UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

GRADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

"SISTEMA ELECTRÓNICO PARA LA CARACTERIZACIÓN Y MODELADO DE PANELES SOLARES EN TIEMPO REAL"

TRABAJO FIN DE GRADO

Julio -2021

AUTOR: Pablo Casado Pérez

DIRECTOR/ES: José Manuel Blanes Martínez

Vicente Galiano Ibarra

Agradecimientos

Este TFG es el resultado de varios meses de trabajo y aprendizaje, y terminarlo no hubiera sido posible sin la ayuda y el apoyo de varias personas a las que me gustaría mostrar mi agradecimiento.

En primer lugar, al doctor José Manuel Blanes Martínez, por su gran apoyo, su paciencia y su dedicación durante el desarrollo de este trabajo. Ha sabido guiarme en todo momento y ha sido de gran ayuda para resolver los distintos problemas que han surgido por el camino.

Además, me gustaría agradecer a mi compañero de laboratorio Cristian Torres Vergara, por su gran apoyo, su tiempo y su ayuda siempre que he tenido alguna duda a cambio de nada. Sin su ayuda no hubiese sido posible completar este TFG.

También me gustaría dar las gracias al departamento de Ciencia de Materiales, Óptica y Tecnología Electrónica por acogerme en su laboratorio y hacerme sentir cómodo desde el primer día.

Para terminar, me gustaría agradecer a mi familia y a los profesores que he tenido a lo largo de mi vida académica que han hecho posible que pueda llegar a finalizar mis estudios de grado.

Índice

Índice de tablas

Índice de figuras

Abreviaturas

Abreviatura	Descripción
RPi	Raspberry Pi
GPIO	Entrada/Salida de Propósito General
AD2	Analog Discovery 2
SO	Sistema Operativo
SMD	Surface Mounted Device (Montaje superficial)
THT	Through-Hole (Orificio pasante)
PGA	Programmable Gain Amplifier (Ganancia)
FS	Full-Scale Range (Rango de entrada de escala completa)
SF	Scale Factor (Factor de escala)
GUI	Graphical User Interface (Interfaz Gráfica de Usuario)
Voc	Tensión de circuito abierto
Isc	Corriente de cortocircuito
Pmpp	Máxima potencia
Vmpp	Tensión del punto de máxima potencia
Impp	Corriente del punto de máxima potencia
FF	Factor de forma o factor de llenado

CAPÍTULO 1. Introducción

La energía se define como la capacidad que tiene un sistema para realizar un trabajo. Para obtenerla, se requiere de una fuente de energía que puede ser de dos tipos. Por un lado, está la renovable (como la solar, la eólica o la hidráulica), que son aquellas que son inagotables y a las que se puede recurrir de manera permanente. Por otro lado, está la no renovable (como el petróleo, el gas natural o el carbón), cuyas reservas son limitadas y disminuyen a medida que se consumen.

Actualmente, las energías renovables están creciendo a una gran velocidad y ya representan entorno al 60% de la producción de energía a nivel mundial. Como hemos citado, una de las fuentes de energía renovable es la energía solar, que es la que llega desde el Sol a la Tierra en forma de radiación electromagnética. Esta se puede convertir en energía eléctrica mediante el efecto fotoeléctrico haciendo uso de paneles solares.

En el diseño de instalaciones fotovoltaicas es muy importante conocer las características de los paneles solares que vayamos a utilizar. Estas características dependerán de variables externas e internas que afectarán al rendimiento de la instalación. Para comprobar su correcto funcionamiento es necesario disponer de un instrumento de medida que sea capaz de detectar el comportamiento del dispositivo fotovoltaico en cuestión.

En el área de Tecnología Electrónica del departamento de Ciencia de Materiales, Óptica y Tecnología Electrónica se dispone de un dispositivo electrónico capaz de medir la curva característica I-V de un panel solar, desde el cual partirá este trabajo.

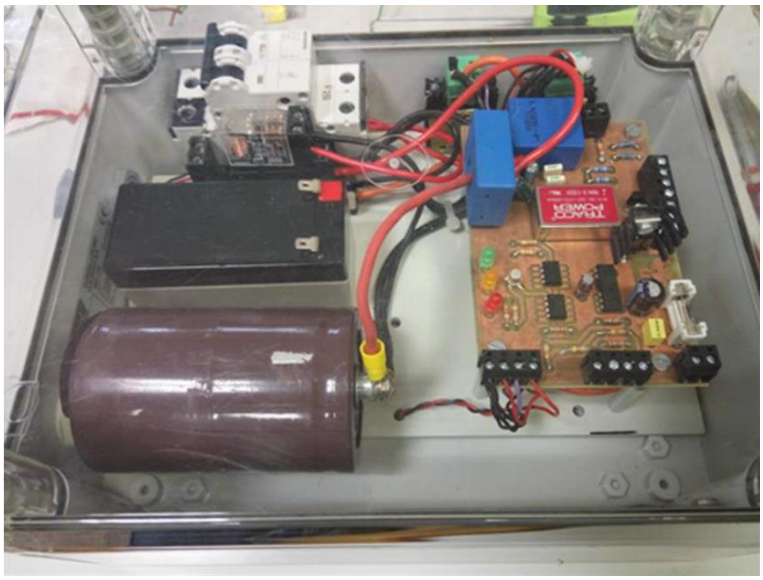


Figura 1. Dispositivo electrónico del área de Tecnología Electrónica

1.1. Objetivos del proyecto

El objetivo de este trabajo es realizar el diseño y la implementación de un sistema electrónico portátil independiente para realizar la caracterización en tiempo real de paneles solares. Este dispositivo electrónico deberá de ser capaz de:

- Monitorizar tanto in situ como de manera remota el comportamiento a la salida de un dispositivo fotovoltaico a través del uso de un microcontrolador.
- Obtener las curvas características adquiriendo los pares de valores de tensión y corriente durante la carga de un condensador.
- Obtener los parámetros fundamentales a partir de los datos de tensión y corriente.
- Adquirir los valores de la irradiancia solar y temperatura del panel para contextualizar las medidas.
- Realizar series de medidas programadas por el usuario cuyos resultados se almacenen en una tabla para que puedan ser contrastados.

1.2. Estructura del proyecto

El desarrollo del diseño y la implementación de este proyecto está estructurado en ocho capítulos distintos:

- Capítulo 1: Es el capítulo actual, donde se ha introducido el trabajo desde el que partimos además de los objetivos de este proyecto.
- Capítulo 2: En él explicaremos como funciona una célula solar, como están constituidos los distintos paneles solares que existen en la actualidad y cuáles son los parámetros fundamentales que los caracterizan. De esta manera, contextualizaremos el uso y la importancia de trazadores de curvas características.
- Capítulo 3: En este capítulo se desarrollará tanto el diseño hardware del dispositivo electrónico como su montaje y ensamblado.
- Capítulo 4: Aquí explicaremos la parte del diseño software de las distintas aplicaciones que tiene el dispositivo de medida.
- Capítulo 5: En este se mostrarán los resultados experimentales obtenidos con distintas configuraciones de paneles solares de distintos modelos.
- Capítulo 6: En él se encontrarán las conclusiones del proyecto.
- Bibliografía: En ella se citarán las distintas fuentes que han sido útiles para el desarrollo de este trabajo.
- Anexos: En este último apartado se encontrarán el presupuesto, los distintos planos relacionados con el diseño hardware y los distintos códigos de Python de la parte de software.

CAPÍTULO 2. Estado del arte

En este capítulo estudiaremos el funcionamiento de una célula fotovoltaica y las distintas partes de un panel solar. Además definiremos los parámetros fundamentales de un módulo fotovoltaico y que influencia tiene la climatología sobre su rendimiento. Por último, presentaremos algunos ejemplos de trazadores de curva IV que hay actualmente en el mercado.

2.1. Célula fotovoltaica

Una célula solar fotovoltaica es un dispositivo que tiene la capacidad de convertir energía luminosa en energía eléctrica a través del efecto fotovoltaico. Para que se produzca este fenómeno se utilizan materiales semiconductores en la forma de unión P-N, donde N es la carga negativa expuesta a la luz y P la positiva situada en la zona de oscuridad. Gracias a esta composición, un fotón es capaz de excitar a un electrón para que pase a un estado de energía más alto, desencadenando así el movimiento de electrones libres de un semiconductor a otro en busca de un hueco que llenar. Esto se traduce en una corriente eléctrica y una diferencia de potencial que variarán en función de la luz a la que están expuestas.

Actualmente, se emplean una gran variedad de materiales para la fabricación de células fotovoltaicas. Estas se pueden clasificar en tres generaciones distintas en función de su material y la importancia que han tenido a lo largo de la historia.

2.1.1. Primera generación

Las células de primera generación son aquellas que están compuestas por silicio cristalino (c-Si). Son las células solares más comunes y económicas. Se pueden clasificar según su estructura interna:

- Silicio monocristalino (m-Si). Son aquellas que están compuestas por un único cristal de silicio. Posee una estructura muy uniforme que facilita el movimiento de electrones en su interior, otorgándole así un mayor rendimiento que el resto de tecnologías en condiciones de baja radiación solar (entorno al 18-25%).
  Debido a que las obleas con las que se fabrican se cortan a partir de lingotes cilíndricos, las esquinas de las celdas se recortan, dándole así su forma octogonal característica.
- Silicio policristalino (p-Si). Son aquellas que están formadas por muchos cristales de silicio. Los lingotes a partir de los que se fabrican son cuadrados, por lo que son menos costosas que las monocristalinas al no ser necesario recortar las esquinas. No obstante, tienen un menor rendimiento, entorno al 16-20%.

2.1.2. Segunda generación

Las células de segunda generación son aquellas que están formadas por una película delgada. Son de menor coste que las de la primera generación, aunque su rendimiento es menor, pues se requiere del doble de espacio para producir la misma potencia. Algunos ejemplos son las células de silicio amorfo, que son las células más desarrolladas de esta generación hasta la fecha; las células de teluro de cadmio (CdTe), cuyo proceso de fabricación es muy tóxico aunque son las únicas que pueden competir con las de silicio; y las de seleniuro de cobre, indio y galio(CIGS).

2.1.3. Tercera generación

Las células de tercera generación son aquellas que tienen una eficiencia mucho mayor y un precio de producción mucho menor. Actualmente, se encuentran en mayor parte en investigación y desarrollo, por lo que no se han extendido en el mercado. Muchas de ellas están basadas en materiales orgánicos, como el caso de las células solares de perovskita, con las que se ha llegado a conseguir una eficiencia del 25% aunque con una vida útil muy corta.

2.2. Panel solar fotovoltaico

Un panel solar o módulo fotovoltaico está formado por un conjunto de células solares, las cuales están conectadas eléctricamente en serie o paralelo en función del nivel de tensión e intensidad para el que esté diseñado el panel. A continuación, expondremos la constitución con la que están formados.

2.2.1. Cubierta frontal y posterior

El panel está cubierto por una cubierta tanto por la parte superior como por la inferior. La cubierta frontal está compuesta por vidrio templado con un contenido en hierro muy bajo, el cual es muy buen transmisor de radiación solar. Su función principal es la de proteger ante adversidades que pueda presentar el clima, como el impacto del granizo.

En cuanto a la cubierta posterior, también se utiliza como protección mecánica además de protección contra la humedad. No obstante, al no ser necesario que deje pasar la luz, suele ser de polímeros o plásticos completamente opacas de color blanco.

2.2.2. Encapsulado

El encapsulado es el encargado de proteger el módulo solar de la intemperie. Suele estar formado por materiales con una buena transmisión a la radiación y con una baja degradación frente a radiaciones ultravioletas. El material más común es el etil-vinil-acetileno (EVA). Es muy importante que el proceso de encapsulado se realice correctamente, pues de lo contrario, las células podrían oxidarse si entran en contacto con aire o agua.

2.2.3. Cajas de conexiones y diodos de bypass

En la parte posterior del panel se encuentra la caja de conexiones, donde todas las celdas se interconectan y salen dos cables, uno positivo y otro negativo. Además, en esta caja hay colocados unos diodos de protección, también denominados diodos de bypass, que permiten que la corriente fluya en una sola dirección. Estos diodos son los encargados de minimizar los efectos negativos en el rendimiento que pueden producir las sombras o las diferencias de temperatura entre células. Pues si una célula sombreada tuviera que disipar una potencia igual que la del resto de células, se calentaría y se podría generar sobre ella un “hot spot” o punto caliente.

2.3. Curva característica I-V

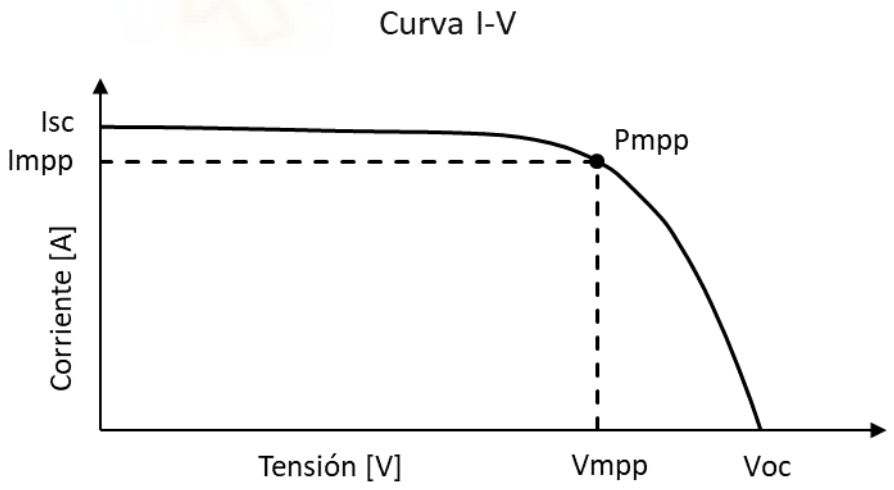


Figura 2. Ejemplo de curva característica I-V

La curva característica I-V (Figura 2) se utiliza para poder estudiar el comportamiento a la salida de una célula, módulo, panel o sistema fotovoltaico. Esta está formada por pares de valores de tensión y corriente que pueden encontrarse

funcionando en una célula. A partir de estos valores, se pueden extraer los parámetros fundamentales, que son:

- Tensión de circuito abierto (Voc). Es la tensión que hay en el panel cuando la corriente es nula. Es la máxima tensión que podemos encontrar en la curva.
- Corriente de cortocircuito (Isc). Es la corriente que entrega el panel cuando se cortocircuitan sus terminales. Es la máxima intensidad que podemos encontrar en la curva.
- Punto de máxima potencia. Es la pareja de valores de la curva que mayor potencia tiene. Está formado por la corriente de máxima potencia (Impp) y por la tensión de máxima potencia (Vmpp). El producto de ambas es igual a la potencia de pico del panel (Pmpp).
- Factor de forma (FF). También conocido como factor de relleno, nos indica que tan cerca está la curva característica I-V de la forma rectangular. Se calcula a partir de la fórmula:

$$FF = \frac{I_{mpp} \cdot V_{mpp}}{V_{OC} \cdot I_{SC}}$$

2.4. Efectos climatológicos

En un panel solar, las condiciones climatológicas son determinantes, pues en función de estas cambiará su curva característica asociada. Dos efectos ambientales que influyen en esta variación son la radiación solar incidente sobre el plano del panel, y la temperatura en el panel.

