UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

SISTEMA DE MONITORIZACIÓN INALÁMBRICA DE TEMPERATURA MEDIANTE SENSOR DE INFRARROJOS Y MICROCONTROLADOR ESP32

TRABAJO FINAL DEL

Máster Universitario en Sensores para Aplicaciones Industriales

REALIZADO POR

Jorge Peris Martinez

TUTORIZADO POR

Rafael Masot Peris

CURSO ACADÉMICO: 2019/2020

TÍTULO

Sistema de monitorización inalámbrica de temperatura mediante sensor de infrarrojos y microcontrolador ESP32.

RESUMEN

Desde la implantación de las comunicaciones inalámbricas en los distintos sectores de la industria, el cableado físico ha pasado a un segundo plano. Esto se debe a la mayor simpleza de los sistemas inalámbricos respecto a los cableados, así como a la reducción económica que supone prescindir de una red física de comunicaciones. Así pues, el presente proyecto desarrolla un sistema de comunicaciones inalámbrico, haciéndose valer de una de las herramientas más útiles e importantes desde la segunda mitad del siglo XX: Internet.

El sistema desarrollado engloba una etapa de sensado, una de comunicación y una última de monitorización y tratamiento de datos. De esta manera, el sistema conjunto es capaz de medir la variable física temperatura, la cual es procesada por un microcontrolador y enviada mediante una antena WiFi a una plataforma de almacenamiento en la red. Cualquier dispositivo provisto con conexión a internet puede acceder a esta plataforma y consultar los datos recibidos. Finalmente, la información de temperatura es enviada de nuevo vía WiFi desde la plataforma de almacenamiento a un *software* de programación, en el que se estudian los valores recibidos mediante una interfaz gráfica. Adicionalmente se dispone de una conexión Bluetooth, para visualizar los datos desde un *smartphone* ubicado en las proximidades del proceso.

Habiendo sido provisto de todos los elementos necesarios para ello, el proyecto abre una puerta al IoT, cada vez más presente y que, sin lugar a dudas, ocupará un lugar fundamental en la vida cotidiana de la población en los próximos años.

SUMMARY

Since the introduction of wireless communications in different sectors of the industry, physical cabling has taken a back seat. This is due to the greater simplicity of wireless systems with respect to wiring, as well as to the economic reduction involved in dispensing with a physical communications network. In this way, this project develops a wireless communications system, making use of une of the most useful and important tools since the second half of the 20th century: the Internet.

The system developed includes a sensing stage, a communication stage and a last one for monitoring and data processing. In this way, the system is capable of measuring the physical variable temperatura, which is processed by a microcontroller and sent through a WiWi antenna to a sotrage platform on the network. Any device with

an internet connection can access this platform and check the data received. Finally, the temperature information is sent again via WiFi from the storage platform to a programming software, in which the received values are studied through a graphical interface. Additionally, there is a Bluetooth connection to view the data from a Smartphone located near the process.

Having been provided with all the necessary elements for this, the project opens a door to the IoT, which is inceasingly present and which, whitout a doubt, will occupy a fundamental place in the daily life of the population in the coming years.

PALABRAS CLAVE

Sensor de infrarrojos; microcontrolador; WiFi; Bluetooth; IoT; ThingSpeak; Matlab; APP inventor;

KEYWORDS

Infrared sensor; microcontroller; WiFi; Bluetooth; IoT; ThingSpeak; Matlab; APP inventor;

Autor del TFM

Jorge Peris Martínez

Localidad y fecha

Valencia, Septiembre de 2020

Tutor Académico

Prof. D.Rafael Masot Peris

ÍNDICE DE LA MEMORIA

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

1. Introducción

Desde finales del siglo XX, el desarrollo de la tecnología de telecomunicaciones ha cambiado por completo el estilo de vida de la sociedad respecto a los siglos anteriores, suponiendo sin duda alguna una revolución a escala mundial. Entre los distintos tipos de tecnologías, hay una que sin duda destaca sobre el resto debido a su "libre" accesibilidad e infinidad de servicios que ofrece: internet.

Es por este motivo que en el primer cuarto del siglo XXI se está arraigando de manera significativa el concepto IoT (*Internet of Things*) en la sociedad, siendo cada vez mayor el número de personas que deciden apostar por esta tecnología para afrontar la vida cotidiana. El IoT es un concepto que, en grandes términos, se define como el uso de internet para la monitorización o control de distintos elementos o dispositivos con los que trata una persona en su vida diaria.

Por ejemplo, una persona puede controlar la temperatura del termostato de su casa desde su puesto de trabajo, siempre y cuando el termostato disponga de una conexión WiFi que le permita enviar los valores de temperatura a internet, con tal de ser descargados por el teléfono móvil del usuario, y que este a su vez pueda regular la temperatura a su antojo desde el mismo móvil.

Así pues, el IoT es visto como una manera de facilitar o simplificar la cotidianidad de la sociedad, suponiendo una revolución tecnológica que marcará la vida de la gente en los años futuros.

De la misma manera que puede regularse el termostato de un hogar, se puede controlar cualquier dispositivo de manera remota siempre y cuando dicho dispositivo cuente con una antena WiFi que le permita establecer una red de comunicación y vincularse con el dispositivo maestro, el cual le ordene cómo debe de comportarse. Así pues, resulta alentador incluir este tipo de tecnología en un sistema, que permita recoger cierta información de un dispositivo o proceso para ser controlado de manera remota.

La temperatura es una magnitud escalar relacionada con la energía interna de un sistema termodinámico, y resulta una variable que tiene cabida en la mayoría de procesos físicos. En prácticamente todos los procesos industriales es requerido el conocimiento de temperatura de uno o varios elementos específicos (motores, líquidos de refrigeración, etc.) que intervienen en el proceso; o directamente en el proceso en sí (calentamiento de piezas metálicas, tratamiento de aceros, etc.).

A continuación se presentan algunos procesos en los que la temperatura juega un papel crucial, siendo determinante en la calidad del mismo:

Fermentación del vino: Durante el tratamiento de la uva para su conversión en vino, se producen distintos procesos que dependen fundamentalmente de la temperatura

alcanzada por el jugo, ya que su aumento se traduce en una aceleración de los procesos enzimáticos de fermentación. No obstante, se debe tener un control riguroso

de estos aumentos térmicos ya que una subida excesiva de temperatura produce una disminución de la actividad enzimática.

Detección de fiebre: Tras los hechos acontecidos en el año 2020 por la expansión del virus COVID-19 (cuyo principal síntoma es un aumento significativo de la temperatura corporal), se han incluído cámaras termográficas en aeropuertos y lugares públicos de todo el mundo con tal de discriminar a personas enfermas de sanas.

Tratamiento de acero: Existen distintos procesos para modificar las propiedades mecánicas del acero, como pueden ser el temple o el revenido. Estos procesos se basan en la aplicación de cantidades elevadas de calor a las piezas metálicas con tal de lograr una modificación estructural de las mismas que les permita aumentar su dureza o tenacidad, características necesarias para ciertas aplicaciones. La temperatura juega de nuevo un papel crucial, puesto que se debe buscar el punto óptimo de calentamiento para no estropear la pieza.

Se hace evidente pues, que la temperatura resulta una variable física bastante interesante para ser sometida a estudio ya que interviene en una infinidad de procesos, ya sean industriales o de otra índole.

Bajo todo este contexto, resulta verdaderamente alentador el desarrollo de un sistema basado en el IoT, que permita una monitorización remota de la temperatura de undeterminado proceso para permitir tomar soluciones en consecuencia del estado de dicho proceso en cada momento.

2. Objetivos

El objetivo principal del proyecto es desarrollar un sistema que permita abordar cualquier proceso en el que sea requerido el conocimiento de la temperatura de un elemento, cuyo gradiente de temperaturas esté comprendido en el rango de valores medibles por el sensor. También es pretensión del proyecto ofrecer un sistema de comunicaciones que permita transferir la información captada por el sensor a un servidor de internet, cuyo acceso pueda restringirse o no, en función de la aplicación que se quiera dar al sistema.

Para la monitorización del proceso, se pretende desarrollar una interfaz gráfica con la que un usuario pueda visualizar la evolución del proceso, además de ofrecer establecer parámetros de calidad que permitan tener un control básico, generando alarmas según la aplicación que se le dé al sistema.

De esta manera aparece una serie de objetivos básicos que a la finalización del proyecto deben ser cumplidos:

- Obtener la variable física temperatura de manera continuada.
- Adaptar los datos recibidos relativos a la temperatura para su posterior transferencia a una plataforma de almacenamiento de datos.
- Establecer una red de comunicaciones que permita la transferencia de los valores medidos de manera inalámbrica.
- Desarrollar una interfaz gráfica que permita la visualización y el tratamiento de los valores recibidos.
- Desarrollar una APP móvil para monitorizar el proceso desde un *smartphone* controlado por un operario próximo al lugar de medición.

Por lo explicado anteriormente, se hace patente que el sistema no pretende ser incorporado en un sector o industria específicos, sino en cualquiera que requiera de la monitorización de un proceso térmico de manera remota.

3. Materiales y métodos

Para cumplir los objetivos anteriormente planteados, el sistema debe ser provisto con todos los elementos que permitan, en conjunto, obtener resultados indicadores de la eficacia y fiabilidad del mismo.

Para ello se cuenta con distintos componentes físicos y digitales que operan de manera combinada, así como protocolos de comunicación que permiten la interacción entre dichos componentes.

3.1. Diagrama de bloques

Para ofrecer una idea visual del funcionamiento en conjunto de todos los elementos anteriormente explicados, se muestra a continuación el diagrama de bloques descriptivo del sistema **(Ilustración 1)**. Además, se ofrece una rigurosa explicación de cada uno de los bloques para aclarar su funcionalidad así como la operación completa del sistema:

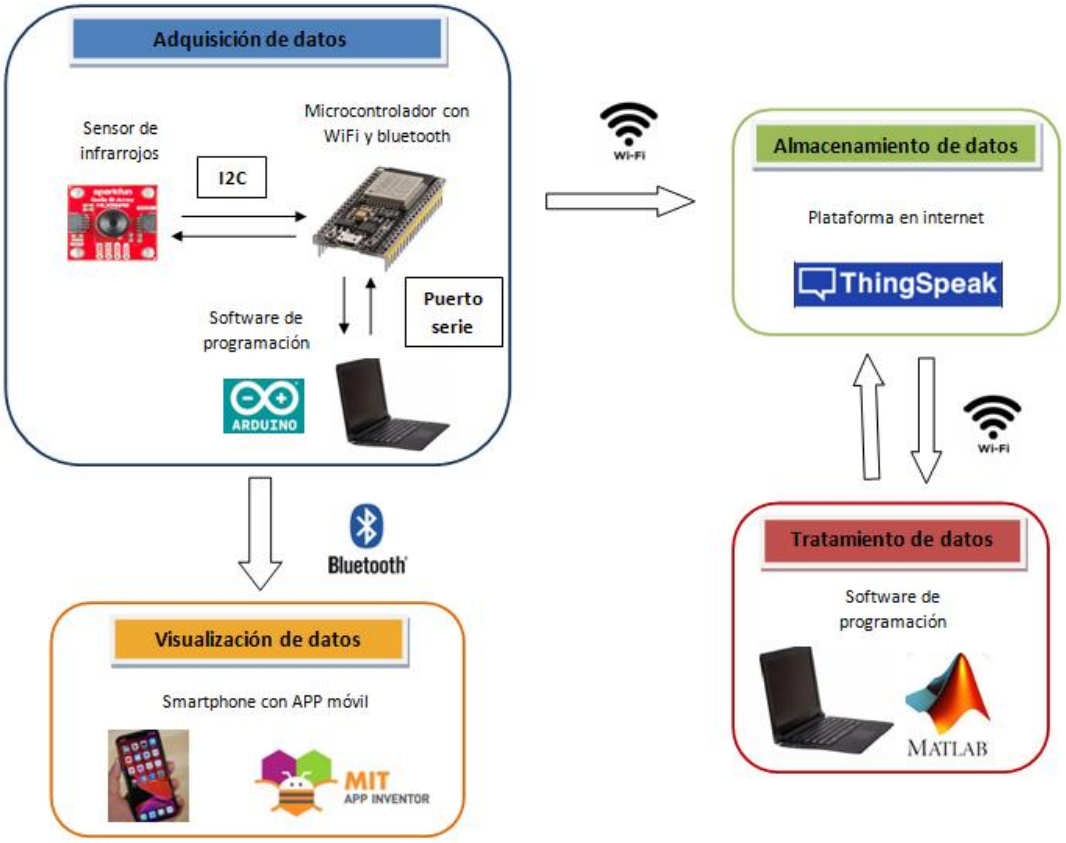


Ilustración 1: diagrama de bloques auto descriptivo del funcionamiento e interconexión de los elementos del sistema.

Bloque de adquisición de datos.

En este bloque se produce la adquisición de temperaturas, en distintos pixeles de la cámara termográfica MLX90640 mediante la tecnología de sensores de infrarrojos en la que se basa y bajo la orden del microcontrolador. Acto seguido, los valores medidos son enviados al bus I2C, de donde los adquiere el microcontrolador y son mostrados por el hyperterminal del IDE arduino para verificar la correcta adquisición de temperaturas.

Una vez se ha verificado que las temperaturas corresponden a valores posibles o que se adaptan a la situación del entorno medido; son enviadas vía WiFi a un servidor de ThingSpeak y vía Bluetooth a un dispositivo móvil cercano.

Bloque de almacenamiento de datos

Cuando los datos llegan al servidor ThingSpeak van siendo almacenados por orden de entrada, y expuestos en una gráfica que muestra la evolución del proceso térmico. El rango de tiempos que puede visualizarse es variable, pudiendo incluso visualizarse desde el primer momento que el módulo WiFi envió la primera temperatura al servidor.

De este bloque los datos pasan al bloque de tratamiento de datosa través de la red WiFi.

Bloque de tratamiento de datos

En este bloque se desarrolla el programa así como la interfaz gráfica de tratamiento y visualización de datos. Es bajo una orden emitida por este programa que los datos viajan de ThingSpeak hasta la interfaz, siendo visibles para un usuario que la esté manipulando. Una vez que los datos llegan, son expuestos en una gráfica que permite ver su evolución. Además, se aplica la lógica de comparación con los parámetros establecidos por el usuario que permiten certificar la calidad del proceso térmico que está teniendo lugar.

Bloque de visualización de datos

En este bloque se la aplicación para la monitorización del proceso mediante un dispositivo móvil. El dispositivo se empareja previamente con el módulo ESP32 a través de la aplicación, y una vez vinculados comienza a recibir la información correspondiente a las temperaturas medidas por el sensor de infrarrojos vía Bluetooth.

3.2. Hardware

El *hardware* engloba todos los elementos físicos necesarios para el montaje y desarrollo del sistema. Todos estos elementos son necesarios ya que permiten enviar y/o recibir señales físicas bajo la orden de un microcontrolador. Todos los elementos de *hardware* empleados en el proyecto disponen de circuitos impresos en su interior que permiten el intercambio de datos y señales entre ellos.

A continuación, se ofrece una descripción detallada de dichos elementos físicos y la funcionalidad que ocupan en el sistema.

3.2.1. Sensor MLX90640

El sensor MLX90640 de Sparkfun **(Ilustración 2)** es una cámara termográfica de 768 pixeles ysupone el elemento más importante en la etapa de sensado del proyecto, ya que es el dispositivo encargado de medir la temperatura en un proceso térmico, siendo su rango de medición desde -40ºC hasta 300ºC.

Ilustración 2: cámara termográfica o sensor de infrarrojos MLX90640 de SPARKFUN.

Concretamente, se trata de una matriz de 32x24 pixeles que miden la intensidad de radiación infrarroja incidente. Este diseño permite obtener una imagen térmica de 768 pixeles, en la que se puede diferenciar la temperatura medida en cada uno de ellos**(Ilustración 3).**

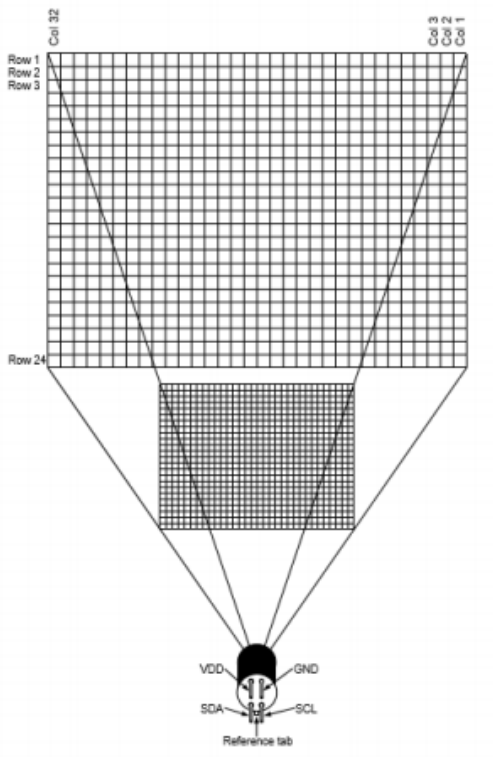


Ilustración 3: matriz de pixeles del sensor MLX90640. Dispone de 24 líneas y 32 columnas, dando un total de 768 pixeles de captación de infrarrojos.

La posición de cada pixel es identificada mediante dos subíndices, que indican la fila y columna a la que pertenece. De esta manera se puede almacenar la información relativa a la temperatura de cada pixel en una matriz de datos que posteriormente sea transferida a un *software* de programación, permitiendo así la visualización de la matriz de datos.

Para ofrecer una descripción más rigurosa de la operación del sensor, se detalla a continuaciónsu tecnología de funcionamiento, así como sus especificaciones técnicas

- **Tecnología y principio físico de funcionamiento**

El suceso que permite y rige el funcionamiento del sensor es la excitación del mismo debido a la radiación infrarroja que incide sobre él. Según la física, cualquier cuerpo de la naturaleza que esté a una temperatura superior al cero absoluto (0 grados Kelvin), emitirá radiación, en su mayor parte, infrarroja. De este hecho se deduce que el tipo de radiación asociada al calor percibido es la radiación infrarroja, cuya longitud de onda puede variar desde el infrarrojo cercano (800 nm) hasta el infrarrojo lejano (1000 μm).

Para ello, se hace valer de un elemento denominado bolómetro, y que supone la base de cualquier sensor de radiación electromagnética.

El bolómetro es un dispositivo electrónico que normalmente contiene un depósito de películas delgadas amorfas compuestas por aleaciones de silicio-germanio y dopadas con oxígeno, cuya resistencia eléctrica varía en función de la temperatura del mismo, que a su vez depende de la energía absorbida por la radiación infrarroja incidente. De esta manera puede medir la cantidad de energía calorífica que está recibiendo en forma de radiación infrarroja **(Ilustración 4).**

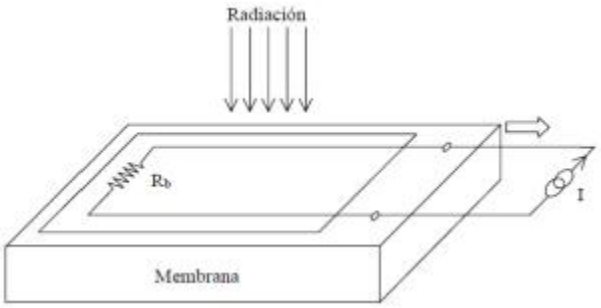


Ilustración 4: estructura típica de un bolómetro. La radiación incide en la parte superior de la membrana, donde se encuentra la resistencia termosensible, cuya variación óhmica define la intensidad de la radiación absorbida.

Las ecuaciones que rigen el comportamiento físico de un bolómetro se detallan a continuación:

Ecuación de equilibrio térmico en ausencia de radiación:

$$G.\left(T_{pol} - T_0\right) = I^2.R_b \textbf{(1)}$$

Donde:

G : Conductancia térmica entre el bolómetro y el sumidero térmico (W/K)

T_{pol} : Temperatura del bolómetro en ausencia de radiación (K).

T_0 : Temperatura ambiente (K).

$I^2.R_b$: Potencia disipada por el efecto Joule en forma de calor (W).

Cuando un objeto caliente comienza a emitir radiación sobre el bolómetro, su temperatura aumenta ligeramente, al igual que la potencia disipada por el efecto Joule (P). Sabiendo que $P = I^2.R_b$, se obtiene la siguiente ecuación:

$$C.\frac{d(T_{pol}+\Delta T)}{dt} + G.(T_{pol} + \Delta T - T_0) = P + \Delta P + \varepsilon.\delta.\varphi_e \textbf{(2)}$$

Donde:

C: Capacidad térmica del bolómetro (sin incluir el sustrato).

$\delta.\varphi_e$: Potencia incidente desde la fuente.

ε : emisividad del absorbente del bolómetro.

Teniendo en cuenta la ecuación **(1)** se puede simplificar la ecuación **(2)** obteniendo la siguiente:

$$C.\frac{d(T_{pol}+\Delta T)}{dt} + G.\Delta T = \Delta P + \varepsilon.\delta.\varphi_e \textbf{(3)}$$

La variación de potencia eléctrica disipada depende fundamentalmente del circuito de polarización del bolómetro, que consiste en la resistencia R_b conectada a una fuente de corriente constante de valor I.

