CAMPUS
DE EXCELENCIA
INTERNACIONAL

Graduado en Ingenería Informática

Universidad Politécnica de Madrid

Escuela Técnica Superior de
Ingenieros Informáticos

TRABAJO FIN DE GRADO

Captura y procesamiento de imágenes de una cámara térmica

Memoria Final

Autor: Rubén Pascual Arribas

Director: José Luis Pedraza Domínguez

MADRID, JULIO 2016

1. ÍNDICE

1. Índice 1
2. Resumen 2
3. Introducción y objetivos 3
3.1 Motivación 3
3.2 Objetivos 3
4. Desarrollo 5
4.1 Trabajos previos 5
4.2 Entorno de desarrollo 6
4.3 Hardware utilizado 7
4.3.1 Cámara térmica Flir Tau 2 7
4.3.2 Capturadora de vídeo USB 8
4.4 Planificación 8
4.4.1 Organización de las tareas 8
4.4.2 Problemas con la cámara 10
4.4.3 Problemas encontrados durante el desarrollo 12
4.5 Proceso de obtención de la temperatura a partir de imágenes 12
4.6 Descripción de la API 16
4.6.1 Requisitos 16
4.6.2 Configuración de la aplicación 17
4.6.3 Funciones auxiliares 18
4.6.4 Funciones principales 19
5. Resultados 34
6. Conclusiones 38
7. Bibliografía 39
8. Agradecimientos 41

2. RESUMEN

El uso de la termografía para el estudio del comportamiento de las plantas sometidas a diferentes condiciones ambientales puede revelar diferencias en la temperatura en la superficie de las hojas según la salinidad a la que estén sometidas. Para estudiar este fenómeno en un proyecto del VI Plan Nacional de I+D+i es necesaria la utilización de una cámara térmica que tome imágenes de las plantas sometidas a diferentes niveles de salinidad. En este trabajo fin de grado se ha desarrollado una interfaz de programación de aplicaciones software que permita automatizar el proceso de toma de imágenes y la medida de la temperatura en éstas. Esta interfaz se ha diseñado para cámaras térmicas Flir Tau en versiones 2.0 y posteriores, siguiendo las especificaciones de la guía de comandos software del fabricante de la cámara. El código de la aplicación se ha desarrollado en lenguaje C para funcionar en plataformas Linux.

Abstract

The use of thermal imaging amongst researchers studying plants can expose differences between the temperatures of plants depending on the salinity of the irrigation. To study this phenomenon and perform research on it we can use thermal cameras that allow us to take pictures of plants under different salinity levels. In this final degree dissertation, I have developed an application programming interface that allows the user to automate the process of taking images from the thermal camera and the measurement of temperatures. This interface is only meant to work with Flir Tau cameras in versions 2.0 and later, that follow the specifications detailed in the software reference guide. The application code has been developed on Linux platform and is written in C language.

3. INTRODUCCIÓN Y OBJETIVOS

El tema se este trabajo fin de grado se enmarca en el ámbito de un proyecto del VI Plan Nacional de I+D+i denominado *"Entrada de Na+ en la raíz y tolerancia al NaCl en las plantas: aplicación para su estudio de un análisis funcional y genético"* [13]. En él se intenta automatizar la captación de características de crecimiento de plantas sometidas a riego salino. Concretamente, se estudia *Arabidopsis*, una planta que crece paralela al terreno, lo que facilita la tarea de observar toda la superficie de ésta.

La característica que nos interesa medir es la temperatura. Para ello contamos con una cámara térmica con módulo de conectividad USB (VPC module), cuyas características se encuentran disponibles en el documento [5].

3.1 Motivación

La cámara que vamos a utilizar para medir temperaturas se puede manejar a través de una interfaz gráfica de usuario en sistemas Windows, pero el objetivo del proyecto es poder controlar la cámara desde un brazo robótico en el laboratorio que vaya posicionándose encima de cada planta. Lógicamente no podemos montar un equipo muy grande en el brazo robótico o situar el equipo muy lejos de la cámara, motivo por el cual ésta se controlará a través de un computador de tamaño muy pequeño (*Raspberry Pi*) que ejecutará un sistema operativo Linux.

Para acceder a la funcionalidad de la cámara de forma automática no podemos utilizar la interfaz gráfica que proporciona el fabricante de forma gratuita, sino que deberíamos utilizar un kit de desarrollo software (SDK) específico que ofrece el fabricante [enlace], que en este caso es de pago y tiene un coste muy elevado. Es en este momento donde surge la necesidad de implementar una API específica para esta cámara y para las condiciones del proyecto de investigación descritas anteriormente. La descripción de la API se encuentra más adelante. También se ha implementado un software que hace uso de esta API y se encarga de almacenar las imágenes de las plantas y obtener las medidas de temperatura de esas imágenes.

3.2 Objetivos

Los objetivos que se han establecido para este proyecto son los siguientes:

- Conocer y describir las características e interfaz de comunicación de un módulo controlador de cámara térmica.
- Diseñar e implementar una API sencilla que permita al programador la comunicación con el módulo controlador de la cámara térmica.

- Diseñar e implementar una aplicación elemental mínima que permita validar la API implementada.
- Implementar un software básico de análisis y filtrado de imágenes obtenidas que permita medir de temperatura de diferentes zonas de la imagen captada por la cámara térmica, utilizando la API mencionada en el apartado anterior.

4. DESARROLLO

4.1 Trabajos previos

El proyecto de investigación para el que se va a utilizar la cámara térmica estudiará el efecto que tiene la salinidad del agua en la temperatura y crecimiento de la planta *Arabiopsis* [13].

Existen investigaciones previas sobre este tema, que han encontrado relación entre la salinidad o la sequía y la temperatura de las plantas, como por ejemplo las referencias [7] y [12].

"we compared the potential of high-resolution thermography and infrared thermometry to discriminate among stress treatments (control, drought, salt and combined salt and drought) (...) with differences varying between 1–9 °C" [7].

La parte en la que se centra este trabajo fin de grado es el desarrollo de un software cuyo fin último es obtener la temperatura de diversos ejemplares de *Arabidopsis* para estudiar su relación con la salinidad.

Ya existen APIs que permiten acceder a la señal de vídeo producida por una cámara (en este caso sería imagen térmica) como por ejemplo Video4Linux2 [8], que está integrada en el kernel Linux, y se podría usar en una Raspberry Pi [9]. Se podría capturar una imagen fija del video digital para después calcular la temperatura por zonas. El problema de esta solución es que nuestra cámara no tiene el módulo de vídeo digital (Camera Link Expansion Board). En cambio, sí que dispone de salida de vídeo analógico, pero la calidad disminuye respecto a una imagen fija y el procesado de imagen sería computacionalmente costoso. Por tanto que debemos capturar las imágenes fijas mediante comandos y posteriormente recibir los datos de las imágenes a través del puerto serie USB. Se ha optado por desarrollar una interfaz ad-hoc ya que los comandos necesarios para acceder a la funcionalidad de la cámara están definidos en el documento [1].

Otro de los aspectos que hay que desarrollar para poder enviar y recibir comandos de la cámara es el cálculo de códigos de comprobación de errores (CRC). Para el cálculo del CRC existen librerías que calculan estos códigos de forma optimizada utilizando tablas previamente calculadas. En [4] hay disponible una librería implementada usando tablas en las que se han calculado previamente todas las combinaciones posibles entre el byte de entrada de datos y el polinomio desplazado. La implementación que se ha realizado para usarse en la API de la cámara térmica ha sido realizada por mí, apoyándome en el pseudocódigo del algoritmo disponible en [3]. Es cierto que el cálculo se podría optimizar

utilizando las tablas previamente calculadas pero es una mejora que dejo para añadir en un futuro en función de la planificación del proyecto.

El software implementado trabaja con imágenes que descarga de la cámara y se deben almacenar para su posterior procesado. Para almacenar los datos de imágenes como ficheros de imagen comunes me he decantado por utilizar el formato PNG (Portable Network Graphics), que permite compresión e incluir metadatos en las imágenes en forma de *text chunks*(trozos de texto). Existe una librería llamada libPNG [9] que permite crear imágenes png que viene incluida en las distribuciones Ubuntu, y ha sido ampliamente utilizada, motivos por los que he decidido utilizarla para este proyecto. Para generar las imágenes a partir de los datos recibidos por la cámara, me he basado en un código previo [10] que he modificado para ajustar a mis necesidades.

Para el cálculo de la temperatura de las zonas de la imagen, que se describe en el apartado 4.5, he utilizado

4.2 Entorno de desarrollo

Para el desarrollo del código de la aplicación se ha instalado una máquina virtual con sistema operativo Ubuntu 14.04 64-bit. Se ha escogido este sistema operativo debido a que se ha trabajado con él a lo largo de muchas asignaturas del grado y, por tanto, es familiar. Además, el software desarrollado se utilizará en una placa Raspberry Pi con sistema operativo Raspbian, siendo ambos derivados del mismo sistema operativo (Debian). El entorno de desarrollo utilizado ha sido CodeBlocks ya que deseaba una interfaz gráfica y ligera que me permitiese compilar y depurar con más facilidad.