Por un lado, la radiación solar incidente perpendicularmente sobre el plano del panel es directamente proporcional a la corriente que entrega el panel. Cuanto mayor es la radiación, mayor es la intensidad (Figura 3).

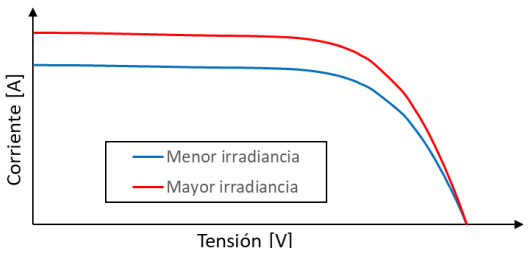


Figura 3. Comparativa de curvas con distinta irradiancia

Por otro lado, la temperatura a la que están las células que componen el panel afecta a los valores de tensión de la curva. Cuando la temperatura aumenta, la tensión disminuye, mientras que cuando la temperatura disminuye, la tensión aumenta (Figura 4).

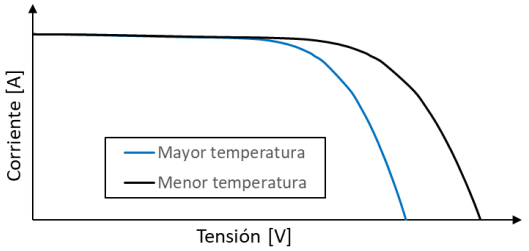


Figura 4. Comparativa de curvas con distinta temperatura de panel

2.5. Trazadores de curva IV del mercado

Hoy en día existen algunos modelos comerciales de trazadores de curva I-V. Los valores máximos de tensión e intensidad que son capaces de medir rondan entorno a 1kV y 10A. Un ejemplo es el modelo SOLAR I-Ve de la marca HT Instruments (Figura 5). Este es capaz de medir curvas con 128 puntos en instalaciones de hasta 1500V y 10A o de hasta 1000V y 15A. Actualmente puede encontrarse por 4796.98€ en Amazon. Otro ejemplo es el modelo FTV200 del fabricante Chauvin Arnoux, que realiza medidas de 500 puntos en instalaciones desde 10V y 0.1A hasta 1000V y 10A. Su precio actual en la tienda online mediaprec.es asciende a los 3760€.

Figura 5. Imagen del SOLAR I-Ve de HT Instruments extraída de la web del fabricante

CAPÍTULO 3. Diseño hardware

Este dispositivo de medida será diseñado para que sea capaz de medir valores nominales de tensión y corriente de hasta 200V y 20A con total seguridad. Además, utilizará el método de medida de la carga capacitiva, que destaca por su sencillez, inmediatez y precisión. Este método se basa en conectar al dispositivo fotovoltaico un condensador descargado, que se cargará hasta la tensión de circuito abierto. Durante esta carga, se realizará la adquisición de los pares de valores de tensión y corriente a la salida del panel solar, que corresponderán con los de la curva característica I-V del panel.

Debido a que el sistema deberá de ser portátil, se utilizará un osciloscopio portátil de USB para la adquisición de los valores de tensión y corriente asociados a la carga del condensador. Este, irá conectado a una Raspberry Pi 4 que procesará toda la información que le envíe, además de regular la carga y descarga y de leer los datos de irradiancia y temperatura de cada medida.

En la figura 6 se muestra un diagrama de bloques de la estructura del circuito completo que se va a desarrollar en este trabajo.

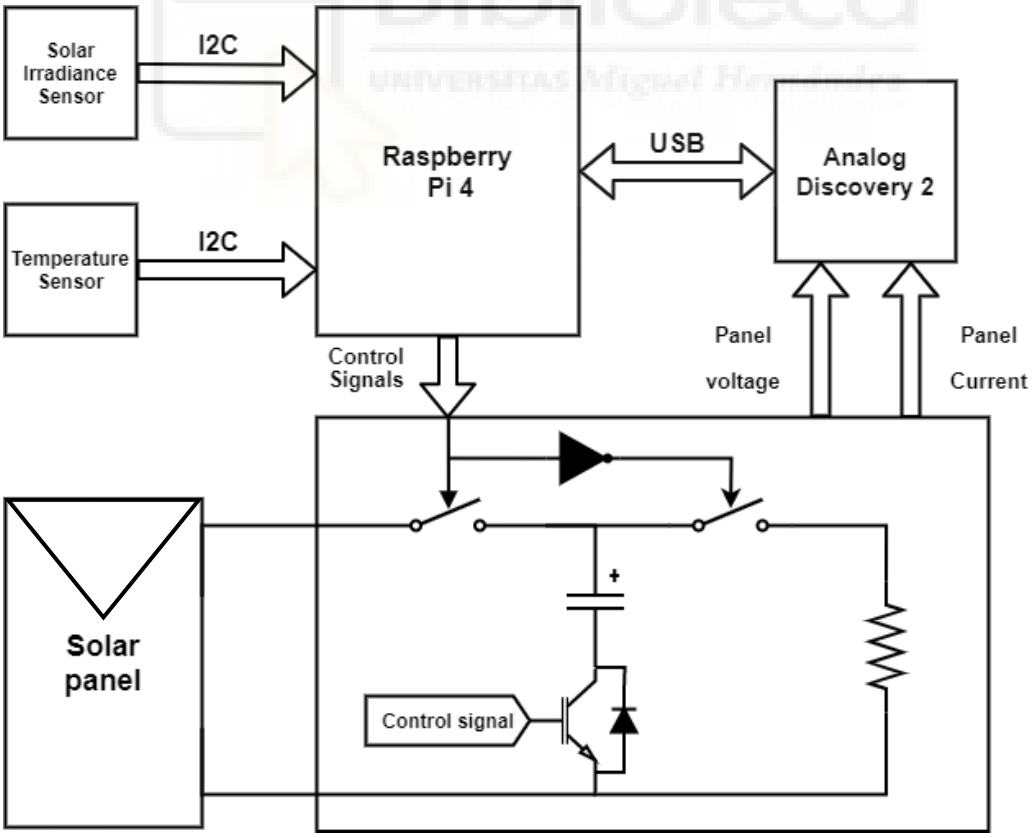


Figura 6. Diagrama de bloques de la estructura del circuito

3.1. Circuito de medida

Esta parte del circuito del dispositivo será la encargada de adquirir los datos de la curva característica IV del panel. Este circuito estará compuesto principalmente por un condensador, una resistencia de descarga, relés, un transistor IGBT, y dos transconductores de tensión y corriente.

3.1.1. Condensador

Uno de los puntos más significativos del diseño es la elección del condensador que se cargará con el panel solar que conectemos. De este dependerá directamente el tiempo que se tarde en realizar la medida, que no podrá ser demasiado bajo, porque perderíamos resolución en la adquisición, ni demasiado alto, ya que la temperatura e irradiancia cambiar y alterar la medida.

Como hemos explicado anteriormente, un panel solar está compuesto por N_p cadenas de N_s celdas en serie que tienen una corriente de cortocircuito I_{sc}. Esto se traduce en que la corriente i(t) que suministra un panel se puede expresar como:

$$\mathrm{i(t)} = \mathrm{I_{SC}} = \mathrm{I_{SC_{celda}}} \cdot \mathrm{N_{paralelo}} \quad \text{para} \quad 0 < \mathrm{t} < \mathrm{t_0}$$

En cuanto a la variación del voltaje durante la carga, sabemos que el condensador comienza comportándose como un cortocircuito. Después, el voltaje aumenta rápidamente hasta que se llega al último tramo de la carga en el que aumenta cada vez menos rápido. Si despreciamos el voltaje inicial del condensador, ya que en nuestra aplicación siempre partimos desde cero voltios, y si suponemos que la corriente que lo carga es constante, la expresión que calcula la tensión vendría dada por:

$$v(t) = \frac{1}{C}\int_0^{t_0} I_{SC} \cdot dt = \frac{I_{SC}}{C} \cdot t \qquad para \qquad 0 < t < t_0$$

Por lo tanto, en el instante $t = t_0$, que es cuando se alcanza la tensión en circuito abierto del panel, tenemos que:

$$v(t_0) = V_{OC_{celda}} \cdot N_{Serie} = V_{OC} = \frac{I_{sc} \cdot t_0}{C}$$

De esta última ecuación podemos despejar la expresión ideal que enlaza la capacidad del condensador con el tiempo de carga:

$$C = \frac{I_{sc} \cdot t_0}{V_{OC}}$$

Si queremos realizar medidas de 10 milisegundos y tenemos en cuenta los valores de Isc y Voc para los que vamos a diseñar el dispositivo de medida, tendremos que utilizar un condensador con una capacidad de al menos 1 mF.

En cuanto al tipo de condensador, se ha optado por utilizar uno electrolítico. Estos son capaces de soportar altos voltajes y, además, existe una gran oferta con capacidades de entorno a unos pocos de miles de faradios. No obstante, la polaridad en este tipo de capacitores es muy importante, por lo que será necesario que el dispositivo detecte antes de medir si la polaridad con la que se ha conectado el panel es correcta.

Figura 7. Condensador electrolítico de 2200 uF de Nichicon. Imagen extraída de la web de la tienda Digikey

Finalmente, se ha seleccionado un condensador electrolítico de 2200 uF con una tensión nominal de 400 V (superior a los 200 V de Voc para los que está diseñado el dispositivo) de la marca Nichicon como el de la figura 7. Las conexiones con la placa de este componente se pueden ver en la tabla 1.

Conector	Conectado a
X1-3	Terminal negativo condensador
X1-4	Terminal positivo condensador

Tabla 1. Conexiones del condensador con la PCB

3.1.2. Resistencia de descarga

Otro de los puntos más significativos del diseño es la elección de la resistencia de descarga, pues el tiempo mínimo entre medida y medida dependerá directamente del valor que tenga.

En un circuito RC, el tiempo de carga y descarga del condensador viene dada por la expresión de la constante de tiempo:

$$\tau = R \cdot C$$

Sabemos que cuando transcurre τ, el condensador se descarga un 63.2%. Para considerar que está completamente descargado tiene que transcurrir cinco veces la constante de tiempo (5 · τ). Por lo tanto, si queremos que la descarga sea entorno a un segundo:

$$\tau = \frac{1[s]}{5} = 0.2\ [s]$$

$$R = \frac{\tau}{C} = \frac{0.2}{0.0022} = 90.90\ [\Omega]$$

Finalmente se ha seleccionado una resistencia de potencia de 120 Ω de la marca ARCOL modelo HS50, por lo que el tiempo de descarga será:

$$t_{descarga} = 5 \cdot \tau = 5 \cdot R \cdot C = 5 \cdot 120 \cdot 0.0022 = 1.32\ [s]$$

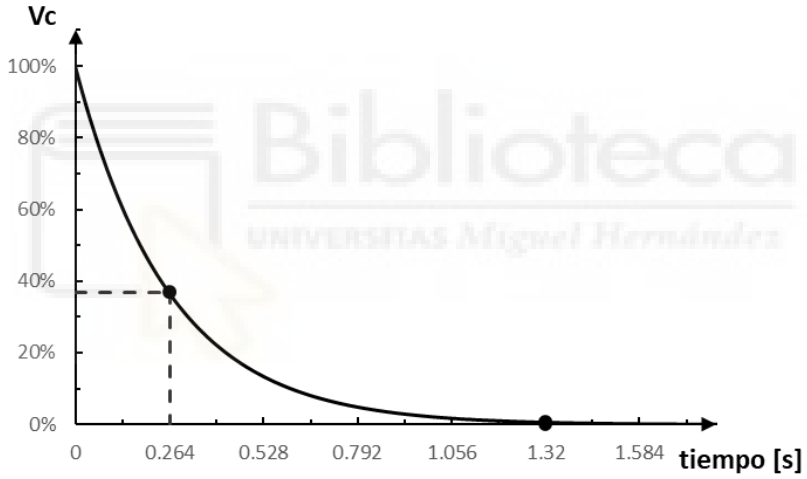


Figura 8. Gráfica de la descarga de un condensador

En cuanto a las conexiones con la placa con esta resistencia, en la tabla 2 se muestran los conectores a los que tendrán que ir conectados.

Conector	Conectado a
X1-1	Terminal resistencia
X1-2	Terminal resistencia

Tabla 2. Conexiones de la resistencia con la PCB

3.1.3. Relés y transistor

Para controlar la conexión y desconexión del condensador con la resistencia y el panel, utilizaremos relés de potencia. Estos tendrán que ser capaces de aguantar tensiones en continua de como mínimo 200 V, además de ser apto para que puedan circular 20 A por sus terminales y contactos. Tras mirar en el mercado, se encontró el modelo de relé G7L-2A-X-L de la marca OMRON (Figura 9) que toleraba cargas nominales de 20 A a 1000 V en continua. Su tiempo de conmutación, según la hoja de datos del fabricante, es de 30 ms como máximo.

Figura 9. Relé G7L-2A-X-L de la marca OMRON. Imagen extraída de la web de la tienda Digikey

Una de las desventajas que tiene utilizar relés en el circuito de medida es el posible ruido que se puede introducir en la señal cuando estos se accionan. Además, como se verá más adelante, el tiempo de adquisición de datos es muy limitado, por lo que necesitamos que el tiempo que se tarde en empezar la carga del condensador sea lo más pequeño posible. Para resolver ambos problemas, se ha decidido incluir un transistor IGBT en serie con el condensador que se accionará 30 ms después que el relé asociado a la carga. En concreto, será el modelo IHW20N120R5 de la marca Infineon, cuyos valores nominales son I_{CE} = 20 A y V_{CE} = 1200 V. Además, su tiempo de conmutación es de 10us como máximo, que es bastante menor que el del relé.

En el diseño se han incluido las conexiones para poder utilizar dos transistores, uno para el circuito de medida, y otro auxiliar por si se requiere en alguna futura modificación. En la tabla 3 se pueden ver las conexiones de ambos con la placa.

Conector	Conectado a
X6-1	Colector IGBT 1
X6-2	Puerta IGBT 1
X6-3	Emisor IGBT 1
X5-1	Colector IGBT 2
X5-2	Puerta IGBT 2
X5-3	Emisor IGBT 2

Tabla 3. Conexiones de los IGBTs con la PCB

3.1.4. Transconductores

Puesto que el instrumento de adquisición de datos que utilizaremos no está diseñado para medir corrientes y tampoco para registrar tensiones tan elevadas, haremos uso de un transconductor de corriente y otro de tensión.