Así pues, la variación de la potencia en función de la variación de temperatura se define desarrollando la siguiente ecuación:

$$\Delta P = \frac{dP}{dt}.\Delta T \rightarrow \Delta P = \frac{d(I^2.R_b)}{dT}.\Delta T \rightarrow \Delta P = I^2.R_b.\left(\frac{1}{R_b}.\frac{dR_b}{dT}\right).\Delta T$$

Para simplificar la ecuación se introduce el concepto de coeficiente de temperatura de resistencia eléctrica α, el cual se define de la siguiente manera:

$$\alpha = \frac{1}{R_b}.\frac{dR_b}{dT}$$

Obteniendo así la ecuación:

$$\Delta P = I^2 . R_b . \alpha . \Delta T \textbf{(4)}$$

Se define la conductancia térmica efectivaG_{ef}:

$$G_{ef} = G - \alpha . I^2 . R_b$$

Y se reescribe la ecuación **(3)**como**:**

$$C . \frac{d\Delta T}{dT} + G_{ef} . \Delta T = \varepsilon . \delta . \varphi_e \textbf{(5)}$$

Si se representa ahora la potencia radiada por la fuente mediante una función periódica del tipo:

$$\delta . \varphi_e = \varphi_{e0} . e^{j.w.t} \textbf{(6)}$$

Esta función permite estudiar la evolución del bolómetro en relación a la frecuencia de sus componentes. Incluyendo la función **(6)** en la ecuación **(5)** se obtiene:

$$C . \frac{d\Delta T}{dT} + G_{ef} . \Delta T = \varepsilon . \varphi_{e0} . e^{j.w.t} \textbf{(7)}$$

La resolución de la ecuación diferencial resultante describe el comportamiento dinámico del bolómetro en función de la temperatura:

$$\Delta T = \frac{\varepsilon . \varphi_{e0}}{G_{ef} . (1 + j.w.\tau)} . \left(e^{j.w.t} - e^{\frac{-t}{\tau}} \right) \textbf{(8)}$$

Donde τ es la constante de tiempo térmica, que está directamente relacionada con la respuesta en frecuencia del dispositivo y determina el tiempo que tarda en enfriarse el bolómetro.

Las cámaras termográficas de infrarrojo lejano integran un tipo de bolómetro especial llamado microbolómetro, de tamaño muy reducido y que puede ser incluido acompañado de otros microbolómetros constituyendo una matriz.

Este es el caso del MLX90640, el cual dispone de una matriz 32x24 de microbolómetros, dando un total de 768 temperaturas que conforman el gradiente que compone una imagen termográfica.

❖ Características generales

Las características generales del MLX90640 más relevantes de acuerdo a la hoja de datos son citadas a continuación:

- Tamaño reducido, matriz de 32x24 pixeles de bajo coste.
- Fácilmente integrable.
- Paquete TO39 estándar de cuatro derivaciones.
- Diferencia de temperatura diferencia en ruido (*Noise equivalent temperature difference*, NETD) 0.1K RMS @ 1 Hz frecuencia de actualización.
- Calibrado en fábrica.
- Compatibilidad con interfaz digital I2C.
- Frecuencia de actualización programable 0.5Hz…64Hz.
- Voltaje de suministro: 3.3V.
- Consumo de corriente menor de 23 mA.
- Temperatura de operación: -40°C ÷ 85°C
- Temperatura objetivo: -40°C ÷ 300°C
- 2 opciones: FOV – 55ºx35º y 110ºx75º
- Peso del paquete: 0.025 kg
- Tamaño del paquete: 10 cm x 10 cm x 2 cm

❖ Características específicas

Algunas de las características más específicas del sensor deben ser explicadas para justificar el uso que se hace de los elementos que lo componen:

- Presenta 4 pines de conexión, correspondientes a alimentación, tierra y los pines SCL y SDA de comunicación I2C **(Ilustración 5).**

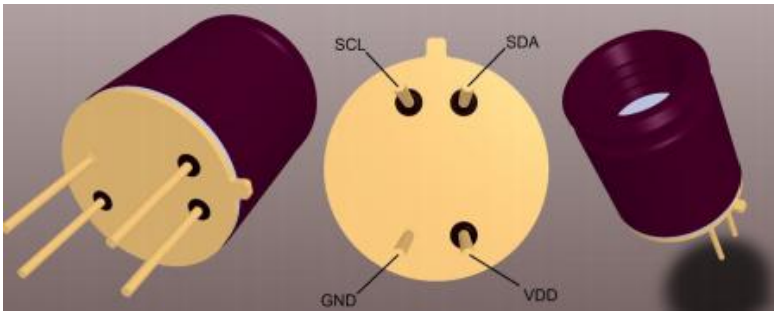


Ilustración 5: terminales o pines de la matriz de píxeles. SDA y SCL se emplean para la comunicación I2C, mientras que VDD y GND se utilizan para la alimentación a 3,3V.

- La memoria EEPROM se emplea para almacenar variables de calibración y los parámetros de configuración del dispositivo.

- Dispone de un modo de medición en el cual se ejecutan medidas de temperatura de manera constante que se actualizan en la memoria RAM con una velocidad dependiente del fps (*frames per second*) establecido en el registro de control 1 (0x800D) **(ilustración 6).**

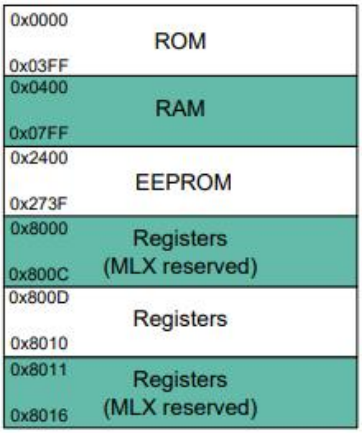


Ilustración 6: mapa de memorias y registros del MLX90640 obtenido de su hoja de datos.

- Dispone de dos modos de lectura: modo ajedrez y modo intercalado de TV. En cada caso se subdivide de una manera el conjunto de pixeles, dando un tipo de patrón de lectura u otro **(Ilustración 7).** La configuración en el bit 12 del "Registro de control 1".

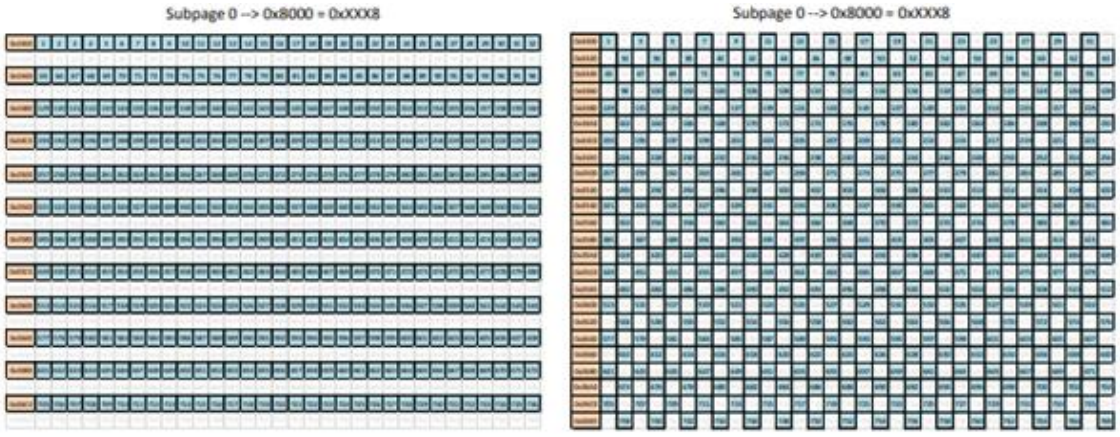


Ilustración 7: patrones de lectura del MLX90640. La configuración de la izquierda corresponde al patrón de TV intercalado, en el que se los píxeles se encuentran dispuestos por líneas no consecutivas. En la parte derecha se observa la configuración tipo ajedrez, con los píxeles ocupando posiciones no consecutivas.

- Ofrece registros internos accesibles para el usuario que permiten cierto grado de personalización del rendimiento.

Por todo lo anteriormente comentado, sale a la superficie una aplicación realmente interesante que se le puede dar a la cámara termográfica MLX90640, aparte de detectar focos de temperatura, poder ubicarlos. Esto puede hacerse con gran precisión si la transferencia de datos correspondientes a cada pixel se hace mediante un protocolo UART, como es el ejemplo mostrado en la **Ilustración 8**, obtenido de un proyecto realizado por alumnos del Máster Universitario en Sensores para Aplicaciones Industriales de la UPV.

Ilustración 8: imagen térmica desarrollada por alumnos de la UPV. En esta imagen se puede identificar la cerilla que supone un foco cálido, apareciendo señalizada en la imagen mediante los colores rojo y amarillo.

3.2.2. Módulo ESP32

El módulo ESP32 **(Ilustración 9)** constituye un dispositivo verdaderamente útil y con una elevada variedad de aplicaciones especialmente gracias a dos elementos integrados: una antena de WiFi y otra de Bluetooth. Mediante estas antenas, es posible realizar una transferencia de datos de manera inalámbrica, comunicándose con otro dispositivo provisto de cualquiera de estas antenas.

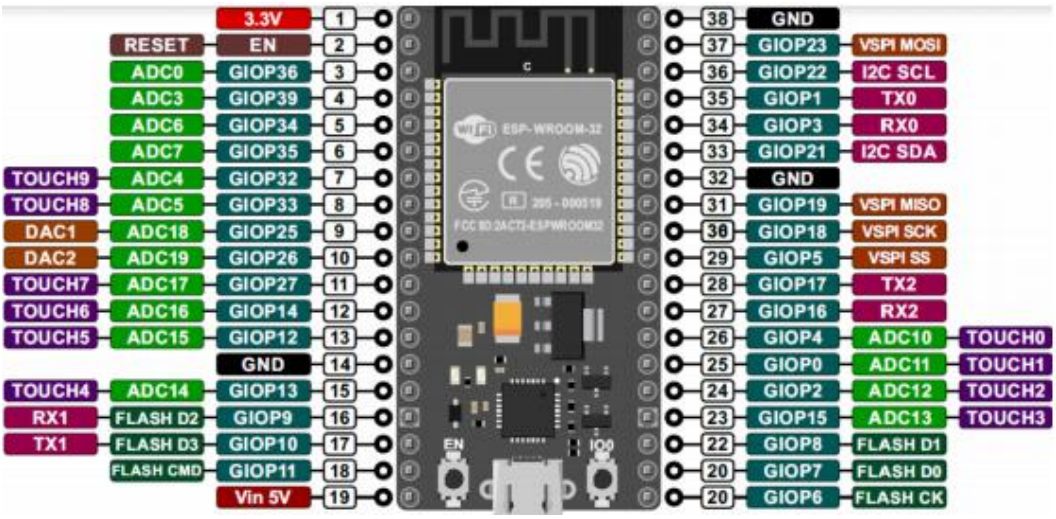


Ilustración 9: esquema indicativo de todos los pines que constituyen el módulo ESP32. GPIO22 y GPIO21 son utilizados para la comunicación I2C.

De manera que este dispositivo es el encargado de brindar ese carácter IoT al sistema, siendo además el más propicio para ello debido a su reducido coste de adquisición y consumo energético.

Este módulo cuenta con un microprocesador de 32 bits *Xtensa LX6*, de núcleo simple o doble y que opera a160 o 240 MHz. Este dispositivo es utilizado como microcontrolador tomando el papel de maestro en la comunicación con el sensor de infrarrojos, teniendo un control de las medidas captadas por el mismo.

Por lo anteriormente explicado, se resuelve que la funcionalidad principal de este módulo es la de gobernar al sensor MLX90640 mediante las instrucciones emitidas por el microcontrolador ESP32, además de enviar los datos relativos a la temperatura medida por el sensor vía WiFi y Bluetooth.

Todo este proceso se gestiona mediante la programación del microcontrolador ESP32 mediante el IDE de arduino.

Además de los elementos integrados anteriormente mencionados, el módulo ESP32 también incluye los siguientes: detector de radiofrecuencia, amplificador de potencia, amplificador receptor de bajo ruido, filtros, y módulos de administración de energía.

Adicionalmente, el módulo debe incluir un condensador de 10µF conectado entre los pines EN y GND del mismo **(Ilustración 10).** Esto es debido a que algunas placas de desarrollo ESP32 no entran de manera automática en el modo de carga/parpadeo cuando se carga el código de programa. Una manera de solventar este problema, es mantener presionado el botón BOOT de la placa mientras se está cargando el código. No obstante, resulta más cómodo incluir un condensador de tamaño pequeño que realice la misma función.

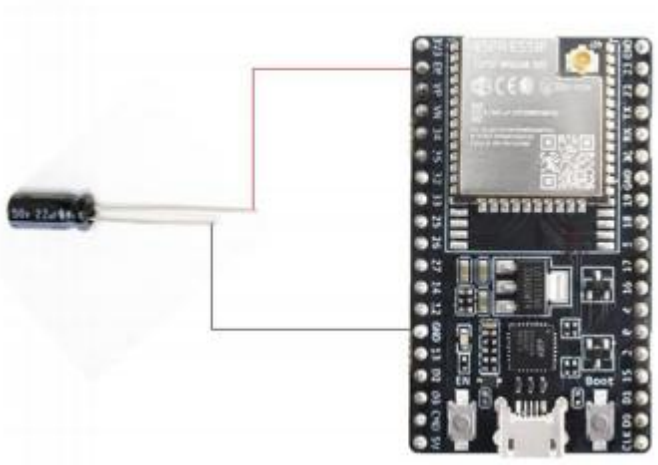


Ilustración 10: condensador de 10µF conectado entre los pines EN y GND del módulo ESP32 para facilitar su conexión al router.

❖ **Especificaciones del módulo ESP32**

- CPU: Microprocesador de 32-bit Xtensa LX6 de doble núcleo (o de un solo núcleo), operando a 160 o 240 MHz y rindiendo hasta 600 DMPIS.
- Coprocesador de ultra baja energía (ULP).
- Memoria: 520 KiB SRAM.
- Conectividad inalámbrica vía WiFi y Bluetooth.
- 36 pines GPIO.
- 4 interfaces periféricas SPI.
- 2 interfaces periféricas I2C.
- 3 interfaces periféricas UART.
- 2x8 bits DACs.
- Soporta todas las características de seguridad estándar IEEE 802.11, incluyendo WFA, WPA/WPA2 y WAPI.
- 12 bit SAR ADC de hasta 18 canales.
- Frecuencia de reloj ajustable desde 80 MHz hasta 240 MHz.
- Controlador host SD/SDIO/CE-ATA/MMC/eMMC Controlador esclavo SDIO/SPI.
- Controlador esclavo SDIO/SPI.
- Incluye sensor de efecto hall y de temperatura interna del chip.
- Pre-amplificador analógico de ultra baja potencia.

3.2.3. PC

Constituye el elemento principal de trabajo para un usuario que decida aprovechar la aplicación desarrollada. Desde el ordenador se carga la interfaz gráfica, desde la cual se puede observar los valores medidos por el sensor y su evolución en tiempo real, así como establecer diferentes parámetros. Es necesario que el PC empleado disponga de conexión a internet para poder acceder al servidor de almacenamiento y verificar la correcta transmisión de datos.

3.2.4. Smartphone

Los *smartphones* suponen el dispositivo inteligente más utilizado en la actualidad, ya que prácticamente toda la población cuenta con uno. Si bien los recursos que ofrece un *smartphone* pueden asemejarse a los que ofrece un PC, la principal diferencia que presentan los primeros es que disponen de una antena de telefonía, mediante la cual se puede producir una comunicación entre dos dispositivos separados por cientos o miles de kilómetros.

Además de disponer de WiFi, Bluetooth y radio; los *smartphones* actuales también cuentan con la tecnología 4G, la cual permite al dispositivo enviar y recibir señales de antenas 4G que brinden la posibilidad de conectar el dispositivo a internet sin la necesidad de disponer de un punto de acceso.

Otro detalle a tener en cuenta es el hecho de que los móviles de hoy disponen de la funcionalidad de zona WiFi, en la cual pueden actuar como un punto de acceso tomando la red de 4G de datos móviles y generando una señal WiFi a la que pueden conectarse otros dispositivos cercanos. Esto puede resultar útil para conectar el PC o el propio módulo ESP32 en momentos puntuales en los que no se disponga de un router.

No obstante, únicamente se requiere hacer uso del Bluetooth del *smartphone* para el envío de datos desde el microcontrolador ESP32; así como de su interfaz gráfica para desarrollar la APP de visualización del proceso.

3.3. Software

El concepto de *software* hace referencia a todos los programas e instrucciones informáticas almacenadas en una memoria y ejecutadas por un microcontrolador o microprocesador y que son esenciales para la automatización y control de cualquier proceso.

Los elementos relativos al *software* requeridos por el sistema a desarrollar son explicados con detalle en los sucesivos subapartados.

3.3.1. IDE arduino

Este IDE (*Integrated Development Environment*) supone el entorno de programación de las placas arduino, aunque con núcleos terceros puede emplearse también en otras placas en desarrollo.

Constituye una aplicación multiplataforma desarrollada en Java, y admite los lenguajes de programación C y C++ bajo ciertos estándares de estructuración de códigos. Normalmente en todo IDE de arduino se podrá contemplar la misma estructura de programa:

- Una definición previa de librerías.
- Una función de ajuste o preparación previa al bucle principal.
- Una función de bucle principal que se ejecuta de manera infinita.

En este caso la IDE de arduino contiene las librerías relativas a:

- Sensor MLX90640.
- Módulo ESP32 (WiFi y Bluetooth).
- Conexión I2C.
- APP Inventor.

Así como el conjunto de instrucciones que rigen la medición de temperatura y el envío de los datos obtenidos vía WiFi y Bluetooth.

3.3.2. ThingSpeak

ThingSpeak es una plataforma a la que se puede acceder mediante un registro con una cuenta MathWorks. El servicio que ofrece esta plataforma es una serie de servidores a los que se puede acceder para consultar información que les es transmitida desde una fuente distinta. Para ello, ThingSpeak permite la creación canales, que desde el momento de su validación generan un identificador único y una serie de claves de lectura y escritura **(Ilustración 11)** cuyo contenido deberá conocerse si se quiere acceder a dichos canales desde otras fuentes. En el caso del proyecto presente se quiere acceder a un canal creado para monitorizar efecto Hall y temperatura desde el IDE arduino y Matlab.

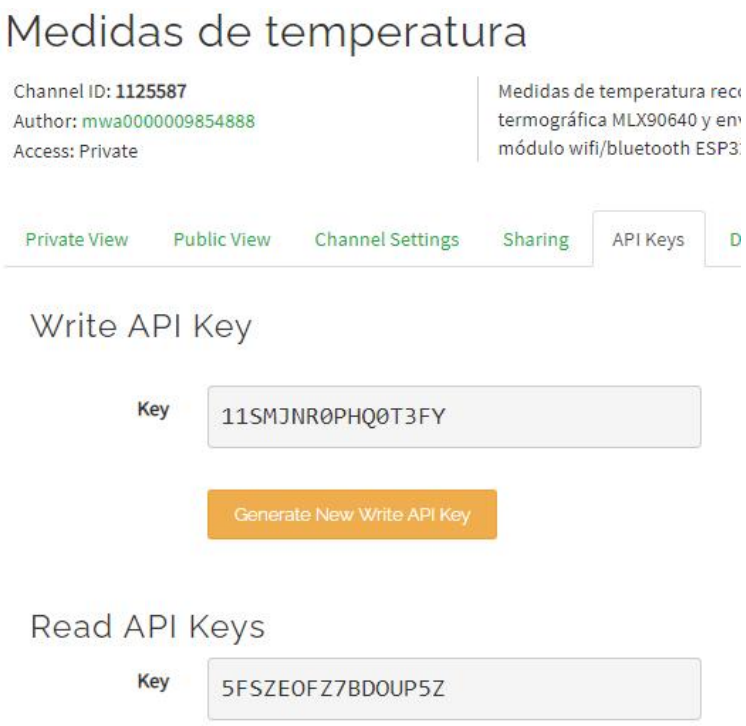


Ilustración 11: pantalla de claves de acceso al canal de ThingSpeak creado. Mediante la clave de escritura (Write API Key) se puede modificar los valores visualizados por el canal. La clave de lectura (Read API Keys) permite transferir los valores del canal a otro programa o dispositivo.

El identificador ubica el canal, sería el análogo a una dirección IP, mientras que las claves de escritura y lectura permiten modificar y obtener datos del canal, respectivamente. Siempre que se desee se puede generar nuevas claves de lectura y escritura, y el canal puede privatizarse para que su visualización solo sea posible para un conjunto de miembros establecido. ThingSpeak necesita mínimo 15 segundos entre medidas para cargarlas correctamente desde arduino.

3.3.3. Matlab

Es el *software* de programación sobre el que se desarrolla toda la interfaz. En este punto hay dos partes a tener en cuenta. La primera es relativa al diseño de la propia interfaz, cuya estética es modificada añadiendo los distintos elementos necesarios (botones, gráficos, etiquetas, etc.) desde la GUIDE de Matlab **(Ilustración 12)**. El segundo punto hace referencia a la programación de los elementos incluidos, la cual va a regir cómo se comporta la aplicación en base a la interacción con dichos elementos (en caso de los botones), o qué se va a visualizar en cada uno (en caso de las gráficas y etiquetas).

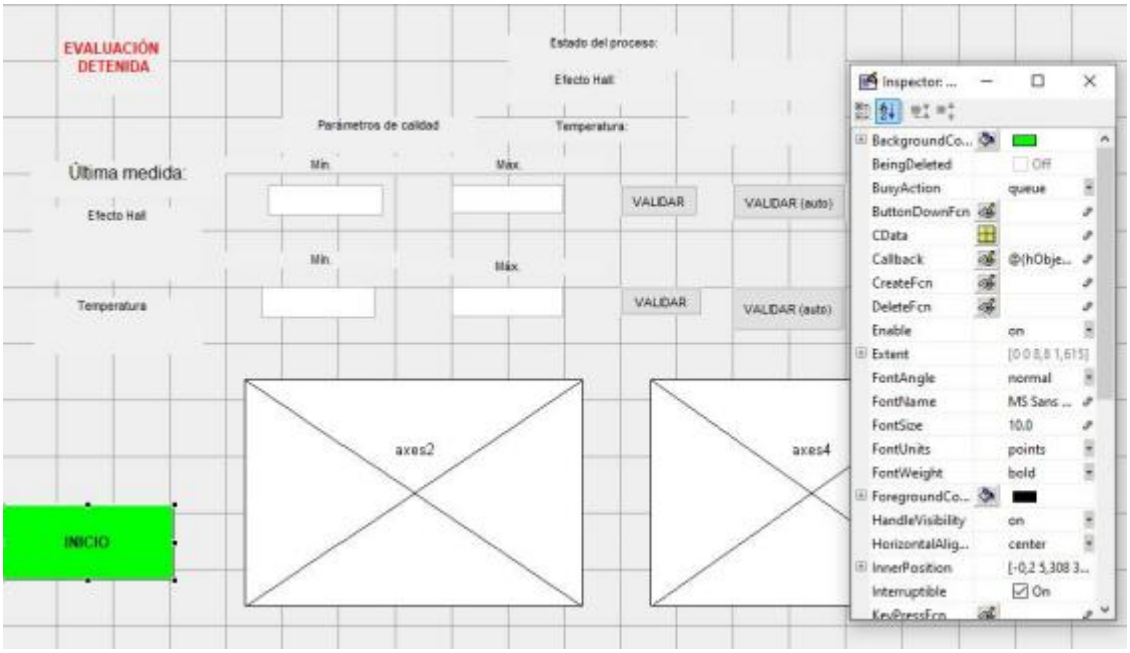


Ilustración 12: interfaz ejemplo desarrollada en el *software* Matlab. Se puede observar la presencia de elementos como ejes, botones, etiquetas y alarmas.

