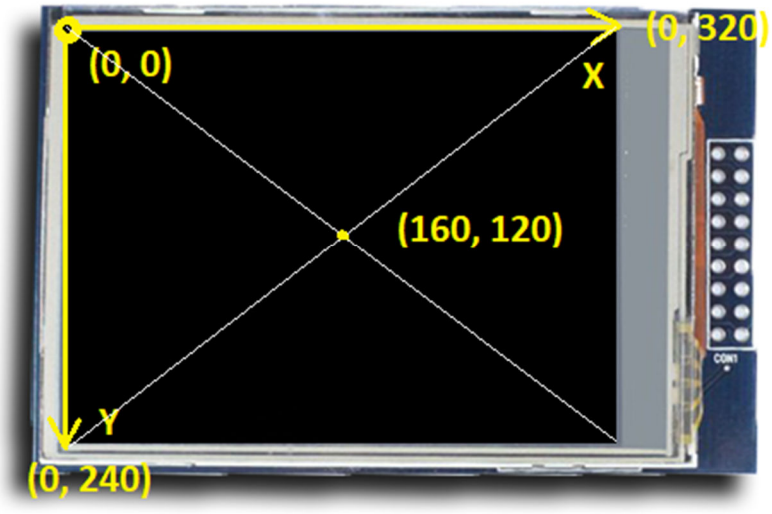



Ilustración 16 Calibrado táctil.

Dado que la imagen tarda cierto tiempo en actualizarse, se actualiza con una frecuencia de 1.05 segundos, durante la actualización el procesador no reconocerá que se está pulsando sobre la pantalla táctil. El problema será solucionado en el siguiente apartado, de momento se concluye con la explicación de la implementación de este apartado. Bastará decir que cuando se pretende usar esta función, la imagen deseada que se quiere analizar se congela en la pantalla.

Así, cuando se pulsa en cualquier punto del área delimitada por la imagen, el controlador del táctil de la pantalla devuelve al procesador de la tarjeta Arduino unas coordenadas exactas, x e y, tal como se muestran en la ilustración 16. Así, por medio de un proceso inverso al que se realiza para mostrar la imagen por pantalla, es posible obtener a partir de las coordenadas x e y, y del vector que almacenaba las temperaturas interpoladas, la temperatura exacta del punto que ha sido pulsado. Una vez obtenida dicha temperatura, se programa un indicador que se coloque en la posición que le corresponde en la barra térmica.
La implementación del código se puede observar en el Anexo E.7. A continuación se muestra el resultado obtenido:

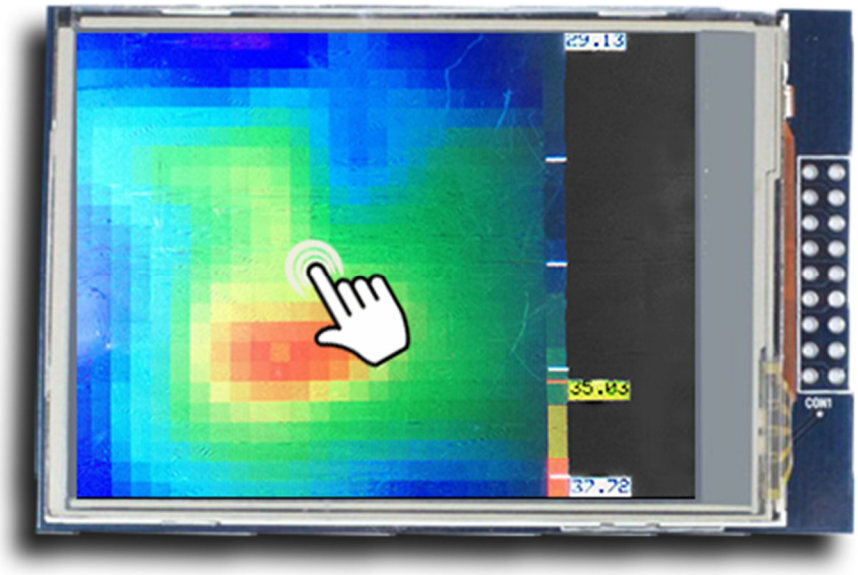



Ilustración 17. Analizador.

### 3.2.3.3. Modos de contraste.

En este apartado se va a desarrollar dos modos de contraste de la imagen que se muestra por pantalla. La explicación de este apartado viene dada porque cuando se trabaja entre rangos de temperaturas cortas, si se mantienen fijas las temperaturas mínimas y máximas a las cuales se establece el rango de colores, en la representación de la imagen captada no se hace apreciable las diferencias de color. Esto se puede observar en la siguiente imagen comparativa (ilustración 18), donde se ha preestablecido el rango de temperaturas entre 0ºC y 80ºC.

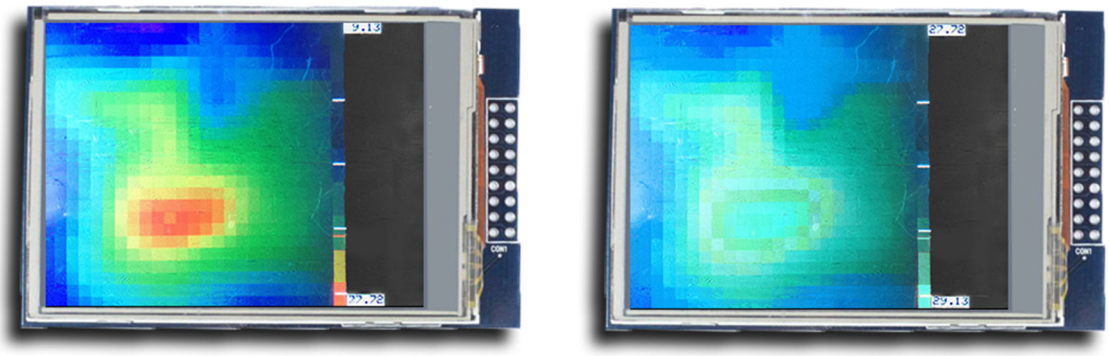


Ilustración 18. Modos de contraste.

A la de la izquierda, el rango de temperaturas que capta la cámara está entre 9ºC y 77ºC, como se muestra en la imagen, mientras que a la derecha está entre 27ºC y 29 ºC. Como se ve, no se puede distinguir bien el area mas caliente, en comparación con la imagen de la derecha. Para solucionarlo, se ha implementado dos modos de contraste. En uno, el rango de temperaturas está predefinido, y va de 0 a 80 ºC. Este modo es útil para cuando se quiere tener una escala común de temperaturas, sirviendo de parámetro de referencia entre varias imágenes. Otro modo desarrollado, es el llamado de alto contraste. En este modo, se le asigna a la temperatura más baja captada por la cámara el color más azulado, mientras que a la temperatura más alta se le asigna el color más rojizo. Es útil para distinguir zonas más calientes y más frías en la imagen cuando los rangos de temperatura son muy débiles. La implementación de este código puede verse durante todo el Anexo A y en el Anexo E.6

### 3.2.3.4 Almacenamiento de imágenes.

**Inicialización de tarjeta**

Este apartado se centrará en el almacenamiento de la imagen retenida mediante el sistema del apartado anterior en una tarjeta microSD. Para comenzar se debe de realizar unas modificaciones en las librerías que la IDE de Arduino proporciona para manejar la conexión con tarjeta microSD. Esta modificación radica en que con la conexión pin a pin que hemos realizado entre la tarjeta Arduino y la pantalla, la cual integra el lector de tarjetas, no conecta adecuadamente la parte del hardware necesario para los pines del lector. Esto es, dicha distribución compacta de pines esta prediseñada para la tarjeta Arduino Uno, el cual tiene su bus SPI que utiliza el lector para comunicarse en los pines 10, 11, 12 y 13. Sin embargo, estos pines no corresponden al bus SPI en Arduino Mega, que en su caso están en 53, 51, 50, 52.

Por lo tanto, las propias librerías ofrecen una solución para poder mantener esta distribución compacta en Arduino Mega. Consiste básicamente en

simular en los pines 10, 11, 12 y 13 un puerto SPI, mediante el cual lograr la comunicación con la tarjeta.

**Creación de archivo de imagen.**

Para poder extraer la imagen, en formato BMP (11), que se está mostrando por pantalla a la tarjeta microSD, se debe remontarse a donde se tiene almacenados los valores que se muestran por pantalla, esto es el vector de valores interpolados de temperaturas al que hacemos referencia en el apartado 1.2. A partir de este vector, por medio de un proceso similar al que se realiza la muestra por pantalla de los colores asociados a cada temperatura, se consigue obtener que color pertenece a cada temperatura.

Debemos de tener en cuenta, que, en este caso, crear la imagen no se realizará de la misma forma que lo hacemos cuando mostramos por pantalla y, sin embargo, se deben obtener los mismos resultados. Cuando mostramos por pantalla, cada color asociado a cada temperatura simplemente lo mostramos creando un cuadrado de 10x10 píxeles, utilizando de las librerías proporcionadas por el fabricante para crear dichos cuadrados. Para este caso, tenemos que crear el fichero BMP, de acuerdo a como lo establece el formato.

Se debe tener en cuenta que este fichero no solo necesita de la escritura del mapa de colores, el fichero necesita que le introduzcamos una cabecera con la información necesaria. Para ver la implementación de este código, se tiene que consultar el Anexo F.1 .Además, se puede añadir que se ha desarrollado el código necesario para mostrar en la misma imagen la barra térmica correspondiente con la imagen (Anexo F.6) También comentar que se ha implementado un sistema que avisa al usuario de que la tarjeta microSD no está introducida y evita que se lance la aplicación de escritura, que puede originar cuelgues en el sistema.

El código referido a la implementación de esta utilidad puede observarse en el Anexo F.7.

**Resultados finales.**

A continuación, se pueden ver algunas imágenes extraídas por medio de este sistema, comparadas con la misma imagen en la cámara térmica (ilustración 19.)

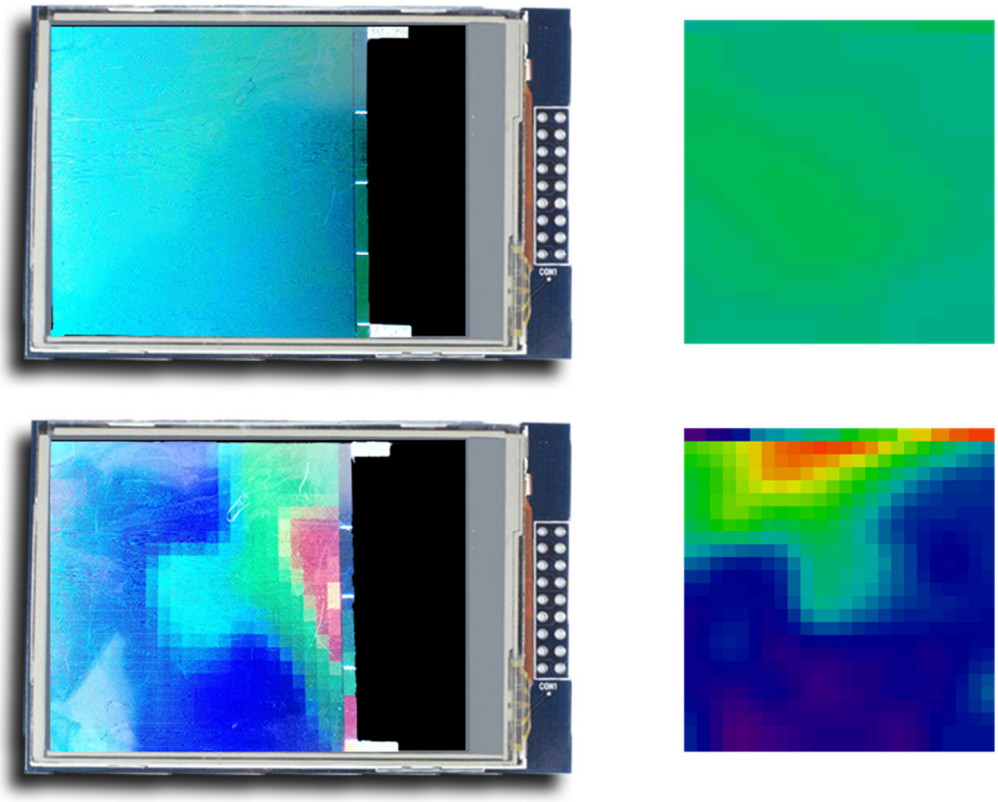


Ilustración 19. Ejemplos de acabado de imagen en BMP.

### 3.2.4. Software para detección de eventos.

Este apartado abarcará el desarrollo del software necesario para la detección de eventos que fuercen al procesador a realizar la tarea que se le indica. Esto se conseguirá mediante interrupciones al sistema. Pueden plantearse dos formas de lograr estas interrupciones. Por un lado, integrar un sistema que detecte una señal externa (13), esto puede ser un pulsador, conectado a cualquiera de los pines capaces de provocar una interrupción a la CPU y a partir de ello habilitar el modo táctil de la pantalla para a través de ella indicar que acción queremos realizar. Por otro lado, un sistema de interrupciones temporales (14), que cada intervalo de tiempo interrumpa la rutina ejecutada por la CPU y permita verificar si la pantalla táctil ha sido pulsada, llevando directamente al evento seleccionado.

Antes de decantarse por una opción, se proponen un conjunto de pros y contras para cada uno. Las interrupciones externas permiten que durante el funcionamiento normal del sistema, este se ejecute a la máxima velocidad, dado que las interrupciones temporales cada vez que se ejecutan suponen un coste computacional. Por otro lado, las interrupciones externas suponen la necesidad de implementar un añadido en el hardware, en este caso un

pulsador, mientras que las temporales no es necesario, pudiendo implementarse con el hardware disponible hasta el momento.

Así pues, para evitar la necesidad de integrar un elemento nuevo al sistema, se ha decidido optar por el desarrollo de interrupciones temporales. Estas se realizan mediante una biblioteca que permite el uso del Timer1 de Arduino Mega, sin la necesidad de acceder a los registros de este. Las interrupciones se realizarán cada 0.1 s y en su rutina de servicio (ISR), se implementará el código que detecte si se ha pulsado la pantalla. Si ésta es pulsada, se enviará a un modo pausa, donde es posible el uso de la interfaz de la pantalla, pudiendo seleccionar otra tarea.

Para la implementación de esto, se debe de tener en cuenta que los pines que controlan la comunicación la pantalla, son compartidos con los que controlan la comunicación del táctil con la tarjeta. Luego habrá que evitar que el sistema se interrumpa durante la comunicación tarjeta-pantalla, con el fin de no provocar errores de escritura. Se llevará a cabo deshabilitando las interrupciones de la CPU durante cualquiera de estos procesos de escritura en la pantalla, y posteriormente habilitándolas.

La implementación de este sistema se hace en varias partes del código, pero donde se inicializan las interrupciones y se pone en marcha el Timer1 es en el Setup (ver Anexo A.2). Gran parte de la posterior sincronización entre tareas se realiza directamente en el Loop (ver Anexos A.3)

### 3.2.5. Interfaz.

La interfaz de usuario es un aspecto importante a desarrollar, pues es lo que el usuario de la cámara maneja y observa constantemente. Para su implementación, se ha hecho uso de las librerías que el fabricante proporciona para la creación de formas y la representación de texto por pantalla. Además, se debe de tener en cuenta que la interfaz se lanzará directamente en el Setup del sketch.

#### 3.2.5.1. Selector de contraste.

En primer lugar, se desarrolla la interfaz que permite elegir entre modos de contraste, ésta es la primera pantalla interactiva que se lanzará una vez encendida la cámara. Para la elección de cualquier modo simplemente se tiene que pulsar sobre él en la pantalla:


Ilustración 20. Selector de contraste.

C: modo de alto contraste, rangos ajustables. R: modo real, rangos fijos de 0 a 80ºC

### 3.2.5.2. Interfaz de usuario para selección de tareas.

En segundo lugar, se desarrolla la interfaz que permite el acceso a los distintos modos que se han desarrollado. Consiste en un conjunto de botones que darán acceso a los diferentes modos. Para su implementación debe de parametrizarse la región de pixeles que ocupa cada botón y cuando se pulsa en esa región, lanzar la aplicación prediseñada.

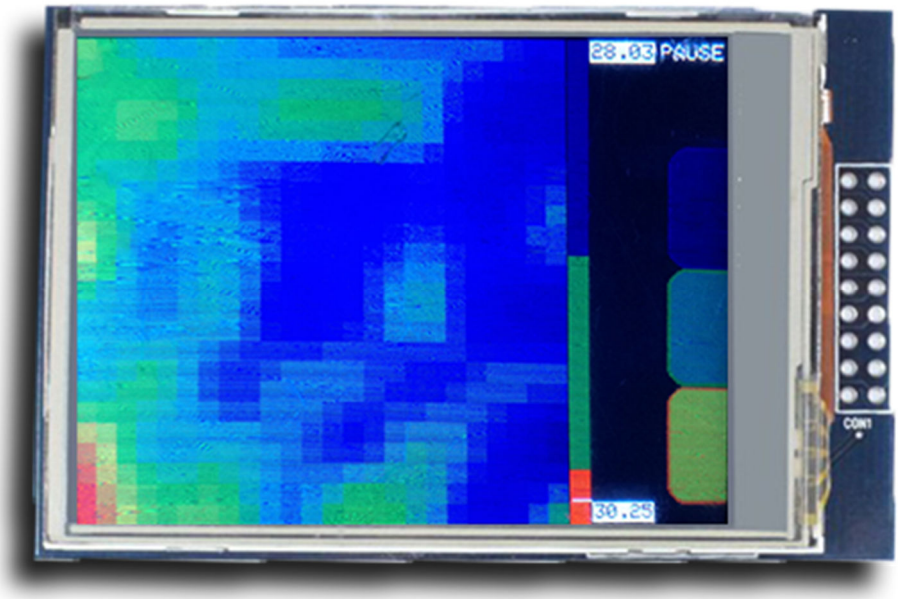



Ilustración 21. Interfaz de usuario para selección de tareas.