El software desarrollado consta de un conjunto de funciones enfocadas a enviar comandos a la cámara y otras funciones que se encargan de obtener información de las imágenes capturadas con esta, establecer ajustes, calcular temperaturas etc. Además, se han separado los algoritmos que se repetirían en varias funciones en funciones auxiliares para reducir la complejidad y mejorar la mantenibilidad del código.

4.3 Hardware utilizado

4.3.1 Cámara térmica Flir Tau 2

El componente hardware principal de este trabajo es una cámara térmica Flir Tau 2. Las características principales de esta cámara son las siguientes:

- Resolución de imagen: 336x256 píxeles
- Rango de temperatura de la escena: -40 a 160ºC (alta ganancia), -40 a 550ºC (baja ganancia)
- Sensor de imagen formado por micro bolómetros de 17µm
- Campo espectral 7.5 - 13.5 µm

El número de serie de la cámara utilizada inicialmente es 46336025H-SPNLXDC1445. Aunque debido al fallo del funcionamiento de esta cámara también se ha utilizado otra prestada similar a la anterior, pero con una resolución de 640x512 píxeles.

Ninguna de ellas cuenta con funcionalidad de radiometría avanzada, con lo cual sólo se tiene información de la temperatura en la zona central de la imagen y el resto se debe calcular según una relación lineal. Según la nota de aplicación de radiometría ([6]) a cada uno de los valores de los píxeles de la imagen le corresponde una temperatura si está configurada la opción *Tlinear* en la cámara. Lamentablemente esta funcionalidad no está disponible en el modelo de cámara disponible, lo que dificulta la medición de la temperatura. En caso de activar *Tlinear* el valor 0 de un píxel de la imagen se correspondería con el valor mínimo de temperatura que acepta la cámara, y el valor 2^n-1 sería el valor máximo de temperatura que acepta la cámara, para una imagen de n bits. Se obtendría entonces una precisión de 0.04 grados Kelvin al discretizar la medida en la imagen. La fórmula para calcular la temperatura de un píxel cualquiera de la imagen sería la siguiente:

$$temperatura\ del\ pixel\ i = \frac{temperatura\ del\ pixel\ central}{valor\ del\ pixel\ central} * valor\ del\ píxel\ i$$

La única información de temperatura de la que dispone esta cámara es un medidor de temperatura en el centro de la imagen, que muestra valores de flujo radiométrico entre 0 y 2^{14}-1 o valores enteros de grados centígrados. A partir de ahí se podría hacer una estimación de diferencias de temperatura entre el valor de los 4 píxeles centrales de la imagen y el resto.

Otro impedimento que nos encontramos es que los datos de las imágenes tomadas con una profundidad de 14 bits están comprimidos para almacenarse en la cámara y al descargarlas no podemos acceder a los datos sin comprimir, ya que la documentación no especifica cuál es el algoritmo de compresión utilizado. Por lo tanto, nos quedan datos de imágenes de 8 bits, esto es, 256 valores posibles de temperatura, que debemos transformar previamente a 14 bits para después hallar la equivalencia en temperatura según la relación lineal establecida.

4.3.2 Capturadora de vídeo USB

Modelo: EasyCAP usb2.0. Chip Syntek Stk1160
Para probar la implementación de los ajustes de brillo y contraste se ha utilizado una capturadora de vídeo analógico con interfaz USB. Esto ha permitido comprobar visualmente el efecto de forma inmediata que tenían los comandos de contraste y brillo enviados a la cámara, en lugar de tener que capturar una imagen y descargarla para visualizar el resultado.

4.4 Planificación

4.4.1 Organización de las tareas

La planificación original de este proyecto no se ha podido seguir según lo planeado, debido al fallo en el funcionamiento de la cámara térmica explicado en el siguiente apartado.

La distribución del tiempo empleado en diferentes tareas es la siguiente:

- Reuniones con el tutor: durante el semestre de la realización de este trabajo he mantenido reuniones con mi tutor todas las semanas lectivas, con una duración media de 1 hora y 30 minutos. En ellas se han planteado y resuelto dudas, se ha revisado el trabajo realizado y la planificación y se han solucionado errores en el código. He de agradecer a mi tutor la gran ayuda que me ha prestado y su implicación.
- Lectura inicial de la documentación: 12 horas.
- Diseño de la interfaz de comunicación: 3 horas. El diseño inicial de la interfaz se basaba en un conjunto reducido de funciones que realizaban la funcionalidad básica de la API, y que están basadas en los comandos de la cámara descritos en el documento [1]. Más tarde se ha ampliado con alguna función que se ha visto necesaria durante el desarrollo.
- Implementación de las funciones de la API relacionadas con la comunicación con la cámara: 80 horas. La carga de trabajo de esta parte se encuentra repartida en dos períodos, principalmente. El primero abarca desde el día 4 de octubre hasta el 1 de noviembre. A partir de esa fecha la cantidad de tiempo dedicado a este apartado ha sido baja pero constante, con modificaciones pequeñas debido a que no se podían depurar ya que no disponíamos de la cámara en ese momento. El segundo período abarca desde el 21 y 22 de diciembre y del 7 al 10 de enero, fechas en las que he podido disponer de una cámara que funcionaba de forma similar a la original. En esos días se han podido probar las funcionalidades relacionadas con el acceso a la memoria flash de la cámara, la descarga de imágenes y la medición de temperatura de la cámara. Durante el periodo de Navidad no he podido disponer de la cámara, pero he podido revisar la comunicación que se intercambia entre la cámara y el PC-Windows al utilizar el software de interfaz gráfica del fabricante gracias a las capturas realizadas previamente con la herramienta Wireshark.
- Depuración y pruebas de las funciones de la API relacionadas con la comunicación con la cámara: 40 horas. En el inicio del proyecto, la puesta en marcha de la primera funcionalidad de la cámara me llevó bastante tiempo debido a que no conseguía configurar correctamente el puerto serie y el cálculo del código de redundancia cíclica, como describo en el apartado 4.3.3. Estos eran requisitos previos para poder enviar mensajes a la cámara. A mediados de noviembre pudimos utilizar una cámara con una versión de firmware anterior a la soportada por el software desarrollado, que sólo nos hizo perder el tiempo porque el código no funcionaba correctamente y no sabíamos la causa.
- Estudio de algoritmos de filtrado: 10 horas. Se han estudiado los algoritmos de medida de temperatura comunes. Ante la posibilidad de que la cámara que nos prestasen contase con radiometría avanzada pensé en utilizar el algoritmo descrito en el apartado 3. Puesto que finalmente no hemos utilizado ninguna cámara con

esta característica he optado por obtener el valor de temperatura con el medidor de la cámara en la zona central de la imagen para después hacer un cálculo lineal y obtener el valor de temperatura del resto de píxeles de la imagen.

- Implementación de las funciones de la API relacionadas con la generación de imágenes y cálculo de temperatura: 35 horas. Se han estudiado los distintos formatos de imagen, sobre todo bmp y png. Y se ha optado por utilizar este último ya que la librería libPNG simplifica mucho el proceso de creación de imágenes. Se han implementado distintas funciones, para varios formatos, ya que inicialmente no podíamos saber qué datos nos iban a llegar de la cámara porque no podíamos probarlo.
- Depuración de las funciones de la API relacionadas con la generación de imágenes y cálculo de temperatura: 20 horas.
- Documentación del proyecto: 20 horas.