3.3.4. APP inventor

APP inventor es un entorno de programación de *software* diseñado para dispositivos móviles tales como *smartphones* o *tablets*. Desde este entorno, se puede desarrollar aplicaciones mediante el uso de elementos visuales u objetos y el enlazamiento de unos bloques que emulan el código de programación **(Ilustración 13)**. A pesar de que el grado de programación ofrecido sea bastante básico, APP inventor puede resultar una herramienta útil para la inclusión de los *smartphones* ya sea en procesos industriales o de otra índole.

Ilustración 13: ejemplo de interfaz desarrollada en APP inventor, con los bloques de programación enlazados para dar lugar a una determinada aplicación.

Así pues, APP inventor constituye el entorno sobre el que se desarrolla la interfaz de monitorización para un usuario que, mediante un dispositivo móvil situado dentro del alcance de la red de Bluetooth, pueda presenciar la evolución del proceso térmico.

El principal inconveniente que presenta es que en la actualidad esta APP no está totalmente desarrollada para el sistema operativo iOS, por lo que únicamente pueden hacer uso de todo el potencial de la aplicación los usuarios de Android. Es por este motivo que se ha utilizado una *tablet* con sistema operativo Android en lugar de un *smartphone* Iphone 7 para el desarrollo del proyecto. No obstante, puesto que por motivos de cotidianidad social es más probable disponer de un *smartphone* que de una *tablet*, se continuará referenciando a la transmisión de datos desde un *smartphone* durante el resto de la memoria.

3.4. Protocolos de comunicación

3.4.1. I2C

La comunicación I2C (*inter integrated circuits*) constituye uno de los protocolos de comunicación serie, de manera síncrona, que permite el intercambio de datos entre dos o más dispositivos conectados al bus, siendo uno de ellos maestro y el resto esclavos. Para lograrlo, se hace valer únicamente de dos vías de comunicación: SDA (*Serial Data*) y SCL (*Serial Clock*).

Ambas líneas son del tipo drenador abierto, pero asociados a un transistor de efecto campo (o FET), lo que resulta en un estado similar al de colector abierto. Para que un chip pueda establecer un nivel alto en cualquiera de las líneas, se debe incluir en ambas resistencias de polarización a 3.3V.

La línea SDA es por la que se transmiten los bits de datos, y por cada uno de ellos es necesario un pulso de reloj emitido por el maestro por la línea SCL. Únicamente puede transmitirse un bit de datos cuando el puerto SCL se encuentra en nivel bajo o 0 lógico.

El protocolo I2C establece que la comunicación comienza cuando el maestro envía al bus la condición de inicio (Start). Esta condición se genera cuando el SDA ocupa el bus, pasando de un nivel alto a un nivel bajo mientras el SCL permanece en nivel alto **(Ilustración 14).**

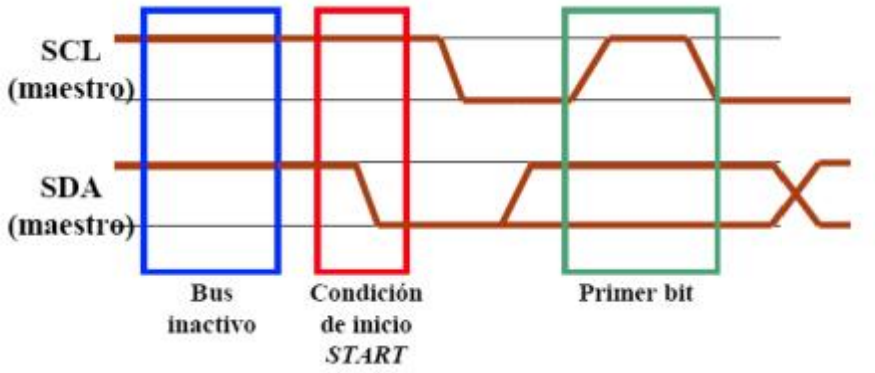


Ilustración 14: condición de arranque generada por el SDA del maestro. La zona roja corresponde a dicha condición, mientras que en la región verde se transmite el primer bit y la azul corresponde al reposo inicial antes de la comunicación con el esclavo.

Seguidamente, el maestro envía 8 bits, de los cuales, los 7 primeros corresponden a la dirección del esclavo con el que se desea comunicar y el último indica si va a producir escritura (0) o lectura (1).

El siguiente bit es enviado del esclavo hacia el maestro, y supone un bit de reconocimiento (ACK). Si cuando el maestro lee el estado de la línea SDA y recibe un 0, la transferencia de datos continúa. Si por el contrario recibe un 1, significa que el esclavo no valida la comunicación y el maestro genera la condición de parada para liberar el bus. A partir de este punto, cada 8 bits que envía el maestro el esclavo envía uno de reconocimiento (ACK).

Cuando la transmisión finaliza, el maestro genera igualmente la condición de parada, que se da cuando la línea SDA pasa de un estado bajo a un estado alto, quedando ambas líneas en estado alto.**(Ilustración 15).**

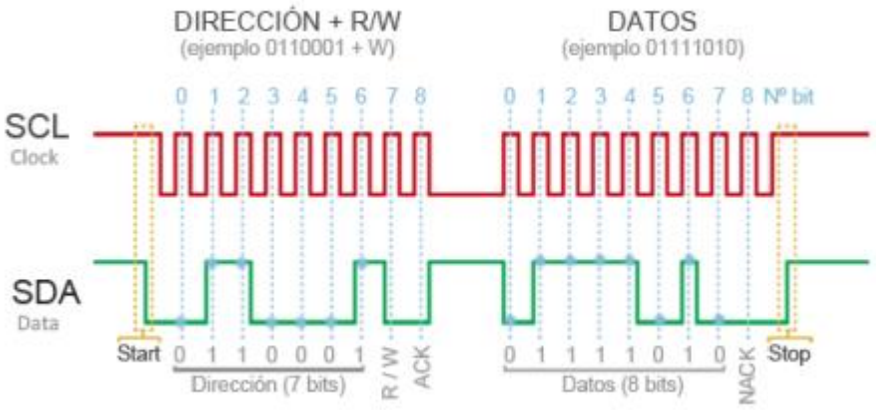


Ilustración 15: protocolo de comunicación I2C. El maestro genera el SCL a un nivel alto por defecto. SDA se encarga de marcar el inicio y la parada de la comunicación entre maestro y esclavo, así como de transmitir los datos del esclavo al maestro.

En el caso del presente proyecto, la comunicación I2C se emplea para transferir los datos de temperatura desde la MLX90640 (esclavo) hacia el microcontrolador ESP32 (maestro). Para ello se alimentan ambos a 3.3V respecto a tierra y se conectan los respectivos puertos SDA y SCL **(Ilustración 16).** El módulo ESP32 dispone de las resistencias pull-up en las líneas SDA y SCL por lo que no se deben ser añadidas adicionalmente.

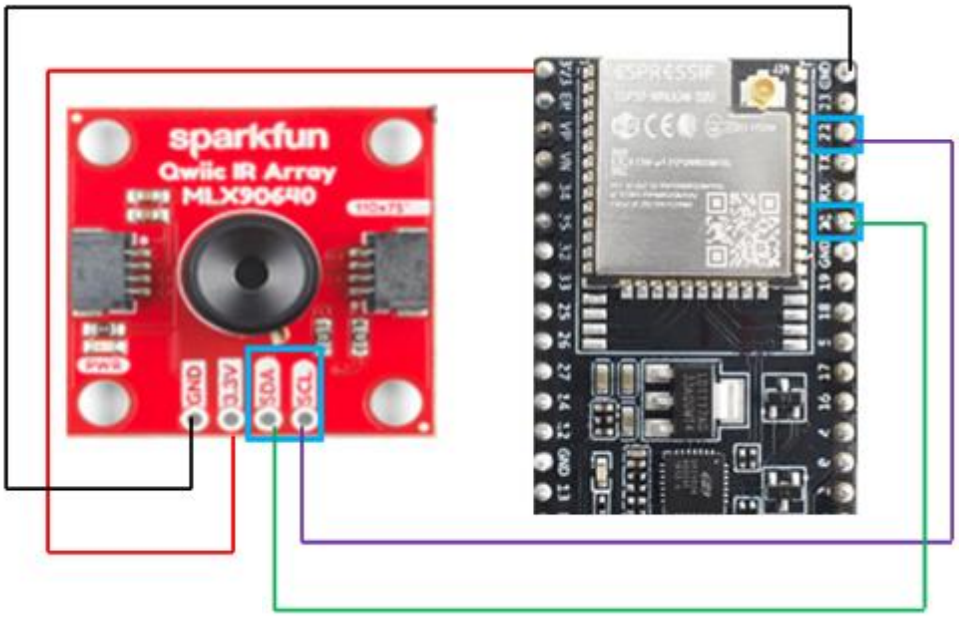


Ilustración 16: conexión entre MLX90640 y ESP32. Se emplean los puertos SDA y SCL (marcados en azul) del ESP32 como maestro y los del MLX90640 como esclavo. Además, MLX90640 se alimenta con la salida de 3,3V del módulo ESP32.

La librería wire.h en el IDE de arduino permite acceder automáticamente al registro que contiene la dirección del bus I2C del esclavo, generando de manera automática la comunicación I2C entre los elementos conectados.

3.4.2. WiFi

WiFi es una abreviación de la marca comercial "*Wireless Fidelity*" (fidelidad inalámbrica), y consiste en una tecnología capaz de conectar de manera inalámbrica dos dispositivos electrónicos, permitiendo así una comunicación entre ellos sin la necesidad de conectar ningún cable. Para ello, es necesario que ambos dispositivos cuenten con una antena WiFi, además de requerirse un punto de acceso de red inalámbrica **(Ilustración 17)** en este caso, un router.

Ilustración 17: conexión de distintos dispositivos electrónicos a internet a través de un punto de acceso o router que genera la señal WiFi.

En las redes WiFi podemos encontrar dos frecuencias de emisión: una de 2.4 GHz y otra de 5 GHz.

Existen diversos tipos de WiFi, cada uno basado en estándar IEEE 802.11. Los estándares o protocolos de comunicación WiFi que emplea el módulo ESP32 es el IEE 802.11 b/g/n. Este último (802.11n) le permite aumentar la velocidad de transmisión a 150 Mbps.

En el sistema, la red WiFi es utilizada para comunicar el módulo ESP32 con un determinado router, permitiendo que la información captada por el sensor de

infrarrojos sea transferida a un servidor web de almacenamiento de datos mediante el mencionado router.

3.4.3. Bluetooth

El Bluetooth constituye el segundo protocolo de comunicaciones inalámbricas incluido en el sistema y uno de los estándares WPAN existentes en la industria. WPAN son las siglas de *Wireless Personal Area Network*, que se traduciría al español como “Red de Área Personal Inalámbrica”.

Dentro de este tipo de redes, el Bluetooth es comúnmente utilizado en la comunicación de dispositivos de bajo consumo y solamente tiene alcance de unos pocos metros, por lo que necesariamente debe cumplirse la condición de que ambos dispositivos se encuentren cerca el uno del otro para posibilitar la transferencia de información entre ellos.

La frecuencia de emisión de Bluetooth oscila entre los 2.4 GHz y los 2.5GHz. Esto se debe a que al compartir la banda de los 2.4 GHz con la señal WiFi se producen interferencias entre ambas; las cuales se solucionan con un pequeño salto de frecuencia en la señal de Bluetooth.

Existen muchas versiones del protocolo de comunicaciones Bluetooth, que han sido desarrollados a lo largo de los años y corrigiéndose los fallos en las versiones más actuales respecto a las anteriores. El módulo ESP32 utiliza la versión Bluetooth v4.2 BR/EDR con especificación BLE. Este protocolo supone la última actualización de la cuarta versión, ofreciendo un consumo energético muy reducido y con la inclusión del concepto BLE que permite una sincronización más duradera. Además, esta versión aumentó la tasa de transferencia y permite transmitir desde 25 Mbps hasta 32 Mbps.

En el sistema, se utiliza la comunicación Bluetooth para transferir directamente desde el módulo ESP32 los datos relativos a las temperaturas medidas por el sensor de infrarrojos a un *smartphone* ubicado en las proximidades del proceso y previamente vinculado.

4. Descripción detallada de la solución adoptada

El principal inconveniente que surge en el intercambio de información entre los componentes del IoT es la capacidad de almacenamiento del servidor web. ThingSpeak permite la actualización de valores en todos los campos de un canal de manera simultánea en intervalos de (como mínimo) 15 segundos. No obstante, cada campo permite la actualización de un único valor o byte. Siendo que cada canal del servidor permite el almacenamiento de 8 canales y que la matriz de temperaturas contiene 768 valores cada uno de ellos compuesto por 2 bytes; se concluye una adquisición de 1536 bytes de información que han de ser transmitidos a 8 campos como cada 15 segundos.

Con tal de no perder información de ningún pixel de la cámara termográfica, se ha optado por dividir la matriz de pixeles en 8 regiones o bloques exactos **(Ilustración 18)**, coincidiendo el número de regiones con el máximo número de posibles bytes transferidos simultáneamente a un canal de ThingSpeak. Después, se realiza la media aritmética de cada uno de los bloques y se envían por WiFi los 8 valores.

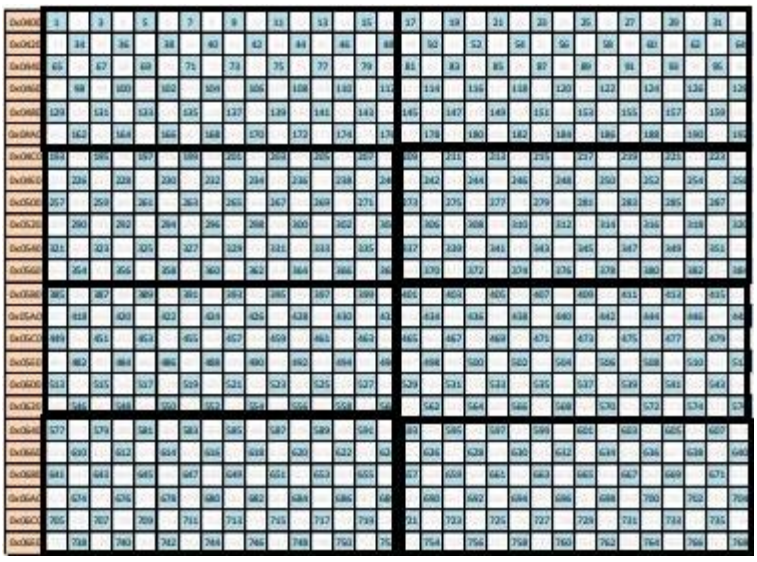


Ilustración 18: división de la matriz 32x24 en una matriz 2x4 para agrupar los 768 pixeles que componen la imagen en 8 grupos simétricos de 96 pixeles cada uno.

A pesar de permitir la actualización de datos cada 15 segundos, se ha aumentado este tiempo de muestreo a 21 segundos, ya que en algunos casos el desfase entre tiempos de actualización entre los distintos elementos provocaba la pérdida de imágenes térmicas puntuales.

En los siguientes subapartados se explica la lógica seguida en el desarrollo de los distintos programas que conforman el proyecto. Los programas completos del IDE

arduino y Matlab pueden hallarse en los documentos anexos a la memoria, mientras que el programa de APP inventor se detalla en el apartado ***4.3.-APP Inventor***

4.1. IDE Arduino

El programa desarrollado en la plataforma IDE de arduino contiene y define las librerías y funciones respectivamente necesarias para la adquisición de las temperaturas medidas por parte de la MLX90640. Tras definirse las variables esenciales, en la función de ajuste se inicia la conexión WiFi al router especificado en el programa, en este caso el router personal del autor. Asimismo, se inicia también la transmisión de señal Bluetooth para la conexión con un dispositivo cercano.

Una vez comprobada la conexión WiFi, se comprueba la conexión de la cámara termográfica en la dirección previamente especificada, y se ejecuta un código de depuración con tal de prevenir posibles errores en la extracción de parámetros.

Todas las conexiones así como el estado de los diferentes registros son mostrados por el puerto serie para comprobar su correcta operatividad.

En el bucle principal se realiza la lectura de todos los pixeles de manera secuencial, y se desarrollan otros 8 bucles para discernir entre las ocho regiones que van a formar la nueva matriz 2x4 de temperaturas. Definidas dichas regiones o bloques, se realiza una media aritmética sumando todas las temperaturas leídas y dividiendo el valor resultante por 96 valores que componen cada bloque.

Acto seguido, se ejecutan las funciones de transferencia Bluetooth por parte del módulo ESP32, que será emparejado con un dispositivo móvil provisto con la APP de adquisición de datos detallada en el apartado ***4.3.- APP Inventor.***

Por último, se hace uso de las funciones pertenecientes a las librerías del servidor ThingSpeak, obteniendo el permiso de las claves para escribir en el canal y los campos especificados el valor de cada promedio.

4.2. Matlab

El programa desarrollado en el *software* Matlab se encarga de leer los datos que se actualizan en ThingSpeak cada 21 segundos y los ordena reconstruyendo la matriz de 2x4 temperaturas o pixeles. Esta matriz es mostrada en la pantalla principal de la interfaz gráfica, detallada en el siguiente subapartado.

4.2.1. Interfaz gráfica

La interfaz gráfica engloba todos los elementos de interacción que aparecen en las pantallas y que brindan la posibilidad a cualquier usuario que pretenda hacer uso de la aplicación de obtener información valiosa del proceso térmico.

La interfaz gráfica del *software* Matlab se gestiona a través de un ordenador con conexión a internet y en ella se pueden encontrar 4 pantallas:

- Pantalla principal

La pantalla principal **(Ilustración 19)** es desde la que se monitoriza el estado del proceso térmico a través de la matriz de temperaturas reconstruida.

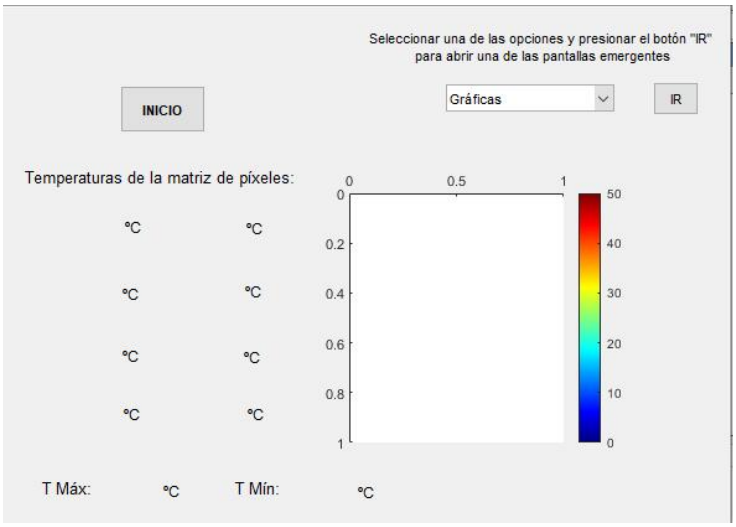


Ilustración 19: pantalla principal de la interfaz gráfica desarrollada en Matlab. Desde esta pantalla se observa de manera visual el gradiente de temperaturas y se accede a todas las demás (gráficas, parámetros y alarmas).

En esta pantalla encontramos una gráfica en la que se representa la matriz de temperaturas mediante un gradiente de colores. Asimismo, se muestra también en forma numérica dicha matriz. De esta manera, se puede obtener de una manera visual la información, aunque poco precisa, de la región focalizada por la cámara en la que se encuentra el foco caliente o frío. La adquisición de valores desde el servidor web comienza cuando se pulsa el botón de inicio y los valores máximo y mínimo registrados son mostrados en la parte inferior.

Se encuentra también en esta pantalla la lista desplegable con el resto de pantallas a las que se puede acceder mediante la selección correspondiente y pulsando el botón "IR".

❖ Pantalla de gráficas

La pantalla de gráficas **(Ilustración 20)** permite realizar un seguimiento de la evolución de la temperatura en el tiempo mediante 8 gráficas que muestran las fluctuaciones de cada bloque de píxeles.

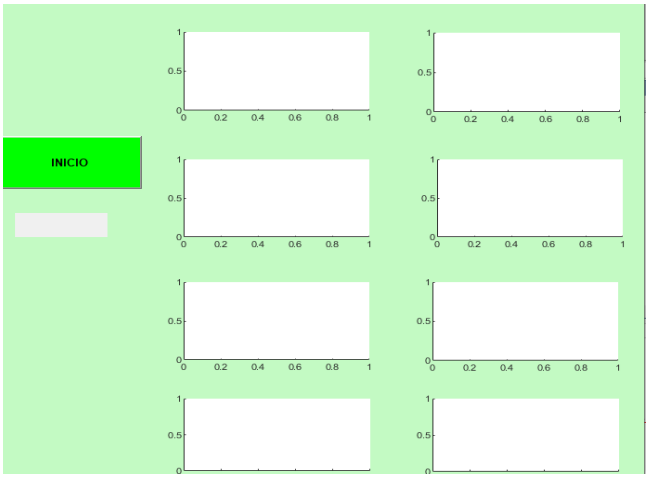


Ilustración 20: pantalla de gráficas de la interfaz de usuario desarrollada en Matlab. En esta pantalla hay 8 gráficas dispuestas en orden que muestran la evolución de la temperatura de cada bloque de pixeles.