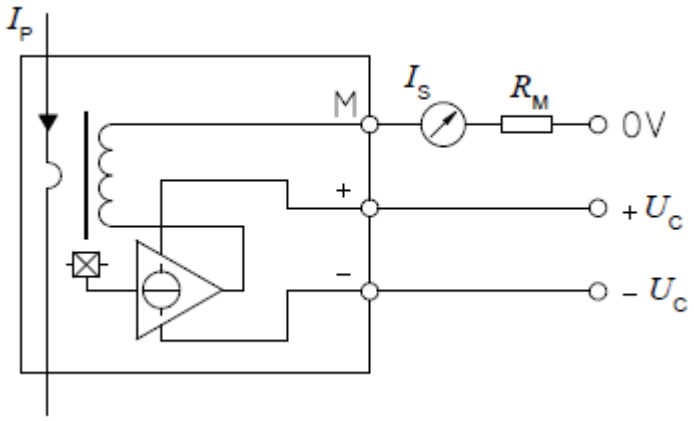


Figura 10. Esquema extraído del datasheet del transconductor LA 100P SP13

En referencia al transconductor de corriente, utilizaremos el modelo LA 100-P SP13 de la marca LEM, que según el fabricante puede medir corrientes de hasta 100 A. Como podemos ver en el esquema de la figura 10, este componente de bucle cerrado convierte la corriente I_P en otra proporcionalmente menor en el bobinado secundario (I_S). Esta segunda corriente se puede determinar con la Ley de Ohm:

$$I_S = V_M \cdot R_M$$

Por lo tanto, si le damos a R_M un valor de 100Ω, solo tendremos que medir la caída de voltaje en esta resistencia. A partir de I_S se puede calcular I_P aplicando el ratio de conversión de 1:1000 que nos da el fabricante. No obstante, puesto que este transconductor de corriente utiliza tecnología de efecto Hall, y que permite medir hasta 100A y solo vamos a medir hasta 20A, se ha pasado 5 veces el conductor de I_P por el agujero primario para multiplicar por 5 la salida I_S. De esta manera aumentamos la precisión de la medida de la corriente. Finalmente, el factor de conversión quedaría:

$$K_I = \frac{1000}{5} \cdot \frac{1}{100} = 2$$

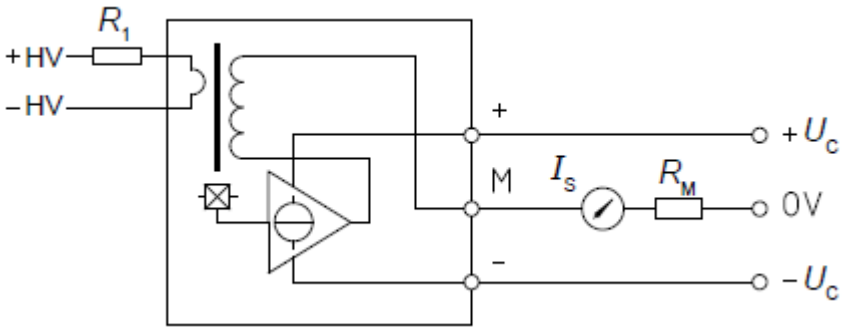


Figura 11. Esquema extraído del datasheet del transconductor LV 55-P

En cuanto al transconductor de tensión, utilizaremos el modelo LV 25-P de la marca LEM, que según su fabricante puede medir tensiones de hasta 500 V. Como podemos ver en el esquema de la figura 11, este componente posee dos partes diferenciadas y aisladas galvánicamente. Por un lado, el circuito primario, al que tendremos que colocarle una resistencia en serie (R_1) para poder medir tensión. Y por otro lado, el circuito secundario, que nos proporcionará una corriente proporcionalmente menor a la que pasa por el primario. Para medir esta intensidad, colocaremos una resistencia (R_M) de 100 Ω referenciada a 0V.

En la hoja de datos del fabricante se indica que la máxima precisión se obtiene cuando la tensión que queramos medir corresponda a una corriente de 10 mA en el circuito primario. Es decir, la resistencia R_1 tendrá que ser:

$$R_1 = \frac{V_P}{10\ mA}$$

Además, si tenemos en cuenta toda la resistencia del devanado primario y no solo R_1, podremos medir con mayor precisión las bajas tensiones. Esto es debido a que la resistencia que hay entre +HV y -HV está entorno a los 230Ω, por lo que cuanto menor sea R1, peor será la precisión del transconductor. Por ejemplo, si colocamos una resistencia de 1kΩ en R_1 para medir 10 V, realmente estaremos midiendo con una resistencia de 1.23kΩ. Por este motivo, se utilizarán distintas resistencias en el circuito primario en función del Voc que se vaya a medir. Se han seleccionado cuatro resistencias distintas: 2kΩ, 4.7kΩ, 14.7kΩ, 19.4kΩ. Para cambiar de una a otra se ha diseñado un selector. Este será explicado más adelante junto con el factor de conversión.

3.1.5. Circuito completo

Una vez seleccionados los componentes, pasaremos a explicar el funcionamiento del circuito completo, cuyo esquema se puede ver en la figura 12.

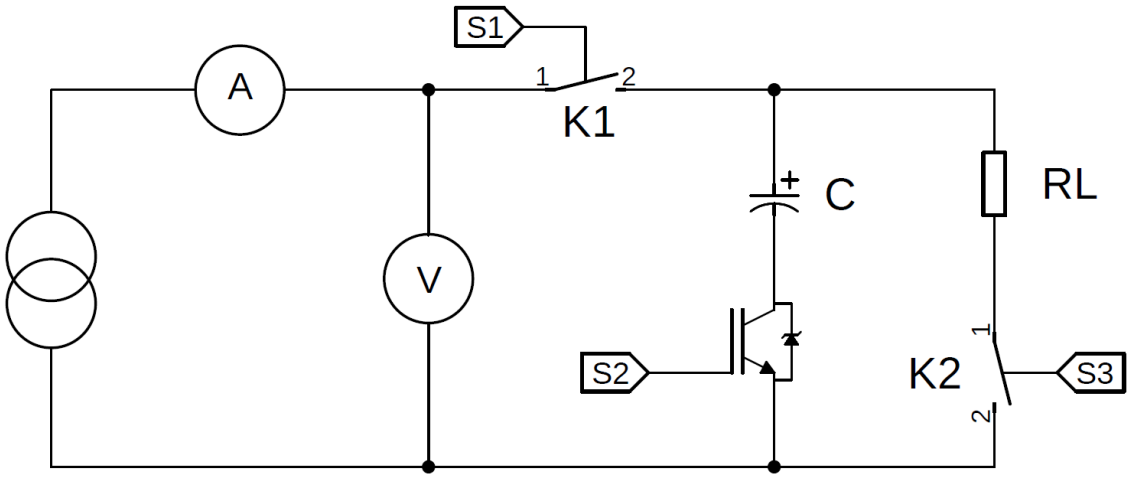


Figura 12. Esquema del circuito de medida

Como se puede ver en el circuito, existen 3 señales de control distintas. La función de cada una es:

- S1 : Controla el relé K1, que es el encargado de conectar el panel solar al circuito de medida.
- S2 : Esta señal de control es la que gobierna la puerta del transistor IGBT. Cada vez que se active S1 y transcurran los 30 ms que tarda el relé en accionarse, este se activará y comenzará la adquisición.
- S3 : Controla el relé K2, que es el encargado de conectar la resistencia de descarga en paralelo con el condensador.

En la tabla 4 se muestra la secuencia que se tienen que seguir las señales de control para realizar una medida:

	S1	S2	S3
PASO 1	OFF	ON	ON
PASO 2	ON	OFF	OFF
PASO 3	ON	ON	OFF
PASO 4	OFF	ON	OFF
PASO 5	OFF	ON	ON

Tabla 4. Secuencia de las señales de control

- PASO 1: Descargar el condensador durante 1.32 s. Este paso no es necesario si el condensador está descargado, pero de esta manera nos aseguramos de que sí lo está.
- PASO 2: Apagamos todas las señales y tras 30ms activamos S1. Con este paso evitamos perturbaciones del relé en la medida.
- PASO 3: Adquisición de datos. Activamos el IGBT y comenzamos el barrido.
- PASO 4: Después del barrido desconectamos el panel del circuito y esperamos los 30 ms que tarda el relé en conmutar.
- PASO 5: Descargamos el condensador durante 1.32 s. Por seguridad, esta será la configuración por defecto de las señales de control.

3.2. Circuito de control

El circuito de control es el encargado de interpretar las órdenes de la RPI para regular de manera aislada el selector de resistencias del transconductor de tensión y las distintas señales de control del circuito de medida.

3.2.1. Señales de control

Como hemos comentado en el apartado del circuito de medida, utilizaremos de 3 señales de control. No obstante, existe una cuarta señal auxiliar por si se necesita para alguna modificación del circuito en un futuro. Cada una de estas señales está asociada a un pin de salida de la RPI (Tabla 5), las cuales activaremos desde el programa de medida.

Señal	Asociada a
S1	GPIO 21
S2	GPIO 20
S3	GPIO 16
S4	GPIO 12

Tabla 5. Pines GPIO asociados a las señales de control

Puesto que los pines de salida GPIO de la RPI tienen un voltaje máximo de 3.3VDC, y que tanto los relés como el transistor se activan con 10VDC, se ha optado por

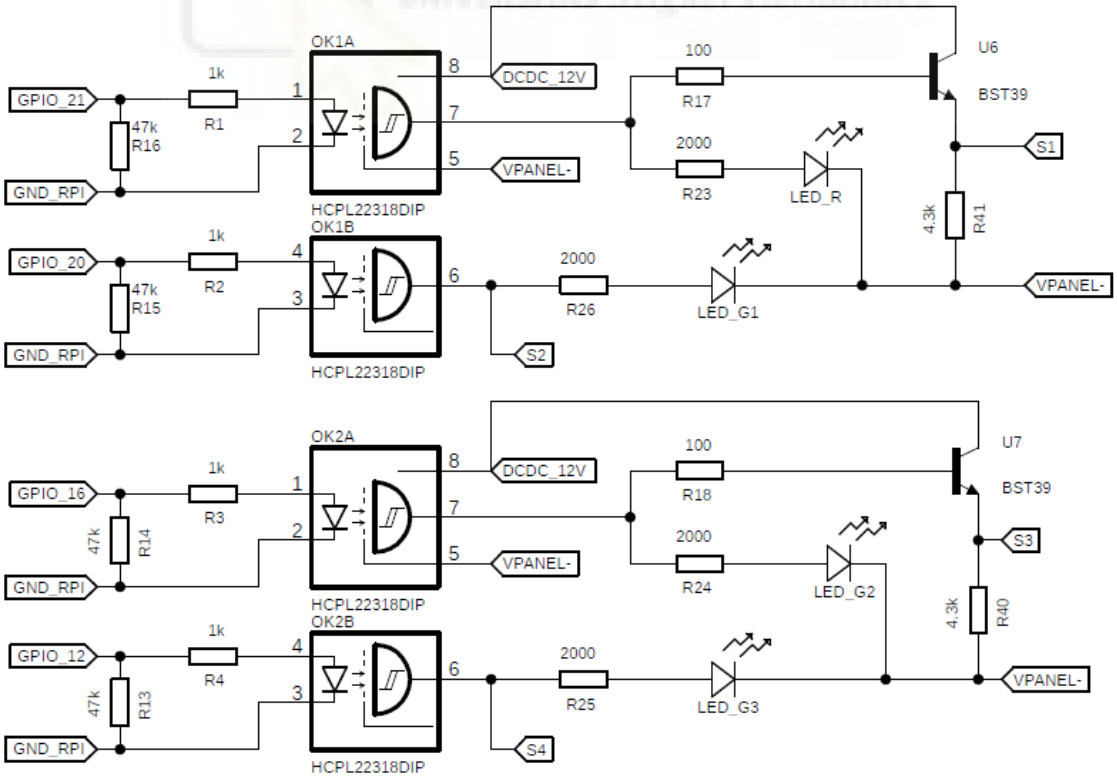


Figura 13. Esquema del circuito asociado a las señales de control

utilizar dos optoacopladores HCPL-2231-500E de la marca Broadcom Limited. También cabe destacar que, además de utilizarlos como interruptores, los usaremos como aislamiento eléctrico entre la RPI y el circuito de medida para aumentar la seguridad del circuito.

En la figura 13 podemos ver el esquema del circuito. Como se puede apreciar, cada señal tiene un diodo led asociado que se iluminará cuando estén activas. Además, en cada una de las señales de control asociadas a los relés, se ha introducido un transistor BJT, que se saturará cada vez que se active la señal y pondrá 12V en los conectores de la bobina del relé.

3.2.2. Selector de resistencias

Como hemos comentado anteriormente, se ha diseñado un selector de resistencias para el circuito primario del transconductor de tensión. Este circuito es similar al anterior, pues utiliza los mismos optoacopladores. No obstante, no se utilizan transistores BJT, ya que introducirían una pequeña corriente que alteraría la medida de la tensión. En su lugar se han colocado MOSFETs, en concreto el modelo BSS87H6327FTSA1 con una tensión nominal V_{DS} de 240V. Estos no introducen ninguna corriente, pero si presentan una resistencia entre el drenador y la fuente que deberemos tener en cuenta a la hora de calcular el factor de conversión del transconductor. El esquema del circuito del selector se puede ver en la figura 14.

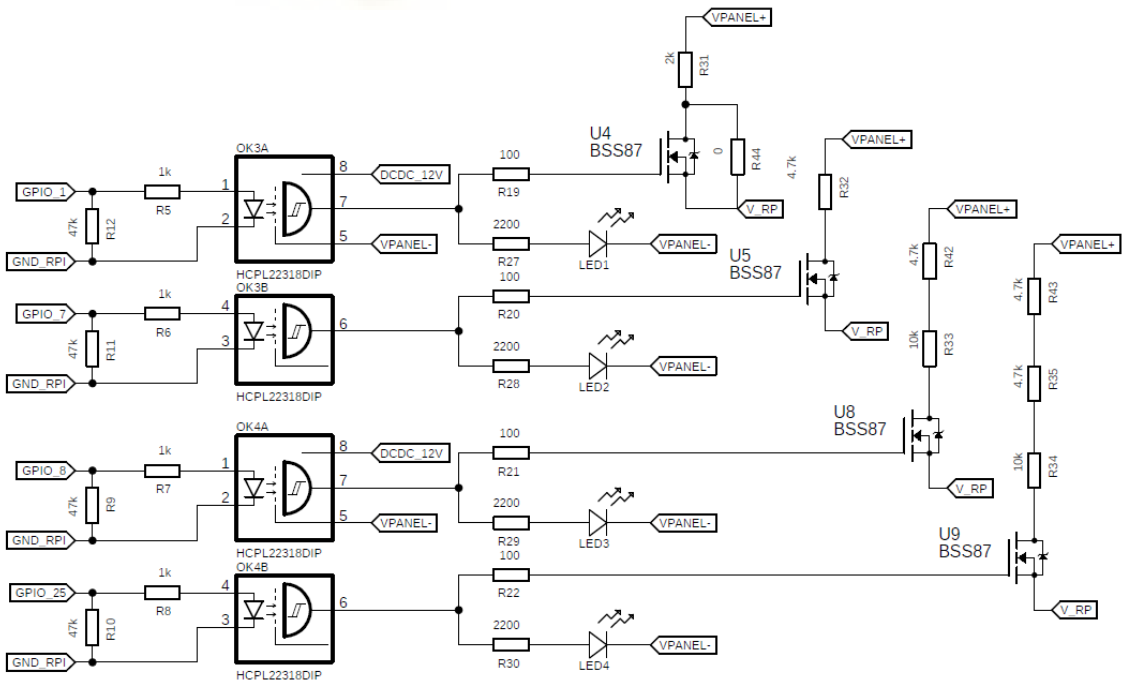


Figura 14. Esquema del circuito asociado al selector de resistencias

Tras realizar el soldado de todos los componentes del selector, se midió la resistencia real asociada a cada hilo. En la Tabla 6 se han representado todos los factores de conversión con sus respectivas resistencias teniendo en cuenta que el ratio de conversión dado por el fabricante es de 2500:1000.