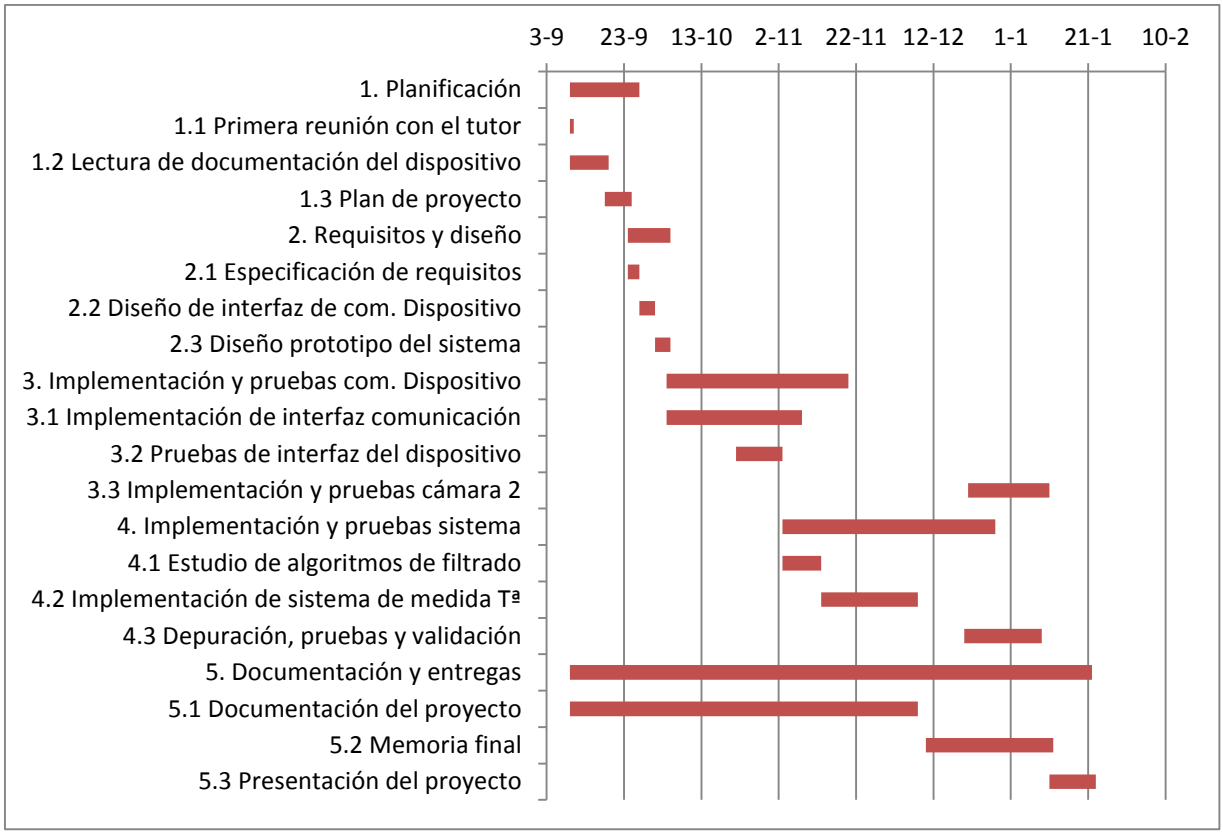


Figura 1. Tareas en diagrama de Gantt

4.4.2 Problemas con la cámara

La cámara original dejó de funcionar el día 27 de octubre, mientras probaba las funciones que accedían a la memoria flash de la cámara. Al conectarla, el puerto serie aparecía

correctamente en la lista de dispositivos conectados, pero no conseguía respuesta al enviarle mensajes. La cámara se envió al fabricante para su reparación. Esto ha supuesto un trastorno en la planificación del proyecto, ya que he tenido que posponer parte de la programación de la API de la cámara y algunas pruebas, para comenzar otras tareas en las que no sea necesario el uso de la cámara, como la obtención de la temperatura en una imagen. Un mes después tuvimos la oportunidad de probar otra cámara similar, prestada por la empresa distribuidora. Después de varias horas probando el código ya desarrollado que ahora no funcionaba descubrimos que la versión del firmware de esa cámara no soportaba los comandos descritos en el documento [1] para los que habíamos desarrollado la API.

Durante el tiempo que no pude utilizar la cámara para probar las funciones desarrolladas, me dediqué a desarrollar la parte de generación de imágenes en formato png y la medida de la temperatura de las imágenes. En este tiempo no podía saber cómo eran los datos que me iban a llegar desde la cámara (si iban a ser simplemente bytes de datos de píxeles en escala de grises, si iban a ser datos de 16, 14 u 8 bits, si las imágenes tendrían una estructura como en un fichero bmp, etc.). Todo esto supuso que perdiera tiempo en desarrollar funciones de prueba con datos que suponía que serían como los que iba a recibir.

Los días 21 y 22 de diciembre pudimos probar otra cámara que nos prestó la empresa distribuidora, similar a la primera, que en este caso tenía un módulo de comunicación distinto. Estuvimos un tiempo intentando encontrar los drivers para poder comunicarnos por el puerto serie con la cámara. También ocurre algo peculiar con esta cámara, las llamadas a read() y write(), a pesar de estar configurado como bloqueante, no se bloquean, y provocaban errores en las funciones por recibir un número inadecuado de bytes. Se ha solucionado incluyendo bucles que terminan una vez leídos los bytes pedidos.

Como no se podía disponer de la cámara durante las Navidades y el proyecto tenía que seguir avanzando con las pruebas de la cámara, nos planteamos capturar el intercambio de mensajes entre la cámara y la aplicación gráfica proporcionada por el fabricante con el software Wireshark. Con este método se consiguió corregir bastantes errores en el código. También se descubrió que la aplicación enviaba mensajes a la cámara que no están documentados, o que los argumentos son distintos a las opciones que aparecen en el manual [1].

La cámara llegó reparada en el mes de marzo, y desde entonces se ha desarrollado la parte de medida de temperatura. En esta parte ha habido varios cambios de rumbo en el proyecto, ya que lo que inicialmente se había planificado hacer no era posible porque la cámara no tiene todas características habilitadas, como se comenta en el apartado 4.5.

Una vez se consiguió obtener datos de temperatura a partir de imágenes descargadas, se hizo necesario vaciar la memoria flash de la cámara, ya que con las pruebas casi se había llenado de imágenes. Se utilizó la aplicación de Windows del fabricante para borrar las imágenes, pero hubo algún problema y ésta se bloqueó. Se esperó un tiempo para ver si se terminaban de borrar y la cámara respondía, pero no fue así y se desconectó la cámara. Al volver a conectarla al puerto USB, la aplicación no la reconocía. La cámara dejó de funcionar de nuevo.

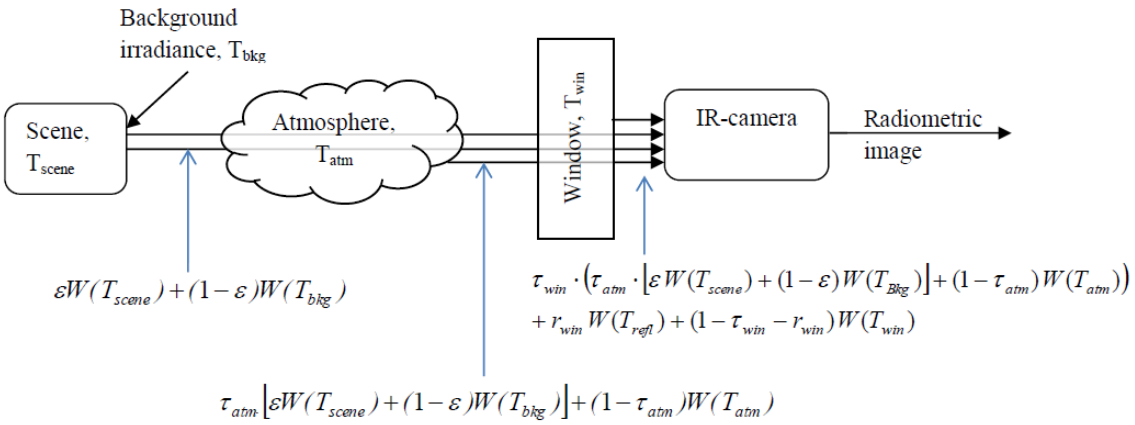
4.4.3 Problemas encontrados durante el desarrollo

Durante la implementación de esta parte de la práctica han surgido problemas y errores, de los cuales cabe destacar los siguientes porque su solución no ha sido trivial:

- Conexión inicial con la cámara. Al tratar de escribir en el puerto serie de la cámara, tuve que cambiar los permisos a lectura y escritura para todos los usuarios, ya que aun estando en el mismo grupo de usuarios que el puerto serie de la cámara (*dialout*) y teniendo permisos para el grupo no conseguía escribir en el puerto.
- Configuración del puerto serie. Después de configurar la paridad, los bits por carácter, etc. según el manual, no conseguía enviar correctamente los mensajes. Lo solucioné al usar la función *cfmakeraw*, que desactiva el procesado de caracteres especiales, el eco y permite la entrada de caracteres individuales.
- Error en el algoritmo del CRC. El manual de la cámara explicaba el algoritmo para calcular el CRC para un único byte, pero no explicaba cómo había que combinar el dato calculado con el siguiente byte. El algoritmo utilizado es de uso habitual y se denomina *CCITT-16 xmodem* [5]. Para comprobar la corrección del código CRC calculado se ha usado una calculadora online, disponible en [4].
- Errores al calcular las direcciones en get_addr_size(). Las direcciones que devolvía la función se calculaban mal porque el orden de los bytes estaba equivocado y no se incrementaba el puntero al argumento recibido en el byte menos significativo, con lo que el tamaño (que era lo siguiente a leer) también era erróneo.

4.5 Proceso de obtención de la temperatura a partir de imágenes

Según el documento [6] apartado 4.2, la compensación de la temperatura usa el siguiente modelo:



$$S = \tau_{win} \cdot \left(\tau_{atm} \cdot \left[\varepsilon W(T_{scene}) + (1-\varepsilon) W(T_{Bkg})\right] + (1-\tau_{atm}) W(T_{atm})\right) + r_{win} W(T_{refl}) + (1-\tau_{win}-r_{win}) W(T_{win})$$

Notation	Description
S	Value of the 14-bit digital video in counts
ε	Emissivity of the scene.
τ_{win}	Transmission coefficient of the window
T_{win}	Window temperature
r_{win}	Window reflection
T_{refl}	Temperature reflected in the window
τ_{atm}	Transmission coefficient of the atmosphere
	between the scene and the camera
T_{atm}	Atmospheric temperature
T_{bkg}	Background temperature (reflected by the scene)
T_{scene}	Scene temperature
$W(T)$	Radiated flux (in units of counts) as function of the temperature of the radiating object. The conversion from temperature to flux can be done using the camera command GET_FLUX_FROM_TEMP (function code 0xB6)

Ilustración 1. Modelo de compensación de temperatura

La cámara incorpora mecanismos de corrección de la transmisión de energía por parte de la atmósfera, eliminando el flujo radiométrico que ésta emite y compensando su coeficiente de transmisión. Lo mismo sucede con la óptica de la cámara (Window), sólo que en este caso también se aplica una corrección para compensar la reflexión de la temperatura. Los coeficientes de este modelo vienen calibrados de fábrica y sólo se pueden modificar en cámaras con radiometría avanzada.