Tras pulsar el botón de inicio, comienza la adquisición de temperaturas que se actualiza cada 21 segundos.

❖ Pantalla de parámetros

La pantalla de parámetros **(Ilustración 21)** permite establecer los parámetros de calidad que determinan el estado del proceso.

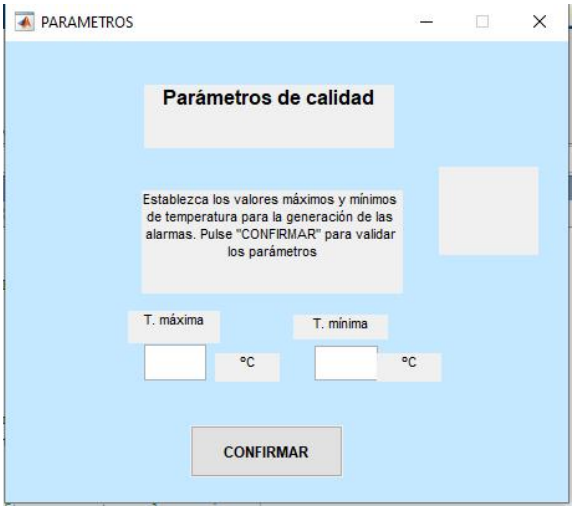


Ilustración 21: pantalla de parámetros de calidad. En esta pantalla se seleccionan los límites de temperatura por parte del usuario en función de la aplicación que se le vaya a dar al sistema y de los valores permitidos en el proceso pertinente.

Tras establecer dichos parámetros y pulsar el botón “CONFIRMAR”, se ejecuta la comparación constante de los parámetros con los valores máximos y mínimos recogidos por el sensor. Si el valor medido excede alguno de los límites, esta pantalla genera la alarma pertinente y es mandada a la pantalla de alarmas. Esto se realiza haciendo uso de un canal adicional de ThingSpeak.

- Pantalla de alarmas

La pantalla de alarmas **(Ilustración 22)** contiene el histórico de todas las alarmas producidas desde el inicio del proceso. Para ello, la adquisición se realiza en todo momento cada 21 segundos, sin necesidad de pulsar ningún botón.

Ilustración 22: pantalla de las alarmas. En esta pantalla aparece un listado de todas las alarmas que se han generado desde que el sistema se pone en marcha. Cada alarma se ubica mediante la fecha y hora en la que se ha producido.

Esta pantalla aporta la información sobre cómo se está ejecutando el proceso en base a la cantidad de alarmas generadas en el tiempo. Para el acuse de las alarmas, el programa de esta pantalla lee el valor establecido en el canal adicional de ThingSpeak por la pantalla de parámetros, y en base al valor leído genera un tipo de alarma u otra, en caso de que la haya.

4.3. APP Inventor

La aplicación de APP Inventor es desarrollada en un ordenador con acceso a internet, y después es cargada en el dispositivo móvil correspondiente como una APP móvil. En este caso, se desarrolla otra interfaz gráfica aunque más simple que la de Matlab, la cual muestra la información continua de las 8 temperaturas relativas a los promedios realizados.

El conjunto bloques de programación que permite la adquisición de las temperaturas se muestra en la **Ilustración 23** numerados por orden de aparición, mientras que en la **tabla 1** se describe la operación de cada uno de los bloques.

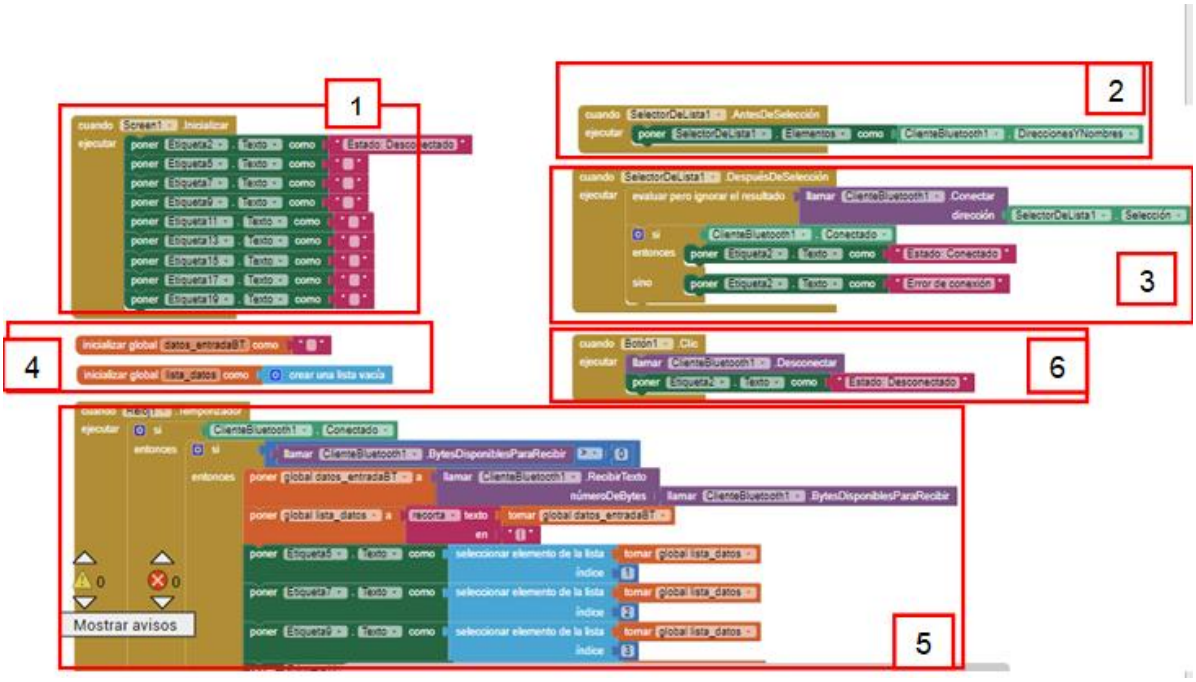


Ilustración 23: conjunto de bloques empleados en la programación de la APP. Cada conjunto de bloques está numerado por orden de aparición en el programa.

Bloque 1	Se establece el estado del programa en su inicialización. Las etiquetas en las que se va a mostrar el valor de la variable temperatura permanecen vacías.
Bloque 2	Se configura que al presionar el selector de lista se muestre una lista con los dispositivos Bluetooth disponibles.
Bloque 3	Tras seleccionar el dispositivo Bluetooth con el que se desee realizar el emparejamiento, se intenta realizar la conexión. Posteriormente, se evalúa si hay conexión o no. La etiqueta 2 aparecerá un texto informativo sobre el estado de la conexión en cualquiera de los casos.
Bloque 4	Se inicializan las variables con las que se van a manipular los datos de entrada. Una se utilizará para almacenar los datos de las lecturas del sensor, mientras que la otra se empleará para crear una lista con dichos datos.
Bloque 5	En primer lugar, se verifica que la APP esté lista para recibir los datos. En caso afirmativo, se asignan los datos de entrada vía Bluetooth a la variable datos_entradaBT. Después se utiliza lista_datos para crear una lista con los valores de la variable anterior y se establece que el separador entre elementos de la lista sea el carácter "|". Desde el ESP32 se mandará las temperaturas vía Bluetooth estableciendo siempre un "|" entre temperaturas para que se discierna entre cada una de ellas durante su recepción en la APP
Bloque 6	Cambiar el texto de la etiqueta que informa sobre el estado de la conexión a "Desconectado" si se pulsa el botón de desconexión.

Tabla 1: Conjunto de bloques que forman la programación de la APP móvil desarrollada mediante APP Inventor. La segunda columna detalla la operación de cada bloque y menciona sus elementos principales.

5. Resultados obtenidos y discusión

Tras definir la manera de abordar las especificaciones y objetivos que pretende satisfacer el proyecto, realizando la programación de todos los elementos anteriormente detallados, se obtienen una serie de resultados que, posteriormente serán evaluados para calificar la viabilidad del sistema.

En este apartado se muestran diferentes pruebas, correspondientes a diferentes posibles situaciones que pueden darse en cualquier proceso térmico y que resulta conviene abordar.

En primer lugar, se muestra las gráficas y la interfaz principal (**Ilustración 24**) cuando el proceso que está teniendo lugar no está superando los parámetros límite de calidad, es decir, cuando el proceso está transcurriendo correctamente sin la aparición de alarmas de ningún tipo.

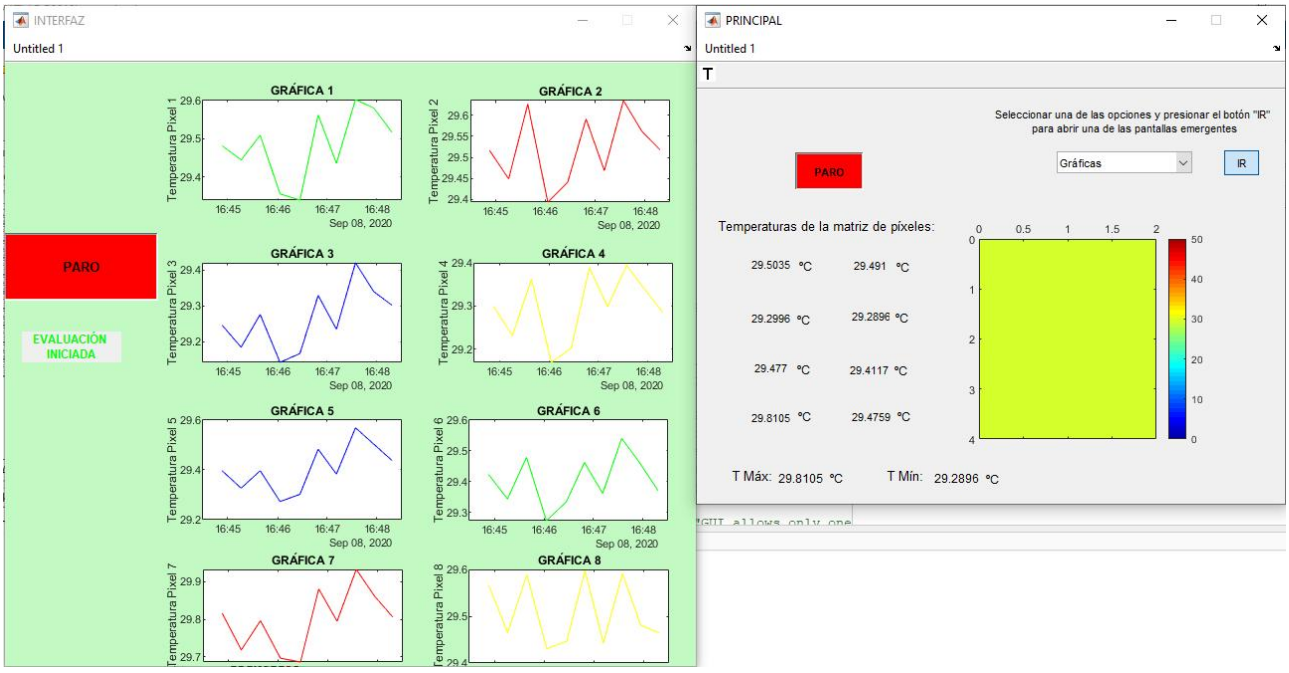


Ilustración 24: pantalla principal y gráficas en funcionamiento para un proceso dentro de los valores límite.

En este caso se puede observar una matriz de temperaturas completamente amarilla, sin prácticamente cambios de tonalidad entre los pixeles que la conforman debido a la ínfima variación de temperatura entre ellos. La **Ilustración 25** muestra las pantallas de parámetros y alarmas cuando el proceso se encuentra en la fase explicada.

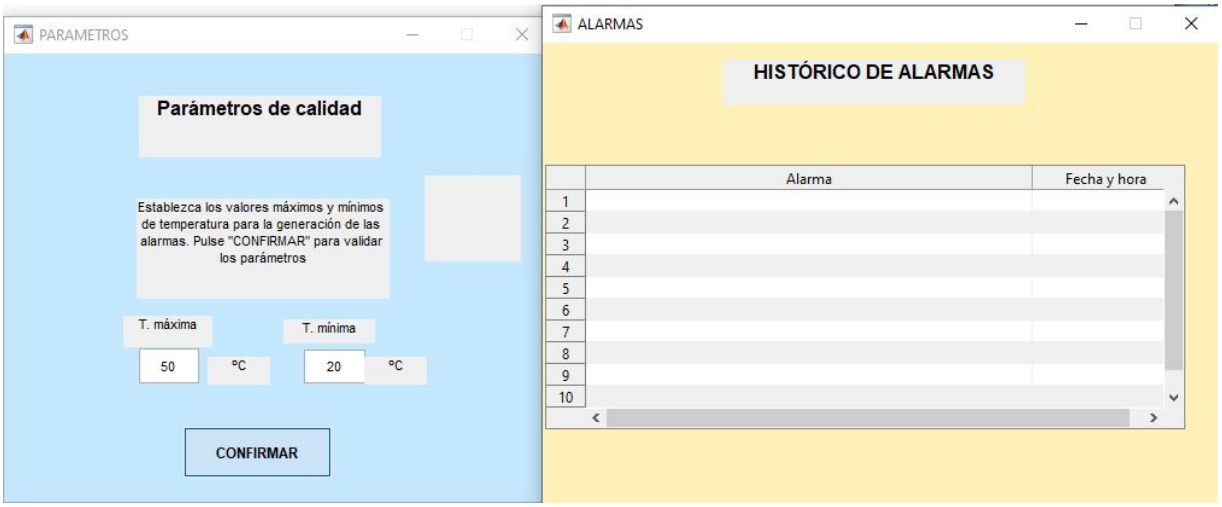


Ilustración 25: pantallas de parámetros de calidad y alarmas. En la primera se seleccionan los valores límite que generan una alarma al superarse. En la segunda se almacenan las alarmas por fecha. En la situación actual las temperaturas registradas pertenecen al rango por lo que no se acusa ninguna alarma.

❖ Detección de un foco caliente

Para comprobar la detección de un foco caliente, se ha procedido a encender la llama de un mechero en una zona específica de la cámara termográfica **(Ilustración 26)**, con tal de ponerla a prueba viendo con qué exactitud es capaz de detectar y ubicar dicho foco

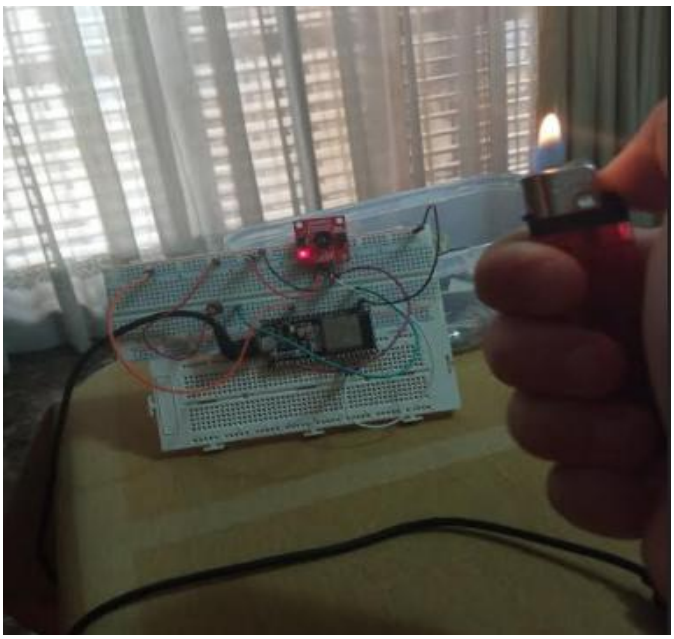


Ilustración 26: prueba de foco caliente realizada con un mechero. La llama se ubica en la parte derecha de la zona de barrido de la cámara termográfica.

Se hace evidente el éxito de la prueba al comprobar en la interfaz principal de Matlab la correcta ubicación del foco caliente por parte de la cámara termográfica **(Ilustración 27),** coincidiendo el píxel más caliente con la zona en la que se ha encendido el mechero.

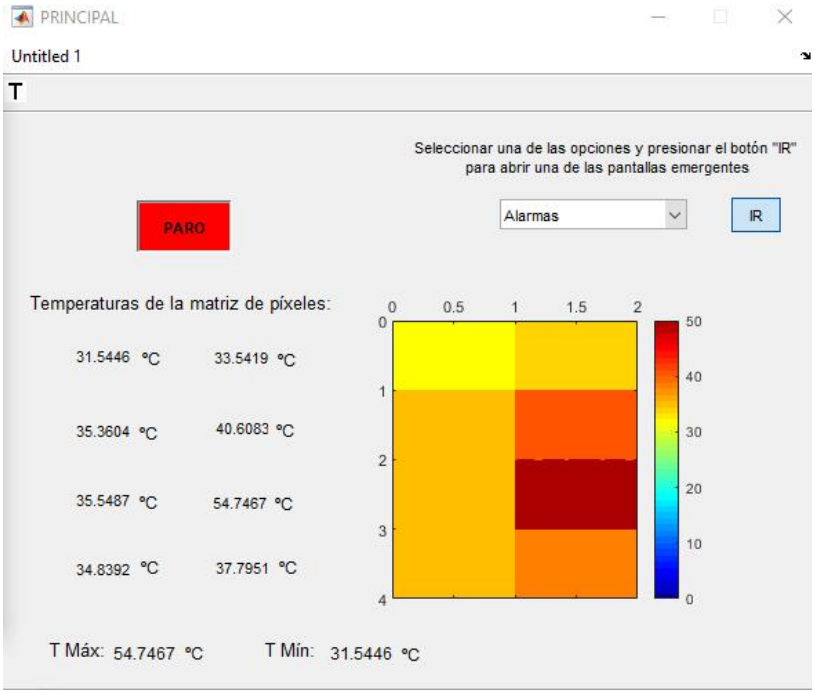


Ilustración 27: pantalla principal de la interfaz gráfica con los valores de temperatura medidos. La zona roja intensa corresponde a la ubicación de la llama respecto a la cámara termográfica.

En la **Ilustración 28** se puede observar la aparición de la alarma generada debido a la detección de una temperatura máxima que excede el límite superior establecido en los parámetros de calidad, con la fecha y hora en la que se produce dicha alarma.

Ilustración 28: pantalla de las alarmas registrando el aviso del foco caliente detectado por la cámara termográfica.

- **Detección de un foco frío**

En esta ocasión se elige una placa de hielo para realizar la prueba de la detección y ubicación del foco frío, ubicándose este en la zona inferior de toda la región captada por la cámara **(Ilustración 29)**

Ilustración 29: prueba de foco frío realizada con una placa de hielo. El foco se ubica en la parte inferior de la zona de barrido de la cámara termográfica.

De la misma manera que en la prueba anterior, se concluye una detección exitosa por parte de la cámara al coincidir los pixeles más fríos mostrados en la matriz de temperaturas con la zona en la que se encuentra la placa de hielo **(Ilustración 30).**

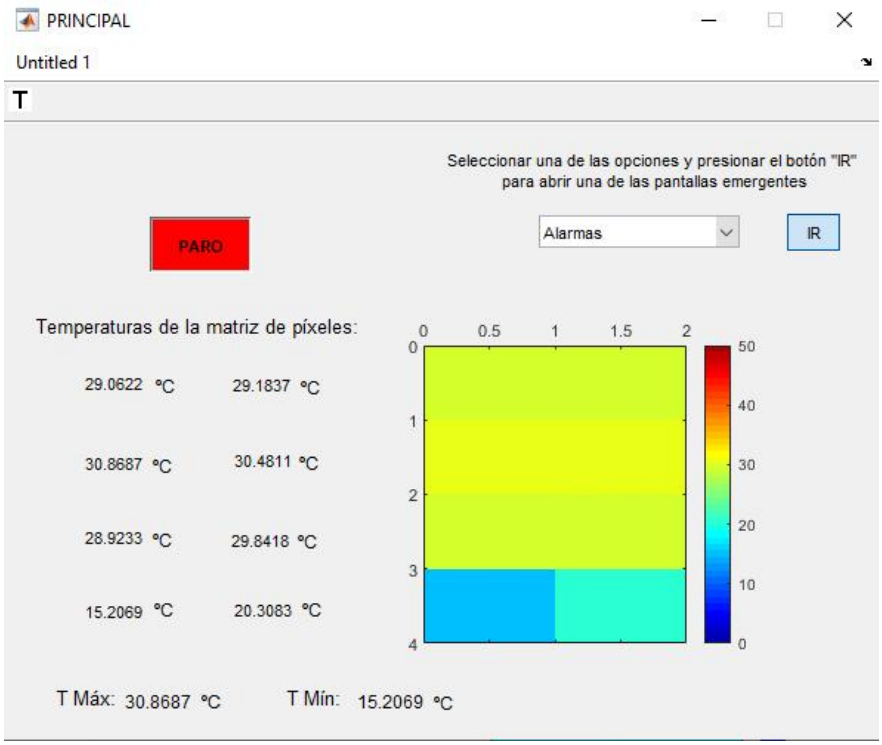


Ilustración 30: pantalla principal de la interfaz gráfica con los valores de temperatura medidos. La zona azul corresponde a la ubicación del hielo respecto a la cámara termográfica.

En la **Ilustración 31** se puede observar la aparición de la alarma generada debido a la detección de una temperatura mínima que excede el límite inferior establecido en los parámetros de calidad, con la fecha y hora en la que se produce dicha alarma.

Ilustración 31: pantalla de las alarmas registrando el aviso del foco frío detectado por la cámara termográfica.

En último lugar, se muestra en la **Ilustración 32** la interfaz de la APP móvil desarrollada para la monitorización de los valores de temperaturas vía Bluetooth desde la *tablet*. En este caso, se trata un proceso que está teniendo lugar exitosamente sin exceder valores límite.