R_1 [kΩ]	2	4.7	14.7	19.4
R_{TOTAL} [kΩ]	2.260	4.928	14.941	19.637
Vp [V]	22.60	49.28	149.41	196.37
Factor Kv	9.041	19.712	59.764	78.548

Tabla 6. Factores de conversión asociados a cada resistencia del selector

3.3. Adquisición de irradiancia y temperatura

Además de la tensión y la corriente asociadas a las curvas características de un panel solar, otros parámetros relevantes en la caracterización son la irradiancia y la temperatura. Ambos datos pueden adquirirse a través de sensores analógicos, los cuales emiten una señal instantánea comprendida entre dos valores (en nuestro caso entre 0V y 5V) proporcional al efecto que miden.

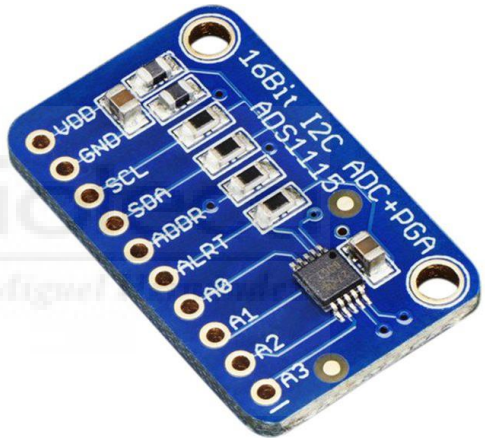


Figura 15. Convertidor AC/DC ADS1115 de Adafruit. Imagen extraída de la página del fabricante

Sin embargo, puesto que la RPI no dispone de entradas analógicas o de algún convertidor AC/DC que nos permita leer la salida de estos sensores, se ha optado por utilizar un convertidor analógico digital I2C. Se ha seleccionado el modelo ADS1115 del fabricante Adafruit, que funciona con conexión I2C y cuenta con una librería para RPI en Python. Este convertidor requiere de una tensión de alimentación entre 2V y 5V (en este caso se ha alimentado a 3.3V) y dispone de cuatro canales de entrada, de los cuales utilizaremos 3 (uno para el sensor de irradiancia y dos para la temperatura).

En cuanto a la resolución, cada uno de los canales en modo single ended (que será el modo que utilicemos) dispone de 16 bits, 15 para la medición y uno para el signo. Esto nos daría N valores posibles:

$$N = 2^{15} = 32768 \text{ valores posibles}$$

Por lo tanto, si ajustamos el PGA (ganancia) a 1, ya que por defecto es 2/3, tendremos una escala completa (FS) de:

$$\mathrm{FS} = \pm\frac{4.096}{\mathrm{PGA}} = \pm\frac{4.096}{1} = \pm 4.096\ [\mathrm{V}]$$

A pesar de que el FS sea de ±4.096 V, las entradas a los canales no podrán exceder:

$$\mathrm{VDD} + 0.3 = 3.3 + 0.3 = 3.6\ [\mathrm{V}]$$

No es lo mismo la referencia del PGA que el FS, y si excedemos este voltaje, podríamos dañar el chip. Finalmente, el factor de escala (SF) que tendremos con la configuración que hemos elegido será de:

$$\mathrm{SF} = \frac{4.096}{32767} = 0.000125\ [\mathrm{V}] = 0.125\ [\mathrm{mV}]$$

Una vez conocido el factor de escala y los rangos de los que dispondremos, pasaremos a exponer los distintos sensores que conectaremos al ADS1115.

3.3.1. Sensor de irradiancia

Para la medida de la irradiancia se ha seleccionado el sensor 7821 del fabricante DAVIS. Según su hoja de características, tiene 3 conexiones. Por un lado, el cable negro, que corresponde con la entrada de alimentación que será de 5 V DC, por lo que tendremos que incluir un terminal de dicho voltaje en nuestra PCB. Por otro lado, el cable rojo y verde, el cual será la referencia GND. Y por último, el cable amarillo, que corresponde con la salida del sensor. Esta posee un rango de 0 a 3 VDC donde cada 1.67 mV corresponden a 1 W/m2, que se traduce en un rango de 0 W/m2 a:

$$\frac{3000\ \mathrm{mV}}{1.67\ \mathrm{mV}} = 1796.40\ [\mathrm{W/m2}]$$

Teniendo en cuenta el SF del convertidor, el factor KG que utilizaremos será:

$$1\ \text{mV} = \frac{1}{1.67} = 0.5988\ \text{W/m2} \rightarrow \text{K}_\text{G} = \frac{0.5988}{0.125} = 4.79$$

Finalmente, el factor de escala con el que podremos medir la irradiancia será de 0.07485, que nos da una precisión de un decimal. En cuanto a las conexiones del sensor con la PCB se muestran en la tabla 7.

Conector	Conectado a
X3-1	5V (Negro)
X3-2	GND (Rojo y verde)
X3-3	Salida (Amarillo)

Tabla 7. Conexiones del sensor de irradiancia con la PCB

3.3.2. Sensor de temperatura

Para la adquisición de la temperatura se ha optado por utilizar el sensor LM35. Al igual que el sensor de irradiancia, estará alimentado a 5V y referenciado al GND de la RPI. El factor de escala en este componente es lineal con unos 10 mV/ºC. Por lo tanto, si tenemos en cuenta que somos capaces de medir la tensión con un factor de escala de 0.125 mV, tendremos una precisión de 0.0125ºC.

En cuanto a las conexiones, en la figura 16, extraída de la hoja de datos del sensor, podemos ver a qué corresponde cada pin según su vista inferior. Y en la tabla 8 se muestran las conexiones con la PCB.

Conector	Conectado a
X3-1	5V
X3-2	GND
X3-4	Salida sensor 1
X3-5	Salida sensor 2

Tabla 8. Conexiones de los sensores de temperatura con la PCB

Figura 16. Esquema del sensor LM35 extraído de su datasheet

3.4. Analog Discovery

Para adquirir las señales de tensión del devanado secundario de los transconductores se ha optado por utilizar el osciloscopio por USB Digilent Analog Discovery 2 (AD2). Este dispositivo, como veremos posteriormente, es lo suficientemente potente para desarrollar esta tarea, y lo suficientemente pequeño para poder incluirlo dentro del dispositivo de medida.

Figura 17. Imagen del Analog Discovery 2 extraída de la página del fabricante

El AD2 tiene entradas y salidas tanto analógicas como digitales. En nuestro caso, utilizaremos únicamente los dos canales de osciloscopio que posee. Estos tienen una alta impedancia de entrada de 1MΩ y pueden medir tensiones de ±25 V, que es suficiente para poder medir paneles de hasta 200 V de tensión de circuito abierto con cualquiera de las resistencias del selector y hasta 20 A de corriente de cortocircuito que pueden soportar los relés.

En cuanto a la frecuencia de muestreo, es capaz de llegar a medir 100 MS/s. No obstante, cada canal posee un buffer con una capacidad máxima de 16k muestras, y teniendo en cuenta que necesitaremos medir durante al menos media décima de segundo, nunca se llegará a medir a más de 320 kHz. Además, si intentamos aumentar la frecuencia tomando varias medidas seguidas de 16k muestras, nos veremos condicionados por la comunicación USB 2.0 con la RPI. Esto es debido a que la velocidad máxima de transmisión de datos es de 12 Mbps y afectaría a la frecuencia de actualización.

En referencia a la resolución y a la precisión, tiene dos opciones: 0.32 mV con ± 0.05 mV ,y 3.58 mV con ± 0.5 mV. Teniendo en cuenta que la resolución con la que trabaja el AD2 es de 14 bits, los rangos de medida para cada resolución son de:

$$\text{Rango}_1 = \pm 2^{14-1} \cdot 0{,}32 \cdot 10^{-3} = \pm 2{,}62144\ [\text{V}]$$

$$\text{Rango}_2 = \pm 2^{14-1} \cdot 3{,}58 \cdot 10^{-3} = \pm 29{,}32736\ [\text{V}] \rightarrow \pm\ 25\ [\text{V}]$$

Por lo tanto, los rangos con cada resolución para cada resistencia del selector serían los de la Tabla 9. Tras realizar varias pruebas con las dos resoluciones, se optó por utilizar siempre el rango 1.

Resistencia [Ω]	Rango 1 [V]	Rango 2 [V]
2k	± 23.70	± 226.03
4.7k	± 51.67	± 492.80
14.7k	± 156.67	± 1494.10
19.4k	± 205.91	± 1963.70

Tabla 9. Rangos de medida de cada precisión asociados a cada resistencia

En relación con las conexiones del AD2, tanto la alimentación como la comunicación de datos con la RPI se realiza por conexión USB 2.0, mientras que los dos canales del osciloscopio se conectan a la PCB a través de los pines mediante cables simples. En la figura 18, la cual ha sido extraída de la página del fabricante, se pueden ver la disposición de los pines del AD2.

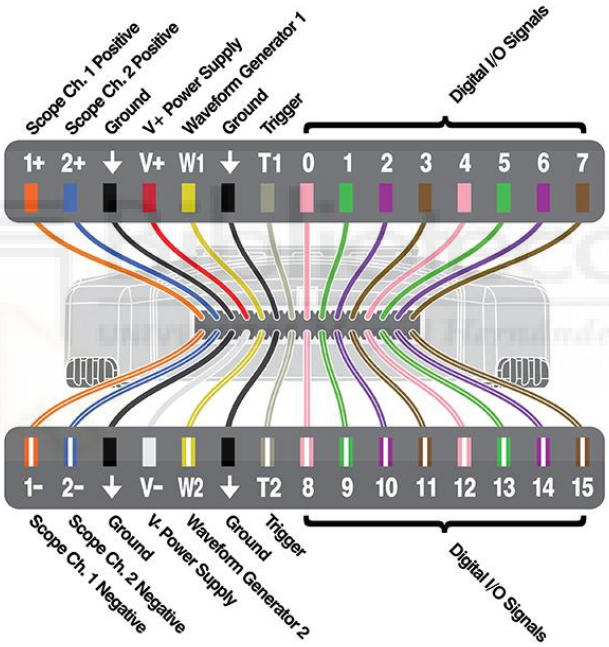


Figura 18. Esquema de pines del Analog Discovery 2 extraído de la página del fabricante

3.5. Raspberry Pi

Una de las particularidades del dispositivo de medida de este proyecto es que es completamente portátil. Es por este motivo que se ha elegido una Raspberry Pi 4 Modelo B de 8GB de RAM (Figura 19) como PC.

Figura 19. Imagen de la Raspberry Pi 4 Model B extraída de la página del fabricante

Otra de las ventajas de la RPI es el encabezado GPIO de 40 pines, para el que existe una librería desarrollada en Python para manejar las entradas y salidas. Gracias a él se podrá regir el circuito de control y se podrá establecer una comunicación I2C con el convertidor analógico digital para obtener la irradiancia y la temperatura. En la figura 20 se pueden ver las distintas conexiones que se han realizado en este proyecto.

Es muy importante no conectar nada en los pines GPIO 0 y GPIO 1, ya que podría dar problemas en el arranque.

En cuanto a la parte de potencia, existen dos maneras de alimentarla. Por un lado, se puede conectar desde la toma USB tipo C a la red eléctrica con un adaptador de corriente de 5V a 3A . Por otro lado, se puede alimentar directamente por los pines GPIO. Se ha optado por esta última alternativa debido a que de esta manera se puede alimentar desde la PCB con un convertidor AC/DC que expondremos más adelante.

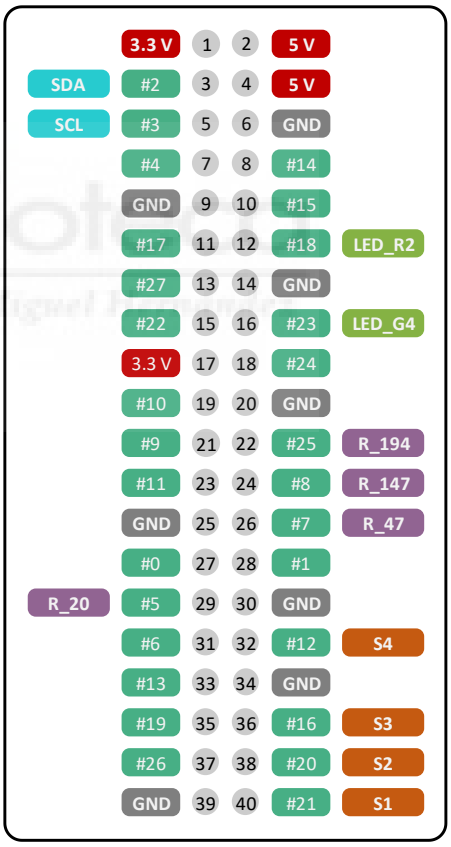


Figura 20. Conexiones de los pines GPIO la PCB

3.6. Alimentación

Como acabamos de comentar, el circuito de control se ha diseñado para que funcione a 12V, y la RPI funciona a 5V. Aprovechando esta situación, para alimentar este circuito se ha utilizado un convertidor AC/DC con salidas de 5V y 12V. Concretamente es el modelo TMP 30252C de la marca TRACO POWER (Figura 21). Gracias a este componente, solo habría que poner una toma hembra IEC C13 en la carcasa del dispositivo para poder alimentarlo. Las conexiones con la placa de este componente se pueden ver en la tabla 10.

Figura 21. Convertidor TMP 30252C de TRACO POWER. Imagen extraída de la web de la tienda DigiKey

Conector	Cableado a
X2-1	+Vout1
X2-2	-Vout1
X2-3	+Vout2
X2-4	-Vout2

Tabla 10. Conexiones del convertidor AC/DC a la PCB

Como se puede ver en el esquema de la figura 22, para aumentar la seguridad en el circuito, se ha colocado un fusible de 2A en la parte que trabaja a 12V para evitar

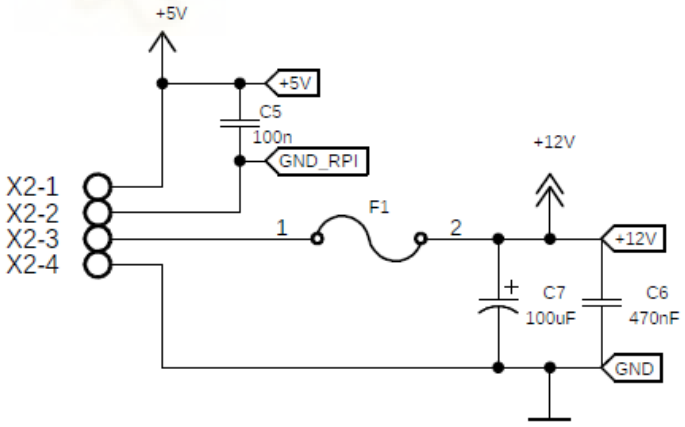


Figura 22. Esquema de la entrada de alimentación de la PCB

daños en caso de que hubiera algún cortocircuito. Además, se han añadido un condensador electrolítico de 100uF y otro cerámico de 470 nF (ambos SMD) para estabilizar la alimentación.

Esta tensión de 12 V se utiliza para alimentar dos convertidores DC/DC. Por un lado, el DCP021212U (Figura 23), que es el encargado de aislar la entrada del circuito con los componentes que funcionan a esta tensión y están conectados a la RPI.