Como se ha mencionado anteriormente, la cámara de la que se dispone no tiene habilitadas las funcionalidades de radiometría avanzada, que permitirían obtener valores de temperatura de forma sencilla a partir de una imagen ya capturada. Según el documento [6] apartado 6.3, la temperatura de un píxel de una imagen capturada se puede obtener de la siguiente manera:

$$T_{pix} = Res \cdot S$$

Siendo Res la resolución (0.4 o 0.04 según se configure), S el valor del píxel en 14 bits y T_{pix} la temperatura de la escena correspondiente a ese píxel en la imagen.

Nos encontramos con dos problemas. El primero es que para que sea posible hacer esta conversión la cámara debe tener activada la característica *TLinear*, que en nuestra cámara no está disponible. El segundo tiene que ver con la obtención de imágenes en 14 bits, puesto que la cámara permite capturar estas imágenes, pero al descargarlas se utiliza un algoritmo de compresión que se desconoce. Sólo se pueden obtener imágenes sin comprimir en 8bits.

Según el documento [6] apartado 4.2 una alternativa para obtener la temperatura a partir de un valor de 14 bits de flujo radiométrico sería mediante el comando 0xB7 GET_TEMP_FROM_FLUX. Al depurar con la cámara este comando siempre devuelve un código de error 0x0A "CAM _FEATURE_NOT_ENABLED". Tampoco aparece en la lista de comandos del documento [1].

Otra forma de obtener la temperatura sería definir el área de la imagen que se desea medir y utilizar el *spot meter* de la cámara, que nos devolvería la temperatura de esa zona en grados centígrados. Desgraciadamente en nuestra cámara no se puede definir la zona de la imagen a medir, sino que ésta es fija y está situada en los 4 píxeles centrales de la imagen. Precisamente esta característica nos abre la puerta a establecer una relación entre el valor de flujo radiométrico en 14 bits de la imagen y la temperatura que le corresponde.

Para poder hacer la conversión necesitamos obtener primero el valor de flujo radiométrico en 14 bits a partir del valor de 8 bits del píxel en la imagen capturada. Según el documento [5] apartado 3.3.2.6.3 la relación entre estos valores es la siguiente:

$$val8 = m \cdot val14 + b$$

Esta fórmula es una transformación lineal del histograma de la imagen. El factor m representa la pendiente de la recta y b es la ordenada en el origen. Estas variables son equivalentes en fotografía al contraste y el brillo, respectivamente.

Esta fórmula se aplica en la cámara sólo cuando el control de ganancia (AGC) se configura en modo manual. Se puede definir el brillo y contraste deseado mediante comandos enviados a la cámara. En particular, el brillo óptimo se puede obtener en cada momento con el comando READ_ARRAY_AVERAGE, que calcula el valor medio en 14 bits de los píxeles de la imagen. Según el documento [1] también debería devolver el ancho del histograma, lo cual nos permitiría estimar el contraste óptimo, pero en esta cámara no sucede así y deberá hacerse manualmente.

La fórmula que se aplica para calcular la ordenada en el origen y la pendiente de la transformación lineal, según el documento [5] apartado 3.3.2.6.4 es la siguiente:

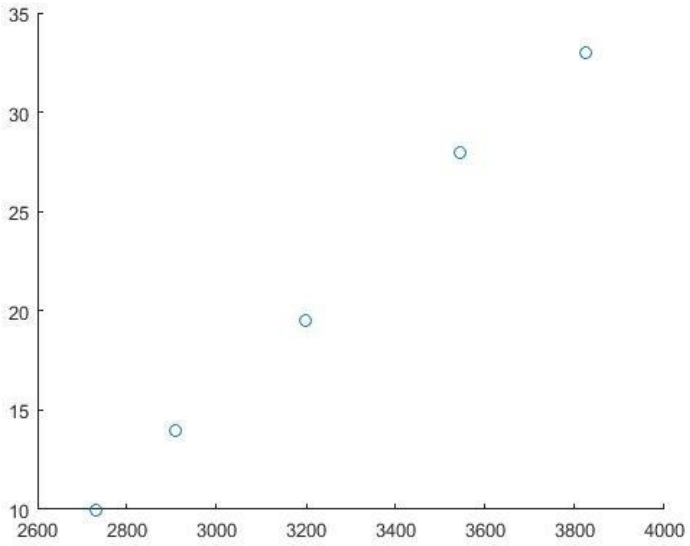
$$m = \frac{contraste}{64}$$

$$b = 127 - brillo * m$$

El contraste se fija mediante el comando CONTRAST y admite valores entre 0 y 255. El brillo se fija mediante el comando BRIGHTNESS y admite valores entre 0 y 16383.

De esta forma, un brillo de valor 127 y contraste 1 asignaría a un valor de 0 en 14 bits el valor 0 en 8 bits, y a un valor de 16383 en 14 bits le asignaría 255 en 8 bits. Este contraste nos sería útil para una escena en la que haya temperaturas tan extremas como el rango que puede capturar la cámara, pero en nuestra aplicación la escena tendrá temperaturas similares, por lo que tendremos que aumentar el contraste para distinguirlas y fijar el brillo en el valor medio de la escena.

El último paso de esta conversión es obtener la temperatura en grados centígrados de cada píxel de la imagen obtenida. Nos apoyamos en el *spot meter* de la cámara, una funcionalidad que permite obtener el flujo radiométrico o la temperatura en grados centígrados de una zona de la imagen. En esta cámara en concreto la zona de medida es fija y se usan los cuatro píxeles centrales de la imagen para calcular el dato. Para conseguir una equivalencia entre flujo radiométrico y temperatura aprovechamos que el medidor se puede configurar para obtener estos tipos de resultados. La API calcula la equivalencia tomando la medida en flujo radiométrico y a continuación se toma la medida en grados centígrados. Se supone que en el tiempo que pasa entre las dos medidas es muy escaso, que la cámara no se mueve y que por tanto esas dos medidas son equivalentes. Como queremos ser capaces de medir la temperatura de toda la imagen y no sólo la zona central necesitamos una función que a cada valor del píxel (0-256) le asigne una temperatura en grados centígrados.



La imagen muestra una serie de medidas de flujo radiométrico (eje x) y su temperatura correspondiente en grados centígrados (eje y). Se puede apreciar que los puntos forman una recta.

Ilustración 2. Medidas de flujo frente a temperatura

Al tomar varias medidas de flujo-temperatura se aprecia que la relación entre ambas es directamente proporcional y su representación forma una recta, por lo que la función de ajuste será lineal.

Según las especificaciones de la cámara, las medidas de temperatura en grados Celsius que da el *spot meter* son números enteros, con lo que se producirá error aleatorio en cada medida por la resolución. Por este hecho, que tendremos que encontrar una función de ajuste a partir de varios puntos que minimice los errores al medir, y este ajuste se puede lograr con el método de mínimos cuadrados. Puesto que este método es muy utilizado en análisis numérico hay librerías con implementaciones probadas y optimizadas que vamos a utilizar en nuestra API.

La librería que vamos a utilizar es GNU Scientific Library (GSL), que incluye una función llamada gsl_fit_linear(), que dados unos datos x,y encuentra los coeficientes de la ecuación lineal utilizando el método de mínimos cuadrados.

Finalmente, una vez se han capturado las imágenes y se han descargado, se pueden obtener las temperaturas de cada uno de sus píxeles según se explica en el apartado 5.

4.6 Descripción de la API

4.6.1 Requisitos

El código desarrollado funciona con cámaras térmicas FLIR TAU 2.0 y posteriores, siempre que sigan las especificaciones del fabricante descritas en el documento [1].

La cámara debe conectarse a un puerto USB. En el caso de Ubuntu (14.04) no es necesario instalar ningún controlador, y aparece como dispositivo en /dev/ttyUSB0. Para poder enviar y recibir mensajes a través del puerto serie es necesario asignarle los permisos correspondientes de lectura y escritura mediante el comando *chmod* en Ubuntu cada vez que se conecte la cámara, o hacer al usuario miembro del grupo *dialout* para tener esos permisos activos siempre. Las funciones de la API reciben como argumento el descriptor de fichero del puerto serie abierto, por lo que antes de llamar a cualquiera de ellas habrá que llamar *setSerial(<path_puerto>, <baudios>),* que abre el puerto serie en modo de lectura y escritura. Se debe definir el nombre del puerto serie al que se conecta la cámara en la variable de entorno CAMERA_PORT. A continuación se muestra el puerto correspondiente a la máquina de desarrollo:

```
export CAMERA_PORT=/dev/ttyUSB0
```

El código hace uso de tres librerías dinámicas:

- LibPNG: para la creación de imágenes en formato PNG
- LibGSL y LibGSLcblas: GNU Scientific Library. Para el algoritmo de mínimos cuadrados

Para instalarlas en Ubuntu se usa la herramienta de gestión de paquetes apt-get:

```
sudo apt-get install libpng-dev
sudo apt-get install libgsl0-dev
```

4.6.2 Configuración de la aplicación

En este apartado se van a detallar los pasos a seguir para configurar la aplicación en la máquina cliente, así como calibrar la cámara para capturar imágenes y temperaturas correctamente. Se va a mostrar un programa de ejemplo que hace uso de la API. El código completo del ejemplo se encuentra en el fichero main.c.