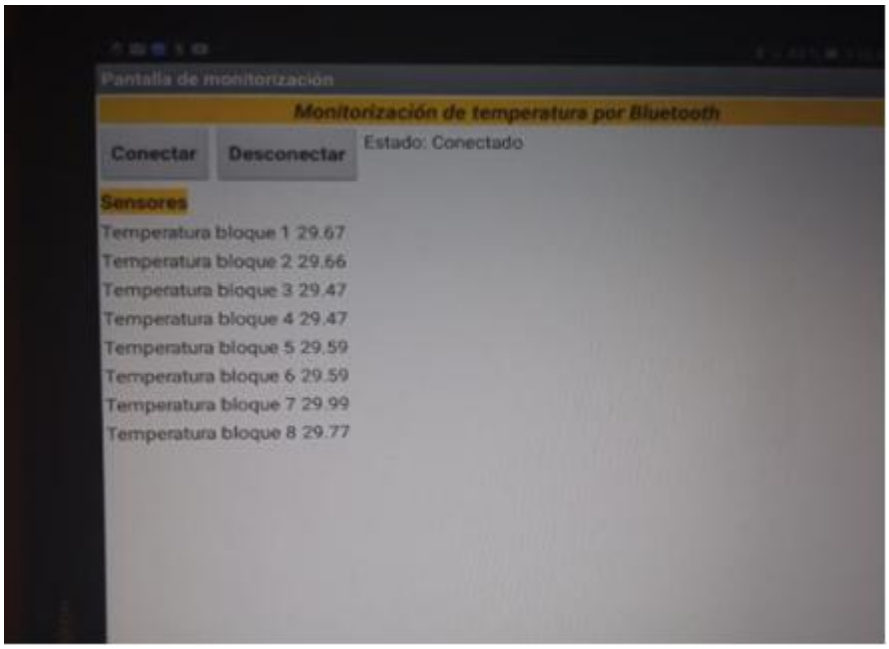


Ilustración 32: vista de la *tablet* desde la que se carga la APP móvil. Los valores de temperatura medidos por el sensor aparecen por orden tras iniciar la comunicación Bluetooth con el módulo ESP32.

6. Conclusiones

En base a los resultados obtenidos durante el desarrollo del proyecto, así como teniendo en cuenta los objetivos inicialmente planteados, se justifican las siguientes conclusiones que dan respuesta a las cuestiones más importantes planteadas.

- Se ha desarrollado exitosamente un sistema puramente inalámbrico de monitorización de temperatura y fiable, ya que puede comprobarse su correcta operación a través de las interfaces gráficas. Se ha demostrado que el sistema no sólo detecta puntos térmicos anómalos, sino que además, es capaz de identificarlos en el espacio.

- En comparación con un sistema cableado, la capacidad de transmisión de datos simultáneos se reduce notablemente debido a las limitaciones del servidor web de almacenamiento. No obstante, a pesar de la baja resolución de la imagen térmica resultante, la eliminación del cableado permite una movilidad completa de la cámara, pudiendo ser reubicada fácilmente sin necesidad de mover elementos adicionales.

- Se ha desarrollado un sistema versátil, que puede ser integrado en multitud de procesos térmicos no específicos. Siempre que el proceso no exceda las temperaturas límite medidas por el sensor [-40…300]ºC, se puede emplear el sistema fijando los parámetros deseados en función del proceso que se quiera evaluar.

- Se ha posibilitado la visualización del estado del proceso por parte de un usuario ubicado cerca del mismo, a través de un dispositivo móvil mediante la APP desarrollada. Se ha brindado al sistema con la posibilidad de ser supervisado por cualquiera que disponga de la APP instalada en su dispositivo.

- Al emplear el Microcontrolador del ESP32, se han reducido costes así como el peso global del sistema respecto al uso de una placa arduino.

Para concluir, cabe destacar la facilidad para añadir nuevas funcionalidades a la interfaz gráfica, desarrollando en pocos minutos pantallas útiles que permitan la implementación de la cámara en otro tipo de procesos. Por ejemplo, una pantalla que permita discernir si el proceso va a ser estático o dinámico. En el caso de aprovechar el sistema para un proceso estático, se generaría una pantalla con un botón que permitiese realizar una única medida de temperatura, en vez de ejecutarse mediciones cada cierto tiempo. Este tipo de aplicación puede verse en la actualidad en las cámaras termográficas que se están empleando para la detección de fiebre por el COVID-19.

7. Referencias bibliográficas

- Maik WERNER, Doris RAUHUT.(2009). Código de buenas prácticas vitivinícolas ecológicas. *Infowine.,*12:2-4. (última consulta el 25/08/2020).
- Masot, R. (2018-2019). Temario Redes de sensores. Valencia: UniversitatPolitècnica de València.
- Melexis. (2019). Datasheet MLX90640 32x24 IR array.
- Vergara, L. (2000). Fabricación y caracterización de bolómetros de Si:Ge:O utilizando tecnicas de micromecanizado del silicio. Madrid: Universidad Politecnica de Madrid.
- (Castillo y col., 2020). Sistema de monitorización IoT de temperatura.
- https://randomnerdtutorials.com/solved-failed-to-connect-to-esp32-timed-out-waiting-for-packet-header/ (última consulta el 25/08/2020).
- https://community.appinventor.mit.edu/t/ble-esp32-bluetooth-send-receive-arduino-ide/1980/5 (última consulta el 29/08/2020).
- ESP32-WROOM-32.Datasheet.
- https://github.com/espressif/arduino-esp32 (última consulta el 30/08/2020).

8. Anexos

8.1. Código de arduino IDE

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

#include "MLX90640_API.h"
#include "MLX90640_I2C_Driver.h"
#include "ThingSpeak.h" //incluir librería de thingspeak
#include <WiFi.h> //incluir librería del módulo WiFi

#define SECRET_SSID "Albo_David"    // nombre de red WiFi
#define SECRET_PASS "A62486248A"  // contraseña de red WiFi
#define SECRET_CH_ID 1127965     // ID del canal de thingspeak con el
que se quiere establecer la comunicación
#define SECRET_WRITE_APIKEY "D294ZL4E0RTGEF44"   // replace XYZ with
your channel write API Key
#define SECRET_AUTH "XeOgBsypYhOevTAccv7Bi8edUpnWR9Bf" // clave de
seguridad para establecer la comunicación con APP Blynk
#define TA_SHIFT 8 //Default shift for MLX90640 in open air

char auth[] = SECRET_AUTH;
char ssid[] = SECRET_SSID;
char pass[] = SECRET_PASS;
int keyIndex = 0;            // your network key Index number (needed
only for WEP)

unsigned long myChannelNumber = SECRET_CH_ID;
const char * myWriteAPIKey = SECRET_WRITE_APIKEY;
const byte MLX90640_address = 0x33; //Default 7-bit unshifted address
of the MLX90640


int error=1;
uint16_t Register;
static float mlx90640To[768];
paramsMLX90640 mlx90640;

WiFiClient  client;
BluetoothSerial SerialBT;

void setup()
{
  Wire.begin();
  Wire.setClock(400000); //Incrementar I2C clock speed a 400kHz
SerialBT.begin("ESP32test");

  Serial.begin(115200);
  delay(500);

   WiFi.mode(WIFI_STA);
  ThingSpeak.begin(client);  // Inicializar ThingSpeak
```

```
 if(WiFi.status() != WL_CONNECTED){
    Serial.print("Attempting to connect to SSID: ");
    Serial.println(SECRET_SSID);

    while(WiFi.status() != WL_CONNECTED){
      WiFi.begin(ssid, pass);  // Conectar a red WPA/WPA2.

      Serial.print(".");
      delay(5000);
    }
    Serial.println("\nConnected.");
  }


  while (!Serial); //Wait for user to open terminal
  //Serial.println("MLX90640 IR Array Example");

  if (isConnected() == false)
  {
    Serial.println("MLX90640 not detected at default I2C address.
Please check wiring. Freezing.");
    while (1);
  }
  Serial.println("MLX90640 online!");

  //Obtener parámetros del dispositivo
  int status;
  uint16_t eeMLX90640[832];
  status = MLX90640_DumpEE(MLX90640_address, eeMLX90640);
  if (status != 0)
    Serial.println("Failed to load system parameters");

  status = MLX90640_ExtractParameters(eeMLX90640, &mlx90640);
  if (status != 0)
    Serial.println("Parameter extraction failed");

  //Una vez los parámetros son extraídos, se inicia el array
eeMLX90640

  //DEPURACIÓN

  error = MLX90640_I2CRead(MLX90640_address, 0x800D, 1, &Register);
  Serial.println("ANTES");
  Serial.print("Error lectura: ");Serial.print(error);Serial.print(" /
Registro leído: ");Serial.println(Register);

  Serial.println("Escribiendo...");
  error = MLX90640_I2CWrite(MLX90640_address, 0x800D, 0x1901);
  Serial.print("Error escritura: "); Serial.println(error);

  Serial.println("DESPUÉS");
  error = MLX90640_I2CRead(MLX90640_address, 0x800D, 1, &Register);
  Serial.print("Error lectura: "); Serial.print(error); Serial.print("
/ Registro leído"); Serial.println(Register);


}
```

```
void loop()
{
  for (byte x = 0 ; x < 2 ; x++) //Leer ambas subpaginas
  {
    uint16_t mlx90640Frame[834];
    int status = MLX90640_GetFrameData(MLX90640_address,
mlx90640Frame);
    if (status < 0)
    {
      Serial.print("GetFrame Error: ");
      Serial.println(status);
    }

    float vdd = MLX90640_GetVdd(mlx90640Frame, &mlx90640);
    float Ta = MLX90640_GetTa(mlx90640Frame, &mlx90640);

    float tr = Ta - TA_SHIFT; //Temperatura reflejada basada en la
temperatura ambiente del sensor
    float emissivity = 0.95;

    MLX90640_CalculateTo(mlx90640Frame, &mlx90640, emissivity, tr,
mlx90640To);
  }
float PrimerBloque=0;
float SegundoBloque=0;
float TercerBloque=0;
float CuartoBloque=0;
float QuintoBloque=0;
float SextoBloque=0;
float SeptimoBloque=0;
float OctavoBloque=0;

//BUCLES FOR DEL PRIMER BLOQUE
  for (int a=1;a<17;a++)
  {
    PrimerBloque=PrimerBloque+mlx90640To[a];
  }
    for(int b=33;b<49;b++)
    {
      PrimerBloque=PrimerBloque+mlx90640To[b];
    }
      for(int c=65;c<81;c++)
      {
         PrimerBloque=PrimerBloque+mlx90640To[c];
      }
         for(int d=97;d<113;d++)

         {PrimerBloque=PrimerBloque+mlx90640To[d];
         }

         for (int e=129;e<145;e++)

         {PrimerBloque=PrimerBloque+mlx90640To[e];
         }
         for (int f=161;f<177;f++)
         {PrimerBloque=PrimerBloque+mlx90640To[f];
```

```
          }

          PrimerBloque=PrimerBloque/96;
          Serial.print("Bloque 1: ");
          Serial.print(PrimerBloque);
          Serial.println();

//BUCLES FOR DEL SEGUNDO BLOQUE

        for (int a=17;a<33;a++)
  {
    SegundoBloque=SegundoBloque+mlx90640To[a];
  }
    for(int b=49;b<65;b++)
    {
      SegundoBloque=SegundoBloque+mlx90640To[b];
    }
      for(int c=81;c<97;c++)
      {
         SegundoBloque=SegundoBloque+mlx90640To[c];
      }
         for(int d=113;d<129;d++)

         {SegundoBloque=SegundoBloque+mlx90640To[d];
         }

         for (int e=145;e<161;e++)

         {SegundoBloque=SegundoBloque+mlx90640To[e];
         }
         for (int f=177;f<193;f++)
         {SegundoBloque=SegundoBloque+mlx90640To[f];
         }

        SegundoBloque=SegundoBloque/96;
        Serial.print("Bloque 2: ");
        Serial.print(SegundoBloque);
        Serial.println();

    //BUCLES FOR DEL TERCER BLOQUE

   for (int a=193;a<209;a++)
  {
    TercerBloque=TercerBloque+mlx90640To[a];
  }
    for(int b=225;b<241;b++)
    {
      TercerBloque=TercerBloque+mlx90640To[b];
    }
      for(int c=257;c<273;c++)
      {
         TercerBloque=TercerBloque+mlx90640To[c];
      }
         for(int d=289;d<305;d++)

         {TercerBloque=TercerBloque+mlx90640To[d];
         }
```

```
          for (int e=321;e<337;e++)

          {TercerBloque=TercerBloque+mlx90640To[e];
          }
          for (int f=353;f<369;f++)
          {TercerBloque=TercerBloque+mlx90640To[f];
          }

         TercerBloque=TercerBloque/96;
         Serial.print("Bloque 3: ");
         Serial.print(TercerBloque);
         Serial.println();

  //BUCLES FOR DEL CUARTO BLOQUE

   for (int a=209;a<225;a++)
  {
    CuartoBloque=CuartoBloque+mlx90640To[a];
  }
    for(int b=241;b<257;b++)
    {
      CuartoBloque=CuartoBloque+mlx90640To[b];
    }
      for(int c=273;c<289;c++)
      {
         CuartoBloque=CuartoBloque+mlx90640To[c];
      }
         for(int d=305;d<321;d++)

         {CuartoBloque=CuartoBloque+mlx90640To[d];
         }

         for (int e=337;e<353;e++)

         {CuartoBloque=CuartoBloque+mlx90640To[e];
         }
         for (int f=369;f<385;f++)
         {CuartoBloque=CuartoBloque+mlx90640To[f];
         }

        CuartoBloque=CuartoBloque/96;
        Serial.print("Bloque 4: ");
        Serial.print(CuartoBloque);
        Serial.println();

//BUCLES FOR DEL QUINTO BLOQUE
      for (int a=385;a<401;a++)
  {
    QuintoBloque=QuintoBloque+mlx90640To[a];
  }
    for(int b=417;b<433;b++)
    {
      QuintoBloque=QuintoBloque+mlx90640To[b];
    }
      for(int c=449;c<465;c++)
      {
```

```
        QuintoBloque=QuintoBloque+mlx90640To[c];
    }
        for(int d=481;d<497;d++)

        {QuintoBloque=QuintoBloque+mlx90640To[d];
        }

        for (int e=513;e<529;e++)

        {QuintoBloque=QuintoBloque+mlx90640To[e];
        }
        for (int f=545;f<561;f++)
        {QuintoBloque=QuintoBloque+mlx90640To[f];
        }

      QuintoBloque=QuintoBloque/96;
      Serial.print("Bloque 5: ");
      Serial.print(QuintoBloque);
      Serial.println();

      //BUCLES FOR DEL SEXTO BLOQUE
    for (int a=401;a<417;a++)
{
  SextoBloque=SextoBloque+mlx90640To[a];
}
  for(int b=433;b<449;b++)
  {
    SextoBloque=SextoBloque+mlx90640To[b];
  }
    for(int c=465;c<481;c++)
    {
       SextoBloque=SextoBloque+mlx90640To[c];
    }
       for(int d=497;d<513;d++)

       {SextoBloque=SextoBloque+mlx90640To[d];
       }

       for (int e=529;e<545;e++)

       {SextoBloque=SextoBloque+mlx90640To[e];
       }
       for (int f=561;f<577;f++)
       {SextoBloque=SextoBloque+mlx90640To[f];
       }

          SextoBloque=SextoBloque/96;
      Serial.print("Bloque 6: ");
      Serial.print(SextoBloque);
      Serial.println();


       //BUCLES FOR DEL SEPTIMO BLOQUE
    for (int a=577;a<593;a++)
{
  SeptimoBloque=SeptimoBloque+mlx90640To[a];
}
```

```
  for(int b=609;b<625;b++)
  {
    SeptimoBloque=SeptimoBloque+mlx90640To[b];
  }
    for(int c=641;c<657;c++)
    {
       SeptimoBloque=SeptimoBloque+mlx90640To[c];
    }
       for(int d=673;d<689;d++)

       {SeptimoBloque=SeptimoBloque+mlx90640To[d];
       }

       for (int e=705;e<721;e++)

       {SeptimoBloque=SeptimoBloque+mlx90640To[e];
       }
       for (int f=737;f<753;f++)
       {SeptimoBloque=SeptimoBloque+mlx90640To[f];
       }

      SeptimoBloque=SeptimoBloque/96;
      Serial.print("Bloque 7: ");
      Serial.print(SeptimoBloque);
      Serial.println();

       //BUCLES FOR DEL OCTAVO BLOQUE
    for (int a=593;a<609;a++)
{
  OctavoBloque=OctavoBloque+mlx90640To[a];
}
  for(int b=625;b<641;b++)
  {
    OctavoBloque=OctavoBloque+mlx90640To[b];
  }
    for(int c=657;c<673;c++)
    {
       OctavoBloque=OctavoBloque+mlx90640To[c];
    }
       for(int d=689;d<705;d++)

       {OctavoBloque=OctavoBloque+mlx90640To[d];
       }

       for (int e=721;e<737;e++)

       {OctavoBloque=OctavoBloque+mlx90640To[e];
       }
       for (int f=753;f<769;f++)
       {OctavoBloque=OctavoBloque+mlx90640To[f];
       }

      OctavoBloque=OctavoBloque/96;
      Serial.print("Bloque 8: ");
      Serial.print(OctavoBloque);
      Serial.println();
```

```
        SerialBT.print(PrimerBloque);
        SerialBT.print("|");

        SerialBT.print(SegundoBloque);
        SerialBT.print("|");

        SerialBT.print(TercerBloque);
        SerialBT.print("|");

        SerialBT.print(CuartoBloque);
        SerialBT.print("|");
        SerialBT.print(QuintoBloque);
        SerialBT.print("|");
        SerialBT.print(SextoBloque);
        SerialBT.print("|");
        SerialBT.print(SeptimoBloque);
        SerialBT.print("|");
        SerialBT.print(OctavoBloque);
        SerialBT.print("|");

        //establecer las variables que se van a enviar al servidor y a
qué campo va a pertenecer cada una

    ThingSpeak.setField(1,PrimerBloque);
    ThingSpeak.setField(2,SegundoBloque);
    ThingSpeak.setField(3,TercerBloque);
    ThingSpeak.setField(4,CuartoBloque);
    ThingSpeak.setField(5,QuintoBloque);
    ThingSpeak.setField(6,SextoBloque);
    ThingSpeak.setField(7,SeptimoBloque);
    ThingSpeak.setField(8,OctavoBloque);

   ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);


  delay(21000);
}

//Devuelve verdadero si el MLX90640 se detecta en el bus I2C.

boolean isConnected()
{
  Wire.beginTransmission((uint8_t)MLX90640_address);
  if (Wire.endTransmission() != 0)
    return (false); //Sensor no actua
  return (true);
}
```

8.2. Código de matlab

PANTALLA PRINCIPAL

```
funcgui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @PRINCIPAL_OpeningFcn, ...
                   'gui_OutputFcn',  @PRINCIPAL_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State,
varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before PRINCIPAL is made visible.
function PRINCIPAL_OpeningFcn(hObject, eventdata, handles,
varargin)
%
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
global temporizador pixel matriz;
temporizador=timer;
temporizador.Period=21; %Se establece el periodo
temporizador.ExecutionMode='fixedRate'; % Modo de trabajo
temporizador.TimerFcn={@Timer,handles};
pixel(1:8)=0;
matriz(1:2,1:4)=0;
handles.plt_array.YDir = 'reverse';
handles.plt_array.XAxisLocation = 'top';
colormap(handles.plt_array,jet(256));
caxis(handles.plt_array,[0 50]);
colorbar(handles.plt_array);



% --- Outputs from this function are returned to the command
line.
function varargout = PRINCIPAL_OutputFcn(hObject, eventdata,
handles)
```

```
52.
53. varargout{1} = handles.output;
54.
55. function Timer(hObject, eventdata, handles, varargin)
56. global pixel Tmax Tmin matriz;
57.
58. readChannelID = 1127965;
59.
60.
61. Pixel1Field = 1;
62. Pixel2Field = 2;
63. Pixel3Field = 3;
64. Pixel4Field = 4;
65. Pixel5Field = 5;
66. Pixel6Field = 6;
67. Pixel7Field = 7;
68. Pixel8Field = 8;
69.
70.
71. readAPIKey = '';
72.
73.
74.
75. [ultdata] = thingSpeakRead(readChannelID, 'Fields',[Pixel1Field
    Pixel2Field Pixel3Field Pixel4Field Pixel5Field Pixel6Field
    Pixel7Field Pixel8Field], ...
76.
    'NumPoints', 1, ...
77.
    'ReadKey', readAPIKey);
78.
79. ultimoPixel1 = ultdata(:, 1);
80. set(handles.text2,'String',ultimoPixel1)
81. ultimoPixel2 = ultdata(:, 2);
82. set(handles.text3,'String',ultimoPixel2)
83. ultimoPixel3 = ultdata(:, 3);
84. set(handles.text4,'String',ultimoPixel3)
85. ultimoPixel4 = ultdata(:, 4);
86. set(handles.text5,'String',ultimoPixel4)
87. ultimoPixel5 = ultdata(:, 5);
88. set(handles.text6,'String',ultimoPixel5)
89. ultimoPixel6 = ultdata(:, 6);
90. set(handles.text7,'String',ultimoPixel6)
91. ultimoPixel7 = ultdata(:, 7);
92. set(handles.text8,'String',ultimoPixel7)
93. ultimoPixel8 = ultdata(:, 8);
94. set(handles.text9,'String',ultimoPixel8)
95.
96.
97. for i=1:8
98.     pixel(i)=ultdata(:, i);
```

```
end

    Tmax=max(pixel);
    Tmin=min(pixel);
    set(handles.text24,'String',max(pixel));
set(handles.text25,'String',min(pixel));

    A=reshape(pixel,[2,4])';
    matriz=A;


    image(handles.plt_array,0.5,0.5,matriz,'CDataMapping','scal
ed');

    handles.plt_array.YDir = 'reverse';
    handles.plt_array.XAxisLocation = 'top';
    colormap(handles.plt_array,jet(45));
    caxis(handles.plt_array,[0 50]);
    colorbar(handles.plt_array);tion varargout =
PRINCIPAL(varargin)




    %CONSTRUIR MATRIZ CON ULTIMO PIXEL%


    % --- Executes on selection change in listbox1.
    function listbox1_Callback(hObject, eventdata, handles)
    % hObject    handle to listbox1 (see GCBO)
    % eventdata  reserved - to be defined in a future version
of MATLAB
    % handles    structure with handles and user data (see
GUIDATA)