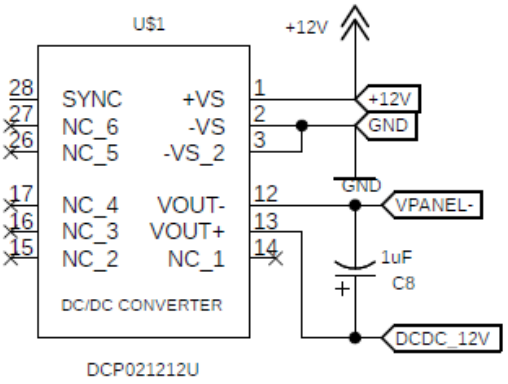


Figura 23. Esquema de las conexiones del convertidor DC/DC DCP21212U

Y por otro lado, el TEN 5-1223, que es el encargado de alimentar los dos transconductores que hemos comentado antes a ±15V. Además, como se puede ver en el esquema de la figura 24, se han añadido varios condensadores cerámicos de 100 nF en las entradas de alimentación de todos los componentes que estaban conectados a la salida de este convertidor.

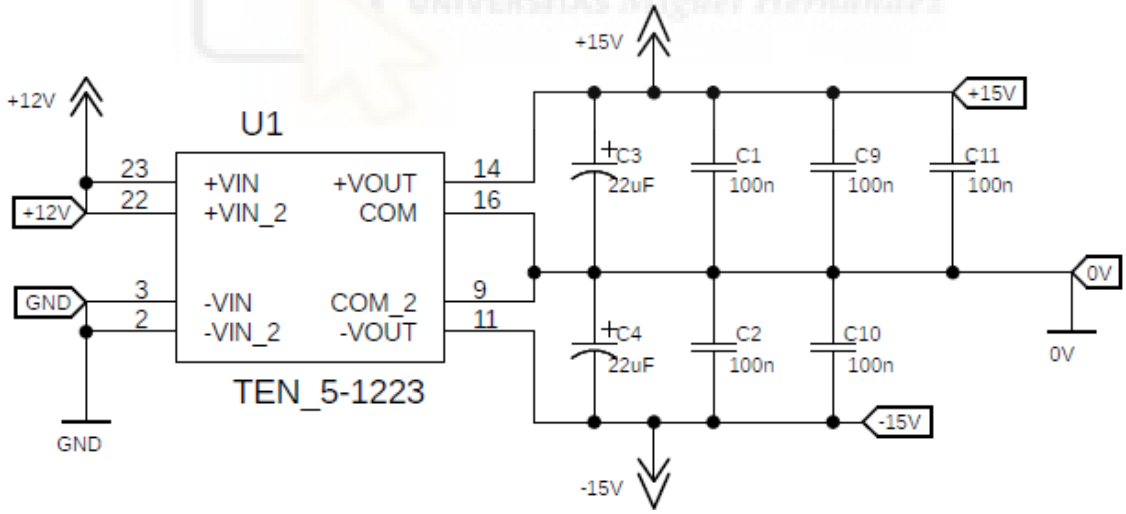


Figura 24. Esquema del circuito asociado al convertidor TEN 5-1223

3.7. Diseño de la PCB y ensamblaje del dispositivo

Una vez desarrollada teóricamente toda la parte de hardware, se ha procedido al diseño de la placa y al ensamblaje del dispositivo de medida.

3.6.1. Diseño de la PCB

Para el diseño de la PCB de nuestro proyecto se ha utilizado el software EAGLE de AutoDesk. Para poder comenzar con el diseño, se ha creado una librería que incluye todos los componentes que se incluyen en el diseño. Algunos de ellos ya venían incluidos en las librerías que EAGLE tiene por defecto, mientras que otros se han tenido que diseñar a acorde con la hoja de datos del componente. Una vez creada la librería, se ha realizado el esquemático del circuito. Puesto que ocupa bastante espacio, se ha dejado adjunto en el ANEXO II.

Tras completar el esquemático, se ha pasado a diseñar el layout de la PCB. Debido al tamaño de los conectores, los relés, los transconductores y los convertidores DC/DC, se ha hecho el diseño en una placa de 125x250mm. En vista del ensamblaje posterior, se ha optado por colocar en la cara superior todos los componentes más voluminosos y los leds indicadores, mientras que los más pequeños que no son THT, en la cara inferior. De esta manera, reducimos la altura total de la PCB ensamblada y ahorramos espacio en la caja.

También cabe destacar que se han posicionado todos los componentes del circuito de medida, los cuales estarán sometidos a una alta potencia, en una misma zona. De esta forma, reduciremos lo máximo posible el camino que recorre la corriente del panel dentro de la placa. Además, se ha aumentado el tamaño de estas pistas todo lo posible para reducir el calor que se pueda generar.

En las figuras 25 y 26 se muestra el diseño final de ambas caras de la placa.

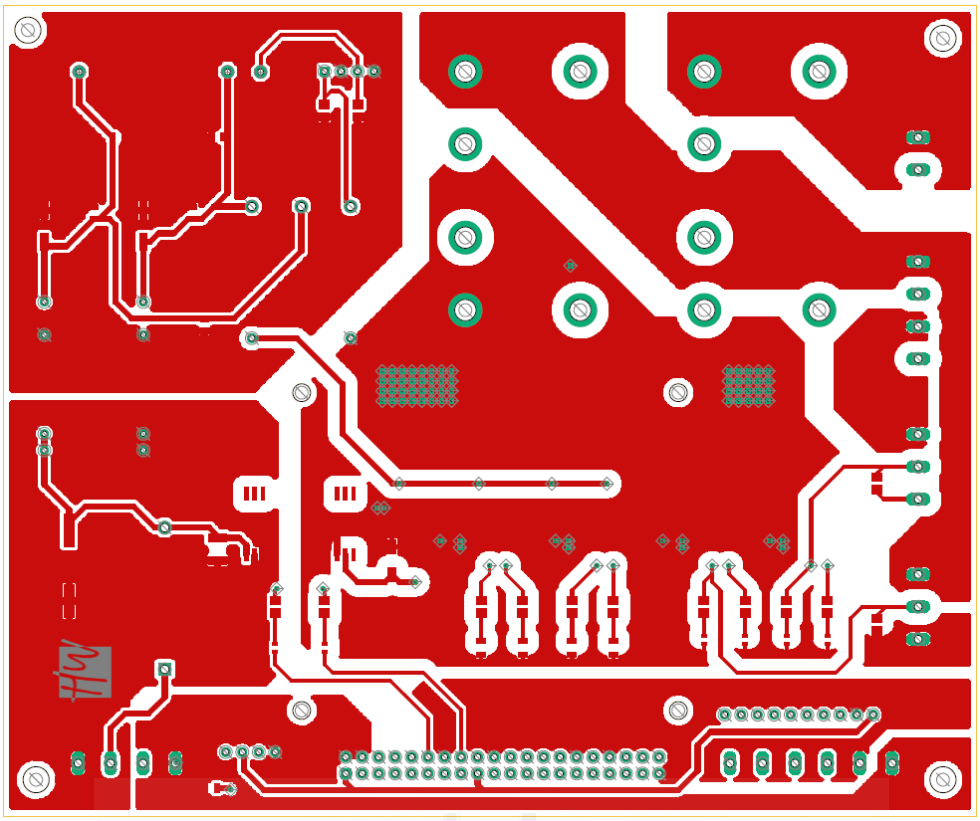


Figura 25. Vista de la cara superior de la placa

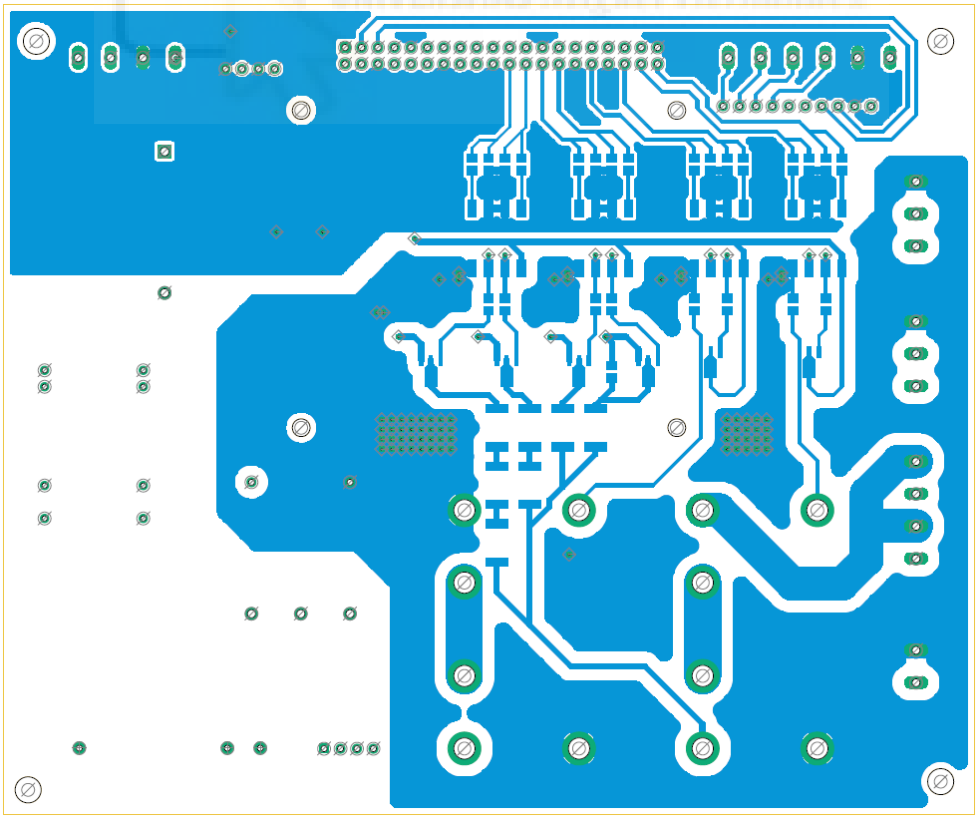


Figura 26. Vista de la cara inferior de la placa

Finalmente, tras completar el diseño, se generaron los gerber del proyecto de EAGLE. Tanto los del cobre como los de la serigrafía de ambas caras se han adjuntado en el ANEXO I de este documento.

Después de diseñar la PCB del dispositivo, se ha mandado a fabricar a la empresa JLCPCB (https://jlcpcb.com/) que vende un servicio de fabricación de prototipos de PCB. En las figuras 27 y 28 podemos ver las placas ya fabricadas.

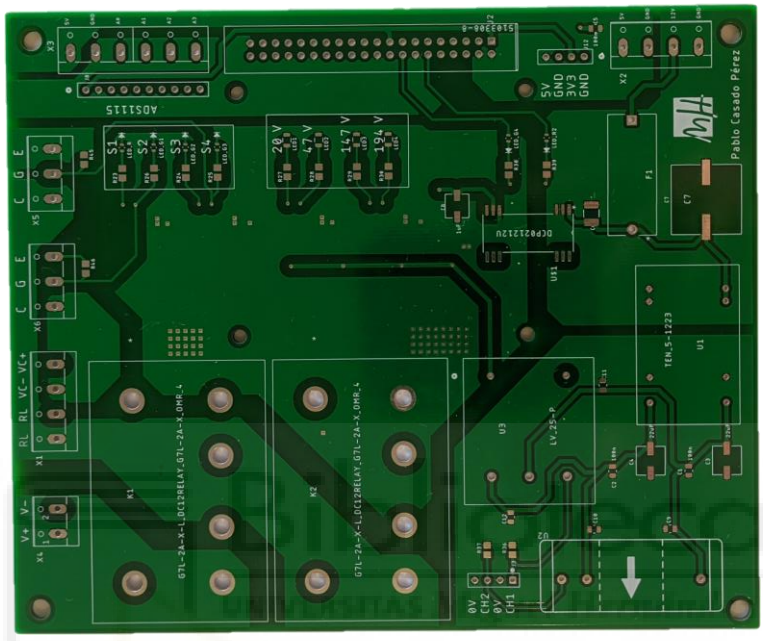


Figura 27. Vista superior de la PCB ya fabricada

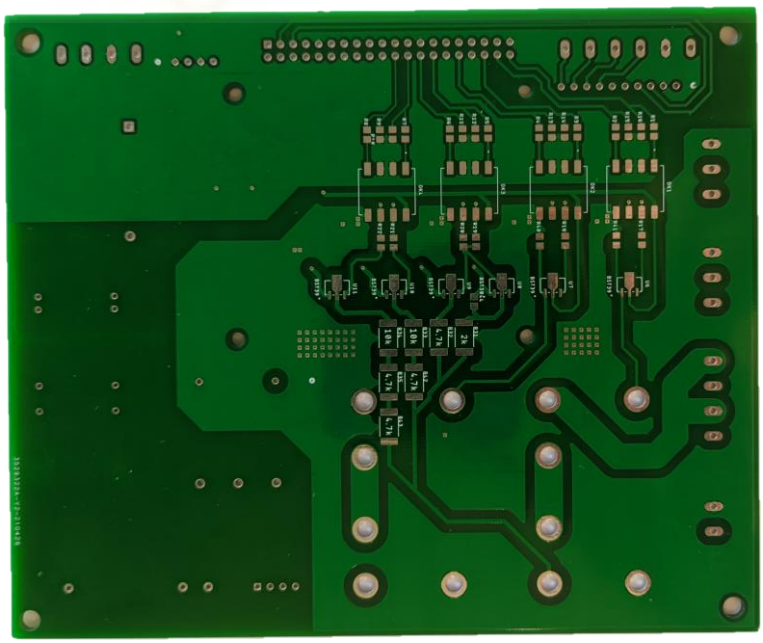


Figura 28. Vista inferior de la PCB ya fabricada

3.6.2. Ensamblaje del dispositivo

El proceso de montaje y soldadura de todos los componentes sobre la PCB se realizó en el laboratorio del edificio Torrevaillo del área de Tecnología Electrónica del departamento de Ciencia de Materiales, Óptica y Tecnología Electrónica de la Universidad de Miguel Hernández. En las figuras 29 y 30 se pueden ver la vista superior e inferior de la placa ya terminada