Para la calibración del brillo y contraste para imágenes con información de temperatura se debe utilizar el siguiente fragmento de código:

```
(…)
int brightness, contrast, res;
contrast = 20;//adjust depending on scene
res=set_AGC_params(fd, buf, &rec_MSG,
                   contrast, AUTO_BRIGHT);
```

De este modo el brillo se ajusta automáticamente según la escena en ese instante (AUTO_BRIGHT). No sucede lo mismo con el contraste, debido a que el ajuste automático según el ancho del histograma está bloqueado en esta cámara. El usuario

tendrá que probar distintos valores hasta encontrar el adecuado para su escena, tarea que se hace más sencilla si se dispone de una capturadora de vídeo que permita visualizar el resultado inmediatamente.

El brillo se debe ajustar siempre que las condiciones de iluminación varíen (por ejemplo en exteriores a lo largo del día). El contraste deberá ajustarse manualmente según la escena.

La compilación se realiza de la siguiente manera:

```
$ export LD_LIBRARY_PATH=/usr/lib:$LD_LIBRARY_PATH
$ gcc -o API_camera main.c functions.c png.c –lpng –lgsl -lgslcblas
```

También se puede utilizar el fichero Makefile incluido, una vez se haya exportado el path de las librerías.

En la especificación de requisitos de la API se ha descrito la funcionalidad de este proyecto. La API incluye la siguiente funcionalidad:

- Conexión por puerto serie a una determinada velocidad en baudios
- Captura de imágenes
- Guardado de una imagen previamente capturada
- Consulta del número de imágenes capturadas
- Consulta del tipo de imagen (bmp-8 o 14 bits)
- Borrado de todas las imágenes
- Restaurado de la cámara a los ajustes de fábrica
- Guardar ajustes modificados
- Modificar brillo y contraste de la imagen
- Modificar el tipo de ajuste de ganancia
- Modificar la unidad de medida de temperatura

4.6.3 Funciones auxiliares

A continuación, se enumeran las funciones auxiliares y se describe su utilidad:

- **genCRC() y checkCRC()**. La primera genera el código de comprobación de errores (CRC) necesario al enviar los mensajes a la cámara y el segundo comprueba la integridad de los mensajes recibidos. El algoritmo utilizado se encuentra disponible en [1], página 15.

- **serialize().** Serializa un mensaje, concatenando los campos y los argumentos de éste uno tras otro para enviarlos por el puerto serie.
- **Byte2Int()**. Convierte uno o varios datos de tipo Byte (uint8_t) a entero, teniendo en cuenta el orden de los bytes (big/little endian).
- **get_timestamp().** Obtiene el número de milisegundos transcurridos desde la fecha 1/1/1970 0:00h GMT, lo que se conoce como UNIX timestamp. Esto se utiliza para el nombrado de las imágenes descargadas de la cámara, con el fin de no sobreescribirlas, ordenarlas y tener la fecha de creación.
- **strImageValues():** genera un string que contiene el nombre de los coeficientes y sus valores correspondientes para escribirlos como metadatos en la imagen png.

4.6.4 Funciones principales

SETSERIAL
Abre un puerto serie a una determinada velocidad en baudios. Abre el puerto serie de la cámara y establece la velocidad de transmisión que se le pasa como parámetro, establece los parámetros de la transmisión (paridad, bits por carácter, flags, etc.) que precisa la cámara para una correcta comunicación (ver [1], página 9).

Sintaxis:

```
int setSerial(const char *port_name, int baud)
```

Argumentos:

- `port_name`: path del puerto serie al que está conectada la cámara
- baud: velocidad de señalización en baudios

Retorno:

La función devuelve 0 si terminó bien o un número menor que 0 si se produjo algún error.

Nota: La cámara térmica tiene activado por defecto la detección automática de la velocidad de transmisión en cada ciclo de encendido, siendo 57600 y 921600 baudios los valores detectables. Ver [1], página 11.

SEND_MSG
Envía un mensaje de comando por el puerto serie a la cámara.

Sintaxis:

```
int send_MSG(int fd, int command, char* args, int nArgs)
```

Argumentos:

- fd: descriptor del puerto serie
- command: comando a enviar (ver [1])
- args: argumentos asociados al comando anterior
- nArgs: número de argumentos

Retorno:

La función devuelve 0 si terminó bien o un número menor que 0 si se produjo algún error.

RECEIVE_MSG
Recibe un mensaje de comando por el puerto.

Sintaxis:

```
int receive_MSG(int fd, char* buf, int buf_max, msg* rec_MSG)
```

Argumentos:

- fd: descriptor del puerto serie
- buf: buffer en el que se almacenan los bytes recibidos por el puerto serie
- buf_max: número máximo de bytes que se espera recibir
- rec_MSG: estructura del mensaje recibido

Retorno:

La función devuelve 0 si terminó bien o un número menor que 0 si se produjo algún error.

GET_ADDR_SIZE
Obtiene direcciones de memoria y tamaños de imagen.

Sintaxis:

```
int get_addr_size(int fd, char* buf, msg* received, int
snapNumber, int* addrOut, int* sizeOut)
```

Argumentos:

- fd: descriptor del puerto serie
- buf: buffer en el que se almacenan los bytes recibidos por el puerto serie
- received: estructura del mensaje recibido
- snapNumber: número de la imagen de la que se desea obtener la información. Si el valor es -1 se devolverá la dirección base de la memoria de imágenes y su

tamaño. Si el valor es -2 se devolverá el número de bytes ocupados en la zona de memoria de imágenes y el número de imágenes almacenadas
- addrOut: argumento de salida en el que se almacenará la dirección solicitada, o el número de bytes utilizados si snapNumber es -2.
- sizeOut: argumento de salida en el que se almacenará el tamaño solicitado, o el número de imágenes almacenadas si snapNumber es -2.

Retorno:

La función devuelve 0 si terminó bien o un número menor que 0 si se produjo algún error.

GET_BLOCK_SIZE
Obtiene la dirección de la memoria no volátil y el tamaño de bloque

Sintaxis:

```
int get_block_size(int fd, char* buf, msg* received, int* addrOut,
int* sizeOut)
```

Argumentos:

- fd: descriptor del puerto serie
- buf: buffer en el que se almacenan los bytes recibidos por el puerto serie
- received: estructura del mensaje recibido
- addrOut: argumento de salida en el que se almacenará la dirección base
- sizeOut: argumento de salida en el que se almacenará el tamaño de bloque

Retorno:

La función devuelve 0 si terminó bien o un número menor que 0 si se produjo algún error.

CHECK_NONBLOCKING
Comprueba el progreso de una operación no bloqueante

Sintaxis:

```
int check_nonBlocking(int fd, char* buf, msg* received)
```

Argumentos:

- fd: descriptor del puerto serie
- buf: buffer en el que se almacenan los bytes recibidos por el puerto serie
- received: estructura del mensaje recibido

Retorno:

La función devuelve 0 si la operación no bloqueante terminó correctamente, 1 si se produjo un error al borrar un bloque, 2 si se produjo un error al escribir un bloque, 3 si no ha terminado la operación o un número menor que 0 si se produjo algún error.

ERASE_BLOCK
Borra un bloque de memoria

Sintaxis:

```
int erase_block(int fd, char* buf, msg* received, int block)
```

Argumentos:

- fd: descriptor del puerto serie
- buf: buffer en el que se almacenan los bytes recibidos por el puerto serie
- received: estructura del mensaje recibido
- block: número del bloque a borrar. Nota: El número de bloque al que pertenece una dirección de memoria se calcula restando la dirección base obtenida con get_block_size() a la dirección deseada y después dividiéndola por el tamaño de bloque de get_block_size().

Retorno:

La función devuelve 0 si terminó bien o un número menor que 0 si se produjo algún error. El borrado del bloque es no bloqueante, por lo que para comprobar su progreso se debe utilizar la función check_nonBlocking():

SHOW_SPOT_METER_GAUGE
Permite mostrar u ocultar el termómetro en el vídeo analógico

Sintaxis:

```
int show_spot_meter_gauge(int fd, char* buf, msg* received, int
mode)
```

Argumentos:

- fd: descriptor del puerto serie
- buf: buffer en el que se almacenan los bytes recibidos por el puerto serie
- received: estructura del mensaje recibido
- mode:
    - 0: no mostrar el termómetro

- 1: mostrar sólo la cifra de temperatura
- 2: mostrar sólo el termómetro
- 3: mostrar la cifra de temperatura y el termómetro

Retorno:

La función devuelve 0 si terminó bien o un número menor que 0 si se produjo algún error.