### 3.2.6. Distribución final del código

Con el fin de clarificar la lectura del código creado, y de darle una organización funcional para futuras posibles ampliaciones y actualizaciones, se ha dotado de la siguiente estructura al Sketch de Arduino. Además, tras este primer esquema (ilustración 22) se muestra un diagrama de flujo (ilustración 23) sobre el ciclo que cumple el programa, en función de las tareas que se realizan en cada momento.

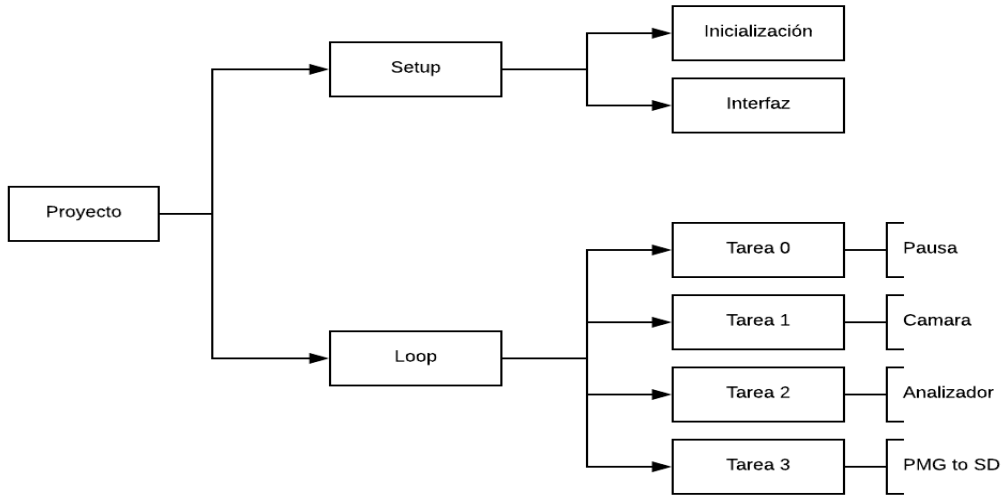



Ilustración 22. Esquema de código.

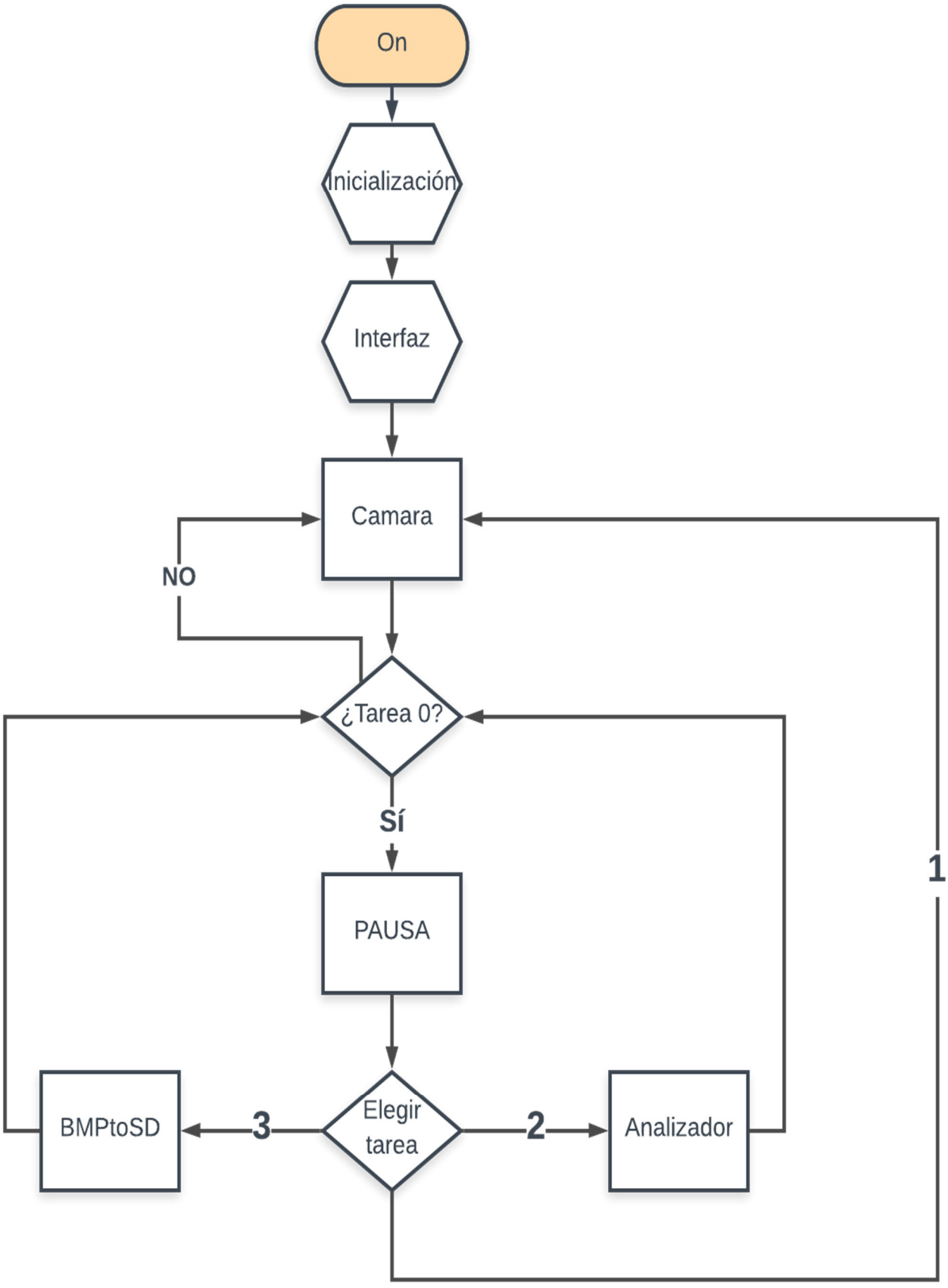



Ilustración 23. Diagrama de flujo de código.

## 3.3 Desarrollo sistema de alimentación autónomo.

El sistema de alimentación es una parte indispensable del proyecto. Con ello se podrá utilizar la cámara sin necesidad de estar conectada a una fuente de alimentación fija. El objetivo será que esta fuente de alimentación fuera de recargable, económica y con un puerto para su carga lo más universal posible, sin necesidad de adaptadores. En primer lugar, se realizará la medición de la potencia consumida por la pantalla, cámara y tarjeta Arduino, ya implementada con el código desarrollado, en su conjunto. Después, se pasará a la elección de los componentes que se adecuen a las demandas exigidas, tanto de potencia por parte de la tarjeta como de tamaño, duración y opciones de carga. Tras esto, se diseñará el esquema de montaje para este sistema de alimentación. Finalmente, se mostrarán los resultados obtenidos.

### 3.3.1. Mediciones.

Las mediciones son imprescindibles para la elección de la tecnología que se va a elegir para el sistema. Para realizarlas se usará de un polímetro, mediante el cual medimos la intensidad y tensión que es suministrada a nuestra tarjeta Arduino. Las mediciones se realizarán en los diferentes modos que se han desarrollado en software, para ver cuáles son las demandas pico de la tarjeta. Notar que las mediciones se han realizado a través de un pin 5V y un GRN (tierra), mediante una fuente estable la cual suministra 5V a la tarjeta. En estas condiciones, los resultados son los siguientes (tabla 2):

| Modo | Voltaje (V) | Intensidad (A) | Consumo (W) |
|---|---|---|---|
| Cámara | 5 | 0,21 | 1,05 |
| Pausa | 5 | 0,20 | 1 |
| Analizador | 5 | 0,21 | 1,05 |
| Pmg to SD | 5 | 0,21 | 1,05 |

Tabla 2. Mediciones iniciales.

Como se puede observar, el consumo permanece prácticamente invariable independientemente del modo que se esté utilizando (tabla 2). Así pues, en función de los datos obtenidos experimentalmente, se concluye que el proyecto consumirá 1,05 W (tabla 2).

### 3.3.2. Elección de componentes.

Una vez realizadas las mediciones oportunas y obtenidos los parámetros mínimos que debe de cumplir el sistema de alimentación, se avanza a la elección de componentes que lo conformarán.

#### 3.3.2.1. Batería.

En primer lugar, se elige el tipo de tecnología que usar en cuanto a la batería. Viendo las alternativas del mercado (15), la opción más atractiva es la batería de Ion-Litio, pues tiene una alta capacidad y su coste, aunque superior al de otras tecnologías, es amortizable por la gran cantidad de ciclos de recarga que posee en comparación con las demás y las exigencias de voltaje mínimo son las que más se acercan al requerido. Se ha elegido el modelo NCR18650B de Panasonic (ilustración 24), Como se puede observar en la hoja de especificaciones técnicas (16), la tensión no se mantiene fija durante la descarga, si no que varía en desde 4.2 V a plena carga, a 2.5 V con la batería vacía. Además, para alimentar Arduino se puede proceder por diferentes maneras


Ilustración 24. Batería Panasonic modelo NCR18650B

#### 3.3.2.2. Convertidor Boost.

Para evitar los bajos rendimientos que se consiguen con el regulador de tensión de 6 a 12 V, el cual Arduino lleva integrado, se va a proceder a alimentarlos por uno de los pines de 5V que incorpora la placa. Para ello, dado que además la tensión de entrada tiene un amplio rango de variación dependiente del estado de carga, se solucionará con un convertidor DC-DC Boost, que permitirá obtener una tensión de salida estable a 5V, aún variando la tensión de entrada.


Ilustración 25. Convertidor Boost.

### 3.3.2.3. Entrada de tensión.

No solo va a ser necesario regular la tensión de salida, también se va a tener que regular la alimentación de la batería, así como proporcionar un sistema de soporte para la carga de la misma y un sistema de seguridad. Para esto, se va a utilizar un sistema BMS, en concreto la placa TP4056 con TE420. Esta placa, proporciona un sistema de carga mediante puerto micro-USB a partir de una tensión de entrada de 5V. Este puerto y dicha tensión es la que proporcionan gran cantidad de los cargadores de dispositivos móviles actuales. Además, también proporciona un sistema de protección contra sobrecorrientes y sobretensiones y corta el flujo de corriente cuando la tensión de la batería sobrepasa su límite mínimo. En los siguientes puntos se pueden ver las características (17) más detalladamente:

- Corriente máxima de carga: 1 A.
- Protección contra sobredescarga: 2.5V.
- Protección contra sobrecorriente: 3A.
- Tensión de entrada: 4.5V-5.5V con una precisión de carga de 1.5%.

Tener en cuenta para el cableado y montaje que la entrada de tensión a la batería y la alimentación de la tarjeta Arduino se hacen directamente a través de las salidas que ofrece este BMS. El modelo elegido se observa en la ilustración 26.

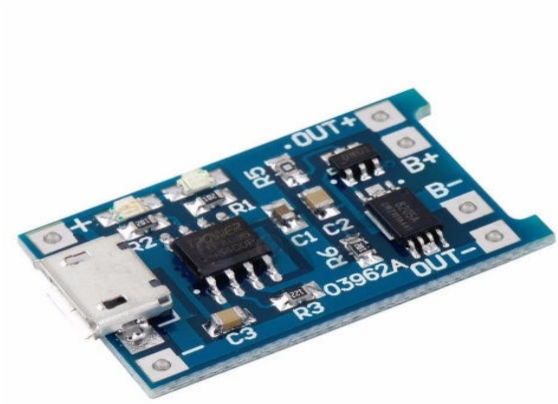



Ilustración 26. TP4056

### 3.3.2.4. Otros.

Se tiene que proporcionar un sistema para apagar y encender la cámara. Este sistema será un interruptor deslizante, que se conectará en uno de los cables que conectan la salida del BMS con la entrada de tensión de la tarjeta Arduino. Para la unión de los componentes, se ha decidido usar un cable AWG 26 (ilustración 27).

Ilustración 27. Interruptor deslizante y cable.

### 3.3.3. Intensidades admisibles estimadas.

Para unas condiciones de uso normales la potencia es de 1,05V, se considera que la tensión en la pila es de 3.7V. Para obtener un porcentaje de rendimiento estimado, se supone que los componentes adiciones que se usen en el sistema tienen un rendimiento conjunto del $\eta_c$ = 90 % (estos sistemas son BMS, cableado e interruptor). Se supone 90% como un porcentaje peyorativo, porque teóricamente los elementos nombrados tienen eficiencias prácticamente cercanas al 100%. Sin embargo, el convertidor Boost sí tiene variaciones en la eficiencia. Lamentablemente el vendedor no proporciona las tablas (18) necesarias para poder obtener los rendimientos necesarios, luego se toma un valor aproximado en función de las tablas que nos ofrece (ilustración 28):

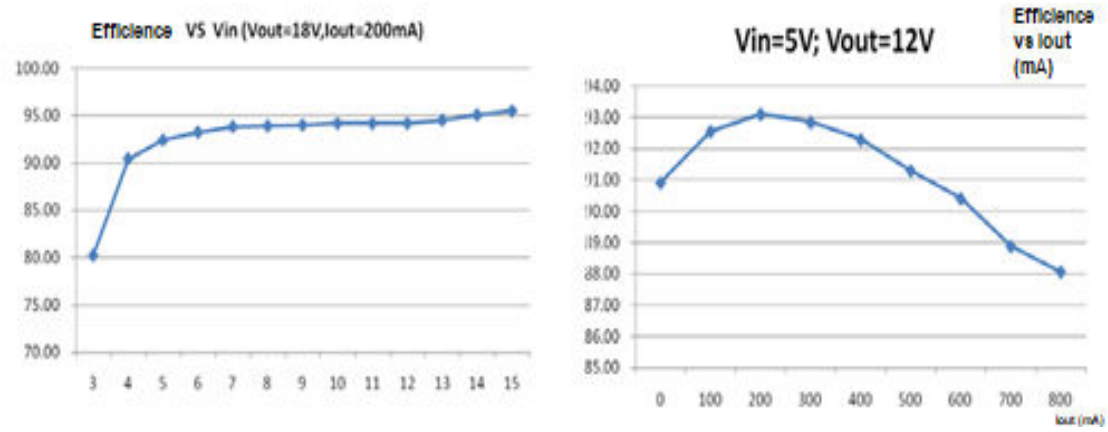



Ilustración Tablas de especificaciones del convertidor Boost.

Para anticiparse a cualquier problema, se cogerá el menor de los rendimientos que aparecen en las tablas de fabricante, en este caso es de $\eta_b$ = 80%.

Se tiene que:

$$P_e * \eta_T = P_s$$

$$P = V * I$$

$$\eta_T = \eta_c * \eta_b$$

$$I_e = P_e / ( V_e * \eta_T )$$

$$I_e = 1{,}05 / ( 3.7 * 0.80 * 0.90 ) = 0.394 \text{ A}$$

Donde: $I_e$: intensidad de entrada. $I_s$: intensidad de salida. $\eta_T$ =rendimiento total. $P_e$: potencia neta consumida. $P_s$: potencia consumida $V_e$: voltaje de entrada. $V_s$: voltaje de salida.

Para unas condiciones más extremas, se va a tener cuenta que la batería trabaja prácticamente descargada a 3 V. Se continúa considerando que el rendimiento total será el del caso anterior. Realizando los cálculos anteriores, tenemos que $I_e$ = 0.486 A.

La intensidad máxima de descarga de la batería ($I_{máx}$) que recomienda el fabricante (19), ateniéndonos a su hoja de especificaciones (13), será $I_{máx}$ = 6700mA. Este resultado, se obtiene multiplicando el ratio de descarga (20) máximo, 2C, que podemos observarlo en la grafica de abajo a la derecha por la capacidad típica de la batería, que es 3350 mAh. Así, se obtiene que la intensidad demandada por nuestra tarjeta está muy por debajo de la intensidad máxima de descarga.

Se puede observar, haciendo una breve retrospección a párrafos anteriores, que la batería nunca alcanzará su intensidad máxima de descarga de 6700 mA, puesto que el sistema BMS cortará el flujo a los 3000mA, evitando posibles problemas con la batería.