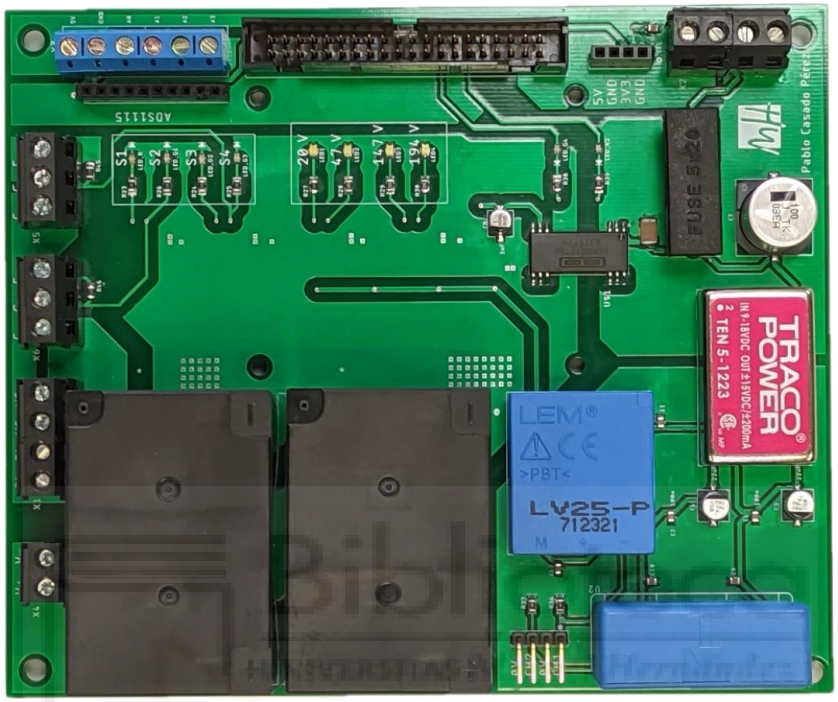


Figura 29. Vista superior de la PCB con todos los componentes soldados

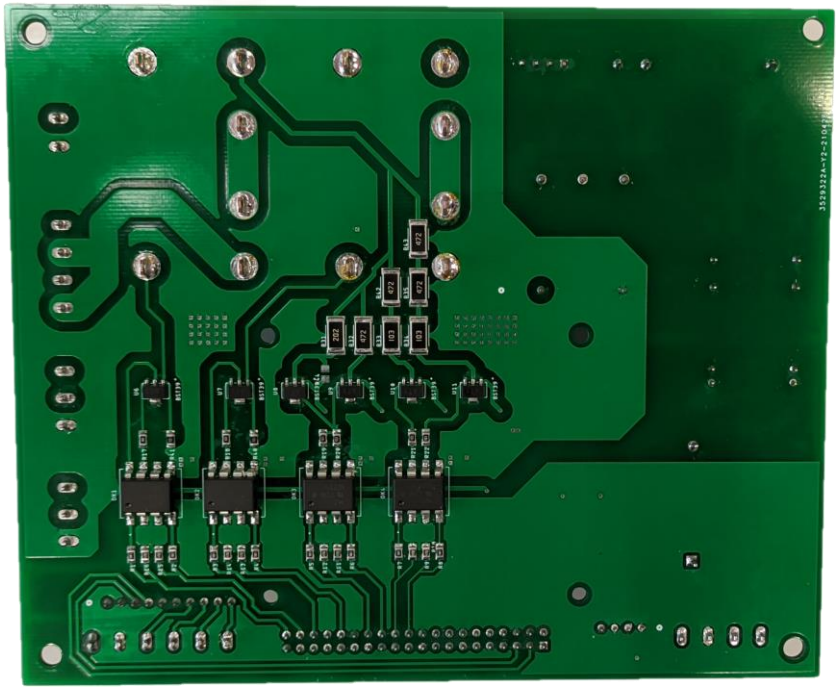


Figura 30. Vista inferior de la PCB con todos los componentes soldados

En la figura 31 se puede ver la PCB en perspectiva para apreciar el tamaño de los componentes THT del circuito.

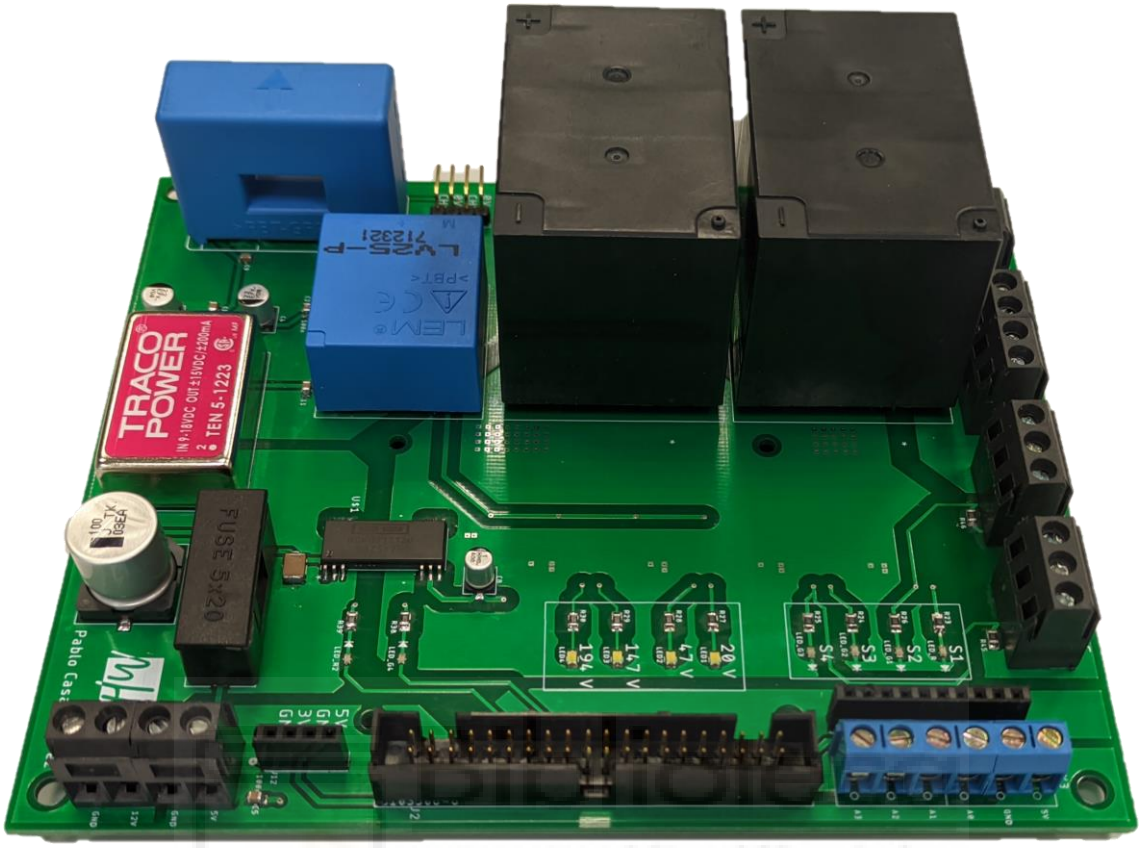


Figura 31. Vista general de la PCB terminada

Una vez ensamblada la placa con todos sus componentes, se ha procedido a realizar el mecanizado de la caja en la que posicionaremos todos los elementos del dispositivo. En primer lugar, se han realizado los planos de la tapa y de los dos laterales donde posicionaremos los distintos conectores. Estos se pueden consultar en el ANEXO III al final de este documento.

Tras realizar el mecanizado, se han ensamblado todos los componentes en la caja. En las figuras 32 y 33 se muestran los laterales con los conectores.

Figura 33. Vista de los conectores de potencia

Figura 33. Vista de los conectores de comunicación

Y por último, en las figuras 34 y 35 se muestra la vista superior del dispositivo con y sin tapa para poder visualizar la disposición interna de los componentes. Para facilitar la vista, no están colocados todos los cables en la figura 35.

Figura 34. Vista superior del dispositivo

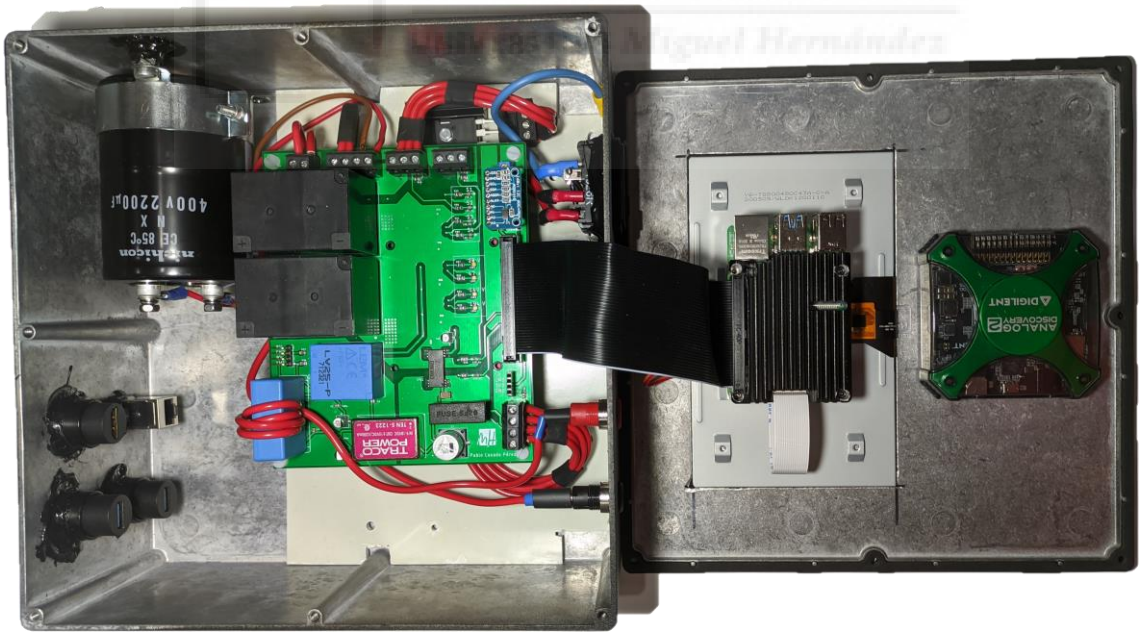


Figura 35. Vista del interior del dispositivo

CAPÍTULO 4. Diseño software

En cuanto al diseño software del caracterizador, se ha optado por utilizar el lenguaje de programación Python. Actualmente es uno de los lenguajes más utilizados y con mayor crecimiento. En nuestro caso, se ha seleccionado debido a que una de sus grandes virtudes es que es un lenguaje de programación interpretado, por lo que solo necesitaremos que la RPi integre su interpretador.

El SO Raspbian viene ya con su propia aplicación para programar y ejecutar tu propio script de Python. Esta es llamada "Thonny" (figura 36), y puede encontrarse en el apartado "Programación" del menú . No obstante, aunque es bastante fácil y sencilla de utilizar, se ha optado por trabajar de manera remota desde otro ordenador. Para ello, se ha establecido una conexión SSH con el programa "MobaXterm", desde el cual se han abierto los distintos scripts en el editor de código fuente "Sublime Text".

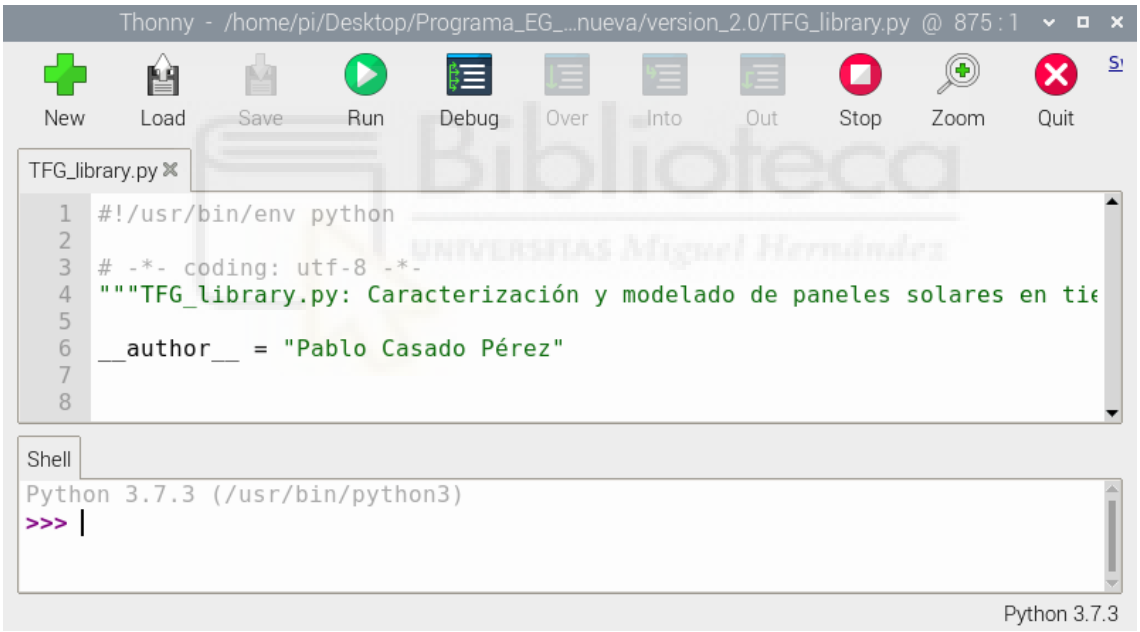


Figura 36. Captura de pantalla de la interfaz del programa Thonny

Tras conocer que lenguaje y que entorno se ha utilizado, en este capítulo explicaremos cómo se ha llevado a cabo la parte de software del proyecto.

4.1. Configuración de la Raspberry Pi

Antes de empezar con el desarrollo del programa que implementará el dispositivo, indicaremos los pasos previos a seguir para que no haya errores al ejecutarlo. En primer lugar, será necesario actualizar el SO Raspbian. Para ello, habrá que estar conectado a Internet y ejecutar los siguientes comandos en el terminal:

```
sudo apt update
sudo apt dist-upgrade
sudo apt clean
sudo reboot
```

Seguidamente, para que el código que se va a desarrollar en los siguientes apartados funcione correctamente, será necesario instalar en la RPI la versión 3 de Python. Antes de esto, es imprescindible descargar los paquetes de instalación PIP, por lo que la secuencia de comandos sería:

```
python get-pip.py
sudo apt install python3-pip
```

Posteriormente, se instalarán todos los paquetes que se necesitarán en el programa:

```
pip install matplotlib numpy
```

Tras instalar todos los paquetes necesarios, habilitaremos la comunicación I2C en la RPI siguiendo los siguientes pasos:

- PASO 1. Este paso se puede efectuar de dos maneras distintas. Por un lado, se podría realizar ejecutando desde la ventana del Terminal el comando:

  ```
  sudo raspi-config
  ```

  El cual nos mostraría en pantalla la herramienta raspi-config (Figura 37).

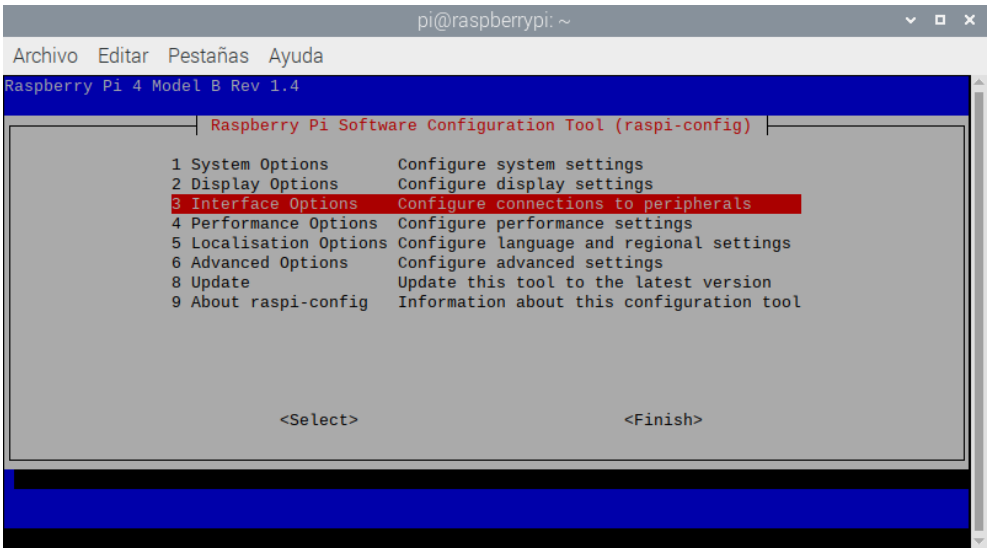


Figura 37. Captura de pantalla del raspi-config

Seleccionamos “Interfacing Options”, y después activamos la opción de I2C de la figura 38.