    % Hints: contents = cellstr(get(hObject,'String')) returns
listbox1 contents as cell plt_array
    %        contents{get(hObject,'Value')} returns selected
item from listbox1


    % --- Executes during object creation, after setting all
properties.
    function listbox1_CreateFcn(hObject, eventdata, handles)
    % hObject    handle to listbox1 (see GCBO)
    % eventdata  reserved - to be defined in a future version
of MATLAB
    % handles    empty - handles not created until after all
CreateFcns called

```

```
140.      % Hint: listbox controls usually have a white background on
    Windows.
141.      %       See ISPC and COMPUTER.
142.      if ispc && isequal(get(hObject,'BackgroundColor'),
    get(0,'defaultUicontrolBackgroundColor'))
143.          set(hObject,'BackgroundColor','white');
144.      end
145.
146.
147.      % --- Executes on selection change in popupmenu1.
148.      function popupmenu1_Callback(hObject, eventdata, handles)
149.
150.      global V1;
151.
152.      V1=get(hObject,'Value');
153.
154.
155.
156.
157.      function popupmenu1_CreateFcn(hObject, eventdata, handles)
158.
159.      if ispc && isequal(get(hObject,'BackgroundColor'),
    get(0,'defaultUicontrolBackgroundColor'))
160.          set(hObject,'BackgroundColor','white');
161.      end
162.
163.
164.
165.
166.      function togglebutton2_Callback(hObject, eventdata,
    handles)
167.
168.      global V1;
169.      % Hint: get(hObject,'Value') returns toggle state of
    togglebutton2
170.      isDown = get(hObject,'Value');
171.      if isDown
172.          switch V1
173.             case 1
174.                INTERFAZ;
175.
176.
177.             case 2
178.                 PARAMETROS;
179.
180.
181.             case 3
182.                  ALARMAS;
183.
184.
185.         end
```

```
186.
187.
188.      end
189.
190.
191.      % --- Executes on button press in pushbutton4.
192.      function pushbutton4_Callback(hObject, eventdata, handles)
193.      % hObject    handle to pushbutton4 (see GCBO)
194.      % eventdata  reserved - to be defined in a future version
   of MATLAB
195.      % handles    structure with handles and user data (see
   GUIDATA)
196.
197.
198.      % --- Executes on button press in togglebutton3.
199.      function togglebutton3_Callback(hObject, eventdata,
   handles)
200.
201.      global temporizador;
202.      isDown = get(hObject,'Value');
203.
204.      if isDown
205.          set(handles.togglebutton3, 'string', 'PARO');
206.          set(handles.togglebutton3,'BackgroundColor','red');
207.
208.          start(temporizador);
209.
210.      else
211.          set(handles.togglebutton3, 'string', 'INICIO');
212.          set(handles.togglebutton3,'BackgroundColor','green');
213.          set(handles.text2,'string', ' ');
   set(handles.text3,'string', ' '); set(handles.text4,'string', '
   ');
214.          set(handles.text5,'string', ' ');
   set(handles.text6,'string', ' '); set(handles.text7,'string', '
   ');
215.          set(handles.text8,'string', ' ');
   set(handles.text9,'string', ' ');
216.          set(handles.text24,'String',' ');
   set(handles.text25,'String',' ');
217.
218.
219.        cla(handles.plt_array);
220.          stop(temporizador);
221.
222.      end
223.
224.
225.      % ------------------------------------------------------
   -----------
226.      function Untitled_1_Callback(hObject, eventdata, handles)
227.      % hObject    handle to Untitled_1 (see GCBO)
```

```
228.      % eventdata  reserved - to be defined in a future version
    of MATLAB
229.      % handles    structure with handles and user data (see
    GUIDATA)
```

PANTALLA DE GRÁFICAS

```
function varargout = INTERFAZ(varargin)

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @INTERFAZ_OpeningFcn, ...
                   'gui_OutputFcn',  @INTERFAZ_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before INTERFAZ is made visible.
function INTERFAZ_OpeningFcn(hObject, eventdata, handles, varargin)

global Temporizador

Temporizador=timer; %Se crea el objeto timer
Temporizador.Period=21; %Se establece el periodo
Temporizador.ExecutionMode='fixedRate'; % Modo de trabajo
Temporizador.TimerFcn={@Timer,handles}; % Nombre función
 %Se inicia el temporizador

% Choose default command line output for INTERFAZ
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes INTERFAZ wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = INTERFAZ_OutputFcn(hObject, eventdata, handles)
```

```
% Get default command line output from handles structure
varargout{1} = handles.output;

function Timer(hObject, eventdata, handles, varargin)
readChannelID = 1127965;
% Temperature Field ID
Pixel1Field = 1;
Pixel2Field = 2;
Pixel3Field = 3;
Pixel4Field = 4;
Pixel5Field = 5;
Pixel6Field = 6;
Pixel7Field = 7;
Pixel8Field = 8;



readAPIKey = '';


[data, timeStamps1] = thingSpeakRead(readChannelID,
'Fields',[Pixel1Field Pixel2Field Pixel3Field Pixel4Field Pixel5Field
Pixel6Field Pixel7Field Pixel8Field], ...

'NumPoints', 10, ...
                                                    'ReadKey',
readAPIKey);


[ultdata, timeStamps2] = thingSpeakRead(readChannelID,
'Fields',[Pixel1Field Pixel2Field Pixel3Field Pixel4Field Pixel5Field
Pixel6Field Pixel7Field Pixel8Field], ...

'NumPoints', 1, ...
                                                    'ReadKey',
readAPIKey);

% Extraer el valor de temperatura de la columna especificada en N -->
(:, N)

TemperaturaPixel1 = data(:, 1);
TemperaturaPixel2 = data(:, 2);
TemperaturaPixel3 = data(:, 3);
TemperaturaPixel4 = data(:, 4);
TemperaturaPixel5 = data(:, 5);
TemperaturaPixel6 = data(:, 6);
TemperaturaPixel7 = data(:, 7);
TemperaturaPixel8 = data(:, 8);


ultimoPixel1 = ultdata(:, 1);
ultimoPixel2 = ultdata(:, 2);
```

```
ultimoPixel3 = ultdata(:, 3);
ultimoPixel4 = ultdata(:, 4);
ultimoPixel5 = ultdata(:, 5);
ultimoPixel6 = ultdata(:, 6);
ultimoPixel7 = ultdata(:, 7);
ultimoPixel8 = ultdata(:, 8);
% Visualize Data



ax1 = subplot(handles.axes2);
x = timeStamps1;
y1 = TemperaturaPixel1;
plot(ax1,x,y1,'g');
title(ax1,'GRÁFICA 1')
ylabel(ax1,'Temperatura Pixel 1')


ax2 = subplot(handles.axes4);
x = timeStamps1;
y2 = TemperaturaPixel2;
plot(ax2,x,y2,'r');
title(ax2,'GRÁFICA 2')
ylabel(ax2,'Temperatura Pixel 2')

ax3 = subplot(handles.axes5);
x = timeStamps1;
y3 = TemperaturaPixel3;
plot(ax3,x,y3,'b');
title(ax3,'GRÁFICA 3')
ylabel(ax3,'Temperatura Pixel 3')


ax4 = subplot(handles.axes6);
x = timeStamps1;
y4 = TemperaturaPixel4;
plot(ax4,x,y4,'y');
title(ax4,'GRÁFICA 4')
ylabel(ax4,'Temperatura Pixel 4')


ax5 = subplot(handles.axes7);
x = timeStamps1;
y5 = TemperaturaPixel5;
plot(ax5,x,y5,'b');
title(ax5,'GRÁFICA 5')
ylabel(ax5,'Temperatura Pixel 5')


ax6 = subplot(handles.axes8);
x = timeStamps1;
y6 = TemperaturaPixel6;
plot(ax6,x,y6,'g');
title(ax6,'GRÁFICA 6')
ylabel(ax6,'Temperatura Pixel 6')
```

```
ax7 = subplot(handles.axes9);
x = timeStamps1;
y7 = TemperaturaPixel7;
plot(ax7,x,y7,'r');
title(ax7,'GRÁFICA 7')
ylabel(ax7,'Temperatura Pixel 7')


ax8 = subplot(handles.axes10);
x = timeStamps1;
y8 = TemperaturaPixel8;
plot(ax8,x,y8,'y');
title(ax8,'GRÁFICA 8')
ylabel(ax8,'Temperatura Pixel 8')


% --- Executes on button press in togglebutton1.
function togglebutton1_Callback(hObject, eventdata, handles)
global Temporizador;
isDown = get(hObject,'Value');

if isDown
    set(handles.togglebutton1, 'string', 'PARO');
    set(handles.togglebutton1,'BackgroundColor','red');

    set(handles.text19,'String','EVALUACIÓN INICIADA');
    set(handles.text19,'ForegroundColor','green');
    start(Temporizador);

else
    set(handles.togglebutton1, 'string', 'INICIO');
    set(handles.togglebutton1,'BackgroundColor','green');

     set(handles.text19,'String','EVALUACIÓN DETENIDA');
    set(handles.text19,'ForegroundColor','red');
    stop(Temporizador);
    %delete(Temporizador);
    %clear Temporizador;

    cla(handles.axes4);
    cla(handles.axes2);
    cla(handles.axes5);
    cla(handles.axes6);
    cla(handles.axes7);
    cla(handles.axes8);
    cla(handles.axes9);
    cla(handles.axes10);

end
% Hint: get(hObject,'Value') returns toggle state of togglebutton1
```

```
% --- Executes on button press in togglebutton2.



% --- Executes when user attempts to close figure1.
function figure1_CloseRequestFcn(hObject, eventdata, handles)
global Temporizador;

stop(Temporizador);
delete(Temporizador);
clear Temporizador;
% Hint: delete(hObject) closes the figure
delete(hObject);


% --- Executes on button press in togglebutton3.
function togglebutton3_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of togglebutton3


% --- Executes on button press in togglebutton4.
function togglebutton4_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of togglebutton4



function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1
as a double


% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)


% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
```

```
end



function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit2 as text
%        str2double(get(hObject,'String')) returns contents of edit2
as a double


% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)

%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit3_Callback(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit3 as text
%        str2double(get(hObject,'String')) returns contents of edit3
as a double


% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit4_Callback(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit4 as text
%        str2double(get(hObject,'String')) returns contents of edit4
as a double


% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit5_Callback(hObject, eventdata, handles)


% Hints: get(hObject,'String') returns contents of edit5 as text
%        str2double(get(hObject,'String')) returns contents of edit5
as a double


% --- Executes during object creation, after setting all properties.
function edit5_CreateFcn(hObject, eventdata, handles)

%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit6_Callback(hObject, eventdata, handles)


% Hints: get(hObject,'String') returns contents of edit6 as text
%        str2double(get(hObject,'String')) returns contents of edit6
as a double


% --- Executes during object creation, after setting all properties.
function edit6_CreateFcn(hObject, eventdata, handles)
```

```
.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit7_Callback(hObject, eventdata, handles)




function edit7_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit8_Callback(hObject, eventdata, handles)



% --- Executes during object creation, after setting all properties.
function edit8_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)



% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --------------------------------------------------------------------
function Untitled_1_Callback(hObject, eventdata, handles)
```

```
% --- Executes during object creation, after setting all properties.
function axes2_CreateFcn(hObject, eventdata, handles)
```

PANTALLA DE PARÁMETROS

```
function varargout = PARAMETROS(varargin)

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @PARAMETROS_OpeningFcn, ...
                   'gui_OutputFcn',  @PARAMETROS_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before PARAMETROS is made visible.
function PARAMETROS_OpeningFcn(hObject, eventdata, handles, varargin)

global temp1;
temp1=timer;
temp1.Period=21; %Se establece el periodo
temp1.ExecutionMode='fixedRate'; % Modo de trabajo
temp1.TimerFcn={@Timer,handles};
% Choose default command line output for PARAMETROS
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);


% --- Outputs from this function are returned to the command line.
function varargout = PARAMETROS_OutputFcn(hObject, eventdata, handles)

varargout{1} = handles.output;

function Timer(hObject, eventdata, handles, varargin)
global pixel ContReport;

readChannelID = 1127965;
```

```
channelID=1038516;
% Temperature Field ID
Pixel1Field = 1;
Pixel2Field = 2;
Pixel3Field = 3;
Pixel4Field = 4;
Pixel5Field = 5;
Pixel6Field = 6;
Pixel7Field = 7;
Pixel8Field = 8;



readAPIKey = '';
writeKey = 'N1CC9LSE0PEAPADL';



[ultdata] = thingSpeakRead(readChannelID, 'Fields',[Pixel1Field
Pixel2Field Pixel3Field Pixel4Field Pixel5Field Pixel6Field
Pixel7Field Pixel8Field], ...

'NumPoints', 1, ...
                                                    'ReadKey',
readAPIKey);
Max=get(handles.edit1,'String');
Min=get(handles.edit2,'String');
Maximo=str2double(Max);
Minimo=str2double(Min);
for i=1:8
    pixel(i)=ultdata(:, i);

end

Tmax=max(pixel);
Tmin=min(pixel);



if Tmax>=Maximo

       set(handles.text11,'string','ALARMA MAX');

thingSpeakWrite(channelID,'Fields',[1],'Values',{1},'WriteKey',writeKe
y);


else
    if Tmin<=Minimo
```

```
                    set(handles.text11,'string','ALARMA MIN');

thingSpeakWrite(channelID,'Fields',[1],'Values',{2},'WriteKey',writeKe
y);
    else
    set(handles.text11,'string',' ');

thingSpeakWrite(channelID,'Fields',[1],'Values',{0},'WriteKey',writeKe
y);
        end


end


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)

global temp1;

isDown = get(hObject,'Value');
if isDown
    start(temp1);

else
    stop(temp1);

end

function edit1_Callback(hObject, eventdata, handles)



% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit2_Callback(hObject, eventdata, handles)



% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
%
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end


% --- Executes when user attempts to close figure1.
function figure1_CloseRequestFcn(hObject, eventdata, handles)

global temp1;

stop(temp1);
delete(temp1);
% Hint: delete(hObject) closes the figure
delete(hObject);


% --- Executes on button press in togglebutton1.
function togglebutton1_Callback(hObject, eventdata, handles)

global temp1;

isDown = get(hObject,'Value');
if isDown
    start(temp1);

else
    stop(temp1);

end
% Hint: get(hObject,'Value') returns toggle state of togglebutton1
```

PANTALLA DE ALARMAS

```
function varargout = ALARMAS(varargin)

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @ALARMAS_OpeningFcn, ...
                   'gui_OutputFcn',  @ALARMAS_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
```

```
% --- Executes just before ALARMAS is made visible.
function ALARMAS_OpeningFcn(hObject, eventdata, handles, varargin)

global temporizador contador;

contador=0;
temporizador=timer;
temporizador.Period=21; %Se establece el periodo
temporizador.ExecutionMode='fixedRate'; % Modo de trabajo
temporizador.TimerFcn={@Timer,handles};
% Choose default command line output for ALARMAS
handles.output = hObject;
start(temporizador);
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes ALARMAS wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = ALARMAS_OutputFcn(hObject, eventdata, handles)

varargout{1} = handles.output;

function Timer(hObject, eventdata, handles, varargin)
global contador;

readChannelID=1038516;
readAPIKey='';
[alarma timeStamps1] = thingSpeakRead(readChannelID, 'Fields',[1], ...

'NumPoints', 1, ...
                                                    'ReadKey',
readAPIKey);


alarm=alarma(:, 1);

 switch alarm

     case 0
         contador=contador;

     case 1
         contador=contador+1;
         get(handles.uitable2,'data');

        a={'SE HA SUPERADO LA TEMPERATURA MÁXIMA ESTABLECIDA'};
         b=cellstr(timeStamps1);
         data = [a b];
         handles.uitable2.Data(contador,1)=a;
         handles.uitable2.Data(contador,2)=b;
```

```
        case 2
            contador=contador+1;

            get(handles.uitable2,'data');

           a={'SE HA SUPERADO LA TEMPERATURA MÍNIMA ESTABLECIDA'};
            b=cellstr(timeStamps1);
            data = [a b];
            handles.uitable2.Data(contador,1)=a;
            handles.uitable2.Data(contador,2)=b;
    end



% --- Executes on button press in togglebutton1.


% --- Executes when user attempts to close figure1.
function figure1_CloseRequestFcn(hObject, eventdata, handles)