GET_TEMP_FROM_FLUX*

*Esta función no es compatible con la cámara utilizada ya que esta no cuenta con radiometría avanzada.

Obtiene la temperatura correspondiente a una cifra de flujo radiométrico.

Sintaxis:

```
int get_temp_from_flux(int fd, char* buf, msg* received, int* flux, int* temp)
```

Argumentos:

- fd: descriptor del puerto serie
- buf: buffer en el que se almacenan los bytes recibidos por el puerto serie
- received: estructura del mensaje recibido
- flux: dato de flujo radiométrico que se desea convertir
- temp: temperatura resultado en grados centígrados

Retorno:

La función devuelve 0 si terminó bien o un número menor que 0 si se produjo algún error.

GET_TEMP_FLUX

Obtiene el valor del flujo radiométrico del spot meter y a continuación el dato de temperatura del spot meter en grados centígrados. Se apoya en la función get_spot_meter_value().

Sintaxis:

```
int get_temp_flux(int fd, char* buf, msg* received, double* celsius, double* counts)
```

Argumentos:

- fd: descriptor del puerto serie

- buf: buffer en el que se almacenan los bytes recibidos por el puerto serie
- received: estructura del mensaje recibido
- celsius: dato de temperatura del spot meter en grados centígrados. Siempre es un número entero
- counts: dato de flujo radiométrico del spot meter

Retorno:

La función devuelve 0 si terminó bien o un número menor que 0 si se produjo algún error.

GET_SPOT_METER_MODE

Obtiene el modo del medidor de temperatura

Sintaxis:

```
int get_spot_meter_mode(int fd, char* buf, msg* received)
```

Argumentos:

- fd: descriptor del puerto serie
- buf: buffer en el que se almacenan los bytes recibidos por el puerto serie
- received: estructura del mensaje recibido

Retorno:

La función devuelve 0 si el medidor no tiene unidad de medida (0-2^{14}-1), 1 si está en Fahrenheit, 2 si está en Celsius o un número menor que 0 en caso de error.

SET_SPOT_METER_MODE
Establece el modo del medidor de temperatura

Sintaxis:

```
int set_spot_meter_mode(int fd, char* buf, msg* received, int
scale)
```

Argumentos:

- fd: descriptor del puerto serie

- buf: buffer en el que se almacenan los bytes recibidos por el puerto serie
- received: estructura del mensaje recibido
- scale: 0 establece medida sin unidades, 1 establece Fahrenheit y 2 establece Celsius

Retorno:

La función devuelve 0 si terminó bien o un número menor que 0 en caso de error.

GET_SPOT_METER_VALUE
Obtiene el valor de temperature en la unidad establecida por set_spot_meter_mode()

Sintaxis:

```
int get_spot_meter_value(int fd, char* buf, msg* received)
```

Argumentos:

- fd: descriptor del puerto serie
- buf: buffer en el que se almacenan los bytes recibidos por el puerto serie
- received: estructura del mensaje recibido

Retorno:

La función devuelve la temperatura del centro de imagen o el valor ($0\text{-}2^{14}\text{-}1$) de la media de los 4 píxeles centrales.

TAKE_SNAPSHOT
Capturar una imagen

Sintaxis:

```
int take_snapshot(int fd, char* buf, msg* received, int nBit)
```

Argumentos:

- fd: descriptor del puerto serie
- buf: puntero al array de caracteres donde se almacenará el mensaje
- received: puntero a la estructura msg donde se almacena el mensaje recibido

- nBit: especifica si se quiere tomar una imagen en 8 bit (temperatura en escala de grises o color según configuración) o 14 bit (temperatura en escala de grises y resolución 14 bit)

Retorno:

La función devuelve 0 si terminó bien o un número menor que 0 si se produjo algún error.

GET_SNAPSHOT
Guardado de una imagen previamente capturada

Sintaxis:

```
int get_snapshot(int fd,char* buf,msg* received,int snapNumber,
uint8_t *imBuf)
```

Argumentos:

- fd: descriptor del puerto serie
- buf: puntero al array de caracteres donde se almacenará el mensaje
- received: puntero a la estructura msg donde se almacena el mensaje recibido
- snapNumber: número de la imagen que queremos guardar, empezando por 0
- imBuf: buffer en el que se almacenarán los datos de imagen recibidos

Retorno:

La función devuelve 0 si terminó bien o un número menor que 0 si se produjo algún error.

GET_NUMBER_SNAPSHOTS
Consulta del número de imágenes almacenadas

Sintaxis:

```
int get_number_snapshots(int fd, char* buf, msg* received)
```

Argumentos:

- fd: descriptor del puerto serie
- buf: puntero al array de caracteres donde se almacenará el mensaje
- received: puntero a la estructura msg donde se almacena el mensaje recibido

Retorno:

La función devuelve el número de imágenes almacenadas en la memoria flash de la cámara o un número menor que 0 en caso de error.

GET_SNAP_HEADER
Consulta del tipo de imagen (bmp-8 o 14 bits)

Sintaxis:

```
int get_snap_header(int fd, char* buf, msg* received, int snapNumber)
```

Argumentos:

- fd: descriptor del puerto serie
- buf: puntero al array de caracteres donde se almacenará el mensaje
- received: puntero a la estructura msg donde se almacena el mensaje recibido
- snapNumber: número de la imagen que queremos consultar, empezando por 0.

Retorno:

La función devuelve 8 si la imagen es un bitmap de 8 bits, 14 si es de 14 bits o un número menor que 0 en caso de error.

ERASE_ALL
Borrado de todas las imágenes

Sintaxis:

```
int erase_all(int fd, char* buf, msg* received)
```

Argumentos:

- fd: descriptor del puerto serie
- buf: puntero al array de caracteres donde se almacenará el mensaje
- received: puntero a la estructura msg donde se almacena el mensaje recibido

Retorno:

La función devuelve 0 si las imágenes se eliminaron correctamente o un número menor que 0 en caso de error.

FACTORY_DEFAULTS
Restaurar los ajustes de fábrica de la cámara

Sintaxis:

```
int factory_defaults(int fd, char* buf, msg* received)
```

Argumentos:

- fd: descriptor del puerto serie
- buf: puntero al array de caracteres donde se almacenará el mensaje
- received: puntero a la estructura msg donde se almacena el mensaje recibido

Retorno:

La función devuelve 0 si los ajustes se restauraron correctamente o un número menor que 0 en caso de error.

GETBMPPROPERTIES

Obtiene el tamaño de una imagen en formato bmp y la dirección de inicio después de la cabecera

Sintaxis:

```
getBMPproperties(uint8_t* imBuf, char* filename, int* start, int*
widthOut, int* heightOut)
```

Argumentos:

- imBuf: buffer en el que se ha almacenado la imagen de la cámara con get_snapshot. Si se desea convertir un BMP a PNG, este argumento debe ser NULL.
- titlename: fichero de imagen BMP. Sólo se utiliza si se desea convertir a PNG
- widthOut: argumento de salida en el que se almacenará el ancho
- heightOut: argumento de salida en el que se almacenará el alto de la imagen
- start: argumento de salida en el que se almacenará la posición en la que empiezan los valores de los píxeles de la imagen.

Retorno:

La función devuelve 0 si terminó correctamente o un número distinto de 0 en caso de error.

WRITEIMAGE8

Lee los datos de una imagen almacenada en un buffer y genera un fichero de imagen png.

Sintaxis:

```
int writeImage8(char* directory, int width, int height, uint8_t
*buffer, char* title, imData* data, int offset, int snapNumber)
```

Argumentos:

- directory: directorio en el que se desea crear la imagen. Ejemplo: /tmp/images
- width: ancho de la imagen del buffer
- height: alto de la imagen del buffer
- buffer: buffer en el que están almacenados los datos de la imagen recibida desde la cámara
- title: texto a incluir en la imagen como metadatos
- data: estructura de datos que almacena los valores medio, máximo y mínimo de la imagen
- offset: posición en la que empiezan los valores de los píxeles de la imagen. Nota: si la imagen es bmp el offset se obtiene con la función getBMPproperties().
- snapNumber: número de imagen de la cámara que se va a escribir. Sirve para obtener los datos de coeficientes de esa imagen concreta.

Retorno:

La función devuelve 0 si terminó correctamente o un número distinto de 0 en caso de error.

BIT8TOTEMP

Calcula la temperatura en grados centígrados a partir del valor de un píxel, tomando como coeficientes de la transformación las variables globales _m, _b, _mTemp y _bTemp.

Sintaxis:

```
int bit8toTemp(int val8, double* ret)
```

Argumentos:

- val8: valor del píxel. Se supone que se trabaja con imágenes en escala de grises de 8 bits por píxel.
- ret: puntero a double donde se almacenará la temperatura en grados centígrados.

Retorno:

La función devuelve 0 si terminó correctamente o un número distinto de 0 en caso de error.

DO_CALIBRATION
Realiza la calibración para la conversión flujo radiométrico-temperatura. Se toman pares de medidas de flujo-temperatura, para después utilizar el método de mínimos cuadrados. Se deben tomar como mínimo 2 medidas para poder calcular los coeficientes de la función de transformación lineal.