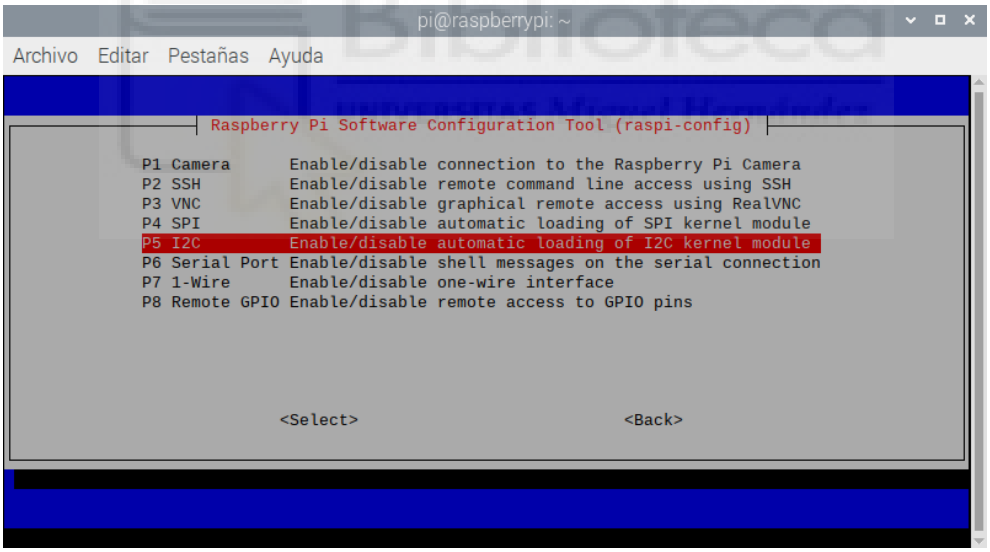


Figura 38. Captura de pantalla del raspi-config 2

Por otro lado, se puede acceder a la configuración de la RPI desde Menú> Preferencias> Configuración de Raspberry Pi. Una vez dentro de la configuración se seleccionaría la pestaña “Interfaces” y se activaría el I2C como en la figura 39.

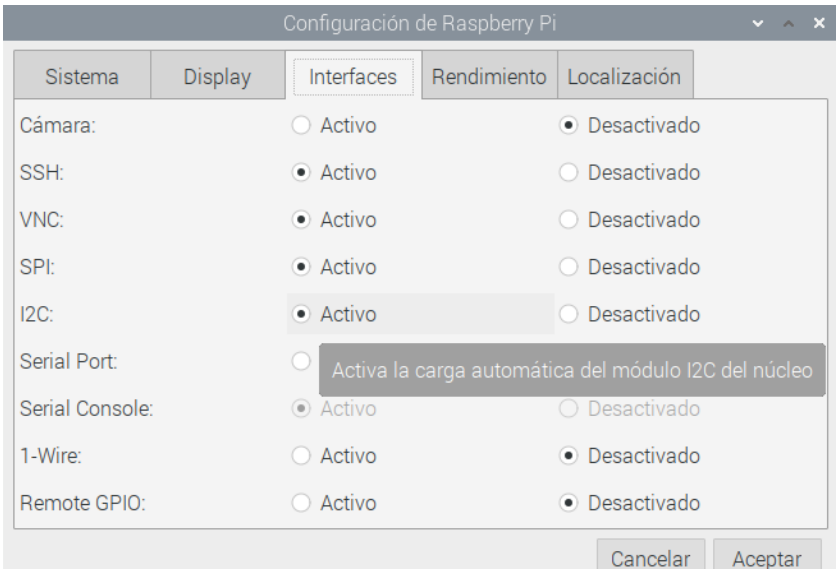


Figura 39. Captura de pantalla de la ventana de configuración de la Raspbian

- PASO 2. Se instala "python-smbus" e "i2c-tools" con los comandos:

```
sudo apt-get update
sudo apt-get install -y python-smbus i2c-tools
```

- PASO 3. Se reinicia la RPI
- PASO 4. Se comprueba si el I2C se ha habilitado correctamente con el comando:

```
lsmod | grep i2c_
```

- PASO 5. Se comprueba que el cableado está puesto correctamente con el comando:

```
i2cdetect -y 1
```

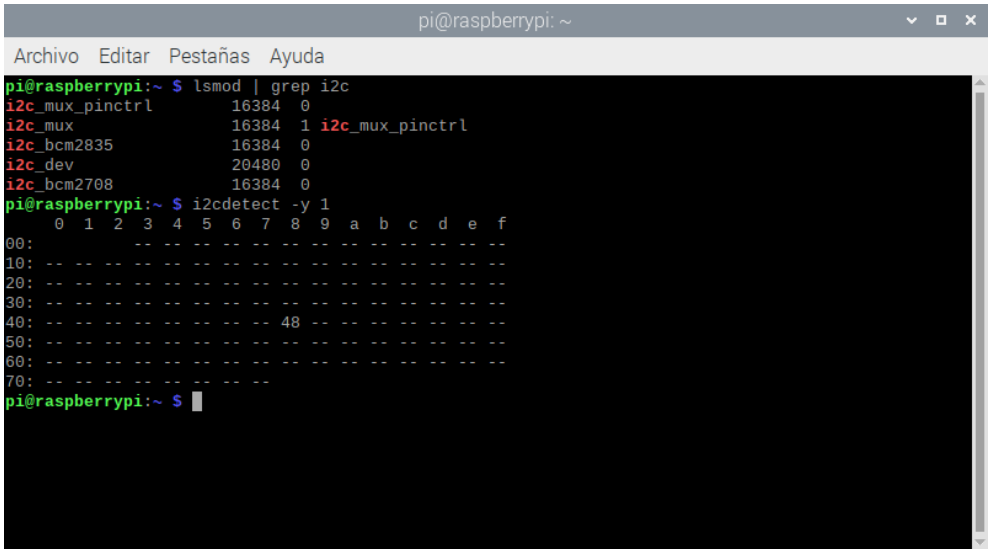


Figura 40. Captura de pantalla del terminal

Tras ejecutar el comando podemos ver que dispositivo está conectado y cuál es su dirección en hexadecimal.

A continuación, para poder hacer uso del AD2 será necesario acudir a la página de Digilent y descargar e instalar “Adept Runtime” para ARM y el software “WaveForms” para ARM respectivamente. Gracias a esto podremos hacer uso del SDK para Python que viene incluido con el programa.

También cabe destacar que, para el desarrollo del programa de manera remota, se ha tenido que activar la opción de SSH desde el menú de preferencias además de crear un servidor local en la RPI con la secuencia de comandos:

```
openssh-server
aptitude install openssh-server
```

4.2. Programa en Python

Este apartado lo dividiremos en tres partes. En primer lugar, explicaremos cuál es la estructura que siguen los programas que utilizan este dispositivo. En segundo lugar, explicaremos la librería que se ha creado. Y en tercer y último lugar, presentaremos el código de un ejemplo de uso de esta librería.

4.2.1. Estructura del programa

Con el fin de que el dispositivo de medida que se ha diseñado en este Trabajo Final de Grado pueda utilizarse de varias maneras posibles, se ha creado una librería. De esta manera, se evitará duplicidad de código y se tendrán funcionalidades comunes, reduciendo el tiempo de desarrollo para nuevas interfaces. Hasta el momento, se han llevado a cabo cuatro interfaces distintas (figura 41).

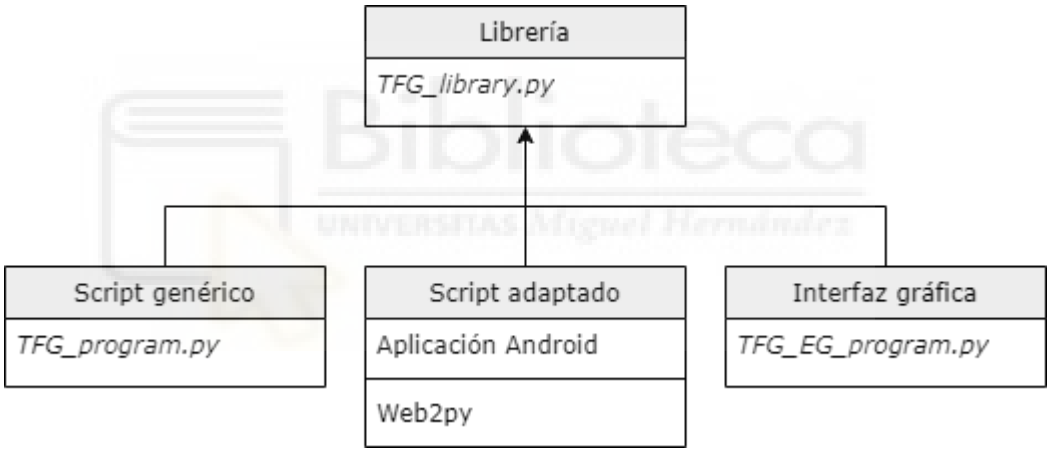


Figura 41. Estructura de los distintos programas

Por un lado, se ha programado un script de Python genérico (*TFG_program.py*), el cual realiza N medidas cada X tiempo en función del valor que le demos a unas constantes en el código. Esta sería la manera más sencilla, aunque menos cómoda, de realizar medidas con el dispositivo. El código de este fichero está explicado en el subapartado 4.2.3.

Por otro lado, se pueden generar scripts similares al anterior pero adaptados a las necesidades de la interfaz gráfica que queramos usar. En este caso, se ha desarrollado una aplicación Android que lanza de manera remota un script adaptado en el dispositivo de medida para que este le devuelva los valores al smartphone de manera correcta. Otro ejemplo de este tipo sería crear un entorno web con “Web2py” desde el que se lanzara un script adaptado.

Por último, otra alternativa que también se ha llevado a cabo es la elaboración de una interfaz gráfica en Python con la biblioteca gráfica Tkinter. En el apartado 4.3 de este capítulo se explica más detalladamente.

5.2.2. Variables, clases y funciones de la librería

Una vez explicada la estructura del programa, se procederá a desglosar la librería con las variables, clases y funciones que se utilizan en los scripts de Python del programa de medida y de la interfaz gráfica.

En primer lugar, se han incluido dos comentarios especiales que necesariamente tienen que estar en las dos primeras líneas del scrcipt. En el primer comentario se especifica qué interprete se va a utilizar (en nuestro caso, Python), y en el segundo se declara la codificación del archivo:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""TFG_library.py: Caracterización y modelado de paneles solares
en tiempo real."""

__author__ = "Pablo Casado Pérez"
```

En segundo lugar, se han incluido las librerías que se utilizarán a lo largo del script:

```
from ctypes import cdll, c_int, byref, create_string_buffer,
c_double
import sys
import RPi.GPIO as gpio
import time
import numpy as np
import Adafruit_ADS1x15
from datetime import datetime
import matplotlib.pyplot as plt
from matplotlib.ticker import AutoMinorLocator
import os
import warnings

# Definimos la librería DWF DLL en función del sistema operativo
if sys.platform.startswith("win"):
    dwf = cdll.dwf    # Windows
else:
    dwf = cdll.LoadLibrary("libdwf.so")   # Linux
```

En tercer lugar, se han declarado todas las constantes que ayudarán a simplificar el código:

```
PIN_S1 = 40     # Pin asociado a la señal S1
PIN_S2 = 38     # Pin asociado a la señal S2
PIN_S3 = 36     # Pin asociado a la señal S3
PIN_S4 = 32     # Pin asociado a la señal S4

PIN_R20 = 29      # Pin asociado a la resistencia para 20 V
PIN_R47 = 26      # Pin asociado a la resistencia para 47 V
PIN_R147 = 24      # Pin asociado a la resistencia para 147 V
PIN_R194 = 22      # Pin asociado a la resistencia para 194 V

C = 0.0022      # Capacidad del condensador
SAMPLES = 16000  # Número de muestras por canal

# Tasa de conversión de la tensión -> 2500:1000
KV_194 = 78.548 # 194 V (19.4K)
KV_147 = 59.764 # 147 V (14.7K)
KV_47 = 19.712  # 47 V (4.7K)
KV_20 = 9.04128 # 20 V (2K)

KI = 1.4  # 5 vueltas y resistencia de 100R

FS = 4.096/32767 # Factor de escala del convertidor ADC
[V/cuentas]
KG = 1000*1/1.67  # Factor del sensor de irradiancia para pasar
de mV a W/m2
KT = 100     # Factor del sensor de temperatura LM35 para pasar
de V a ºC

N_VECTOR = 100  # Número de puntos que se guardan en el fichero

str_action = '' # String para guardar la acción que se está
realizando
```

Para que el código resulte más sencillo y para facilitar la elaboración de una interfaz gráfica, se ha realizado una estructura modular con dos clases. Una clase está destinada a una medida, donde se incluyen todos los datos referentes a esta, además de funciones que nos devolverán dichos datos:

```
class Measure:
    # Tensión e intensidad
    V = np.array(SAMPLES)
    I = np.array(SAMPLES)
    P = np.array(SAMPLES)

    # Tensión e intensidad con N puntos
    V_n = np.zeros(N_VECTOR)
```

```
    I_n = np.zeros(N_VECTOR)
    P_n = 0

    # Tensión e intensidad filtrada
    V_valid = np.array(1)
    I_valid = np.array(1)
    P_valid = np.array(1)

    V_invalid = np.array(1)
    I_invalid = np.array(1)
    P_invalid = np.array(1)

    # Parámetros
    filename = ""   # Nombre del fichero
    date = ""   # Fecha de la medida
    hour = ""   # Hora de la medida
    Voc = 0     # Tensión de circuito abierto
    Isc = 0     # Corriente de cortocircuito
    Pmax = 0    # Máxima potencia
    Vmp = 0     # Tensión de máxima potencia
    Imp = 0     # Corriente de máxima potencia
    FF = 0      # Factor de forma
    G = 0       # Irradiancia
    Eff = 0     # Eficiencia
    T1 = 0      # Temperatura 1
    T2 = 0      # Temperatura 2

    # Constantes
    Kv = KV_194 # Factor del transconductor de tensión para la
medida

    # Función para imprimir por pantalla los parámetros de la
medida
    def Print_Measure(self):
        print ("\n - Parametros:")
        print('\t\tHora : '+ self.hour)
        print('\t\tVoc = ' + str(self.Voc) + ' [V]')
        print('\t\tIsc = ' + str(self.Isc) + ' [A]')
        print('\t\tPmax = ' + str(self.Pmax) + ' [W]\t\t(Vmp,
Imp) = ('+ str(self.Vmp) + ',' + str(self.Imp) + ')')
        print('\t\tFF = ' + str(self.FF))
        print('\t\tG = ' + str(self.G) + ' [W/m2]')
        print('\t\tEficiencia = ' + str(self.Eff * 100) + ' %')
        print('\t\tTemperatura 1 = ' + str(self.T1) + ' ºC')
        print('\t\tTemperatura 2 = ' + str(self.T2) + ' ºC')

    # Función para generar un string con los parametros de la
medida
    def toString(self):
        S_Measure =  '\n' + self.date + '\t' + self.hour + '\t'
```