### 3.3.4. Duración estimada de la batería

Notar que con los resultados obtenidos, la duración normal de la batería será de:

$$\text{Duración} = \text{Capacidad} / \text{Intensidad}$$

$$\text{Duración} = 3350 \text{ mAh} / 390 \text{ mh} = 8.7 \text{ h.}$$

### 3.3.5. Esquema simbólico

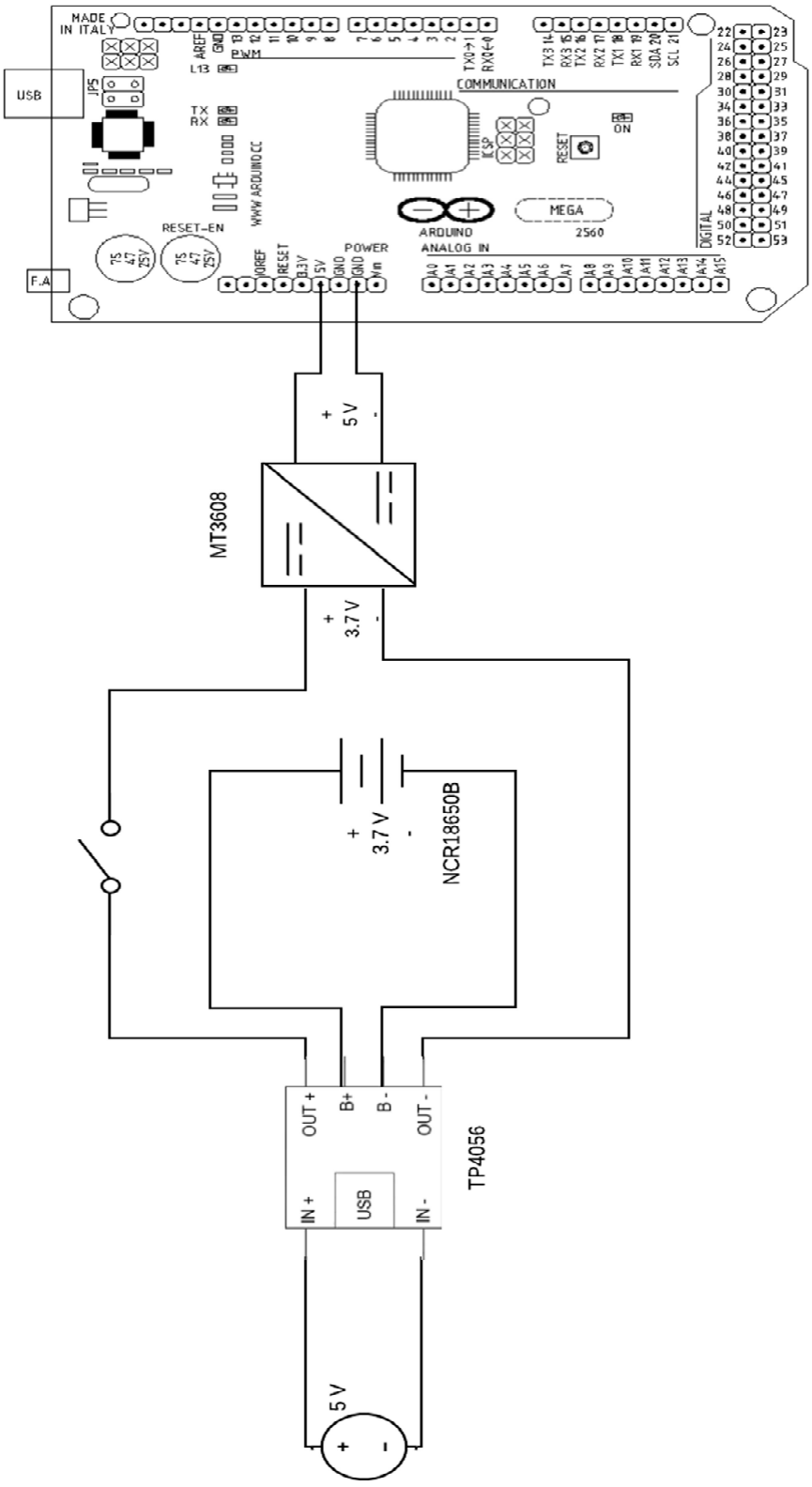



Ilustración 29. Esquema simbólico del montaje.

### 3.3.6. Esquema real.

Finalmente, el montaje ya realizado con los componentes y uniones indicadas. Señalar que los pines que se han elegido para alimentar el Arduino no son específicamente los mostrados en el montaje anterior, siendo indiferente cual de los de GRN (tierra) y V5 se elija:

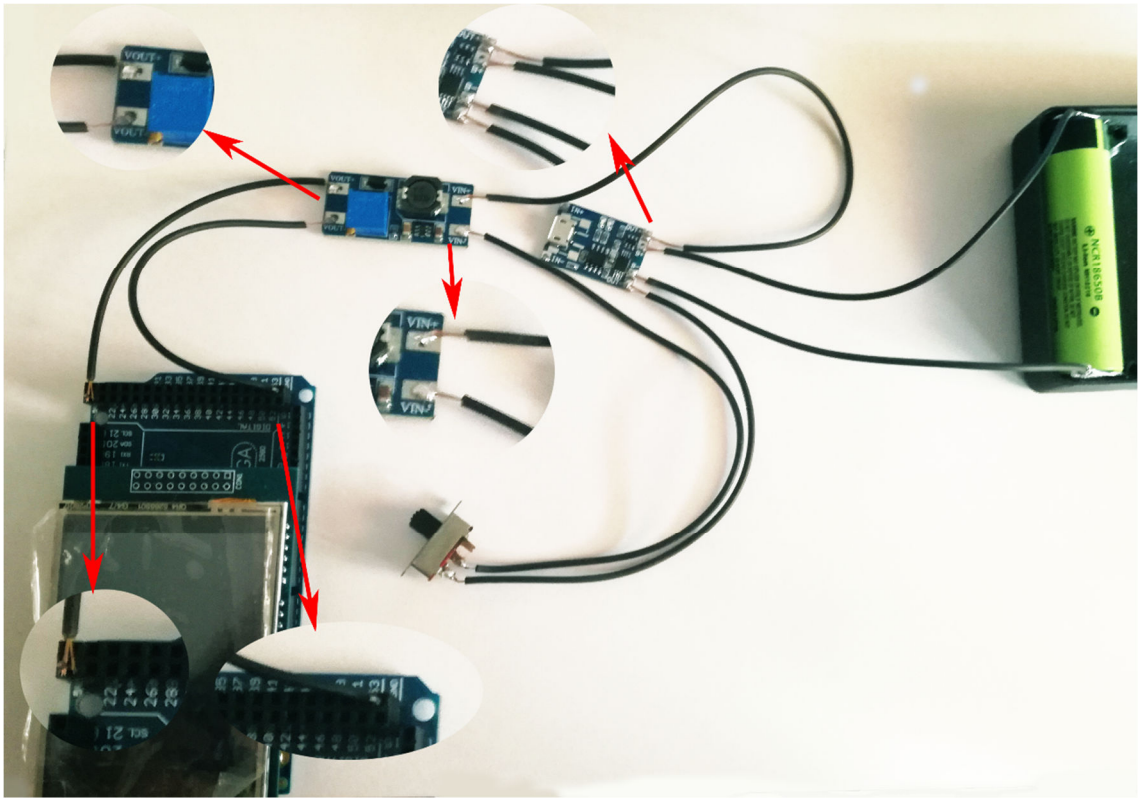



Ilustración 30. Esquema de alimentación real.

### 3.3.7. Mediciones post-montaje.

Para comprobar que los cálculos realizados en apartados anteriores con respecto a consumo energético y duración, se van a realizar las mediciones una vez montado el circuito, tal y como se muestra a continuación. Se debe tener en cuenta que las mediciones se harán a la salida del convertidor Boost y a la salida de la batería. Para llevarlas a cabo, se ha conectado el circuito a la tarjeta y cada cierto periodo de tiempo se ha ido tomando las mediciones, sin cortar en ningún momento el suministro de energía. A continuación, se muestran los resultados obtenidos tras la medición (tabla 3, 4, 5):

| Medición | Voltaje batería (V) | Intensidad bateria (mA) | Potencia bateria (mW) |
|---|---|---|---|
| 1 | 4,09 | 365 | 1492,85 |
| 2 | 3,92 | 375 | 1470 |
| 3 | 3,74 | 400 | 1496 |
| 4 | 3,61 | 425 | 1534,25 |
| 5 | 3,45 | 445 | 1535,25 |
| 6 | 3,23 | 450 | 1453,5 |

Tabla 3. Mediciones post-montaje batería.

| Medición | Voltaje Boost (V) | Intensidad Boost (mA) | Potencia Boost (mW) |
|---|---|---|---|
| 1 | 5 | 200 | 1000 |
| 2 | 5 | 216 | 1080 |
| 3 | 4,9 | 222 | 1087,8 |
| 4 | 4,8 | 227 | 1089,6 |
| 5 | 5,05 | 225 | 1136,25 |
| 6 | 4,85 | 225 | 1091,25 |

Tabla 4. Mediciones post-montaje en salida convertidor Boost.

| Medición | Tiempo (min) | Rendimiento (%) |
|---|---|---|
| 1 | 0 | #¡DIV/0! |
| 2 | 105 | 1,90 |
| 3 | 195 | 1,54 |
| 4 | 270 | 1,48 |
| 5 | 394 | 1,27 |
| 6 | 514 | 1,17 |

Tabla 5. Medición tiempo-rendimiento.

Notar, que el sistema se apagó a los 600 minutos.

### 3.3.8. Análisis y conclusiones del sistema de carga.

En este apartado, se analizarán los resultados experimentalmente, comparándolos con los datos obtenidos en apartados previos para la selección de los componentes. En primer lugar, se analizará un gráfico (ilustración 31), que comprara valores de voltaje y corriente a lo largo de la escarga, con los valores teóricos tomados.

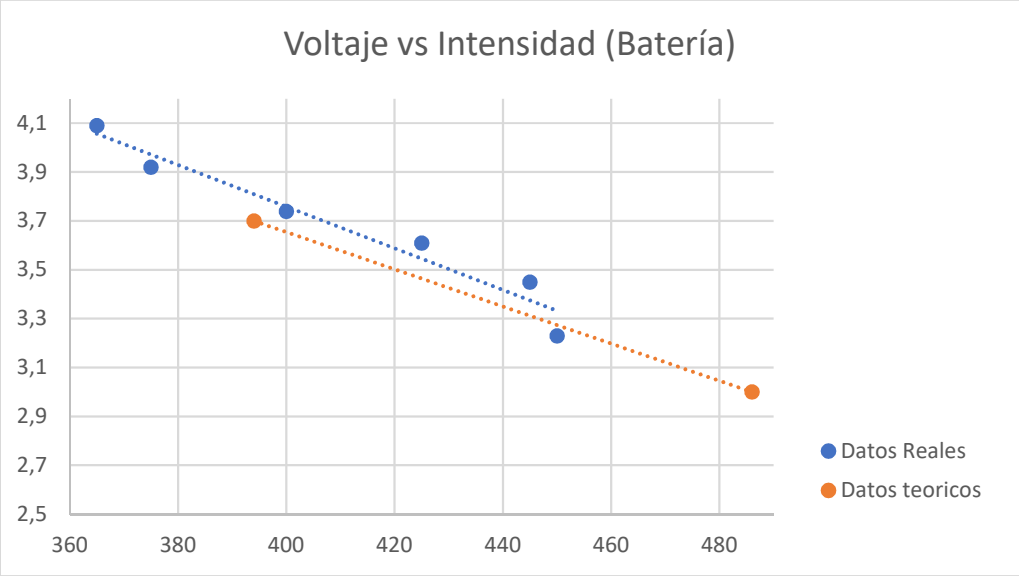



Ilustración 31. Gráfica Voltaje vs Intensidad.

Como se puede observar, los datos teóricos con los reales obtenidos son bastante próximos, aun así, observamos que los reales son superiores (ilustración 31). Anteriormente consideramos de media un rendimiento total de 0.72. Los valores de rendimiento, como podemos observar en la tabla, se encuentran por encima y por debajo de esos valores. A pesar de estos resultados, se puede verificar que la corriente que debe de dar la batería no excede la máxima admisible, ni la de corte del BMS durante todo su proceso de descarga.

Además, se puede hacer también la comparación con la duración real de nuestro sistema, que finalmente ha sido de 600 min, es decir 10 horas. Este también se acerca a las 8,7 horas que estimamos en el apartado anterior, incluso mejorándolo.

## 3.4. Carcasa.

La carcasa es una parte indispensable para hacer de soporte a los distintos componentes que componen la cámara y el sistema de alimentación, además de proporcionar protección frente a posibles manipulaciones del sistema y golpes. Para su modelado se ha empleado Autodesk Inventor. En este apartado se abarcarán los pasos que se han seguido para su modelado, desde las consideraciones previas a tener en cuenta hasta los acabados de la misma.

### 3.4.1. Consideraciones previas.

Antes de comenzar con el propio croquizado del modelo, hay una serie de invariables que se han de tener en cuenta para adaptar el diseño a ellas. Se deben obtener una serie de mediciones de las dimensiones de los distintos componentes que conforman el proyecto. Por lo tanto, todos los elementos que intervienen en el montaje se han medido y posteriormente modelado en el entorno de desarrollo, con el fin de que en el proceso de modelaje de nuestra carcasa, verificar que todo ensambla correctamente.

Puesto que después la impresión se hará en una impresora 3D, que utiliza PLA como material se impresión, tenemos que considerar las retracciones que sufre este material al contraerse, para en el proceso de ensamblaje posterior, evitar que alguna pieza no encaje correctamente. Para ello, se ha aplicado una holgura de 0.03 milímetros por cada milímetro dimensionado, a aquellos agujeros cuyo eje se imprima vertical a la base. Por ejemplo, si se tiene un agujero que debe de hacer de guía a un tornillo de métrica 3, entonces el agujero lo realizaremos de 3 * 1.03 = 3.09 mm, o redondeando 3.1 mm.

Además, se debe adaptar a las medidas normalizas de diferentes elementos, tales como roscas y tornillos, que conformarán la unión mecánica de los diferentes integrantes y medidas de la batería, placa Arduino y demás componentes.

### 3.4.2. Desarrollo de la carcasa.

En este apartado se expondrán algunos detalles sobre el proceso de modelado de la carcasa.

#### 3.4.2.1. Fijaciones.

Dado que las fijaciones entre algunos elementos del proyecto son uniones mecánicas por perno y rosca, se han tenido que diseñar las guías y los huecos respectivamente para ellos. A continuación, se muestran algunas imágenes de las soluciones dadas a lo expuesto (ilustración 32 y 33):

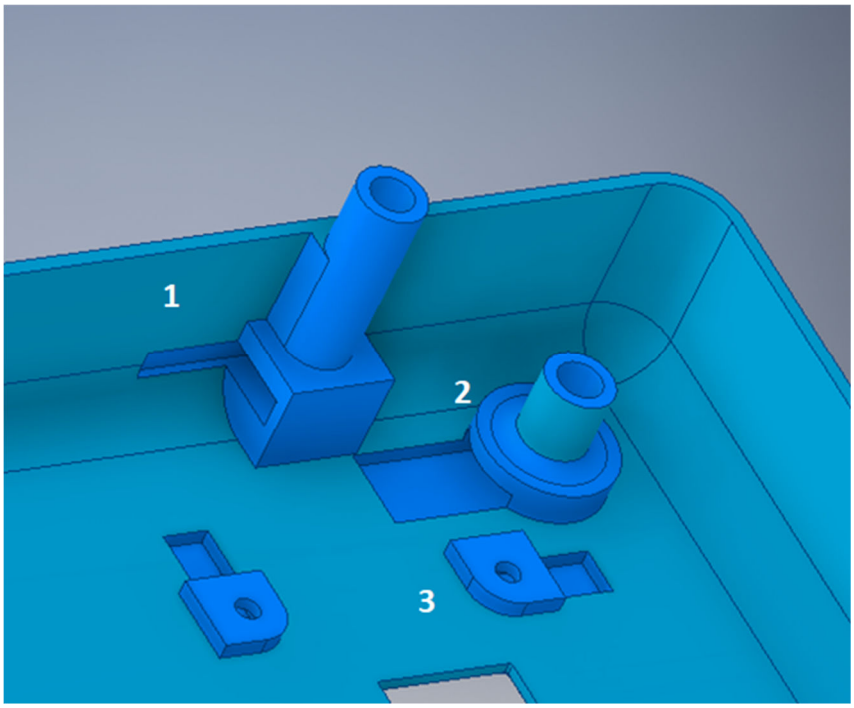



Ilustración 32. Desarrollo de la carcasa.

Donde 1 es el soporte para la unión de las dos partes de la carcasa, 2 es para la fijación de la tarjeta Arduino y la carcasa y 3 es la unión para la cámara y la carcasa. Se puede observar que para ahorrar espacio y darle aún más resistencia al soporte, los huecos para las tuercas los hemos integrado en las propias caras de la carcasa.

En la siguiente imagen (ilustración 33) se ven los acoples para las cabezas de los tornillos:

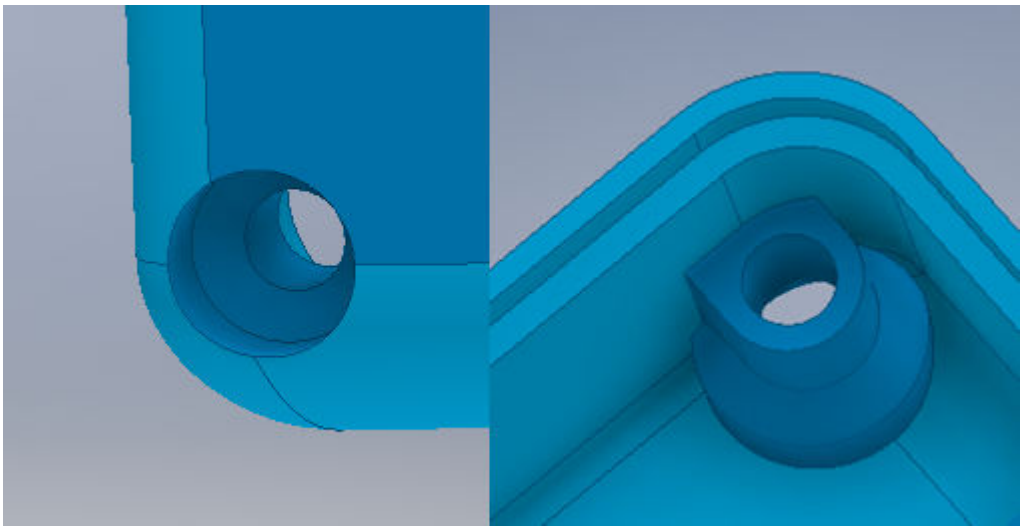


Ilustración 33. Desarrollo de la carcasa: acoples para las cabezas de los tornilos.

Como podemos ver, se ha diseñado con suficiente holgura la parte de la cabeza para poder integrar posteriormente embellecedores.

### 3.4.2.2. Interruptor.

Para embellecer el interruptor de encendido y apagado, se ha diseñado un sistema en el cual, cubriendo el interruptor a modo de funda, conseguimos darle otro aspecto (ilustración 34).

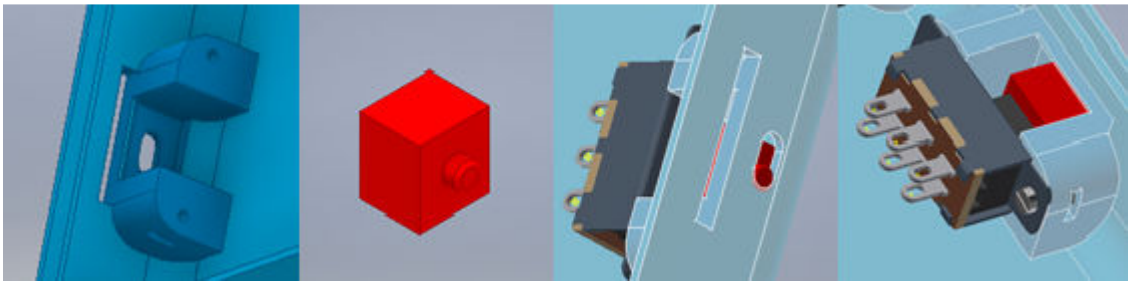


Ilustración 34. Sistema embellecedor del interruptor.