Sintaxis:

```
int do_calibration(int fd, char* buf, msg* received)
```

Argumentos:

- fd: descriptor del puerto serie
- buf: buffer en el que se almacenan los bytes recibidos por el puerto serie
- received: estructura del mensaje recibido

Retorno:

La función devuelve 0 si terminó correctamente o un número distinto de 0 en caso de error.

LEAST_SQUARES
Utilizael método de mínimos cuadrados para encontrar los coeficientes de una función lineal que mejor se ajustan los datos.

Sintaxis:

```
int least_squares(double* arrFlux, double* arrCelsius, int nVals,
double* mRes, double* bRes)
```

Argumentos:

- arrFlux: referencia al array de flujo
- arrCelsius: referencia al array de temperaturas
- nVals: número de valores de los arrays
- mRes: referencia a la variable que guardará la pendiente de la función lineal
- bRes: referencia a la variable que guardará la ordenada en el origen de la función lineal

Retorno:

La función devuelve 0 si terminó correctamente o un número distinto de 0 en caso de error.

GET_ARRAY_AVG
Devuelve el valor medio de flujo radiométrico (14 bit) de la escena actual de la cámara. Se utiliza para ajustar el brillo a ese valor.

Sintaxis:

```
int get_array_avg(int fd, char* buf, msg* received)
```

Argumentos:

- fd: descriptor del puerto serie
- buf: buffer en el que se almacenan los bytes recibidos por el puerto serie
- received: estructura del mensaje recibido

Retorno:

La función devuelve el valor medio de la escena o un número menor de 0 en caso de error.

BIT8TO14
Convierte el valor de 8 bits de la imagen a flujo radiométrico de 14 bits, teniendo en cuenta los coeficientes (las variables globales _m y _b).

Sintaxis:

```
int bit8to14(int val8)
```

Argumentos:

- val8: valor del pixel en escala de grises de 8 bits

Retorno:

La función devuelve el valor de flujo radiométrico correspondiente.

BIT8TO14
Convierte el valor de flujo radiométrico de 14 bits a valor de píxel de 8 bits, teniendo en cuenta los coeficientes (las variables globales _m y _b).

Sintaxis:

```
int bit14to8(int val14)
```

Argumentos:

- val14: valor del flujo radiométrico

Retorno:

La función devuelve el valor de píxel de 8 bits.

GET_AGC_PARAMS
Activa el modo manual del control de ganancia (AGC) y obtiene los valores de brillo y contraste actuales.

Sintaxis:

```
get_AGC_params(int fd, char* buf, msg* received, int* contrast,
int* brightness)
```

Argumentos:

- fd: descriptor del puerto serie
- buf: buffer en el que se almacenan los bytes recibidos por el puerto serie
- received: estructura del mensaje recibido
- contrast: contraste actual de la cámara
- brightness: brillo actual de la cámara

Retorno:

La función devuelve 0 si terminó correctamente o un número menor de 0 en caso de error.

SET_AGC_PARAMS
Activa el modo manual del control de ganancia (AGC) y modifica los valores de brillo y contraste actuales.

Sintaxis:

```
int set_AGC_params(int fd, char* buf, msg* received, int contrast,
int brightness)
```

Argumentos:

- fd: descriptor del puerto serie
- buf: buffer en el que se almacenan los bytes recibidos por el puerto serie
- received: estructura del mensaje recibido
- contrast: contraste que se desea utilizar en la cámara
- brightness: brillo que se desea utilizar en la cámara

Retorno:

La función devuelve 0 si terminó correctamente o un número menor de 0 en caso de error.

5. RESULTADOS

La medida de la temperatura de este proyecto se basa en la única característica de la cámara que ofrece valores de temperatura, el spot meter. Se trata de una característica que muestra la temperatura media o el flujo radiométrico de la zona central de la imagen.

Sobre la fiabilidad de las medidas de temperatura, en el documento [5] apartado 3.3.3.2 se afirma que la precisión del spot meter es de ±20ºC en estado de baja ganancia o el mayor de ±20ºC o 20% en estado de alta ganancia, y que la precisión típica es de ±10ºC. Se trata de una precisión muy pobre para la aplicación que va a tener la cámara. Esta precisión puede verse reducida aún más por los errores acumulados al hacer las transformaciones lineales descritas en el apartado 4.5.

La siguiente ilustración muestra ejemplos de medidas del spot meter que no se corresponden con la temperatura real. En la primera de ellas se puede apreciar la mano de una persona, cuya temperatura corporal normal varía entre los 36-38 grados centígrados, mientras que el valor medido por la cámara es de 29ºC. La segunda imagen muestra una bandeja de cubitos de hielo recién sacada del congelador. Según el termómetro de éste, la temperatura interior está en torno a los -18ºC, bastante alejados de los -47ºC que mide la cámara.

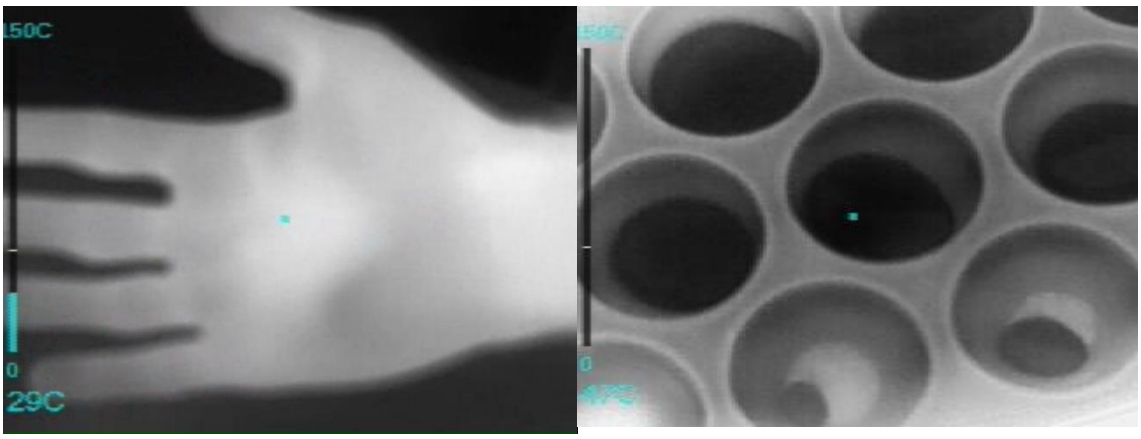


Ilustración 3. Medidas de temperatura con spot meter

En cuanto a la calidad de las imágenes obtenidas, ésta depende de cómo se configure la cámara y el uso que se le pretenda dar. Existen dos opciones de uso: utilizar únicamente las imágenes para tener información de las diferencias de temperatura por zonas sin conocer la temperatura de éstas o utilizar la cámara para obtener imágenes preparadas para calcular la temperatura en cualquiera de sus píxeles.

Si sólo se desea obtener la imagen de la cámara para observar visualmente las diferencias de temperatura por zonas lo recomendable es configurar la cámara con el control

automático de ganancia (AGC), que ajustará el brillo y el contraste automáticamente para cada escena. Para este modo de uso, es recomendable utilizar la función `factory_defaults()`, que activa el AGC y otros filtros que mejoran la apariencia visual de la imagen. A continuación, se muestran ejemplos de imágenes capturadas con estos ajustes.