delete(hObject);
```

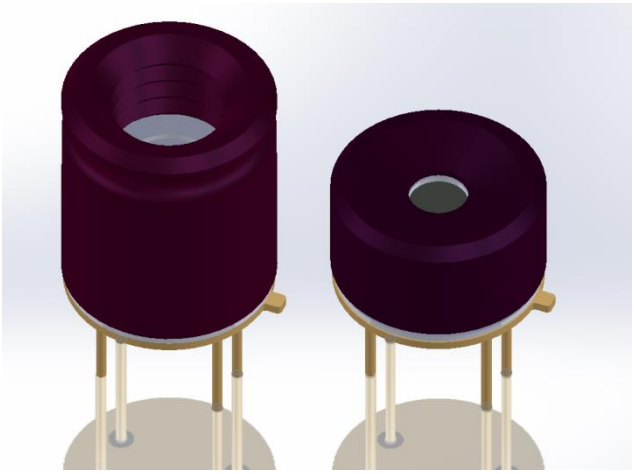
8.3. Hoja de datos del SPARKFUN MLX90640

MLX90640 32x24 IR array

Datasheet

1. Features and Benefits

- Small size, low cost 32x24 pixels IR array
- Easy to integrate
- Industry standard four lead TO39 package
- Factory calibrated
- Noise Equivalent Temperature Difference (NETD) 0.1K RMS @1Hz refresh rate
- I^2C compatible digital interface
- Programmable refresh rate 0.5Hz…64Hz
- 3.3V supply voltage
- Current consumption less than 23mA
- 2 FOV options – 55°x35° and 110°x75°
- Operating temperature -40°C ÷ 85°C
- Target temperature -40°C ÷ 300°C
- Complies with RoHS regulations



2. Application Examples

- High precision non-contact temperature measurements
- Intrusion / Movement detection
- Presence detection / Person localization
- Temperature sensing element for intelligent building air conditioning
- Thermal Comfort sensor in automotive Air Conditioning control system
- Microwave ovens
- Industrial temperature control of moving parts
- Visual IR thermometers
- Driver software for MCU available at: https://github.com/melexis/mlx90640-library.git

3. Description

The MLX90640 is a fully calibrated 32x24 pixels thermal IR array in an industry standard 4-lead TO39 package with digital interface.

The MLX90640 contains 768 FIR pixels. An ambient sensor is integrated to measure the ambient temperature of the chip and supply sensor to measure the VDD. The outputs of all sensors IR, Ta and VDD are stored in internal RAM and are accessible through I^2C.

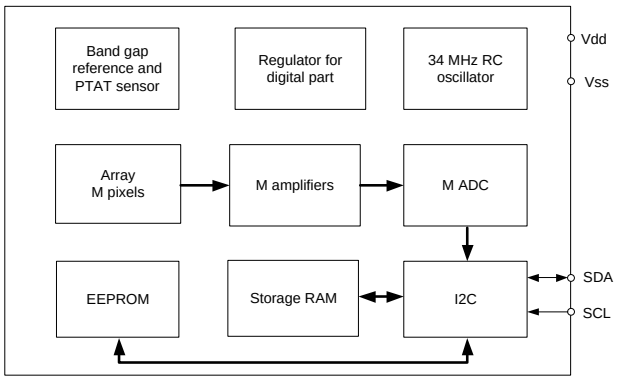


Figure 1 Block diagram

Contents

Tables

Figures

4. Ordering Information

Product	Temperature	Package	Option Code	Custom Configuration	Packing Form	Definition
MLX90640	E	SF	BAA	000	TU	32x24 IR array
MLX90640	E	SF	BAB	000	TU	32x24 IR array

Legend:

Temperature Code:	**E: -40°C to 85°C**
Package Code:	"SF" for TO39 package
Option Code:	xAx – TGC is disabled and may not be changed
Option Code:	xxA – FOV = 110°x75° xxB – FOV = 55°x35°
Custom configuration	000 – standard product
Packing Form:	"TU" - Tubes
Ordering Example:	"MLX90640ESF-BAA-000-TU"

Table 1 Ordering information

5. Glossary of Terms

TC	**T**emperature **C**oefficient (in ppm/°C)
POR	**P**ower **O**n **R**eset
IR	**I**nfra-**R**ed
Ta	**A**mbient **T**emperature – the temperature of the TO39 package
IR data	**I**nfrared **d**ata (raw data from ADC proportional to IR energy received by the sensor)
ADC	**A**nalog To **D**igital **C**onverter
TGC	**T**emperature **G**radient **C**oefficient
FOV	**F**ield **O**f **V**iew
nFOV	**F**ield **O**f **V**iew of the N-th pixel
I^2C	**I**nter-**I**ntegrated **C**ircuit communication protocol
SDA	**S**erial **Da**ta
SCL	**S**erial **Cl**ock
LSB	**L**east **S**ignificant **B**it
MSB	**M**ost **S**ignificant **B**it
Fps	**F**rames **p**er **S**econd – data refresh rate
MD	**M**aster **D**evice
SD	**S**lave **D**evice
ASP	**A**nalog **S**ignal **P**rocessing
DSP	**D**igital **S**ignal **P**rocessing
ESD	**E**lectro **S**tatic **D**ischarge
EMC	**E**lectro **M**agnetic **C**ompatibility
CP	**C**ompensation **P**ixel
NC	**N**ot **C**onnected
NA	**N**ot **A**pplicable
TBD	**T**o **B**e **D**efined

Table 2 Glosarry of terms

6. Pin Definitions and Descriptions

Pin #	Name	Description
1	SDA	I^2C serial data (input / output)
2	VDD	Positive supply
3	GND	Negative supply (Ground)
4	SCL	I^2C serial clock (input only)

Table 3 Pin definition

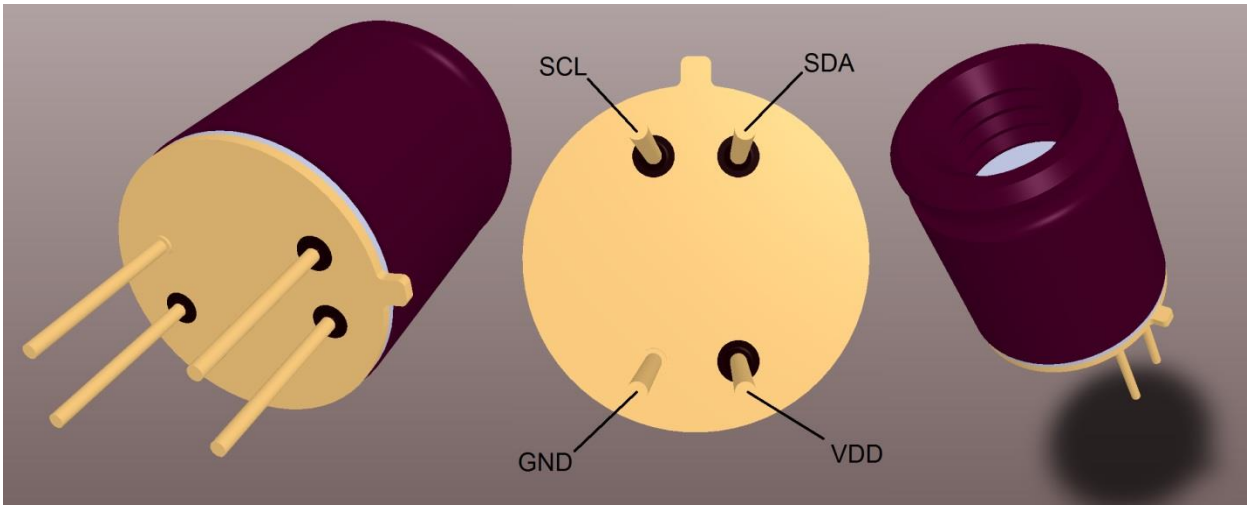


Figure 2 MLX90640 Overview and pin description

7. Absolute Maximum Ratings

Parameter	Symbol	Min.	Typ.	Max.	Unit	Remark
Supply Voltage (over voltage)	V_{DD}			5	V	
Supply Voltage (operating max voltage)	V_{DD}			3.6		
Reverse Voltage (each pin)				-0.3	V	
Operating Temperature	T_{AMB}	-40		+85	°C	
Storage Temperature	T_{ST}	-40		+85	°C	Not in plastic tubes
ESD sensitivity (AEC Q100 002)		4			kV	
SDA DC sink current				40	mA	

Table 4 Absolute maximum ratings

Exceeding the absolute maximum ratings may cause permanent damage. Exposure to absolute maximum-rated conditions for extended periods may affect device reliability.

8. General Electrical Specifications

Electrical Parameter	Symbol	Min.	Typ.	Max.	Unit	Condition
Supply Voltage	V_{DD}	3	3.3	3.6	V	
Supply Current	I_{DD}	15	20	25	mA	
POR level up analog	V_{POR_UP}	2.2		2.6	V	VDD rising
POR level down analog	V_{POR_DOWN}			2.55	V	VDD falling
POR hysteresis	V_{POR_hys}		50		mV	
I^2C address(NOTE 3)		0x01	0x33(default)	0xFF		
Input high voltage (SDA, SCL)	V_{IH}	0.7*V_{DD}			V	Over Ta and V_{DD}
Input low voltage (SDA, SCL)	V_{LOW}			0.3*V_{DD}	V	Over Ta and V_{DD}
SDA output low voltage	V_{OL}			0.4	V	Over Ta and V_{DD} I_{SINK}=3mA
SDA leakage	I_{SDA_leak}			± 10	µA	V_{SDA}=3.6V, Ta=85°C
SCL leakage	I_{SCL_leak}			± 10	µA	V_{SCL}=3.6V, Ta=85°C
SDA capacitance	C_{SDA}			10	pF	
SCL capacitance	C_{SCL}			10	pF	
Acknowledge setup time	$T_{SUAC(MD)}$			0.45	µs	
Acknowledge hold time	$T_{DUAC(MD)}$			0.45	µs	
Acknowledge setup time	$T_{SUAC(SD)}$			0.45	µs	
Acknowledge hold time	$T_{DUAC(SD)}$			0.45	µs	
I^2C clock frequency	F_{I2C}		0.4	1	MHz	
EEPROM erase/write cycles				10	times	
Write cell time	T_{WRITE}	5			ms	

Table 5 Electrical specification

***NOTE 1**: For best performance it is recommended to keep the supply voltage as accurate and stable as possible to 3.3V ± 0.05V*

***NOTE 2**: When a data in EEPROM cell to be changed an erase (write 0x0000) must be done prior to writing the new value. After each write at least 5ms delay is needed in order to writing process to take place.*

***NOTE 3**: Slave address 0x00 must be avoided.*

***NOTE 4**: According to I^2C standard the max sink current is specified to be 20mA, however due to the thermal considerations (the dissipated power into the driver) the max current is limited to 10mA. This is the only parameter which does not comply with the FM+ specification.*

***NOTE 5**: Max EEPROM I^2C speed operations to be done at 400kHz*

9. False pixel correction

The imager can have up to 4 defective pixels, with either no output or out of specification temperature reading. These pixels are identified in the EEPROM table of the sensor and can be read out through the I^2C. The defective pixel result can be replaced by an interpolation of its neighboring pixels.

10. Detailed General Description

10.1. Pixel position

The array consists of 768 IR sensors (also called pixels). Each pixel is identified with its row and column position as Pix(*i*,*j*) where ***i*** is its row number (from 1 to 24) and ***j*** is its column number (from 1 to 32)

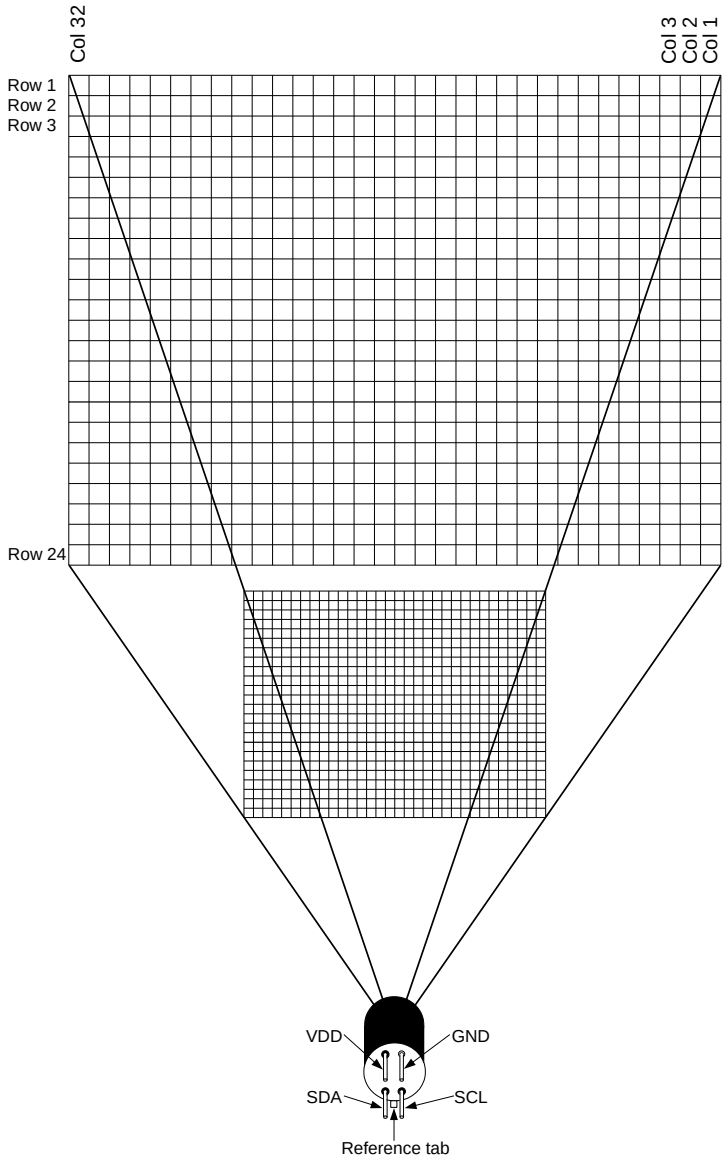


Figure 3 Pixel in the whole FOV

10.2. Communication protocol

The device use I²C protocol with support of FM+ mode (up to 1MHz clock frequency) and can be only slave on the bus. The SDA and SCL ports are 5V tolerant and the sensor can be directly connected to a 5V I²C network.
The slave address is programmable and can have up to 127 different slave addresses.

10.2.1. Low level

10.2.1.1. Start / Stop conditions

Each communication session is initiated by a START condition and ends with a STOP condition. A START condition is initiated by a HIGH to LOW transition of the SDA while a STOP is generated by a LOW to HIGH transition. Both changes must be done while the SCL is HIGH.

10.2.1.2. Device addressing

The master is addressing the slave device by sending a 7-bit slave address after the START condition. The first seven bits are dedicated for the address and the 8th is Read/Write (R/W) bit. This bit indicates the direction of the transfer:

- Read (HIGH) means that the master will read the data from the slave
- Write (LOW) means that the master will send data to the slave

10.2.1.3. Acknowledge

During the 9th clock following every byte transfer the transmitter releases the SDA line. The receiver acknowledges (ACK) receiving the byte by pulling SDA line to low or does not acknowledge (NoACK) by letting the SDA 'HIGH'.

10.2.1.4. I²C command format

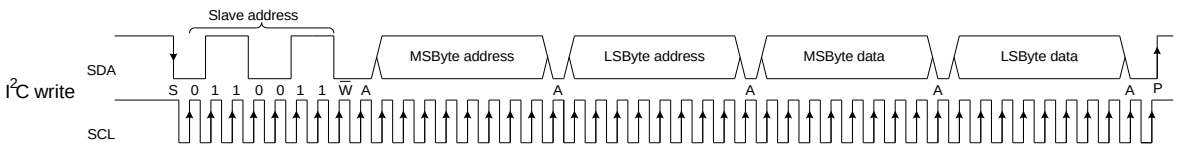


Figure 4 I²C write command format (default SA=0x33 is used)

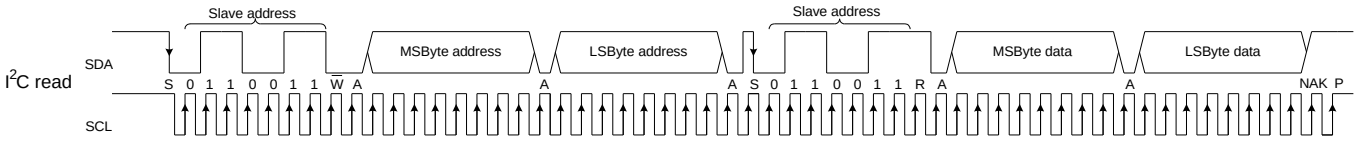


Figure 5 I²C read command format (default SA=0x33 is used)

10.3. Measurement mode

In this mode the measurements are constantly running. Depending on the selected frame rate Fps in the control register, the data for IR pixels and Ta will be updated in the RAM each $\frac{1}{F_{ps}}$ second. In this mode the external microcontroller has full access to the internal registers and memories of the device.

10.4. Refresh rate

The refresh rate is configured by "Control register 1" (0x800D) i.e. if "Refresh rate control" = 011 → 4Hz this would mean that each 250ms a new subpage data is available in the RAM.

NOTE: It is possible to program the desired refresh rate into device EEPROM eliminating the necessity to reconfigure the device every time it is powered on. The corresponding EEPROM cell is at address 0x240C (see Table 8)

Which subpage is updated is indicated by the "Last measured subpage" field.

It is important to read both subpages as the necessary information for the Ta calculations is only available by combining the data from both subpages i.e. the Ta is refreshed with an update speed twice as low as the one set in "Refresh rate control".

When a complete new data set (subpage) is available, a dedicated bit is set to indicate this – bit 3 "New data available in RAM" in "Status register" (0x8000). It is up to the customer to reset the bit once the data has been read.

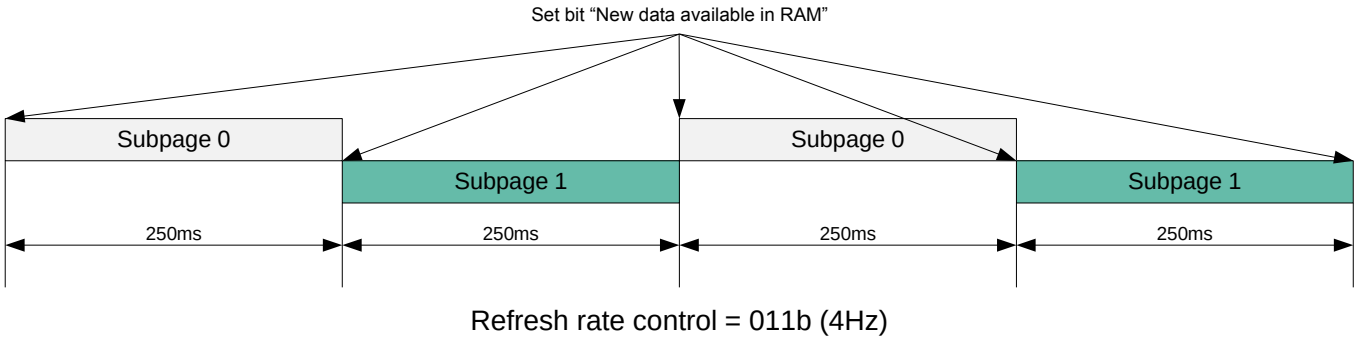


Figure 6 Refresh rate timing

10.5. Measurement flow

Following measurement flow is recommended:

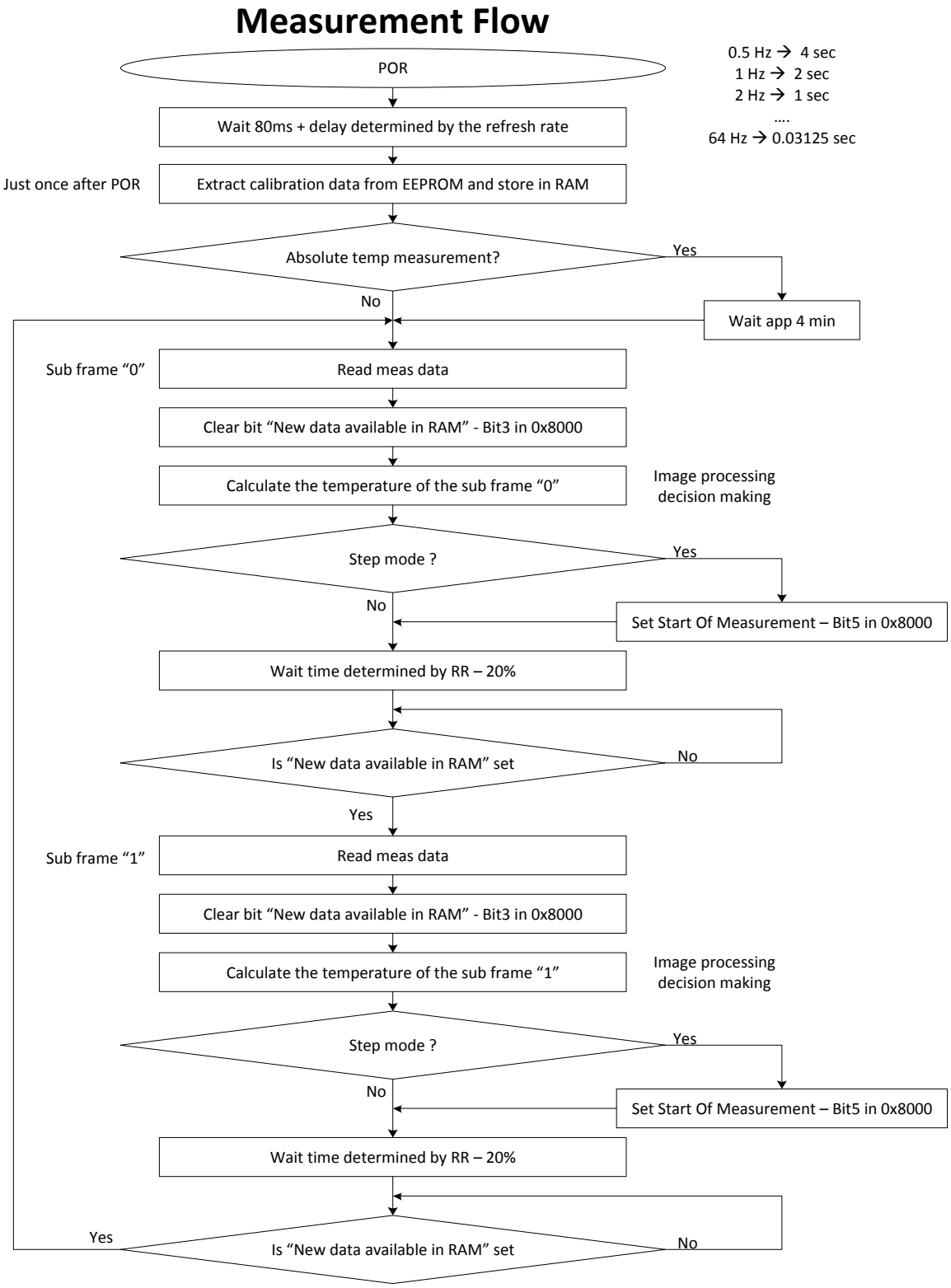


Figure 7 Recommended measurement flow

10.6. Reading patterns

The array frame is divided in two subpages and depending of bit 12 in “Control register 1” (0x800D) – “Reading pattern” there are two modes of the pixel arrangement:

- Chess pattern mode (factory default)
- TV interleave mode

NOTE1: As a standard the MLX90640 is calibrated in Chess pattern mode, this results in better fixed pattern noise behaviour of the sensor when in chess pattern mode. For best results Melexis advices to use chess pattern mode.

NOTE2: Please make sure a proper configuration of the subpage control bit is done. See: Table 6 Priorities of subpage controls

Subpage 0 --> 0x8000 = 0xXXX8

0x0400	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
0x0420	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
0x0440	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
0x0460	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128
0x0480	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160
0x04A0	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192
0x04C0	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224
0x04E0	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256
0x0500	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271	272	273	274	275	276	277	278	279	280	281	282	283	284	285	286	287	288
0x0520	289	290	291	292	293	294	295	296	297	298	299	300	301	302	303	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319	320
0x0540	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335	336	337	338	339	340	341	342	343	344	345	346	347	348	349	350	351	352
0x0560	353	354	355	356	357	358	359	360	361	362	363	364	365	366	367	368	369	370	371	372	373	374	375	376	377	378	379	380	381	382	383	384
0x0580	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399	400	401	402	403	404	405	406	407	408	409	410	411	412	413	414	415	416
0x05A0	417	418	419	420	421	422	423	424	425	426	427	428	429	430	431	432	433	434	435	436	437	438	439	440	441	442	443	444	445	446	447	448
0x05C0	449	450	451	452	453	454	455	456	457	458	459	460	461	462	463	464	465	466	467	468	469	470	471	472	473	474	475	476	477	478	479	480
0x05E0	481	482	483	484	485	486	487	488	489	490	491	492	493	494	495	496	497	498	499	500	501	502	503	504	505	506	507	508	509	510	511	512
0x0600	513	514	515	516	517	518	519	520	521	522	523	524	525	526	527	528	529	530	531	532	533	534	535	536	537	538	539	540	541	542	543	544
0x0620	545	546	547	548	549	550	551	552	553	554	555	556	557	558	559	560	561	562	563	564	565	566	567	568	569	570	571	572	573	574	575	576
0x0640	577	578	579	580	581	582	583	584	585	586	587	588	589	590	591	592	593	594	595	596	597	598	599	600	601	602	603	604	605	606	607	608
0x0660	609	610	611	612	613	614	615	616	617	618	619	620	621	622	623	624	625	626	627	628	629	630	631	632	633	634	635	636	637	638	639	640
0x0680	641	642	643	644	645	646	647	648	649	650	651	652	653	654	655	656	657	658	659	660	661	662	663	664	665	666	667	668	669	670	671	672
0x06A0	673	674	675	676	677	678	679	680	681	682	683	684	685	686	687	688	689	690	691	692	693	694	695	696	697	698	699	700	701	702	703	704
0x06C0	705	706	707	708	709	710	711	712	713	714	715	716	717	718	719	720	721	722	723	724	725	726	727	728	729	730	731	732	733	734	735	736
0x06E0	737	738	739	740	741	742	743	744	745	746	747	748	749	750	751	752	753	754	755	756	757	758	759	760	761	762	763	764	765	766	767	768

Subpage 1 --> 0x8000 = 0xXXX9

0x0400	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
0x0420	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
0x0440	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
0x0460	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128
0x0480	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160
0x04A0	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192
0x04C0	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224
0x04E0	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256
0x0500	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271	272	273	274	275	276	277	278	279	280	281	282	283	284	285	286	287	288
0x0520	289	290	291	292	293	294	295	296	297	298	299	300	301	302	303	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319	320
0x0540	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335	336	337	338	339	340	341	342	343	344	345	346	347	348	349	350	351	352
0x0560	353	354	355	356	357	358	359	360	361	362	363	364	365	366	367	368	369	370	371	372	373	374	375	376	377	378	379	380	381	382	383	384
0x0580	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399	400	401	402	403	404	405	406	407	408	409	410	411	412	413	414	415	416
0x05A0	417	418	419	420	421	422	423	424	425	426	427	428	429	430	431	432	433	434	435	436	437	438	439	440	441	442	443	444	445	446	447	448
0x05C0	449	450	451	452	453	454	455	456	457	458	459	460	461	462	463	464	465	466	467	468	469	470	471	472	473	474	475	476	477	478	479	480
0x05E0	481	482	483	484	485	486	487	488	489	490	491	492	493	494	495	496	497	498	499	500	501	502	503	504	505	506	507	508	509	510	511	512