### 3.4.2.3. Soportes para componentes.

También se muestran ciertos soportes para el resto de componentes. Entre ellos batería, chapas conductora para la batería y BMS (ilustración 35).

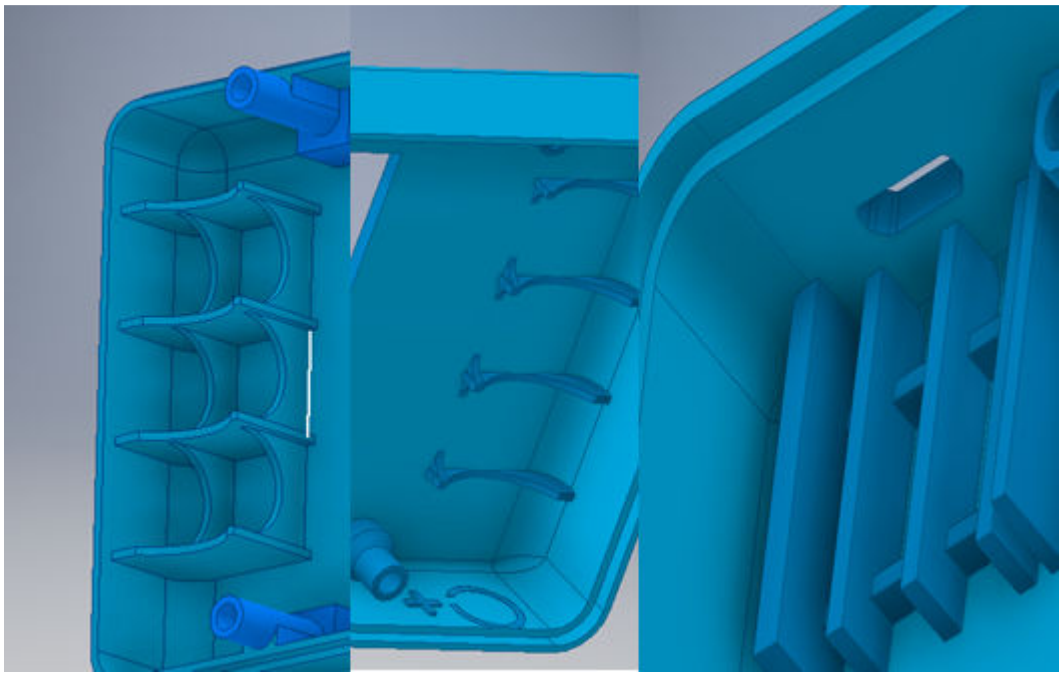


Ilustración 35. Soportes para componentes.

Como vemos se han diseñado teniendo en cuenta que estos elementos pueden liberar calor. Así, para evitar que se sobrecalienten, se ha realizado una estructura que permita un intercambio de calor más eficiente.

## 3.5. Impresión 3D

Una vez hemos diseñado la carcasa, debemos de asegurarnos que todos los componentes encajan perfectamente, antes de llevarla a la impresión 3D, proceso que consume tiempo y recursos. Para ello, se han ensamblado todos los componentes, y se han hecho las modificaciones para verificar que todo encajará correctamente.

### 3.5.1. Enrutamiento del modelo.

El modelo anteriormente creado se pasa al software de la impresora 3D, mediante el cual se enruta y se elige el material de soporte que haga de sustento a las partes en voladizo. El material que se empleará será PLA, con una densidad del 20% y con una calidad de 0.15 mm. En la siguiente imagen (ilustración 36) se observa un pantallazo de el resultado obtenido tras el enrutamiento. Una vez hecho esto, se genera el archivo que se pasa a la impresora para iniciar la impresión.

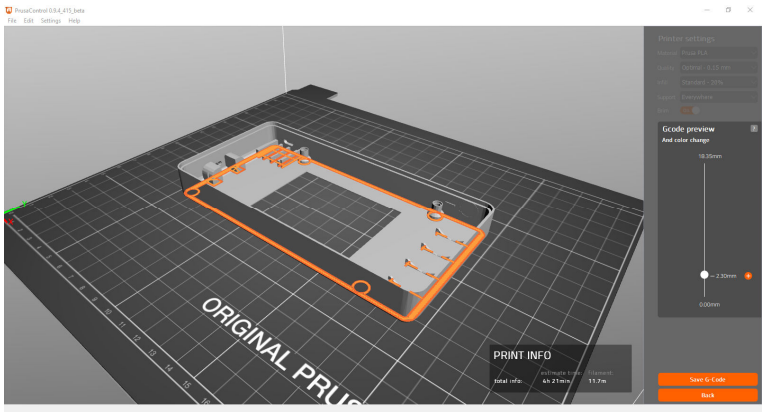



Ilustración 36. Resultado obtenido tras el enrutamiento.

### 3.5.2. Resultados tras la impresión:

Finalmente, tras 8h imprimiendo, ya tenemos el modelo (ilustración 37).

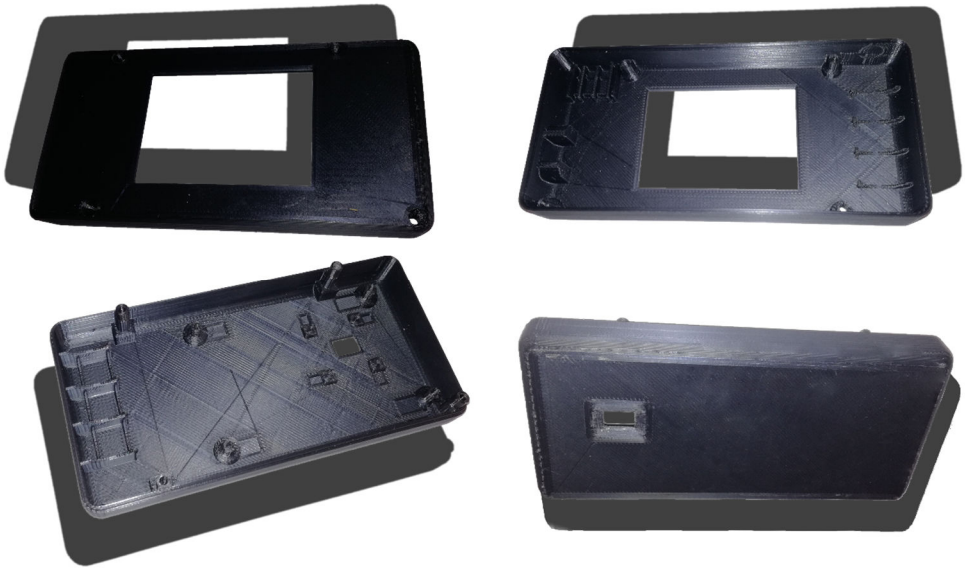


Ilustración 37. Resultados tras la impresión.

## 3.6. Montaje final

### 3.6.1. Elementos necesarios.

Para realizarlo, en primer lugar se han de preparar todos los componentes necesarios, entre ellos será necesario tornillos y tuercas para las uniones, cada uno con la métrica necesaria, cable, adhesivo, la carcasa imprimida, el sistema de alimentación. Además, también se necesitarán destornilladores para el montaje (ilustración 38).


Ilustración 38. Elementos necesarios.

### 3.6.2. Ensamblaje.

Finalmente, se procede al ensamblaje de todo. Primero se comenzará con la carcasa que contiene los huecos para las roscas. Se introducen las roscas en los huecos que se han diseñado para ellas. Se montará la cámara en su hueco correspondiente, con los tornillos M2x8. Lo siguiente, será unir la tarjeta Arduino, se recomienda hacerlo con la pantalla sin ensamblar por mayor comodidad en el atornillado, con los tornillos M3 x10 (ilustración 39).

Ilustración 39. Ensamblaje.

Después, en el otro lado de la pantalla se ensamblará el circuito para la alimentación de la cámara. Para ello se atornilla el interruptor, previamente montado, en el lugar que le corresponde (ilustración 40). La posición de los demás elementos irá como se observa en la ilustración 40. Se debe tener en cuenta que algunos de los elementos, como el BMS, no llevan agujeros para una unión por perno roscado, luego para una correcta fijación se puede usar un adhesivo termoestable. Notar que, en los contactos con la batería, se ha diseñado unas placas metálicas que van soldadas al conductor que conecta con el BMS, estas solo hacen contacto con los electrodos de la batería, sin necesidad de soldarlas.

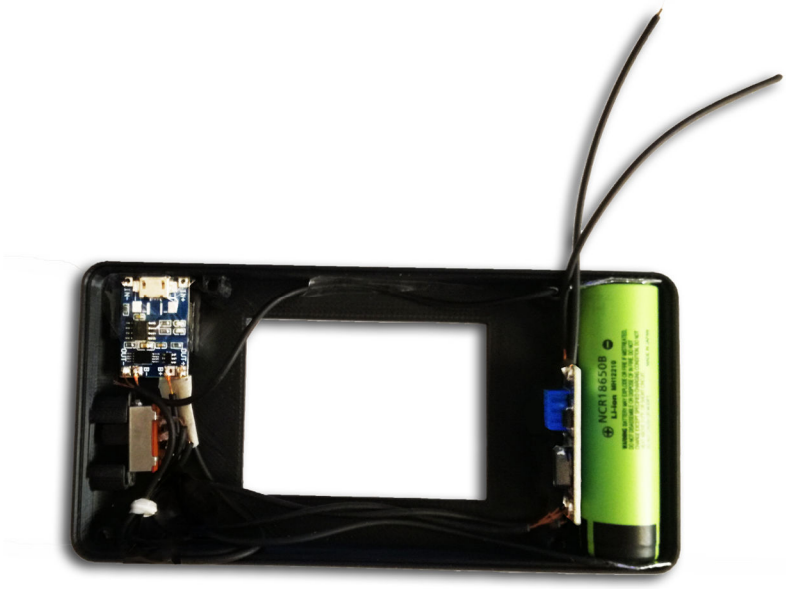



Ilustración 40 Ensamblaje del circuito.

### 3.6.3. Resultado final

A continuación, hay que unir los dos cables que salen del convertidor boost, a los pines por los que se va a alimentar el Arduino (ver ilustración 29). Finalmente, se coge la parte de la carcasa que contiene la tarjeta Arduino y se encaja en su otra mitad. Una vez hecho esto se le da la vuelta, presionando ambas partes, para evitar que se separen y se atornilla con los tornillos M3x25, DIN963. Los resultados son (ilustración 41):

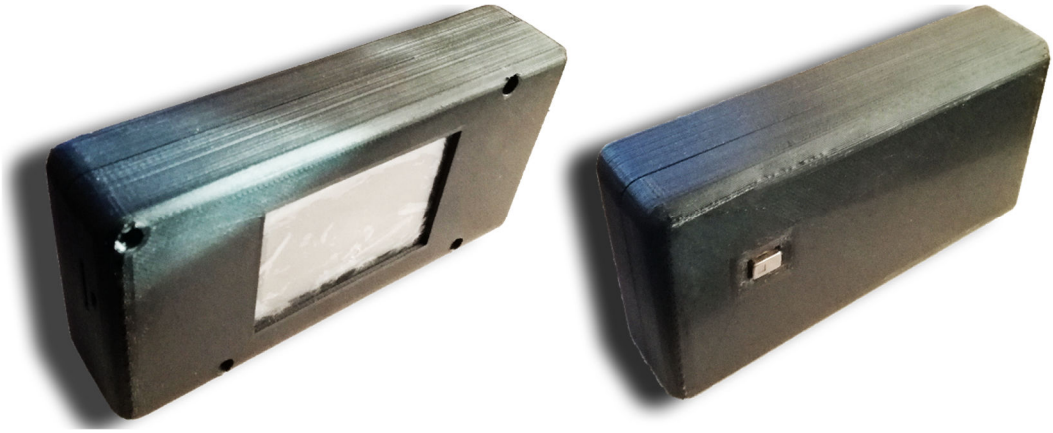


Ilustración 41. Resultado final.

# 4. Conclusiones.

En el presente Trabajo de Final de Grado se ha realizado el desarrollo de una cámara térmica con el objetivo de facilitar a su usuario la detección de focos de calor que pueda haber en un entorno dado.

El proyecto se plantea como una alternativa a los actuales sistemas de esta categoría que hay en el mercado, cuyo valor, en los modelos más económicos no baja de los 150 €, llegando a modelos más complejos que rondan los 10000€.

En este apartado aportaremos las conclusiones que se pueden extraer del proyecto, pero antes se hará un repaso a los objetivos que perseguíamos al iniciar el proyecto con el fin de observar si finalmente se han llevado a cabo. Así, haciendo una retrospección a ellos se tenía que:

- Interconectar por medio de software las distintas partes de hardware que se integran en el proyecto.
- Necesidad de dotar al modelo propuesto de un sistema de alimentación autónoma de energía.
- Dar soporte mecánico al conjunto de componentes del sistema, con el fin de lograr un sistema integrado.

Repasando los resultados obtenidos a lo largo del proyecto, se llega a que los objetivos marcados se han cumplido, teniendo que:

- El primer objetivo marcado se consigue al haber desarrollado un sistema de comunicación sensor IR-pantalla, que, mediante una serie de aplicaciones, permite el uso intuitivo y provechoso del hardware incorporado.
- El segundo objetivo se cumple logrando proporcionar al proyecto un sistema de alimentación, con hasta 10 horas testadas de autonomía y que garantiza con sus componentes la seguridad del proceso de carga.
- El último objetivo se logra con el modelado y posterior impresión 3D de la carcasa, que da soporte e integración a todos los componentes.

Comentar que el proyecto puede llegar a ser competitivo económicamente en el mercado, comparando con los sistemas que se ofertan.

A nivel personal, concluyo que este proyecto ha sido útil para comprender las distintas fases que se han de ahondar en un proyecto de este carácter. El proyecto ha tocado partes diferentes del ámbito ingenieril, que, aunque en un principio parecen estar separadas, como puede ser los sistemas de alimentación y el modelado 3D, todos finalmente se entrelazan unos con otros, permitiendo llegar a la conclusión de que, en el desarrollo de esta actividad profesional, el conocimiento de los diferentes tipos de ciencias y tecnologías, y la capacidad para encontrar puntos en común, son imprescindibles.

# 5. Bibliografía.

1 - Arduino.cc. (2018). *Arduino - ArduinoBoardMega*. [online] Available at: https://www.arduino.cc/en/Main/arduinoBoardMega/ [Accessed 4 Jun. 2018].

2 - Zamudio, J. (2018). *Alimentar el Arduino: La guía definitiva - Geek Factory*. [online] Geek Factory. Available at: https://www.geekfactory.mx/tutoriales/tutoriales-arduino/alimentar-el-arduino-la-guia-definitiva/ [Accessed 4 Jul. 2018].

3 - Crespo, E. (2017). *Memoria Arduino*. [online] Aprendiendo Arduino. Available at: https://aprendiendoarduino.wordpress.com/2017/09/05/memoria-arduino-2/ [Accessed 12 Jun. 2018].

4 - Llamas, L. (2018). *El bus SPI en Arduino*. [online] Luis Llamas. Available at: https://www.luisllamas.es/arduino-spi/ [Accessed 4 Jul. 2018].

5 - LLamas, L. (2018). *El bus I2C en Arduino*. [online] Luis Llamas. Available at: https://www.luisllamas.es/arduino-i2c/ [Accessed 4 Jul. 2018].

6 - Wiki.seeedstudio.com. (2018). *2.8'' TFT Touch Shield v2.0*. [online] Available at: http://wiki.seeedstudio.com/2.8inch_TFT_Touch_Shield_v2.0/ [Accessed 4 Jul. 2018].

7 - Industries, A. (2018). *Adafruit AMG8833 IR Thermal Camera Breakout*. [online] Adafruit.com. Available at: https://www.adafruit.com/product/3538 [Accessed 9 May 2018].

8 - Crespó, E. (2018). *IDE | Aprendiendo Arduino*. [online] Aprendiendoarduino.wordpress.com. Available at: https://aprendiendoarduino.wordpress.com/category/ide/ [Accessed 4 May 2018].

9 - Perez Valderrama, J. and Walteros Parra, Y. (2016). Sistema de gestión de carga para baterías de Ion-Litio. *PontificiaA Uuniversidad Javeriana, Bogotá D. C*-. [online] Available at: https://repository.javeriana.edu.co/bitstream/handle/10554/21433/PerezValderramaJorgeIvan2016.pdf.

10 - González, P. (2018). *Vida y muerte de una batería de ion-litio (parte II)*. [online] forococheselectricos. Available at: https://forococheselectricos.com/2013/05/vida-y-muerte-de-una-bateria-de-ion.html [Accessed 4 Jul. 2018].

11 - Zamudio, J. (2016). *Convertidor Boost con circuito integrado 555 - Geek Factory*. [online] Geek Factory. Available at: https://www.geekfactory.mx/tutoriales/convertidor-boost-con-circuito-integrado-555/ [Accessed 5 May 2018].

12 - Cruz Contreras, A. (2004). Procesamiento de Imagenes: Estructura de Archivos BMP. *PoliBits*. [online] Available at: https://www.polibits.gelbukh.com/2004_30/Procesamiento%20de%20Imagenes_%20Estructura%20de%20Archivos%20BMP.pdf [Accessed 11 Jun. 2018].

13 - Prometec.net. (2017). *Arduino y las interrupciones | Tienda y Tutoriales Arduino*. [online] Available at: https://www.prometec.net/interrupciones/ [Accessed 4 Jul. 2018].

14 - Prometec.net. (2017). *Arduino y los Timers | Tienda y Tutoriales Arduino*. [online] Prometec.net. Available at: https://www.prometec.net/timers/ [Accessed 20 Jun. 2018].