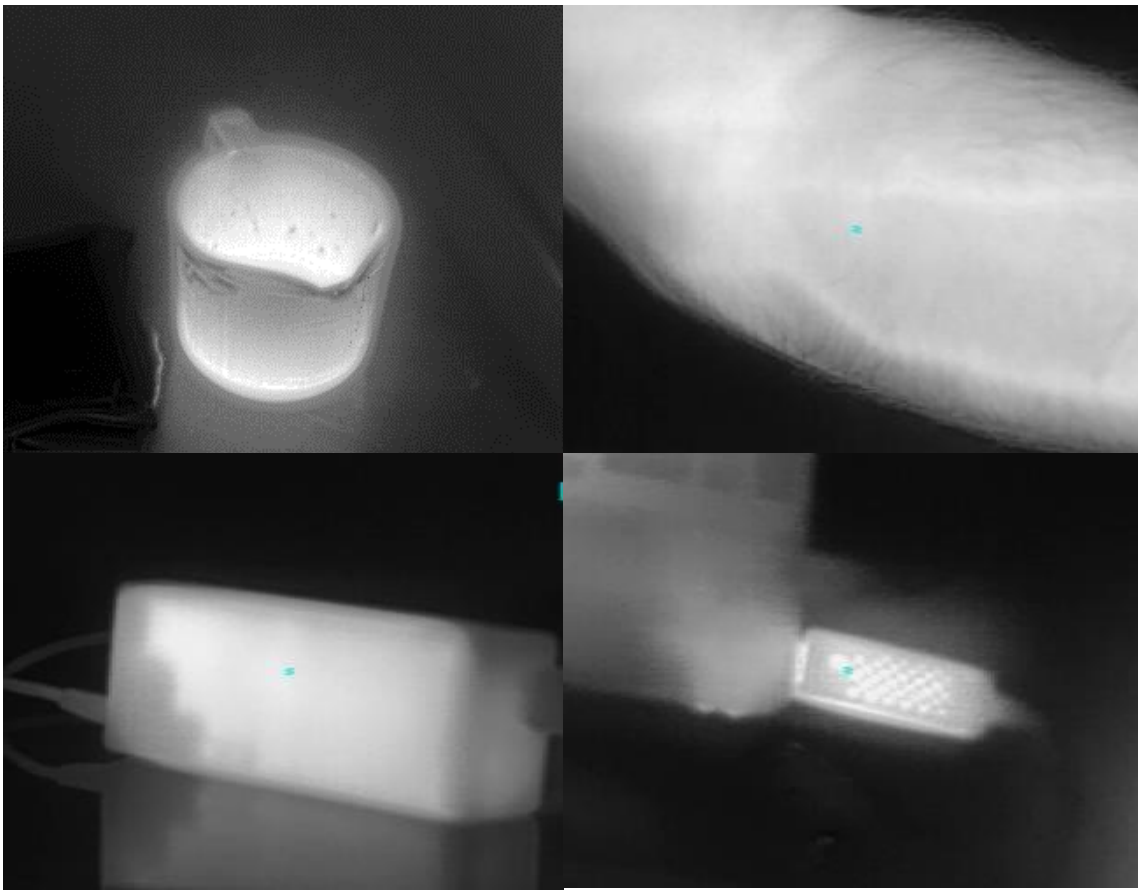


Ilustración 4. Imágenes capturadas con AGC

En las imágenes de la ilustración superior, la primera y la última se capturaron con un enfoque inadecuado, ya que este es manual. En la primera imagen aparece una jarra con agua hirviendo y a su lado izquierdo un bloque de hielo. El contraste se ha ajustado automáticamente para esta escena con temperaturas tan dispares. En la segunda imagen se aprecian las venas del brazo de una persona, lo que demuestra que con el ajuste adecuado se pueden apreciar pequeñas diferencias de temperatura. En la tercera imagen se aprecia la localización de los componentes que generan más calor en el interior de un transformador. En la cuarta imagen se muestra la capturadora de vídeo utilizada para el proyecto.

En el caso de que se necesite saber la temperatura de las zonas de la imagen, se deberá hacer un ajuste o calibración in-situ para fijar los ajustes de brillo y contraste de la imagen capturada ya que, como hemos mencionado antes, necesitamos estos parámetros para poder estimar la función de transformación lineal de flujo radiométrico a valores de píxel (véase apartado 4.6.2). Como ventaja de este modo de uso se encuentra la capacidad de tener una temperatura, más o menos precisa, de cada píxel de la imagen. Los inconvenientes son que requiere una calibración previa del brillo y el contraste, y que las escenas a capturar deben tener estas variables similares a las iniciales. En caso contrario se deberá hacer una nueva calibración para cada escena en la que varíen sustancialmente alguna de ellas, o la imagen aparecerá oscura o sobreexpuesta.

Un ejemplo de una imagen con información de temperatura es la siguiente:

```
AVERAGE: 160.127747
MIN:  0  at  pixel
(0,256)
MAX:  255  at  pixel
(16,245)
-------- VALUES ---
-----
m=0.3125
b=-974.56
mT=0.020220
bT=-44.361886
```

Ilustración 5. Imagen con coeficientes de temperatura

Los datos de la derecha de la imagen provienen de un fichero de texto con el mismo nombre que la imagen descargada. Estos datos incluyen el máximo, mínimo y el valor medio de los píxeles de la imagen, así como su posición. En la parte inferior están los coeficientes de las dos transformaciones lineales para pasar de datos de píxel de 8 a 14 bits y después de 14 bits a temperatura.

$$val14 = (val8 - b)/m$$

$$temp = val14 * mT + bT$$

La fórmula para calcular la temperatura en grados centígrados que le corresponde a un píxel determinado es la siguiente:

$$temp = ((val8 - b)/m) * mT + bT$$

Tomamos dos puntos de la ilustración 4, uno en el fondo y otro en la boca de la persona para obtener sus temperaturas.

El píxel del fondo tiene un valor de 17, y al aplicar la fórmula con los coeficientes de la imagen obtenemos una temperatura de 19,79 grados centígrados, lo cual es una temperatura razonable para el interior de una habitación.

El píxel de la boca tiene un valor de 255, que al aplicar la fórmula da un valor de 35,19 grados centígrados. Es un valor muy aproximado a la temperatura corporal típica.

6. CONCLUSIONES

A lo largo de este proyecto han surgido problemas que han provocado que la entrega se alargue hasta la convocatoria extraordinaria, principalmente el fallo de la cámara térmica y el largo periodo de reparación.

La información que proporciona el fabricante, exceptuando las especificaciones técnicas, contiene imprecisiones, explicaciones vagas, y errores en la descripción y funcionamiento de los mandatos. Esto, ha provocado que la planificación inicial no se haya podido seguir y se haya extendido en el tiempo la fase de desarrollo de la API. El método de obtención de la temperatura también ha sido distinto respecto al inicial debido a que las características de radiometría avanzada no estaban disponibles.

Creo que el proyecto de investigación de la planta Arabidopsis para la que está diseñada esta API, en el caso de que necesite valores precisos de temperatura, debería contar con un instrumento de medida más fiable. En caso de que sólo se necesite observar visualmente las diferencias de temperatura en la superficie de las hojas, la cámara cumplirá su función correctamente, si se ha configurado de la manera adecuada.

No obstante, he conseguido desarrollar una API que controla los ajustes básicos y de imagen de la cámara, captura imágenes, las elimina, las descarga en la máquina a la que está conectada y permite obtener la temperatura de cada píxel una vez descargada la imagen y su fichero de datos.

Con esto se consideran cumplidos los objetivos iniciales del proyecto. Cierto es que el proceso de obtención de la temperatura no es el más intuitivo o fiable, pero cuando se trabaja con limitaciones de hardware o físicas hay que aceptarlas e intentar obtener el mejor resultado posible.

La puesta en funcionamiento de una aplicación basada en esta API es sencilla, siguiendo los pasos descritos en los apartados 4.6.1 y 4.6.2. También se puede observar en el fichero main.c un ejemplo de un programa que utiliza esta API.

7. BIBLIOGRAFÍA

[1] Flir. (2015, Junio). Flir Tau2/Quark2 Software IDD (ver. 133) [Online]. Disponible: http://cvs.flir.com/tau2-quark-software-idd?_ga=1.31836413.2144006524.1446573597

[2] Flir. (2013, Mayo). Flir Image Capture Procedure Application Note (ver. 110) [Online]. Disponible: http://cvs.flir.com/image-capture-note?_ga=1.187632363.2144006524.1446573597

[3] Wikipedia. (2008, Febrero). Computation of cyclic redundancy checks [Online]. Disponible: https://en.wikipedia.org/wiki/Computation_of_cyclic_redundancy_checks

[4] Lammert Bies. (2015, Abril). On-line CRC calculation and free library [Online]. Disponible: http://www.lammertbies.nl/comm/info/crc-calculation.html

[5] Flir. (2015, Junio). Flir Tau2 Product Specification Note (ver. 141) [Online]. Disponible: http://cvs.flir.com/tau2-product-spec?_ga=1.149441398.1323307183.1447252341

[6] Flir. (2015, Junio). Flir Advanced Radiometry Application Note (ver. 120) [Online]. Disponible: http://cvs.flir.com/advanced-radiometry-note?_ga=1.79791063.1323307183.1447252341

[7] Roselyne Ishimwe, K. Abutaleb, F. Ahmed. (2014, Septiembre). Applications of Thermal Imaging in Agriculture - A Review [Online]. Disponible: http://dx.doi.org/10.4236/ars.2014.33011

[8] Autor desconocido. (consultado sep.-dic. 2015). Linux man pages [Online]. Disponible: http://linux.die.net/man/

[9] Greg Roelofs. (2015). Official PNG reference library [Online]. Disponible: http://www.libpng.org/pub/png/libpng.html

[10] A.Greensted. (2015). Creating PNGs with libPNG [Online]. Disponible: http://www.labbookpages.co.uk/software/imgProc/libPNG.html

[11] Wikipedia (2015, octubre). Windows bitmap [Online]. Disponible: https://es.wikipedia.org/wiki/Windows_bitmap

[12] H. Hackl *et al*. (2012, abril). A Comparison of Plant Temperatures as Measured by Thermal Imaging and Infrared Thermometry [Online]. Disponible: http://onlinelibrary.wiley.com/doi/10.1111/j.1439-037X.2012.00512.x/abstract DOI: 10.1111/j.1439-037X.2012.00512.x

[13] B. Benito *et al*. (2013, enero). Proyecto AGL2012-36174: Entrada de Na+ en la raíz y tolerancia al NaCl en las plantas: aplicación para su estudio de un análisis funcional y genético (resumen) [Online]. Disponible: http://www.upm.es/observatorio/vi/index.jsp?pageac=actividad.jsp&id_actividad=192836

8. AGRADECIMIENTOS

Quiero agradecer a mi tutor, José Luis Pedraza, su dedicación y su ayuda durante la realización de este trabajo.