0x0600	513	514	515	516	517	518	519	520	521	522	523	524	525	526	527	528	529	530	531	532	533	534	535	536	537	538	539	540	541	542	543	544
0x0620	545	546	547	548	549	550	551	552	553	554	555	556	557	558	559	560	561	562	563	564	565	566	567	568	569	570	571	572	573	574	575	576
0x0640	577	578	579	580	581	582	583	584	585	586	587	588	589	590	591	592	593	594	595	596	597	598	599	600	601	602	603	604	605	606	607	608
0x0660	609	610	611	612	613	614	615	616	617	618	619	620	621	622	623	624	625	626	627	628	629	630	631	632	633	634	635	636	637	638	639	640
0x0680	641	642	643	644	645	646	647	648	649	650	651	652	653	654	655	656	657	658	659	660	661	662	663	664	665	666	667	668	669	670	671	672
0x06A0	673	674	675	676	677	678	679	680	681	682	683	684	685	686	687	688	689	690	691	692	693	694	695	696	697	698	699	700	701	702	703	704
0x06C0	705	706	707	708	709	710	711	712	713	714	715	716	717	718	719	720	721	722	723	724	725	726	727	728	729	730	731	732	733	734	735	736
0x06E0	737	738	739	740	741	742	743	744	745	746	747	748	749	750	751	752	753	754	755	756	757	758	759	760	761	762	763	764	765	766	767	768

Figure 8 TV mode reading pattern (only highlighted cells are updated)

Subpage 0 --> 0x8000 = 0xXXX8

0x0400	1		3		5		7		9		11		13		15		17		19		21		23		25		27		29		31	
0x0420		34		36		38		40		42		44		46		48		50		52		54		56		58		60		62		64
0x0440	65		67		69		71		73		75		77		79		81		83		85		87		89		91		93		95	
0x0460		98		100		102		104		106		108		110		112		114		116		118		120		122		124		126		128
0x0480	129		131		133		135		137		139		141		143		145		147		149		151		153		155		157		159	
0x04A0		162		164		166		168		170		172		174		176		178		180		182		184		186		188		190		192
0x04C0	193		195		197		199		201		203		205		207		209		211		213		215		217		219		221		223	
0x04E0		226		228		230		232		234		236		238		240		242		244		246		248		250		252		254		256
0x0500	257		259		261		263		265		267		269		271		273		275		277		279		281		283		285		287	
0x0520		290		292		294		296		298		300		302		304		306		308		310		312		314		316		318		320
0x0540	321		323		325		327		329		331		333		335		337		339		341		343		345		347		349		351	
0x0560		354		356		358		360		362		364		366		368		370		372		374		376		378		380		382		384
0x0580	385		387		389		391		393		395		397		399		401		403		405		407		409		411		413		415	
0x05A0		418		420		422		424		426		428		430		432		434		436		438		440		442		444		446		448
0x05C0	449		451		453		455		457		459		461		463		465		467		469		471		473		475		477		479	
0x05E0		482		484		486		488		490		492		494		496		498		500		502		504		506		508		510		512
0x0600	513		515		517		519		521		523		525		527		529		531		533		535		537		539		541		543	
0x0620		546		548		550		552		554		556		558		560		562		564		566		568		570		572		574		576
0x0640	577		579		581		583		585		587		589		591		593		595		597		599		601		603		605		607	
0x0660		610		612		614		616		618		620		622		624		626		628		630		632		634		636		638		640
0x0680	641		643		645		647		649		651		653		655		657		659		661		663		665		667		669		671	
0x06A0		674		676		678		680		682		684		686		688		690		692		694		696		698		700		702		704
0x06C0	705		707		709		711		713		715		717		719		721		723		725		727		729		731		733		735	
0x06E0		738		740		742		744		746		748		750		752		754		756		758		760		762		764		766		768

Subpage 1 --> 0x8000 = 0xXXX9

0x0400		2		4		6		8		10		12		14		16		18		20		22		24		26		28		30		32
0x0420	33		35		37		39		41		43		45		47		49		51		53		55		57		59		61		63	
0x0440		66		68		70		72		74		76		78		80		82		84		86		88		90		92		94		96
0x0460	97		99		101		103		105		107		109		111		113		115		117		119		121		123		125		127	
0x0480		130		132		134		136		138		140		142		144		146		148		150		152		154		156		158		160
0x04A0	161		163		165		167		169		171		173		175		177		179		181		183		185		187		189		191	
0x04C0		194		196		198		200		202		204		206		208		210		212		214		216		218		220		222		224
0x04E0	225		227		229		231		233		235		237		239		241		243		245		247		249		251		253		255	
0x0500		258		260		262		264		266		268		270		272		274		276		278		280		282		284		286		288
0x0520	289		291		293		295		297		299		301		303		305		307		309		311		313		315		317		319	
0x0540		322		324		326		328		330		332		334		336		338		340		342		344		346		348		350		352
0x0560	353		355		357		359		361		363		365		367		369		371		373		375		377		379		381		383	
0x0580		386		388		390		392		394		396		398		400		402		404		406		408		410		412		414		416
0x05A0	417		419		421		423		425		427		429		431		433		435		437		439		441		443		445		447	
0x05C0		450		452		454		456		458		460		462		464		466		468		470		472		474		476		478		480
0x05E0	481		483		485		487		489		491		493		495		497		499		501		503		505		507		509		511	
0x0600		514		516		518		520		522		524		526		528		530		532		534		536		538		540		542		544
0x0620	545		547		549		551		553		555		557		559		561		563		565		567		569		571		573		575	
0x0640		578		580		582		584		586		588		590		592		594		596		598		600		602		604		606		608
0x0660	609		611		613		615		617		619		621		623		625		627		629		631		633		635		637		639	
0x0680		642		644		646		648		650		652		654		656		658		660		662		664		666		668		670		672
0x06A0	673		675		677		679		681		683		685		687		689		691		693		695		697		699		701		703	
0x06C0		706		708		710		712		714		716		718		720		722		724		726		728		730		732		734		736
0x06E0	737		739		741		743		745		747		749		751		753		755		757		759		761		763		765		767	

Figure 9 Chess reading pattern (only highlighted cells are updated)

10.7. Address map

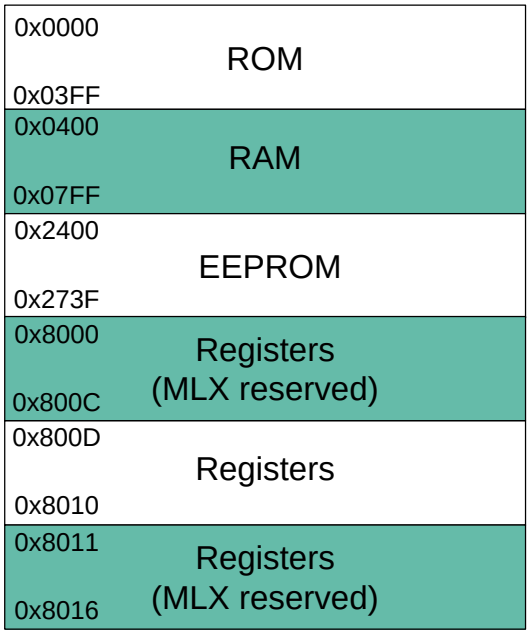


Figure 10 MXL90640 memory map

10.7.1. Internal registers

There are a few internal registers that are customer accessible through which the device performance can be customized:

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0	Status register - 0x8000
Melexis reserved											Enable overwrite	New data available in RAM	Last measured subpage controlled by MLX90641			
													0	0	0	Measurement of subpage 0 has been measured
													0	0	1	Measurement of subpage 1 has been measured
													0	1	0	Melxis reserved
													0	1	1	Melxis reserved
													1	0	0	Melxis reserved
													1	0	1	Melxis reserved
													1	1	0	Melxis reserved
													1	1	1	Melxis reserved
												0	No new data is available in RAM (must be reset by the customer)			
												1	A new data is available in RAM			
											0	Data in RAM overwrite is disabled				
											1	Data in RAM overwrite is enabled				
-	-	-	-	-	-	-	-	-	-	-	Melexis reserved					

Figure 11 Status register (0x8000) bits meaning

10.7.3. EEPROM

The EEPROM is used to store the calibration constants and the configuration parameters of the device

EEPROM address	Access	Meaning
0x2400	Melexis	Melexis reserved
0x2401	Melexis	Melexis reserved
0x2402	Melexis	Melexis reserved
0x2403	Melexis	Configuration register
0x2404	Melexis	Melexis reserved
0x2405	Melexis	Melexis reserved
0x2406	Melexis	Melexis reserved
0x2407	Melexis	Device ID1
0x2408	Melexis	Device ID2
0x2409	Melexis	Device ID3
0x240A	Melexis	Device Options
0x240B	Melexis	Melexis reserved
0x240C	Customer	Control register_1
0x240D	Customer	Control register_2
0x240E	Customer	I2CConfReg
0x240F	Customer	Melexis reserved / I2C_Address

Table 7 Configuration parameters memory

After POR the device read dedicated EEPROM cells and transfers their content to into the control and configuration register of the device. This way the device is configured and prepared for operation. The relation between EEPROM and register address is shown here after (explanation of the bit meaning can be found in section 10.7.1 Internal registers:

EEPROM address	Register address	Access	Name	Data [hex]
0x240C	0x800D	Customer	Control_register_1	1901
0x240D	0x800E	Customer	Control_register_2	0000
0x240E	0x800F	Customer	I2CConfReg	0000
0x240F	0x8010	Customer	Melexis internal use (8 bit) I2C_Address (8bit)	BE33

Table 8 EEPROM to registers mapping

15. Mechanical drawings

15.1. FOV 55°

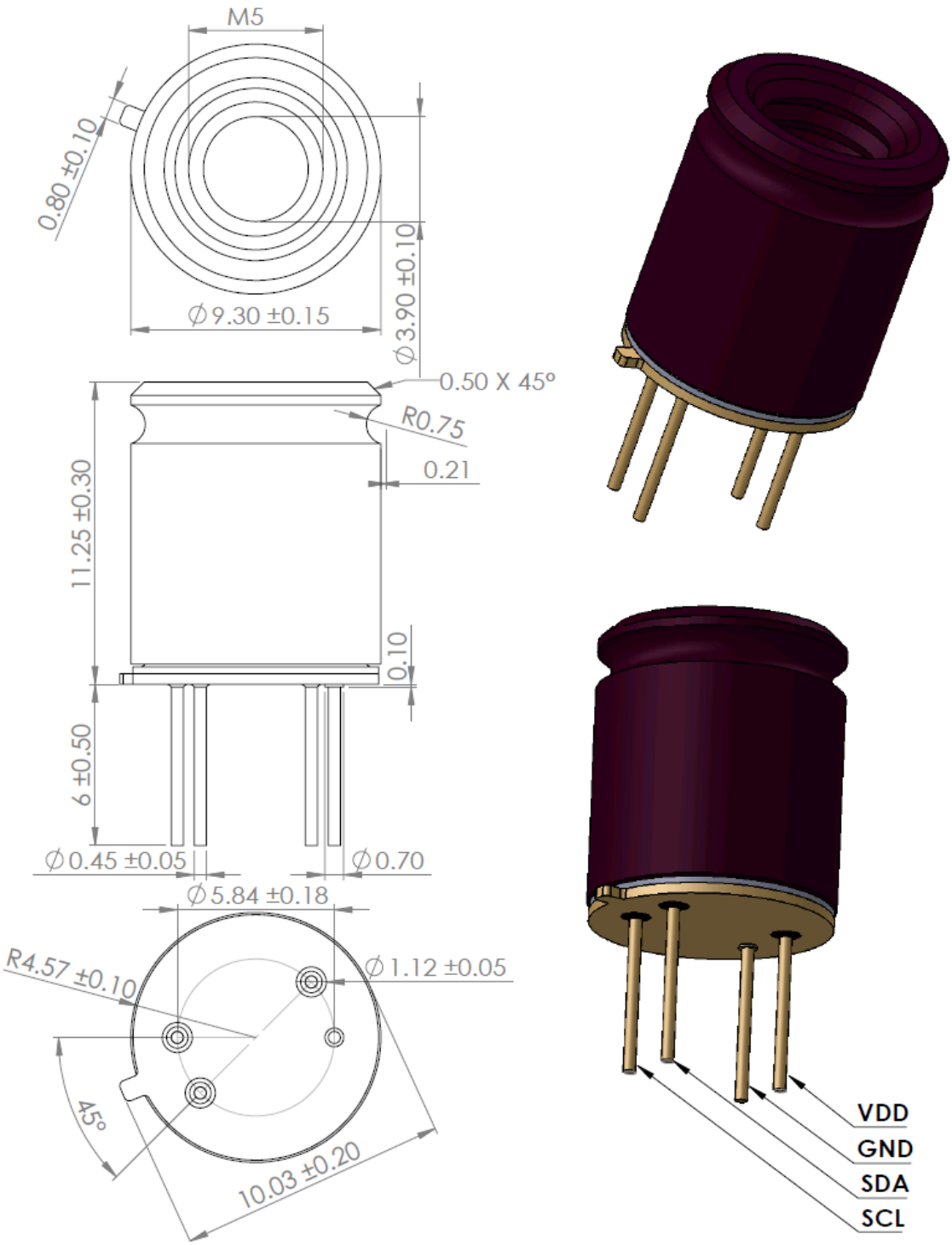


Figure 28 Mechanical drawing of 55° FOV device

15.2. FOV 110°

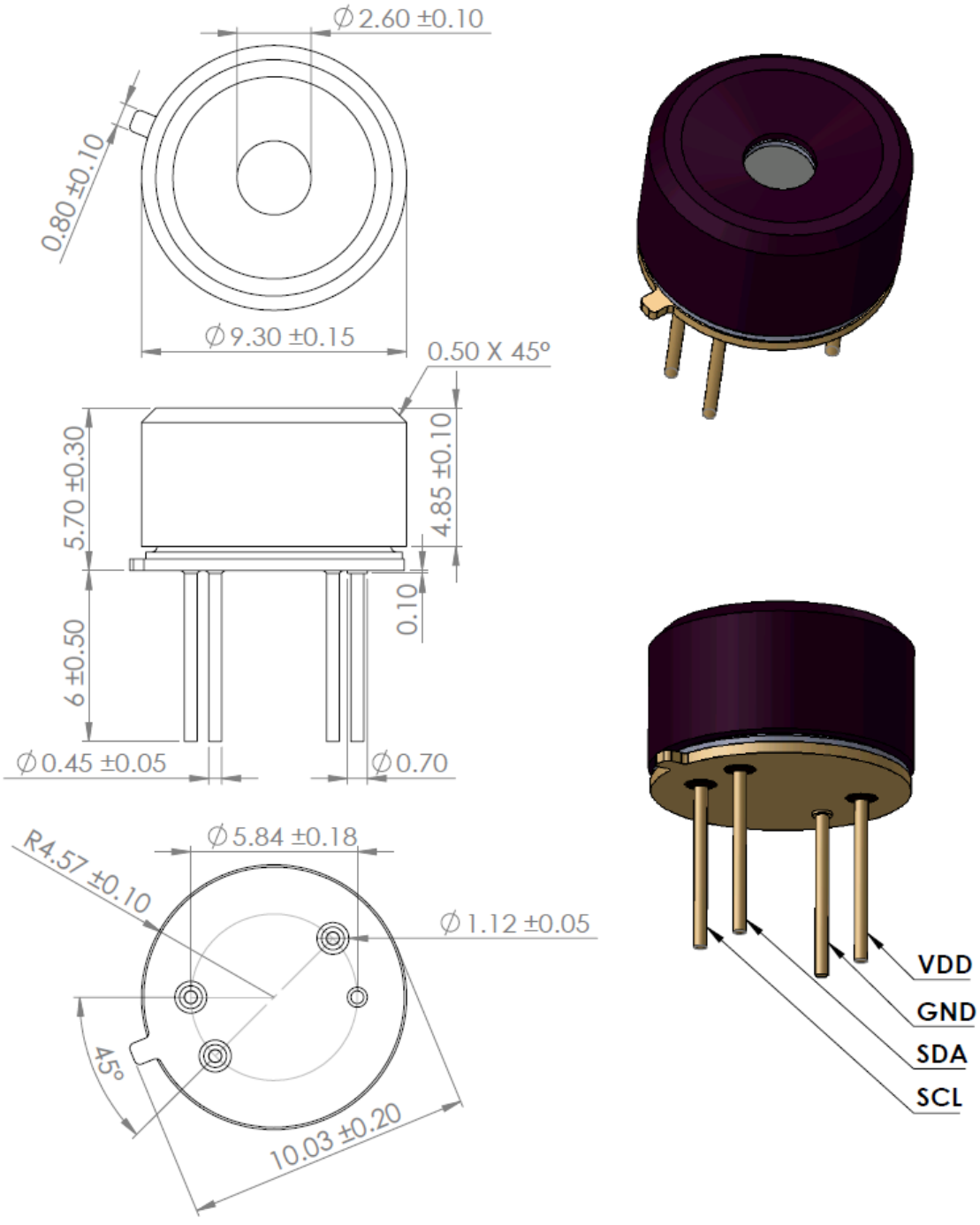


Figure 29 Mechanical drawing of 110° FOV device

16. Standard Information

Our products are classified and qualified regarding soldering technology, solderability and moisture sensitivity level according to standards in place in Semiconductor industry.

For further details about test method references and for compliance verification of selected soldering method for product integration, Melexis recommends reviewing on our web site the General Guidelines soldering recommendation. For all soldering technologies deviating from the one mentioned in above document (regarding peak temperature, temperature gradient, temperature profile etc.), additional classification and qualification tests have to be agreed upon with Melexis.

For package technology embedding trim and form post-delivery capability, Melexis recommends consulting the dedicated trim & forming recommendation application note: lead trimming and forming recommendations

Melexis is contributing to global environmental conservation by promoting **lead free** solutions. For more information on qualifications of **RoHS** compliant products (RoHS = European directive on the Restriction Of the use of certain Hazardous Substances) please visit the quality page on our website: http://www.melexis.com/en/quality-environment

17. ESD Precautions

Electronic semiconductor products are sensitive to Electro Static Discharge (ESD).
Always observe Electro Static Discharge control procedures whenever handling semiconductor products.

18. Revision history table

25/07/2016	Initial release
15/12/2016	Calibration data stored into EEPROM, pixel reading modes explained
17/01/2017	Some errors fixed
07/02/2017	Some calculations errors fixed
24/02/2017	Noise, FOV and accuracy graphs added, some inaccuracies fixed
02/03/2017	Overall rearranged , some typo and grammar mistakes fixed
18/05/2017	Two's complement for IR data from RAM and CP, added outlier identification in EEPROM , added application information
07/07/2017	Slave address changed to 0x240F, default mode is chess, CP RAM address changed 0x0709 -> 0x0708 and 0x0729 -> 0x0728, resolution control included in calculations, PCB under TO can removed
30/08/2017	Laser marking added, Max number of fail pixels added, Measurement flow (continuous and step mode) added, FOV definitions updated
10/10/2017	Added a note regarding CP averaging. Add dimension tolerances in mechanical drawings. Spelling errors corrected

11/04/2018	Updated accuracy table including BAB version, CP for different subpages, compensation for different reading patterns, extended temperature ranges calculations.
03/08/2018	Added: github driver link, ESD changed from 2kV to 4kV, Step mode removed, Internal register tables updated
03/12/2019	Rev 12: Max storage temp changed: 125°C to 85°C, added long term accuracy note, Ta accuracy added, added optical considerations, electrical consideration diagram updated with 10µF, added chamfer info, Doc server № added in the footer

19. Contact

For the latest version of this document, go to our website at www.melexis.com.

For additional information, please contact our Direct Sales team and get help for your specific needs:

Europe, Africa	Telephone: +32 13 67 04 95
	Email : sales_europe@melexis.com
Americas	Telephone: +1 603 223 2362
	Email : sales_usa@melexis.com
Asia	Email : sales_asia@melexis.com

20. Disclaimer

The information furnished by Melexis herein ("Information") is believed to be correct and accurate. Melexis disclaims (i) any and all liability in connection with or arising out of the furnishing, performance or use of the technical data or use of the product(s) as described herein ("Product") (ii) any and all liability, including without limitation, special, consequential or incidental damages, and (iii) any and all warranties, express, statutory, implied, or by description, including warranties of fitness for particular purpose, non-infringement and merchantability. No obligation or liability shall arise or flow out of Melexis' rendering of technical or other services.
The Information is provided "as is" and Melexis reserves the right to change the Information at any time and without notice. Therefore, before placing orders and/or prior to designing the Product into a system, users or any third party should obtain the latest version of the relevant information to verify that the information being relied upon is current. Users or any third party must further determine the suitability of the Product for its application, including the level of reliability required and determine whether it is fit for a particular purpose.
The Information is proprietary and/or confidential information of Melexis and the use thereof or anything described by the Information does not grant, explicitly or implicitly, to any party any patent rights, licenses, or any other intellectual property rights.
This document as well as the Product(s) may be subject to export control regulations. Please be aware that export might require a prior authorization from competent authorities.
The Product(s) are intended for use in normal commercial applications. Unless otherwise agreed upon in writing, the Product(s) are not designed, authorized or warranted to be suitable in applications requiring extended temperature range and/or unusual environmental requirements. High reliability applications, such as medical life-support or life-sustaining equipment are specifically not recommended by Melexis.
The Product(s) may not be used for the following applications subject to export control regulations: the development, production, processing, operation, maintenance, storage, recognition or proliferation of 1) chemical, biological or nuclear weapons, or for the development, production, maintenance or storage of missiles for such weapons: 2) civil firearms, including spare parts or ammunition for such arms; 3) defence related products, or other material for military use or for law enforcement; 4) any applications that, alone or in combination with other goods, substances or organisms could cause serious harm to persons or goods and that can be used as a means of violence in an armed conflict or any similar violent situation.
The Products sold by Melexis are subject to the terms and conditions as specified in the Terms of Sale, which can be found at https://www.melexis.com/en/legal/terms-and-conditions.

This document supersedes and replaces all prior information regarding the Product(s) and/or previous versions of this document.

Melexis NV © - No part of this document may be reproduced without the prior written consent of Melexis. (2016)

ISO/TS 16949 and ISO14001 Certified