15 - Elena, V. (2010). *Técnología de las baterías*. [online] Www2.elo.utfsm.cl. Available at: http://www2.elo.utfsm.cl/~elo383/apuntes/PresentacionBaterias.pdf [Accessed 4 Jun. 2018].

16 - Na.industrial.panasonic.com. (n.d.). Data Sheet ncr18650b. [online] Available at: https://na.industrial.panasonic.com/sites/default/pidsa/files/ncr18650b.pdf [Accessed 4 Jul. 2018].

17 - www.addicore.com. (2016). *TP4056 Lithium Battery Charger and Protection Module*. [online] Available at: https://www.addicore.com/TP4056-Charger-and-Protection-Module-p/ad310.htm [Accessed 18 Jun. 2018].

18 - AEROSEMI, I. (2018). *MT3608 Data Sheet*. [online] Olimex.com. Available at: https://www.olimex.com/Products/Breadboarding/BB-PWR-3608/resources/MT3608.pdf [Accessed 4 Jul. 2018].

19 – Battery Analyse (2018). *Panasonic NCR18650B 3400mah Amp Limit & 18650 Battery Datasheet*. [online] Batteries18650.com. Available at: http://www.batteries18650.com/2015/05/Panasonic-NCR18650B-3400mah-limit-datasheet-li-ion-18650-battery.html [Accessed 4 Jul. 2018].

20 - Aeromodelismo.epiel.com. (2018). *La unidad C en las baterías*. [online] Available at: http://aeromodelismo.epiel.com/c_baterias.html [Accessed 4 Jul. 2018].

# PRESUPUESTOS

## Tabla de contenido



## Indice de tabla

# 1. Introducción.

En este apartado se realizarán los cálculos necesarios para obtener el presupuesto de nuestro proyecto. Para realizarlo, en primer lugar, se debe de considerar una serie de mediciones, es decir, la cantidad que interviene en el proyecto de cualquier bien que suponga un coste. Además, también será necesario conocer el coste de cada unidad cualquiera para, junto con las mediciones realizadas, poder obtener cual ha sido el coste final de que ese bien intervenga en nuestro proyecto. Tras conocerlos, se podrá finalmente presupuestar nuestro proyecto.

## 2. Mediciones.

En este apartado se proporcionarán las mediciones necesarias para la obtención posterior del presupuesto. Para ellos se obtendrá la cuantificación de tiempo, unidades y otros aspectos que intervengan en la realización del proyecto y que supongan un coste.

En cuanto a personal, tenemos que interviene tanto un Ingeniero Industrial y un Ingeniero Industrial considerándolo en prácticas. En la siguiente tabla se recogen las horas invertidas por cada uno:

| ***Ingeniero en prácticas*** | **Horas dedicadas (h)** |
|---|---|
| *Reuniones y planificación* | 24 |
| *Selección y compra de material* | 16 |
| *Diseño de elementos* | 72 |
| *Elaboración de planos* | 16 |
| *Fabricación de piezas* | 8 |
| *Montaje* | 8 |
| *Programación* | 72 |
| *Elaboración de memoria* | 34 |
| *Total* | **250** |

Tabla 1. Horas dedicadas por el ingeniero en prácticas

| ***Ingeniero responsable*** | **Horas dedicadas (h)** |
|---|---|
| *Reuniones y planificación* | 15 |
| *Total* | **15** |

Tabla 2. Horas dedicadas por el ingeniero responsable

Incluimos además elementos de software y hardware, a estos elementos les aplicamos un porcentaje directo con respecto a las horas de invertidas. Lo incluiremos como gastos en maquinaria:

| ***Elemento*** | **Concepto** | **Medición (horas)** |
|---|---|---|
| *Autodesk Inventor* | Modelado | 40 |
| *Windows 10 Pro Edicition* | Soporte Ordenador | 230 |
| *Ordenador* | | 230 |
| *Impresora 3D (Prusa)* | Impresión 3D | 8 |

Tabla 3. Horas de maquinaria

# 3. Precios y rendimiento

## 3.1. Mano de Obra

Se tiene que tener en cuenta que el salario medio de un Ingeniero Industrial es de alrededor 2500 euros. Además, se considerará que un profesional en prácticas cobra el 60% del salario medio proporcionado. Para calcular cuánto cuesta cada miembro mencionado a la hora se considera:

| ***Tipo*** | **Medición** | **Coste (€)** |
|---|---|---|
| *Pagas mensuales* | 12 | 30000 |
| *Pagas extras* | 2 | 5000 |
| *Total* | 14 | 35000 |
| *Total, tras seguridad social* | 130% | 45500 |
| *Ingeniero industrial* | 100% | 45500 |
| *Ingeniero industrial en prácticas* | 60% | 27300 |

**Tabla 4. Retribuciones**

Ese es pues, el coste anual, para calcular el coste mensual sé tendrá que tener en cuenta que la jornada laboral es de 8 horas, durante 5 días a la semana. Un mes tiene 4 semanas y un año 12 meses, luego:

Coste por hora Ingeniero = Coste  Ingeniero / horas / días / semanas / meses

Coste por hora Ingeniero = 45500 / 8 / 5 / 4 / 12

Coste por hora Ingeniero = 23,7 €/h

Realizando el mismo procedimiento, se obtiene que el coste por hora del ingeniero en prácticas es de:

Coste por hora I. Prácticas = 14,22 €/h

Finalmente, se tiene que:

| | **Medición** | **Cantidad €** | **Precio €** |
|---|---|---|---|
| *Ingeniero* | 15 | 23,7 | 355,5 |
| *Ingeniero Prácticas* | 250 | 14,22 | 3555 |
| *Coste Mano de Obra* | | | **3910,5** |

**Tabla 5. Mano de obra**

## 3.2. Materiales

En este apartado presentaremos la tabla de precios de los materiales empleados:

| ***Material*** | **Unidad** | **Coste unitario (€)** | **Medición** | **Precio (€)** |
|---|---|---|---|---|
| *Arduino Mega 2560 rev3* | unidad | 35,32 | 1 | 35,32 |
| *TP4056 con TE420* | unidad | 1,65 | 1 | 1,65 |
| *MT3608* | unidad | 2,09 | 1 | 2,09 |
| *NCR18650B* | unidad | 6,79 | 1 | 6,79 |
| *Interruptor deslizante* | unidad | 0,18 | 1 | 0,18 |
| *Filamento PLA* | metros | 0,07 | 25,27 | 1,7689 |
| *TFT LCD Shield* | unidad | 16,08 | 1 | 16,08 |
| *Tarjeta MicroSd Clase 10 16Gb* | unidad | 5 | 1 | 5 |
| *Tornillo M3 x 25, DIN963* | unidad | 0,09 | 4 | 0,36 |
| *Tornillo M3X10. DIN7945* | unidad | 0,09 | 4 | 0,36 |
| *Tornillo M2X10, DIN84* | unidad | 0,09 | 4 | 0,36 |
| *Tornillo M2X8, DIN84* | unidad | 0,09 | 4 | 0,36 |
| *Estaño* | gramos | 0,032 | 50 | 1,6 |
| *Cable AWG 25* | metros | 0,8 | 0,4 | 0,32 |
| *Lamina aluminio* | unidad | 0,2 | 2 | 0,4 |
| *Muelle* | unidad | 0,1 | 1 | 0,1 |
| *Total* | | | | **73,1** |

Tabla 6. Materiales

## 3.3. Maquinaria.

En este apartado se introducen todas aquellas herramientas que han hecho de vector para lograr el objetivo. Para realizarlo, primero se tiene que obtener el precio por hora de todos ellos.

Consultando en la página web del fabricante, se observa que la licencia anual de Autodesk Inventor cuesta 2.050,95 €. Para obtener su precio por hora, se tiene que las horas aprovechables de esta herramienta son 16 al día, considerando turno de mañana ( 8h ) y de tarde ( 8h ). Teniendo en cuenta que la semana tiene 5 días laborales, 4 semanas en un mes y 12 meses al año se tiene que:

Coste horario Inventor = 2050,98 / 16 / 5 / 4 / 12 = 0.534 €/hora.

Procediendo de igual modo con el resto, sabiendo que Windows 10 Pro Edition cuesta 40 €/año:

Coste horario Windows = 0.01 €/hora

Para el ordenador y la impresora 3D, se le aplicará un periodo de amortización de 3 años. Así, al precio total se le aplicará el denominador

anterior añadiéndole un dividendo de 3. Para un ordenador (1000 €) y la impresora 3D Prusa i3 MK2S (619 €) tenemos:

Coste horario Ordenador = 1000 / 16 / 5 / 4 / 12 / 3 = 0.087 €/h

Coste horario Impresora 3D = 619 / 16 / 5 / 4 / 12 / 3 = 0.054 €/h

Ahora presentamos el cuadro de precios de la maquinaria:

| ***Material*** | **Unidad** | **Coste unitario (€)** | **Medición** | **Precio (€)** |
|---|---|---|---|---|
| *Autodesk Inventor* | hora | 0,534 | 40 | 21,36 |
| *Windows 10 Pro Edition* | hora | 0,01 | 230 | 2,3 |
| *Ordenador* | hora | 0,087 | 230 | 20,01 |
| *Impresora 3D* | hora | 0,054 | 8 | 0,432 |
| *Total* | | | | **44,102** |

**Tabla 7. Maquinaria**

# 4. Resumen de presupuesto

| ***Elemento*** | **Descripción** | **Coste (€)** |
|---|---|---|
| *Mano de Obra* | Compuesta por el Ingeniero Industrial y el Ingeniero en prácticas. | 3910,5 |
| *Materiales* | Todos los elementos que de alguna forma, componen nuestro proyecto material. | 73,1 |
| *Maquinaria* | Elementos de apoyo para el desarrollo del proyecto | 44,102 |
| *Total* | | **4027.702** |

Tabla 8. Presupuesto de ejecución material

| ***Presupuesto de ejecución material*** | **4027.702 €** |
|---|---|
| ***Gastos generales (12%)*** | 483.32 € |
| ***Beneficio industrial (6%)*** | 241.66 € |
| *Presupuesto de EJECUCIÓN POR CONTRATA* | **4751.682** |
| ***21% IVA*** | 997.853 |
| *Presupuesto base de licitación* | **5749.53** |

Tabla 9. Presupuesto base de licitación

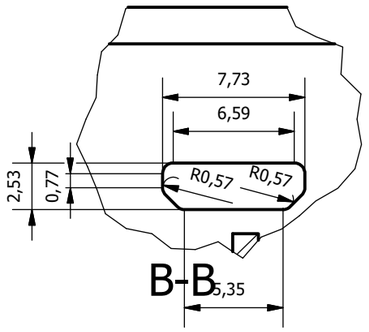
Asciende el presupuesto base de licitación a la expresada cantidad de:

**CINCO MIL SETECIENTOS CUARENTA Y NUEVE EUROS CON CINCUENTA Y TRES CENTIMOS.**

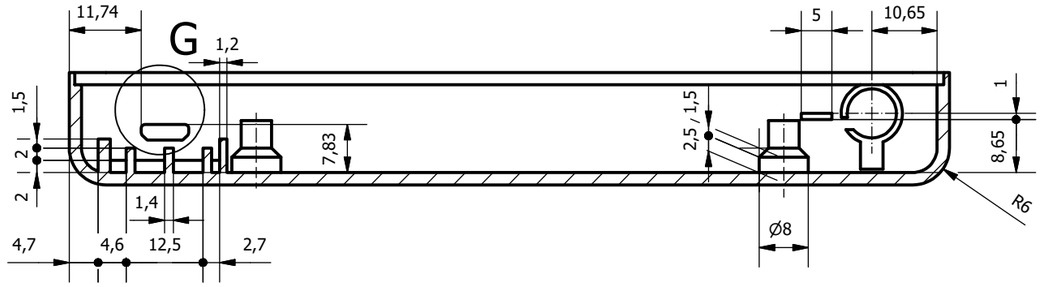
# PLANOS

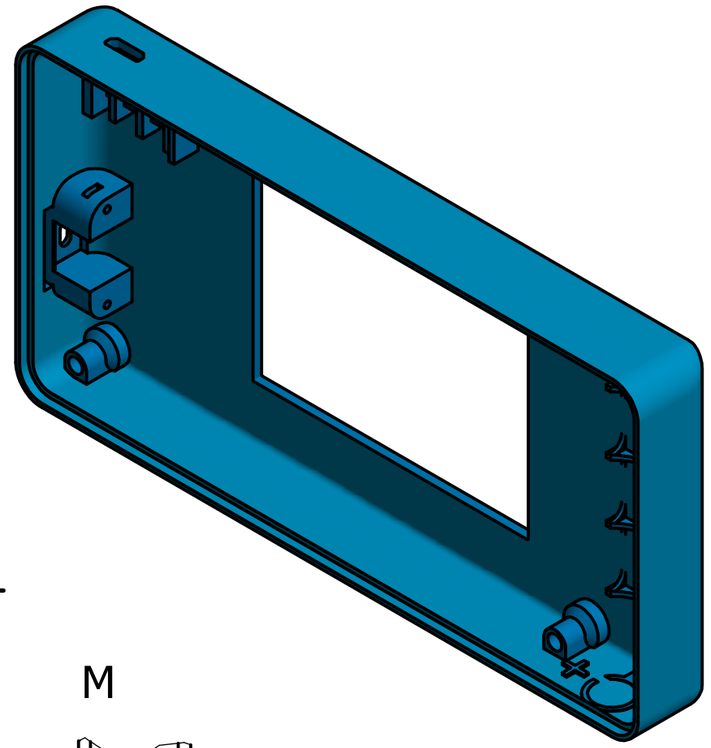
# Índice de planos

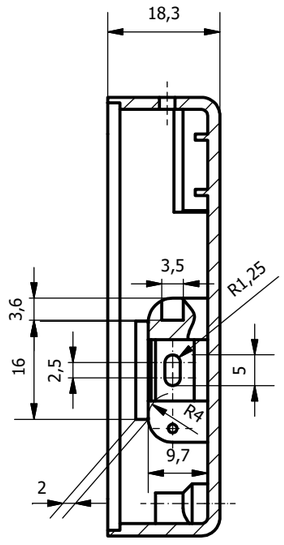
G ( 3 : 1 )

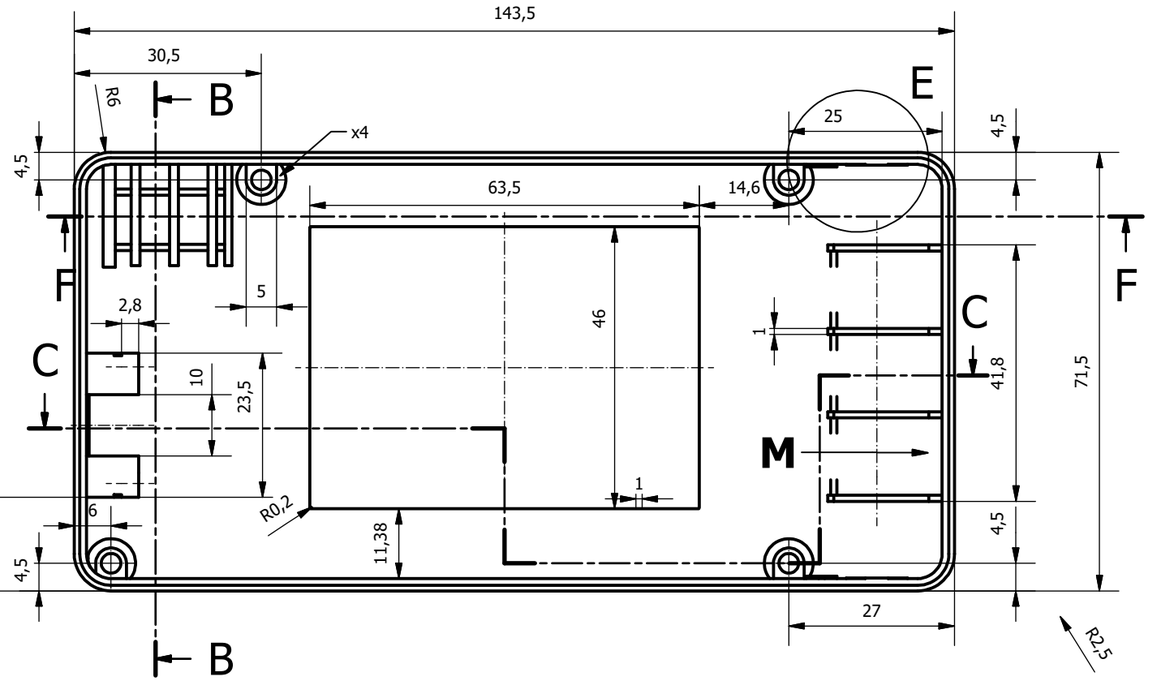


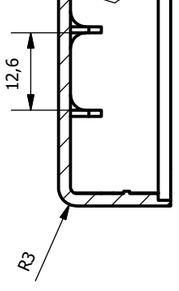

F-F



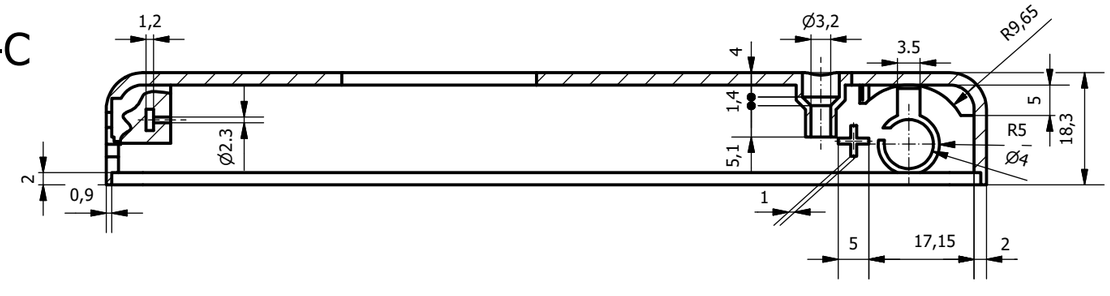


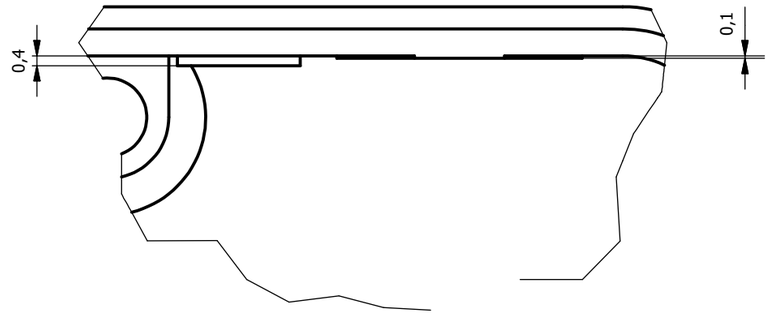



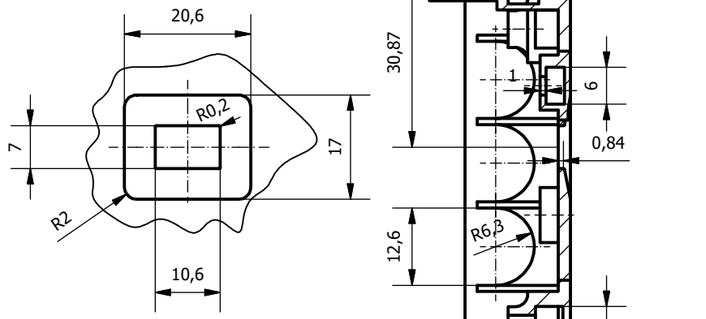

E ( 4 : 1 )

C-C



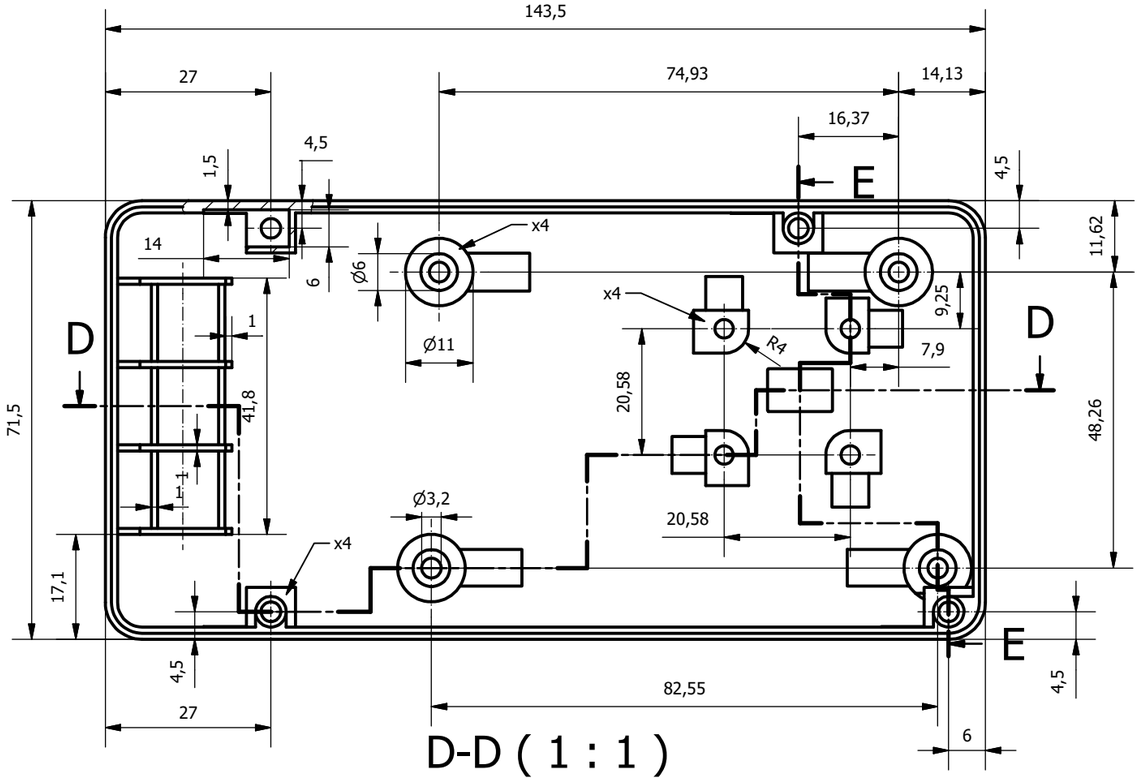




| | | | |
|---|---|---|---|
| TRABAJO FINAL DE GRADO EN TECNOLOGÍAS INDUSTRIALES<br>UNIVERSITAT POLITÈCNICA DE VALÈNCIA<br>ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA | Proyecto: DISEÑO DE CÁMARA TÉRMICA | Plano: Carcasa delantera<br>Autor: Félix Nieto del Amor | Fecha: Julio 2018<br>Escala: 1:1 | Nº Plano: 1 |

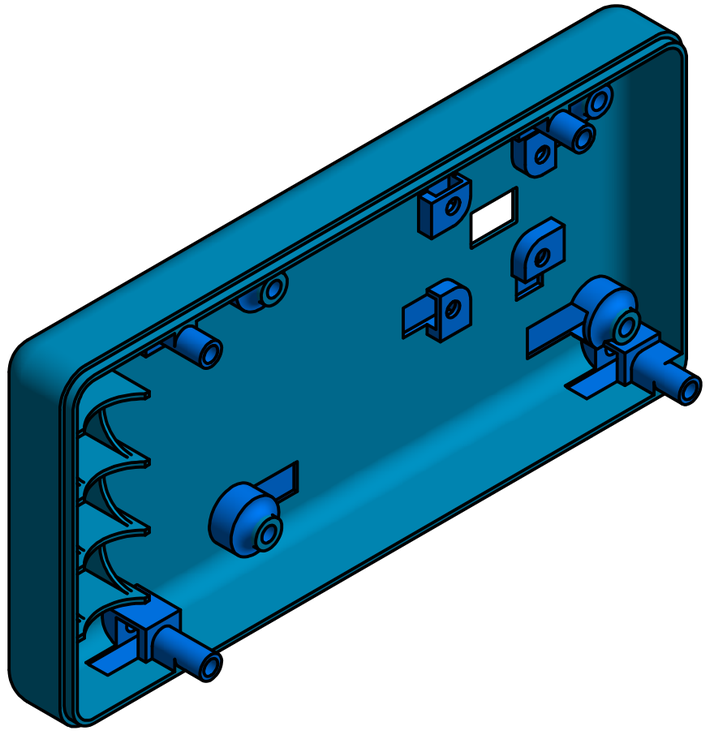


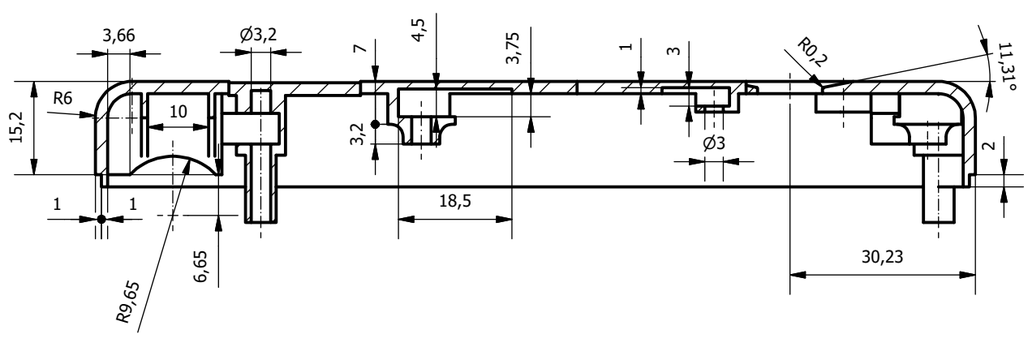

E-E ( 1 : 1 )



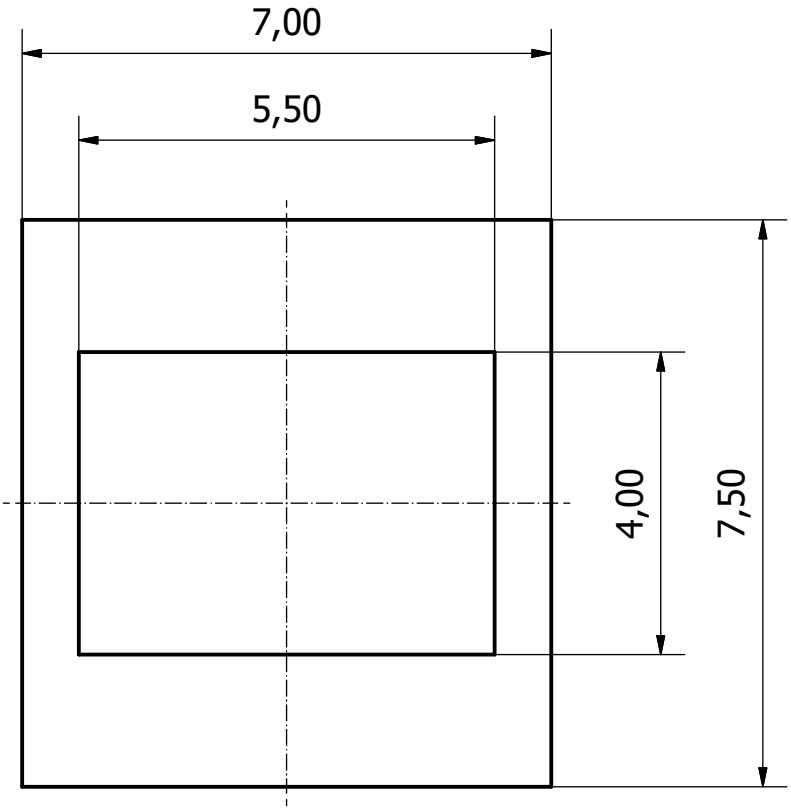



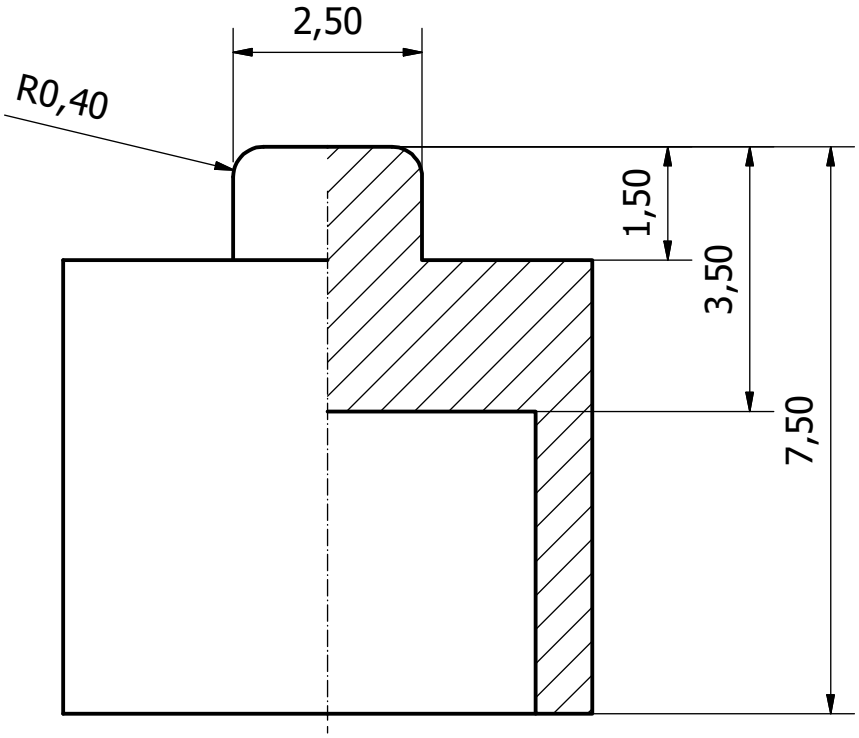

D-D ( 1 : 1 )

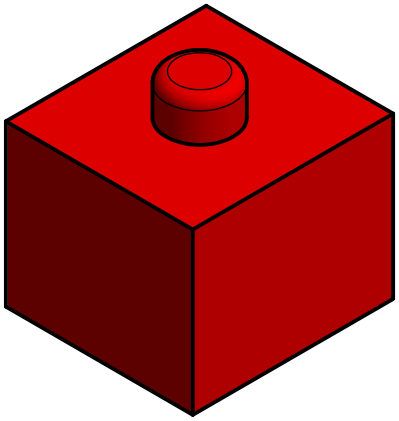





| | | | |
|---|---|---|---|
| TRABAJO FINAL DE GRADO EN TECNOLOGÍAS INDUSTRIALES<br>UNIVERSITAT POLITÈCNICA DE VALÈNCIA · ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA | Proyecto: DISEÑO CÄMARA TËRMICA | Plano: Carcasa trasera<br>Autor: Félix Nieto del Amor | Fecha: Julio 2018<br>Escala: 1:1 | Nº Plano: 2 |



| | | | | |
|---|---|---|---|---|
| TRABAJO FINAL DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES<br>UNIVERSITAT POLITÈCNICA DE VALÈNCIA · ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA | DISEÑO DE CÁMARA TÉRMICA | Plano: Funda de interruptor | Fecha: Julio 2018 | Nº Plano: **3** |
| | | Autor: Félix Nieto del Amor | Escala: 10:1 | |

# ANEXOS

## Tabla de contenido

# Anexo A: Principal

## A.1. Premisas

```
#include "selector.h"
#include "interpolar.h"
#include "tftmanager.h"

#include <Wire.h>
#include <Adafruit_AMG88xx.h>

extern Adafruit_AMG88xx amg;
extern Adafruit_TFTLCD tft;

void contrastMode(void);

volatile uint8_t contrast=2;
volatile uint16_t x_p=1,y_p=1 ;
volatile uint8_t task;
volatile uint8_t delayer=1;
```

## A.2. Setup

```
void setup() {

  Serial.begin(9600);        // abre el puerto serie a 9600 bps

  amg.begin(); //inicialización cámara

  tftInit(); //inicialización pantalla

  SD_init(); //Inicialización tarjeta SD

  contrastMode(); //lanzamiento interfaz elección de contraste
  if(contrast==MODE_REAL) realModeInterface();

  task = 1;

  draw_buttons( 50,20); //lanzamiento de interfaz interactiva

  //inicialización del Timer
  Timer1.initialize(100000);  // 150 ms
  Timer1.attachInterrupt( ISR_Callback) ;

}
```

## A.3. Loop

```
void loop() {

  // put your main code here, to run repeatedly:
if(task==0) {//modo pausa, se mantiene en este modo hasta que se decide que se desea hacer, pulsando uno de los botones
  noInterrupts();
  tft.setCursor(286, 1);  tft.setTextColor(WHITE);  tft.setTextSize(1);  tft.println(F("PAUSE"));
  screemParam(&x_p, &y_p);
  intercambiador( x_p, y_p);


  if (task!=0) tft.fillRect(284,1,35,BOXSIZE,BLACK);

  delayer=0;
}
if(task==1) {
  if(delayer==0){
    for(int i=0;i<200;i++){
      Serial.println("b00");
    }
    interrupts();
  }
  camara(contrast);
  delayer++;
}

if(task==2)
{
  //Timer1.detachInterrupt() ;
  analizador(contrast);
  //Timer1.attachInterrupt( ISR_Callback) ;
  screemParam(&x_p, &y_p);
  intercambiador( x_p, y_p);
  if((contrast==MODE_REAL)&&(task!=2)) {
    realModeInterface();
    tft.fillRect(261 ,11, 29, 218, BLACK);
  }
  if((contrast==MODE_CONT)&&(task!=2)) tft.fillRect(251 ,11, 39, 218, BLACK);
}

if(task==3){
    //Timer1.detachInterrupt() ; //deactiva interrupciones para evitar que se interrunpa la escritura del bmp,
    noInterrupts();

    advertence_triangule_sized();
    BMPtoSD();
    task=1;
  delayer=0;
  }

}
```

## A.4. Intercambiador

```
void intercambiador(uint16_t x, uint16_t y){

  if(((x>290)&&(x<320))&&((y>50)&&(y<110))){
    task=1;
  }
  if(((x>290)&&(x<320))&&((y>110)&&(y<170))){
    task=2;
  }
  if(((x>290)&&(x<320))&&((y>170)&&(y<230))){
    task=3;
  }

}
```

## A.5. Selector de Contraste (ContrastMode)

```
void contrastMode(void){

  drawControlScreem();

  while ( contrast>1 )
  {
    screemParam(&x_p, &y_p);

    if(((x_p>55)&&(x_p<145))&&((y_p>75)&&(y_p<165))){
      contrast=MODE_CONT;
    }
    if(((x_p>175)&&(x_p<265))&&((y_p>75)&&(y_p<165))){
      contrast=MODE_REAL;
    }
  }
  tft.fillScreen(BLACK);
}
```

## A.6. Interfaz Modo Real (realModeInterface)

```
void realModeInterface(void){
  //imprimimos rango de colores
   for(uint8_t y=0;y<220;y=y+1){
   uint8_t colorIndex = map(MINTEMP+(float)y*((MAXTEMP-MINTEMP)/219.0), MINTEMP, MAXTEMP, 0, 255);
   colorIndex = constrain(colorIndex, 0, 255);
   uint16_t color;
   color=pgm_read_word_near(camColors+colorIndex);

   tft.fillRect(241+BOXSIZE ,10+y, BOXSIZE, 1, color);
   }
}
```

# Anexo B Interpolador

## B.1. Archivo de cabecera

```
#ifndef __INTERPOLAR_H
#define __INTERPOLAR_H

#include <stdint.h>
#include <Adafruit_GFX.h>    // Core graphics library
#include <Adafruit_TFTLCD.h> // Hardware-specific library

#include "common.h"

#define MODE_CONT 1
#define MODE_REAL 0


float get_point(float *p, uint8_t rows, uint8_t cols, int8_t x, int8_t y);
void set_point(float *p, uint8_t rows, uint8_t cols, int8_t x, int8_t y, float f);

void interpolate_image(float *src, uint8_t src_rows, uint8_t src_cols,
                       float *dest, uint8_t dest_rows, uint8_t dest_cols);
void drawpixels(float *p, uint8_t rows, uint8_t cols, uint8_t boxWidth, uint8_t boxHeight,uint8_t mode/*, boolean showVal*/);


#endif // __INTERPOLAR_H
```

## B.2. Desarrollo de interpolador

```
#include "interpolar.h"

/////Parte mas especifica, borrar si es para otra pantalla
#include <Arduino.h>
#include <Adafruit_GFX.h>    // Core graphics library
#include <Adafruit_TFTLCD.h> // Hardware-specific library
#include "common.h"
#include "tftmanager.h"
#include "thermalbar.h"
#include <TimerOne.h>
//////////////

#include <avr/pgmspace.h>

extern Adafruit_TFTLCD tft;

static void get_adjacents_1d(float *src, float *dest, uint8_t rows, uint8_t cols, int8_t x, int8_t y);
static void get_adjacents_2d(float *src, float *dest, uint8_t rows, uint8_t cols, int8_t x, int8_t y);
static float cubicInterpolate(float p[], float x);
static float bicubicInterpolate(float p[], float x, float y);

float get_point(float *p, uint8_t rows, uint8_t cols, int8_t x, int8_t y) {
  if (x < 0)         x = 0;
  if (y < 0)         y = 0;
  if (x >= cols)     x = cols - 1;
  if (y >= rows)     y = rows - 1;
  return p[y * cols + x];
}

void set_point(float *p, uint8_t rows, uint8_t cols, int8_t x, int8_t y, float f) {
  if ((x < 0) || (x >= cols)) return;
  if ((y < 0) || (y >= rows)) return;
  p[y * cols + x] = f;
}
```

```
// src is a grid src_rows * src_cols
// dest is a pre-allocated grid, dest_rows*dest_cols
void interpolate_image(float *src, uint8_t src_rows, uint8_t src_cols,
                       float *dest, uint8_t dest_rows, uint8_t dest_cols) {
  float mu_x = (src_cols - 1.0) / (dest_cols - 1.0);
  float mu_y = (src_rows - 1.0) / (dest_rows - 1.0);

  float adj_2d[16]; // matrix for storing adjacents

  for (uint8_t y_idx=0; y_idx < dest_rows; y_idx++) {
    for (uint8_t x_idx=0; x_idx < dest_cols; x_idx++) {
       float x = x_idx * mu_x;
       float y = y_idx * mu_y;
       //Serial.print("("); Serial.print(y_idx); Serial.print(", "); Serial.print(x_idx); Serial.print(") = ");
       //Serial.print("("); Serial.print(y); Serial.print(", "); Serial.print(x); Serial.print(") = ");
       get_adjacents_2d(src, adj_2d, src_rows, src_cols, x, y);
       /*
       Serial.print("[");
       for (uint8_t i=0; i<16; i++) {
         Serial.print(adj_2d[i]); Serial.print(", ");
       }
       Serial.println("]");
       */
       float frac_x = x - (int)x; // we only need the ~delta~ between the points
       float frac_y = y - (int)y; // we only need the ~delta~ between the points
       float out = bicubicInterpolate(adj_2d, frac_x, frac_y);
       //Serial.print("\tInterp: "); Serial.println(out);
       set_point(dest, dest_rows, dest_cols, x_idx, y_idx, out);
    }
  }
}


// p is a list of 4 points, 2 to the left, 2 to the right
float cubicInterpolate(float p[], float x) {
    float r = p[1] + (0.5 * x * (p[2] - p[0] + x*(2.0*p[0] - 5.0*p[1] + 4.0*p[2] - p[3] + x*(3.0*(p[1] - p[2]) + p[3] - p[0]))));
  /*
    Serial.print("interpolating: [");
    Serial.print(p[0],2); Serial.print(", ");
    Serial.print(p[1],2); Serial.print(", ");
    Serial.print(p[2],2); Serial.print(", ");
    Serial.print(p[3],2); Serial.print("] w/"); Serial.print(x); Serial.print(" = ");
    Serial.println(r);
  */
    return r;
}

// p is a 16-point 4x4 array of the 2 rows & columns left/right/above/below
float bicubicInterpolate(float p[], float x, float y) {
    float arr[4] = {0,0,0,0};
    arr[0] = cubicInterpolate(p+0, x);
    arr[1] = cubicInterpolate(p+4, x);
    arr[2] = cubicInterpolate(p+8, x);
    arr[3] = cubicInterpolate(p+12, x);
    return cubicInterpolate(arr, y);
}

// src is rows*cols and dest is a 4-point array passed in already allocated!
void get_adjacents_1d(float *src, float *dest, uint8_t rows, uint8_t cols, int8_t x, int8_t y) {
    //Serial.print("("); Serial.print(x); Serial.print(", "); Serial.print(y); Serial.println(")");
    // pick two items to the left
    dest[0] = get_point(src, rows, cols, x-1, y);
    dest[1] = get_point(src, rows, cols, x, y);
    // pick two items to the right
    dest[2] = get_point(src, rows, cols, x+1, y);
    dest[3] = get_point(src, rows, cols, x+2, y);
}
```

```
// src is rows*cols and dest is a 16-point array passed in already allocated!
void get_adjacents_2d(float *src, float *dest, uint8_t rows, uint8_t cols, int8_t x, int8_t y) {
    //Serial.print("("); Serial.print(x); Serial.print(", "); Serial.print(y); Serial.println(")");
    float arr[4];
    for (int8_t delta_y = -1; delta_y < 3; delta_y++) { // -1, 0, 1, 2
        float *row = dest + 4 * (delta_y+1); // index into each chunk of 4
        for (int8_t delta_x = -1; delta_x < 3; delta_x++) { // -1, 0, 1, 2
            row[delta_x+1] = get_point(src, rows, cols, x+delta_x, y+delta_y);
        }
    }
}

///// parte especifica, borrar de aqui para abajo si es para otra pantallla
//static float past_p[INTERPOLATED_ROWS * INTERPOLATED_COLS]; //Acelerar proyeccion de puntos, ocupa mucha memoria, usar si sobra ->25% de la memoria total
//mode->0=alto contraste 1=real
void drawpixels(float *p, uint8_t rows, uint8_t cols, uint8_t boxWidth, uint8_t boxHeight, uint8_t mode/*, boolean showVal*/) {
  uint16_t colorTemp;

  for (uint16_t y=0; y<rows; y++) {
    for (uint16_t x=0; x<cols; x++) {
      //if(get_point(p, rows, cols, x, y)!=get_point(past_p, rows, cols, x, y)){
      //poner codigo con if, que selecciones antes de procesar el dato, si pixel anterior es igual que el actualizado, si es asÃ- pasar al siguiente pixel
      float val= get_point(p, rows, cols, x, y);

      if(val >= MAXTEMP) colorTemp = MAXTEMP;
      else if(val <= MINTEMP) colorTemp = MINTEMP;
      else colorTemp = val;

      //uint8_t colorIndex = map(colorTemp, MINTEMP /*get_min_t()*/, MAXTEMP /*get_max_t()*/, 0, 255);   //rangos de temp cte //modo alto contrate
       uint8_t colorIndex;
       if (mode==0) colorIndex = map(colorTemp, MINTEMP /*get_min_t()*/, MAXTEMP /*get_max_t()*/, 0, 255);
       if (mode==1) colorIndex= map(val*10, get_min_t()*10, get_max_t()*10, 0, 255);


       colorIndex = constrain(colorIndex, 0, 255); //devuelve si colorIndex esta entre 0 y 255, si es mayor pone 255, si es menor 0
       //draw the pixels!
       uint16_t color;
       color=pgm_read_word_near(camColors+colorIndex);
       noInterrupts();
       tft.fillRect(/*40+(margen)*/boxWidth * (rows - y -1), boxHeight * x, boxWidth, boxHeight, color);
       interrupts() ;
       //aquÃ-, cada pixel de los 24x24 interpolados, pasan a ser cuadrados de 10x10 pixeles-
        /*
       if (showVal) {
         tft.setCursor(boxWidth * y + boxWidth/2 - 12, 40 + boxHeight * x + boxHeight/2 - 4);
         tft.setTextColor(WHITE);  tft.setTextSize(1);
         tft.print(val,1);
       }
       */
       //}
     }
   }
   /*
   for(int i=0;i<(INTERPOLATED_ROWS * INTERPOLATED_COLS);i++){
     past_p[i]=p[i];
   }
   */
}
```

# Anexo C. Selector

## C.1. Archivo de cabecera

```
#ifndef __SELECTOR_
#define __SELECTOR_
#include <stdint.h>

void camara(uint8_t contrast);
void analizador(uint8_t contrast);
void BMPtoSD(uint8_t contrast);

#endif //__SELECTOR_
```

## C.2. Premisas

```
#include "interpolar.h"
#include <Adafruit_GFX.h>    // Core graphics library
#include <Adafruit_TFTLCD.h> // Hardware-specific library
#include <TouchScreen.h>
#include "common.h"
#include "thermalbar.h"
#include "selector.h"
#include "tftmanager.h"
#include "PmgToSD.h"
#include <Wire.h>
#include <Adafruit_AMG88xx.h>

Adafruit_AMG88xx amg;

extern Adafruit_TFTLCD tft;

static float dest_2d [INTERPOLATED_ROWS * INTERPOLATED_COLS];

static void initPixels(float *pixels);
```

## C.3. Modo Cámara (cámara)

```
void camara(uint8_t contrast){
  float pixels[AMG_COLS * AMG_ROWS];

   amg.readPixels(pixels);
  //uint16_t boxsize = min(tft_get_width() / INTERPOLATED_COLS, tft_get_height() / INTERPOLATED_COLS);
  interpolate_image(pixels, AMG_ROWS, AMG_COLS, dest_2d, INTERPOLATED_ROWS, INTERPOLATED_COLS);
  thermalRange(dest_2d);
  thermalBar(get_min_t(),get_max_t(),contrast);
  print_escala();
  drawpixels(dest_2d, INTERPOLATED_ROWS, INTERPOLATED_COLS, BOXSIZE, BOXSIZE, contrast);
}
```

## C.4. Modo Analizador (analizador)

```
void analizador(uint8_t contrast){
  uint16_t x_p,y_p;
  screemParam(&x_p, &y_p);

  if((x_p!=0)&&(y_p!=0)&&(x_p<240)){

    tft.fillRect(251, 10, 35, 230,BLACK);
    thermalBar(get_min_t(),get_max_t(),contrast);
    print_escala();
    camera_analyze_bar( dest_2d, x_p, y_p) ;
  }
  else{

  }

}
```

## C.5. Modo escribir en SD (BMPtoSD)

```
void BMPtoSD(uint8_t contrast){

  bmp_create(dest_2d, contrast );
}
```

# Anexo D. Barra Térmica y Analizador

## D.1. Archivo de cabecera

```
#ifndef __THERMALBAR_
#define __THERMALBAR_

#include <stdint.h>

void thermalRange(float pixels[]);
void thermalBar(uint8_t min_abs, uint8_t max_abs, uint8_t contrast);
void print_escala(void);
void camera_analyze_bar(float *dest, uint16_t x, uint16_t y);
float get_max_t(void);
float get_min_t(void);


#endif //__THERMALBAR_
```

## D.2. Premisas

```
#include "thermalbar.h"
#include <Adafruit_TFTLCD.h>
#include "common.h"
#include "tftmanager.h"
#include "interpolar.h"
#include <avr/pgmspace.h>


extern Adafruit_TFTLCD tft;

static float max_t, min_t;//
```

## D.3. Get para máxima y mínima temperatura

```
float get_max_t(void){
  return max_t;
}
float get_min_t(void){
  return min_t;
}
```

## D.4. Rango de temperaturas (thermalRange)

```
void thermalRange(float pixels[]){ //proporciona el rango de temperaturas max y min que genera la cï¿½mara. //le introduzco dest_2d
max_t=-20;
min_t=200;
  for(int i=0;i<(INTERPOLATED_COLS*INTERPOLATED_ROWS);i++){
    if(pixels[i]>max_t) max_t=pixels[i];
    if(pixels[i]<min_t) min_t=pixels[i];
  }
}
```

## D.5. Barra térmica (thermalBar)

```
void thermalBar(uint8_t min_abs, uint8_t max_abs, uint8_t contrast){
  if(contrast==MODE_REAL){
  for(uint8_t y=0;y<240;y=y+1){
    //Barra térmica modo real, mostrando rango de temperaturas reales
  uint8_t colorIndex = map(min_t+(float)y*((max_t-min_t)/239.0), MINTEMP, MAXTEMP, 0, 255);//rangos variables
  colorIndex = constrain(colorIndex, 0, 255); //devuelve si colorIndex esta entre 0 y 255, si es mayor pone 255, si es menor 0
  uint16_t color;
  noInterrupts();
      color=pgm_read_word_near(camColors+colorIndex);
  tft.fillRect(241 ,y, BOXSIZE, 1, color);
  interrupts();
  }
  }
  else{
  for(uint8_t y=0;y<240;y=y+1){ //si aumentamos alto de pantalla, aumentar ell almacenamiento de y a uint16_t

     uint8_t colorIndex = map(min_t+(float)y*((max_t-min_t)/239.0), min_abs, max_abs, 0, 255); //modo real->tendra min_abs = MIN definido, para el modo alto contrast
     colorIndex = constrain(colorIndex, 0, 255);
     uint16_t color;
     noInterrupts();
     color=pgm_read_word_near(camColors+colorIndex);
     tft.fillRect(241 ,y, BOXSIZE, 1, color);
     interrupts();
  }
  }
  noInterrupts();
  tft.fillRect(252, 0, BOXSIZE*3+1, BOXSIZE, WHITE);
  tft.fillRect(252, 229, BOXSIZE*3+1, BOXSIZE, WHITE);
  tft.setCursor(253, 1);
  tft.setTextColor(BLACK);  tft.setTextSize(1);
  tft.println(min_t);
  tft.setCursor(253, 230);
  tft.setTextColor(BLACK);  tft.setTextSize(1);
  tft.println(max_t);
  interrupts();
}
```

## D.6. Regleta de medición

```
void print_escala(void){//codigo a optimizar
   //tft.fillRect(251 ,12, 10, 240-24, BLACK);
  uint8_t i=2;
  uint8_t dif=(int)(max_t-min_t); //
  noInterrupts();
  while(dif>(i/2)){
    for(uint8_t j=1;j<=(i/2);j++){
      if(i==2)
      tft.fillRect(242 ,12+(int)(((int)(240-24))*j*2/i), 9, 1, WHITE); //(int)(((int)(240-24))*j*2/i)
      else if(i==8) tft.fillRect(242 ,12+(int)(((int)(240-24))*j*2/i), 9, 1, WHITE);
      else tft.fillRect(247 ,12+(int)(((int)(240-24))*j*2/i), 4, 1, WHITE);
    }

    i=i*2;
  }
  interrupts();
}
```

## D.7. Analizador

```
void camera_analyze_bar(float *dest,uint16_t x,uint16_t y){

  float val= get_point(dest, INTERPOLATED_ROWS, INTERPOLATED_COLS, x/BOXSIZE, y/BOXSIZE);

  uint8_t index = map(val, min_t, max_t, 0, 240);
    index = constrain(index, 0, 240);//si no ponermos 10 y 230- incluso 220, se solapa con el valor max y minimo que se muestra por la otra funcion

   tft.fillRect(241 ,index, BOXSIZE, 1, RED);

   //cuadro con valor
   index = constrain(index, 0, 230);
   tft.fillRect(252, index, BOXSIZE*3+1, BOXSIZE, YELLOW);
   tft.setCursor(253, index);
   tft.setTextColor(BLACK);  tft.setTextSize(1);
   tft.println(val);
}
```

# Anexo E. Funcionalidades de la pantalla

## E.1. Archivo de cabecera

```
#ifndef _TFTMANAGER__
#define _TFTMANAGER__

#include <Arduino.h>

void tftInit(void);
void advertence_triangule_sized(void);
void draw_buttons(uint8_t y,uint8_t color);
void drawControlScreem(void);
void screemParam(uint16_t *x, uint16_t *y);
bool touchDetect(void);

#endif //_TFTMANAGER__
```

## E.2. Premisas

```
#include "tftmanager.h"
#include "common.h"

#include <Adafruit_GFX.h>    // Core graphics library
#include <Adafruit_TFTLCD.h> // Hardware-specific library
#include <Arduino.h>
#include <TouchScreen.h>
#include <avr/pgmspace.h>



TouchScreen ts = TouchScreen(XP, YP, XM, YM, 300);
/*Inicializa objeto de las librerías, para manejar la parte tactil del tft
 * XP, YP, XM, YM son pines de la tarjeta para la comunicación. Son compartidos para la escritura en la pantalla
 * 300, necesario para calibrar la presión
*/

Adafruit_TFTLCD tft(LCD_CS, LCD_CD, LCD_WR, LCD_RD, LCD_RESET);
/*Inicializa objeto de las librerías, para manejar tft
 * LCD_CS, LCD_CD, LCD_WR, LCD_RD, LCD_RESET son pines para el control del TFT
*/
```

## E.3. Configuración pantalla (tftInit)

```
void tftInit(void){
    tft.begin(tft.readID());
    tft.setRotation(3);
    tft.fillScreen(BLACK);
}
```

## E.4. Triangulo de advertencia (advertence_triangule_sized)

```
void advertence_triangule_sized(void){


  tft.fillTriangle(  160,  60, 90,  180, 230,  180, YELLOW);

  tft.drawTriangle(  160,  60, 90,  180, 230,  180, RED);
  tft.drawTriangle(  160,  61, 91,  179, 229,  179, RED);
  tft.drawTriangle(  160,  62, 92,  178, 228,  178, RED);

  tft.setCursor(147, 95);
    tft.setTextColor(RED);  tft.setTextSize(6);
    tft.println(F("!"));
  tft.setCursor(129, 152);
    tft.setTextColor(RED);  tft.setTextSize(3);
    tft.println(F("WAIT"));
}
```

## E.5. Lanzador de interfaz (drawbuttons)

```
void draw_buttons(uint8_t y,uint8_t color){
for(uint8_t i=0;i<3;i++){
   |

//Adafruit_TFTLCD tftx(LCD_CS, LCD_CD, LCD_WR, LCD_RD, LCD_RESET);
tft.fillRoundRect(290, y+i*60, 50, 60, 10, pgm_read_word_near(camColors+i*100));

tft.drawRoundRect(290, y+i*60, 50, 60, 10, pgm_read_word_near(camColors+30+i*100));
tft.drawRoundRect(291, y+1+i*60, 50, 60-2, 8, pgm_read_word_near(camColors+10+i*100));
//tft.drawRoundRect(292, y+2, 50, 60-4, 6, camColors[color+20]);
//tft.drawRoundRect(293, y+3, 50, 60-6, 4, camColors[color+10]);



}
}
```

### E.6. Lanzador de selector de contraste (drawControlScreem)

```
void drawControlScreem(void){

  tft.setCursor(35, 85);
  tft.setTextColor(WHITE);  tft.setTextSize(6);
  tft.println(F("WELCOME"));

  tft.setCursor(35, 135);
  tft.setTextColor(WHITE);  tft.setTextSize(1);
  tft.println(F("THERMAL CAM"));


  delay(2500);
  tft.fillScreen(BLACK);


  tft.setCursor(55, 55);
  tft.setTextColor(WHITE);  tft.setTextSize(1);
  tft.println(F("Choose contrast mode:"));

  tft.fillRoundRect(55, 75, 90, 90, 10, 0x53AE);
  tft.drawRoundRect(55, 75, 90, 90, 10, WHITE);
  tft.setCursor(75, 85);
  tft.setTextColor(BLACK);  tft.setTextSize(10);
  tft.println(F("C"));

  tft.fillRoundRect(175, 75, 90, 90, 10, 0x53AE);
  tft.drawRoundRect(175, 75, 90, 90, 10, WHITE);
  tft.setCursor(195, 85);
  tft.setTextColor(BLACK);  tft.setTextSize(10);
  tft.println(F("R"));

}
```

## E.7. Parámetros al pulsar pantalla (screemParam)

```
void screemParam(uint16_t *x, uint16_t *y){
  //TouchScreen ts = TouchScreen(XP, YP, XM, YM, 300); mismo razonamiento que para tft
  digitalWrite(13, HIGH);
  TSPoint p = ts.getPoint();
  digitalWrite(13, LOW);

  // if sharing pins, you'll need to fix the directions of the touchscreen pins
  //pinMode(XP, OUTPUT);
  pinMode(XM, OUTPUT);
  pinMode(YP, OUTPUT);
  //pinMode(YM, OUTPUT);



   if (p.z > MINPRESSURE && p.z < MAXPRESSURE) {

    // scale from 0->1023 to tft.width
    *y = (int)map(p.x, TS_MINX, TS_MAXX, 0, tft.height());//en lo referente al lado estrecho de la pantalla
    //*y=tft.height()-*y;

    *x = (int)(tft.width()-map(p.y, TS_MINY, TS_MAXY, tft.width(), 0)); //en lo referente al lado largo de la pantalla
    //*x=tft.width()-*x;

/* //usar para calibrar pantalla
    tft_fillRect(252, 0, 31, 10, WHITE);
  tft_fillRect(252, 229, 31, 10, WHITE);
  tft_setCursor(253, 1);
  tft_setTextColor(BLACK);  tft_setTextSize(1);
  tft_println(*x);
  tft_setCursor(253, 230);
  tft_setTextColor(BLACK);  tft_setTextSize(1);
  tft_println(*y);
     */
  }
  else{
    *y=0;
    *x=0;
  }
}
```

## E.8. Detector de pulsación (touchDetect)

```
bool touchDetect(void){

return ts.isTouching();


}
```

# Anexo F. Generación de imágenes (PmgToSD)

## F.1. Archivo de cabecera

```
#ifndef __PMGTOSD_
#define __PMGTOSD_

#include <stdint.h>
#include <avr/pgmspace.h>

//inicializar SD
bool SD_init(void);
//crear una nueva pmg
//escribe cabecera,
//mapa de colores y cierra el archivo
// para ser guardado correctamente
void bmp_create(float *p, uint8_t contrast);

#endif //__PMGTOSD_
```

## F.2. Premisas

```
#include "PmgToSD.h"
#include <SD.h>
#include <SPI.h>
#include "interpolar.h"
#include "common.h"
#include <avr/pgmspace.h>
#include "thermalbar.h"

static void new_pmgfile(File *pmgfile);

static void create_Header(File *pmgfile);

static void thermalbarToBmp(File *bmpfile, uint8_t contrast);
```

## F.3. Generador de cabecera BMP (create_Header)

```
/*static*/ void create_Header(File *pmgfile){
  uint16_t rowSize = 4 * ((3*PMG_SIZE_W + 3)/4);      // how many bytes in the row (used
  uint16_t fileSize = 54 + PMG_SIZE_H*rowSize;        // headers (54 bytes) + pixel data

  // create file headers (also taken from StackOverflow example)
  unsigned char bmpFileHeader[14] = {            // file header (always starts with BM!)
    'B','M', 0,0,0,0, 0,0, 0,0, 54,0,0,0   };
  unsigned char bmpInfoHeader[40] = {            // info about the file (size, etc)
    40,0,0,0, 0,0,0,0, 0,0,0,0, 1,0, 24,0   };

  bmpFileHeader[ 2] = (unsigned char)(fileSize      );
  bmpFileHeader[ 3] = (unsigned char)(fileSize >>  8);
  bmpFileHeader[ 4] = (unsigned char)(fileSize >> 16);
  bmpFileHeader[ 5] = (unsigned char)(fileSize >> 24);

  bmpInfoHeader[ 4] = (unsigned char)(       PMG_SIZE_W      );
  bmpInfoHeader[ 5] = (unsigned char)(       PMG_SIZE_W >>  8);
  bmpInfoHeader[ 6] = (unsigned char)(       PMG_SIZE_W >> 16);
  bmpInfoHeader[ 7] = (unsigned char)(       PMG_SIZE_W >> 24);
  bmpInfoHeader[ 8] = (unsigned char)(       PMG_SIZE_H      );
  bmpInfoHeader[ 9] = (unsigned char)(       PMG_SIZE_H >>  8);
  bmpInfoHeader[10] = (unsigned char)(       PMG_SIZE_H >> 16);
  bmpInfoHeader[11] = (unsigned char)(       PMG_SIZE_H >> 24);

  // write the file (thanks forum!)
  pmgfile->write(bmpFileHeader, sizeof(bmpFileHeader));    // write file header
  pmgfile->write(bmpInfoHeader, sizeof(bmpInfoHeader));    // " info header

}
```

## F.4. Inicializar tarjeta (SD_init)

```
bool SD_init(void){
   //Init SD_Card
   //Deben de modificarse los ficheros oportunos para poder usarse en Mega
  pinMode(10, OUTPUT);

return SD.begin(10);
}
```

## F.5. Generador de nuevo archivo BMP (new_pmgfile)

```
void new_pmgfile(File *pmgfile){
  char name[] = "9px_0000.bmp";       // filename convention (will auto-increment)
//esta funciï¿½n crea un nuevo archovo pmg, con un nuevo nombre que no esta en uso;
 // if name exists, create new filename
  for (int i=0; i<10000; i++) {
    name[4] = (i/1000)%10 + '0';     // thousands place
    name[5] = (i/100)%10 + '0';      // hundreds
    name[6] = (i/10)%10 + '0';       // tens
    name[7] = i%10 + '0';            // ones
    if (SD.open(name, O_CREAT | O_EXCL | O_WRITE)) {
      break;
    }
  }
    (*pmgfile) = SD.open(name,FILE_WRITE);
  // set fileSize (used in bmp header)
}
```

## F.6. Generador de barra térmica en el BMP (thermalbarToBmp)

```
void thermalbarToBmp(File *pmgfile, uint8_t contrast){
  uint8_t crear_bmp[3*BOXSIZE];

  uint8_t n;

  for(uint16_t j=0;j<BOXSIZE;j++){

    for(uint8_t y=0;y<24;y=y+1){ //si aumentamos alto de pantalla, aumentar ell almacenamiento de y a uint16_t

      uint8_t colorIndex;

      if(contrast==MODE_REAL){

        colorIndex = map(get_min_t()+(float)y*((get_max_t()-get_min_t())/23.0), MINTEMP, MAXTEMP, 0, 255);
        colorIndex = constrain(colorIndex, 0, 255);
      }
      else if(contrast==MODE_CONT){

        colorIndex = map(get_min_t()+(float)y*((get_max_t()-get_min_t())/239.0), get_min_t(), get_max_t(), 0, 255);
        colorIndex = constrain(colorIndex, 0, 255);
      }

        uint16_t color;
        color=pgm_read_word_near(camColors+colorIndex);

        n=0;

        for(uint16_t i=0;i<BOXSIZE;i++){

            crear_bmp[n+2]=(color>>11)<<3;//R
            crear_bmp[n+1]=((color>>5)&0x3f)<<2;//G
            crear_bmp[n+0]=(color&0x1f)<<3;//B

            n+=3;
        }

        for(uint16_t r=0;r<3*(BOXSIZE);r++){ //se escribe un cuadrado de BOXSIZExBOXSIZE pixeles por cada pixel
          (*pmgfile).write(crear_bmp[r]);
        }
    }
  }
}
```

## F.7. Generador de BMP

```
void bmp_create(float *p , uint8_t contrast){

    File pmgfile;
    new_pmgfile(&pmgfile);
    create_Header(&pmgfile);

    uint8_t crear_bmp[3*BOXSIZE]; //dado que cada color que devuelve camColors es de 16 bits,
    //necesitares 3 bytes para crear caca color para pasarlo al BMP

    //formato bitmap de 24 bits. Cada uno de los 3 bytes asociados a un color, conforma el RGB,
    // R correspontera con la posicion 3 de cada 3, G con la 2 de 3 y B con la 1 de 3
    //ademas, en lo referido a camColors, devuelve colores de 16bits, en el que los
    // 5 MSB se reserban para el RED, 6 intermedios al GREEN y los 5 LSB al BLUE
    //para cada color, de 16 bits, 2bytes, necesitaresmos 3 bytes para sacar el bmp

    float colorTemp;
    uint16_t n=0;

    //for(int i=0; i<(AMG_ROWS*AMG_COLS/*aqui habria que introducir el tamaï¿½o del buffer_t);i++){

    for (uint16_t y=0; y<(PMG_W/BOXSIZE); y++) {
      for(uint16_t j=0;j<BOXSIZE;j++){

    for (uint16_t x=0; x<(PMG_H/BOXSIZE); x++) {
      n=0;
```

```
      float val= get_point(p, (PMG_W/BOXSIZE), (PMG_H/BOXSIZE), x,(PMG_H/BOXSIZE)- y);
      //tener cuidodado con eso, se accede asi a x e y para se la imagen tenga la misma rotacion que en la camara
      if(val >= MAXTEMP) colorTemp = MAXTEMP;
      else if(val <= MINTEMP) colorTemp = MINTEMP;
      else colorTemp = val;
      uint16_t colorIndex;
      if(contrast==MODE_REAL){
        colorIndex = map(colorTemp*10, MINTEMP*10, MAXTEMP*10, 0, 255);
        colorIndex = constrain(colorIndex, 0, 255);
      }
      if(contrast==MODE_CONT){
        colorIndex = map(colorTemp*10, get_min_t()*10, get_max_t()*10, 0, 255);
        colorIndex = constrain(colorIndex, 0, 255);
      }

      for(uint16_t i=0;i<BOXSIZE;i++){

      crear_bmp[n+2]=(pgm_read_word_near(camColors+colorIndex)>>11)<<3;//R
      crear_bmp[n+1]=((pgm_read_word_near(camColors+colorIndex)>>5)&0x3f)<<2;//G
      crear_bmp[n+0]=(pgm_read_word_near(camColors+colorIndex)&0x1f)<<3;//B

      n+=3;
      }
         for(uint16_t r=0;r<3*(BOXSIZE);r++){ //se escribe un cuadrado de BOXSIZExBOXSIZE pixeles por cada pixel
        pmgfile.write(crear_bmp[r]);
        }

      }

  }

}

thermalbarToBmp(&pmgfile, contrast);

pmgfile.close();

}
```

# Anexo G. Archivo de cabecera común

```
#ifndef __COMMON_
#define __COMMON_

#include <Adafruit_TFTLCD.h> // Hardware-specific library
#include <avr/pgmspace.h>

//low range of the sensor (this will be blue on the screen)
#define MINTEMP 0
//high range of the sensor (this will be red on the screen)
#define MAXTEMP 80

#define AMG_COLS 8
#define AMG_ROWS 8


#define INTERPOLATED_COLS 24
#define INTERPOLATED_ROWS 24


////////////////////// necesario para obtener pulsacion pantalla

#define YP A3  // must be an analog pin, use "An" notation!
#define XM A2  // must be an analog pin, use "An" notation!
#define YM 9   // can be a digital pin
#define XP 8   // can be a digital pin

#define TS_MINX 120
#define TS_MINY 80
#define TS_MAXX 920
#define TS_MAXY 920


////////////////////////////////////

#define LCD_CS A3 // Chip Select goes to Analog 3
#define LCD_CD A2 // Command/Data goes to Analog 2
#define LCD_WR A1 // LCD Write goes to Analog 1
#define LCD_RD A0 // LCD Read goes to Analog 0

#define LCD_RESET A4 // Can alternately just connect to Arduino's reset pin


#define BLACK   0x0000
#define BLUE    0x001F
#define RED     0xF800
#define GREEN   0x07E0
#define CYAN    0x07FF
#define MAGENTA 0xF81F
#define YELLOW  0xFFE0
#define WHITE   0xFFFF



#define MINPRESSURE 10
#define MAXPRESSURE 1000


#define BOXSIZE 10
////////////////////77
#define interruptPin 18
////////////////////
```

```
//the colors we will be using
const uint16_t camColors[] PROGMEM= {0x480F,
0x400F,0x400F,0x400F,0x4010,0x3810,0x3810,0x3810,0x3810,0x3010,0x3010,
0x3010,0x2810,0x2810,0x2810,0x2810,0x2010,0x2010,0x2010,0x1810,0x1810,
0x1811,0x1811,0x1011,0x1011,0x1011,0x0811,0x0811,0x0811,0x0011,0x0011,
0x0011,0x0011,0x0011,0x0031,0x0031,0x0051,0x0072,0x0072,0x0092,0x00B2,
0x00B2,0x00D2,0x00F2,0x00F2,0x0112,0x0132,0x0152,0x0152,0x0172,0x0192,
0x0192,0x01B2,0x01D2,0x01F3,0x01F3,0x0213,0x0233,0x0253,0x0253,0x0273,
0x0293,0x02B3,0x02D3,0x02D3,0x02F3,0x0313,0x0333,0x0333,0x0353,0x0373,
0x0394,0x03B4,0x03D4,0x03D4,0x03F4,0x0414,0x0434,0x0454,0x0474,0x0474,
0x0494,0x04B4,0x04D4,0x04F4,0x0514,0x0534,0x0534,0x0554,0x0554,0x0574,
0x0574,0x0573,0x0573,0x0573,0x0572,0x0572,0x0572,0x0571,0x0591,0x0591,
0x0590,0x0590,0x058F,0x058F,0x058F,0x058E,0x05AE,0x05AE,0x05AD,0x05AD,
0x05AD,0x05AC,0x05AC,0x05AB,0x05CB,0x05CB,0x05CA,0x05CA,0x05CA,0x05C9,
0x05C9,0x05C8,0x05E8,0x05E8,0x05E7,0x05E7,0x05E6,0x05E6,0x05E6,0x05E5,
0x05E5,0x0604,0x0604,0x0604,0x0603,0x0603,0x0602,0x0602,0x0601,0x0621,
0x0621,0x0620,0x0620,0x0620,0x0620,0x0E20,0x0E20,0x0E40,0x1640,0x1640,
0x1E40,0x1E40,0x2640,0x2640,0x2E40,0x2E60,0x3660,0x3660,0x3E60,0x3E60,
0x3E60,0x4660,0x4660,0x4E60,0x4E80,0x5680,0x5680,0x5E80,0x5E80,0x6680,
0x6680,0x6E80,0x6EA0,0x76A0,0x76A0,0x7EA0,0x7EA0,0x86A0,0x86A0,0x8EA0,
0x8EC0,0x96C0,0x96C0,0x9EC0,0x9EC0,0xA6C0,0xAEC0,0xAEC0,0xB6E0,0xB6E0,
0xBEE0,0xBEE0,0xC6E0,0xC6E0,0xCEE0,0xCEE0,0xD6E0,0xD700,0xDF00,0xDEE0,
0xDEC0,0xDEA0,0xDE80,0xDE80,0xE660,0xE640,0xE620,0xE600,0xE5E0,0xE5C0,
0xE5A0,0xE580,0xE560,0xE540,0xE520,0xE500,0xE4E0,0xE4C0,0xE4A0,0xE480,
0xE460,0xEC40,0xEC20,0xEC00,0xEBE0,0xEBC0,0xEBA0,0xEB80,0xEB60,0xEB40,
0xEB20,0xEB00,0xEAE0,0xEAC0,0xEAA0,0xEA80,0xEA60,0xEA40,0xF220,0xF200,
0xF1E0,0xF1C0,0xF1A0,0xF180,0xF160,0xF140,0xF100,0xF0E0,0xF0C0,0xF0A0,
0xF080,0xF060,0xF040,0xF020,0xF800,};



////////////// guardar imagenes en SD
//para mostar la imagen
#define PMG_W 240 //tamaÃ±o del vector que contiene los pixeles por boxsize-> para mostar la imagen
#define PMG_H 240
//define el tamaÃ±o completo del pmg (cuando queremos aÃ±adir ademas de la imagen, el termal bar
#define PMG_SIZE_W 240
#define PMG_SIZE_H 259
```