```
+ '{}'.format(self.Voc) + '\t' + '{}'.format(self.Isc) + '\t' +
'{}'.format(self.Pmax) + '\t' +
'{}'.format(self.Vmp).replace("[","").replace("]","") + '\t' +
'{}'.format(self.Imp).replace("[","").replace("]","") + '\t' +
'{}'.format(self.FF) + '\t' + str(self.G) + '\t' +
'{}'.format(self.T1) + '\t' + '{}'.format(self.T2)#+
'{}'.format(self.Eff) + '\t'
        return S_Measure
```

La otra clase está destinada al AD2. En ella se inicializa el dispositivo, y será necesario recurrir a ella cuando se quiera realizar alguna modificación en futuras medidas:

```
class AD2():

    hdev = c_int()          # Dispositivo
    t_muestreo = 0.05        # Tiempo de muestreo o tiempo de
carga del condensador
    Sample_rate = 1000000   # Frecuencia de muestreo

    def __init__(self):

        version = create_string_buffer(16)
        dwf.FDwfGetVersion(version)

        # Consultar el número de AD2 conectados a la RPi
        cdevices = c_int()
        dwf.FDwfEnum(c_int(0), byref(cdevices))

        # Abrimos el AD2
        self.hdev = c_int() # Declaramos un dispositivo AD2
        #dwf.FDwfDeviceOpen(c_int(0), byref(self.hdev))     #
Configuración para hasta 8k muestras por canal
        dwf.FDwfDeviceConfigOpen(c_int(0), c_int(1),
byref(self.hdev))     # Configuración para hasta 16k muestras
por canal

        # Configuramos las entradas analógicas
        dwf.FDwfAnalogInFrequencySet(self.hdev,
c_double(self.Sample_rate)) # Fijamos la frecuencia de muestreo
del AD2
        dwf.FDwfAnalogInChannelFilterSet(self.hdev, c_int(-1),
c_int(1))    # Configuramos de manera individual cada canal para
que alcancen la máxima frecuencia

        # Fijamos los dos canales con la mayor precisión posible
        dwf.FDwfAnalogInChannelRangeSet(self.hdev, c_int(-1),
c_double(4))  # El 4 del tercer elemento marca la máxima
precisión que nos permite el AD2
```

```
    # Función para cambiar la frecuencia de muestreo
    def Set_samplerate(self, Voc, Isc):

        #t_muestreo = C*Voc/Isc                  # Cálculo del
tiempo de carga del condensador
        Sample_rate = int(SAMPLES/t_muestreo)  # Como máximo
puede ser 1 [MHz]

        dwf.FDwfAnalogInFrequencySet(self.hdev,
c_double(Sample_rate))
```

Tras las constantes y las clases, se han declarado todas las funciones necesarias que simplificarán el código. Se ha comenzado con la más importante para la seguridad del dispositivo, que es la función que fija los pines de la RPi. Gracias a ella obtenemos el control de los pines que controlan la carga y descarga del condensador, y los que controlan el selector de resistencias (estarán apagados por defecto para aumentar la vida útil del transconductor):

```
def Set_GPIO():
    gpio.setmode(gpio.BOARD)
    gpio.setup(PIN_S1, gpio.OUT)    # Fijamos la salida del pin
asociado a S1
    gpio.setup(PIN_S2, gpio.OUT)    # Fijamos la salida del pin
asociado a S2
    gpio.setup(PIN_S3, gpio.OUT)    # Fijamos la salida del pin
asociado a S3

    gpio.setwarnings(False)
    gpio.setup(PIN_R20, gpio.OUT)   # Fijamos la salida del pin
asociado a R20
    gpio.setup(PIN_R47, gpio.OUT)   # Fijamos la salida del pin
asociado a R47
    gpio.setup(PIN_R147, gpio.OUT)  # Fijamos la salida del pin
asociado a R147
    gpio.setup(PIN_R194, gpio.OUT)  # Fijamos la salida del pin
asociado a R194

    # Inicializamos el selector de resistencias para que no pase
corriente por ninguna
    gpio.output(PIN_R20, False)
    gpio.output(PIN_R47, False)
    gpio.output(PIN_R147, False)
    gpio.output(PIN_R194, False)
```

La siguiente función también es de suma importancia para la seguridad del dispositivo, pues es la encargada de poner a descargar el condensador:

```
def Discharge_capacitor():
    gpio.output(PIN_S1, False)
    gpio.output(PIN_S2, True)
    gpio.output(PIN_S3, True)
```

Esta función se encarga de deshabilitar el selector de resistencias en el momento que se le llama. Como se ha comentado anteriormente, no es necesaria, pero gracias a ella aumentaremos la vida útil del transconductor de tensión:

```
def Disable_selector():
    gpio.output(PIN_R20, False)
    gpio.output(PIN_R47, False)
    gpio.output(PIN_R147, False)
    gpio.output(PIN_R194, False)
```

Esta función es la encargada de realizar la adquisición de los datos de tensión y corriente de la carga del condensador. Será necesario pasarle el objeto AD2 que hayamos creado:

```
def Analog_In(AD2):

    dwf.FDwfAnalogInConfigure(AD2.hdev, c_int(0), c_int(1))
      # Comienza la adquisición

    sts = c_int()     # Variable que recibirá el estado de
adquisición
    while True:
      dwf.FDwfAnalogInStatus(AD2.hdev, c_int(1), byref(sts))
      #Verifica el estado de adquisición. En el segundo
parámetro de entrada se marca con TRUE para leer datos del
dispositivo. Los datos se adquieren cuando finaliza la
adquisición. El tercer parámetro es la variable que recibe el
estado de adquisición
      if sts.value == 2 :
            break
      time.sleep(0.1)
    # Finaliza la adquisición

    # Leemos las muestras de datos almacenadas en el buffer
    CH_1 = (c_double*SAMPLES)() # Inicializamos CH_1
    CH_2 = (c_double*SAMPLES)() # Inicializamos CH_2
    dwf.FDwfAnalogInStatusData(AD2.hdev, c_int(0), CH_1 ,
len(CH_1))    # Guardamos los datos del canal 1
    dwf.FDwfAnalogInStatusData(AD2.hdev, c_int(1), CH_2 ,
len(CH_2))    # Guardamos los datos del canal 2

    return CH_1, CH_2  # Devuelve las muestras de datos
```

Esta función se encarga de medir la tensión en el panel solar en unas décimas de segundo:

```
def Get_Panel_Voltaje(AD2, Kv):
    # Activamos y desactivamos las señales de control necesarias
(Descargar el condensador por seguridad)
    gpio.output(PIN_S1, False)
    gpio.output(PIN_S2, True)
    gpio.output(PIN_S3, True)

    # Aumentamos la frecuencia de muestreo del AD2
    dwf.FDwfAnalogInFrequencySet(AD2.hdev, c_double(1000000))

    # Obtenemos el voltaje sin ponderar
    V, I = Analog_In(AD2)

    # Calculamos el voltaje medio
    AV = sum(V)/len(V)*Kv

    # Volvemos a fijar la frecuencia de muestreo a la anterior
    dwf.FDwfAnalogInFrequencySet(AD2.hdev,
c_double(AD2.Sample_rate))

    print(" - Tensión del panel = ", AV)    # Mostramos por el
terminal al tensión del panel

    return AV
```

Esta función se encarga de seleccionar la resistencia que más precisión aporte al transconductor de tensión:

```
def Select_Resistor(AD2, Measure):
    # Seleccionamos la resistencia más alta para medir la
tensión del panel por seguridad
    gpio.output(PIN_R20, False)
    gpio.output(PIN_R47, False)
    gpio.output(PIN_R147, False)
    gpio.output(PIN_R194, True)

    # Obtenemos el voltaje del panel
    Vp = Get_Panel_Voltaje(AD2, KV_194)

    # Ponemos el condensador a descargar por defecto
    Discharge_capacitor()

    # Creamos un array con las diferencias del voltaje del panel
con el asociado a las distintas resistencias del transconductor
    Difference = np.array([abs(Vp - 20), abs(Vp - 47), abs(Vp -
147), abs(Vp - 194)])
```

```
    # Creamos un diccionario con los valores de Kv para los
distintos índices del vector diferencia
    Resistor = {
        0: KV_20,
        1: KV_47,
        2: KV_147,
        3: KV_194
    }

    # Le asignamos a  Kv el valor que más se ajusta al panel
    Measure.Kv = Resistor.get(np.argmin(Difference))

    gpio.output(PIN_R194, False)    # Desactivamos el transistor
asoaciado a la resistencia de 19.4 kohms
    time.sleep(0.01)    # Esperamos el tiempo de conmutación del
transistor

    # Saturamos el transistor asociado a la resistencia elegida
    if(Measure.Kv == KV_20):    # Saturamos el transistor
asociado a la resistencia de 2k
        gpio.output(PIN_R20, True)
        gpio.output(PIN_R47, False)
        gpio.output(PIN_R147, False)
        gpio.output(PIN_R194, False)

    elif(Measure.Kv == KV_47):    # Saturamos el transistor
asociado a la resistencia de 4.7k
        gpio.output(PIN_R20, False)
        gpio.output(PIN_R47, True)
        gpio.output(PIN_R147, False)
        gpio.output(PIN_R194, False)

    elif(Measure.Kv == KV_147):    # Saturamos el transistor
asociado a la resistencia de 14.7k
        gpio.output(PIN_R20, False)
        gpio.output(PIN_R47, False)
        gpio.output(PIN_R147, True)
        gpio.output(PIN_R194, False)

    elif(Measure.Kv == KV_194):    # Saturamos el transistor
asociado a la resistencia de 19.4k
        gpio.output(PIN_R20, False)
        gpio.output(PIN_R47, False)
        gpio.output(PIN_R147, False)
        gpio.output(PIN_R194, True)

    else: print("ERROR: No se puede seleccionar ninguna
resistencia")
```

```
        print(" - Resistencia seleccionada")
```

Esta función se encarga de obtener la irradiancia mediante el conversor analógico digital:

```
def Get_Irradiance():
    # Creamos el objeto del modelo del convertidor ADC con la
librería su librería
    adc = Adafruit_ADS1x15.ADS1115()

    # Cogemos varios valores de irradiancia y hacemos la media
    i = 0
    values = [0]*20
    for i in range (len(values)):
        values[i] = adc.read_adc(0, gain = 1)

    # Calculamos la irradiancia
    G = FS*KG*sum(values)/len(values)

    return G
```

Esta función se encarga de obtener las temperataturas a partir del conversor analógico digital:

```
def Get_Temperature():
    # Creamos el objeto del modelo del convertidor ADC con la
librería su librería
    adc = Adafruit_ADS1x15.ADS1115()

    # Cogemos varios valores de temperatura para el primer y
segundo sensor y hacemos la media
    temp_1 = [0]*20     # Tomamos 20 valores
    temp_2 = [0]*20     # Tomamos 20 valores
    i = 0
    for i in range (len(temp_1)):
        temp_1[i] = adc.read_adc(1, gain = 1)
        temp_2[i] = adc.read_adc(2, gain = 1)

    T_1 = round(FS*KT*sum(temp_1)/len(temp_1),1) # Calculamos la
temperatura 1 haciendo la media
    T_2 = round(FS*KT*sum(temp_2)/len(temp_2),1) # Calculamos la
temperatura 1 haciendo la media

    return T_1, T_2
```

Esta función se encarga de obtener todos los parámetros y asignárselos al objeto Medida que se haya creado:

```
def Get_Parameters(Measure):
    global SAMPLES

    # Guardamos la fecha y hora
    Measure.filename = datetime.now().strftime('%Y_%m_%d
%H%M%S')
    Measure.date = datetime.now().strftime('%d/%m/%Y')
    Measure.hour = datetime.now().strftime('%H:%M:%S')

    # Obtenemos la irradiancia y la 'eficiencia'(Habrá que
añadir una opción de introducir el área del panel a medir, de
momento no se ha habilitado la opción)
    Measure.G = int(Get_Irradiance())
    Measure.Eff = 0 #(Measure.FF * Measure.Voc * Measure.Isc) /
(Measure.G) # Falta dividir por el área del panel!!!!

    # Obtenemos las temperaturas
    Measure.T1, Measure.T2 = Get_Temperature()

    # Tensión e intensidad filtrada
    m_v = np.argmin(Measure.V)  # Elemento con la mínima tensión
    m_i = np.argmax(Measure.I)  # Elemento con la máxima tensión

    # Elegimos como corte el elemento que esté más a la derecha
de nuestra gráfica
    if m_v > m_i :
        m = m_v
    else:
        m = m_i

    # Nos quedamos con la parte que nos interesa
    Measure.V_valid = Measure.V[m:SAMPLES-1]
    Measure.I_valid = Measure.I[m:SAMPLES-1]

    # Inicializamos dos vectores para la regresion lineal
    V_rs = Measure.V_valid[0:(int)((SAMPLES-1-m)*0.05)]
    I_rs = Measure.I_valid[0: (int)((SAMPLES-1-m)*0.05)]

    with warnings.catch_warnings():
        warnings.simplefilter('ignore', np.RankWarning)
        p = np.poly1d(np.polyfit(V_rs, I_rs, 1))    # Realizamos
la regresión lineal

    Measure.V_invalid = np.linspace(0, Measure.V[m], m+1)   # Le
damos a la tensión valores entre 0 y V[m]
    Measure.I_invalid = np.full((m+1), Measure.I[m])   #
Inicializamos el vector de la corriente asociada a la regresión
lineal
    i = 1
    while i < m :
```

```
        Measure.I_invalid[i] = p(Measure.V_invalid[i])  # Le
damos a la intensidad los valores resultantes de la regresión
lineal
        i = i + 1

    # Calculamos la potencia válida y la no válida
    Measure.P_valid = Measure.V_valid * Measure.I_valid
    Measure.P_invalid = Measure.V_invalid * Measure.I_invalid

    # Concatenamos la parte válida con la parte de la regresión
lineal
    Measure.V_final = np.concatenate((Measure.V_invalid,
Measure.V_valid),axis = None)
    Measure.I_final = np.concatenate((Measure.I_invalid,
Measure.I_valid),axis = None)
    Measure.P_final = np.concatenate((Measure.P_invalid,
Measure.P_valid),axis = None)


    # Calculamos la potencia
    Measure.P_final = Measure.V_final * Measure.I_final

    # Calculamos la tensión en circuito abierto y la corriente
de cortocircuito
    Measure.Voc = round(np.amax(Measure.V),2)
    Measure.Isc = round(np.amax(Measure.I_final),2)

    # Calculamos la potencia máxima y el punto de máxima
potencia
    Measure.Pmax = round(np.amax(Measure.P_final),2)
    i = np.argmax(Measure.P_valid)
    Measure.Vmp = round(Measure.V_final[i],2)
    Measure.Imp = round(Measure.I_final[i],2)

    # Calculamos el factor de forma
    Measure.FF = round(Measure.Pmax / (Measure.Voc *
Measure.Isc),1)


    # Tensión e intensidad con N medidas
    D_I = Measure.Isc/(N_VECTOR/2)  # Dividimos la corriente de
cortocircuito para tener el tamaño del paso que tenemos que
seguir
    D_V = Measure.Voc/(N_VECTOR/2)  # Dividimos la tensión en
circuito abierto para tener el tamaño del paso que tenemos que
seguir
    # Inicializamos los array que tendrán la resta para cada
iteración
    V_resta = np.array(SAMPLES)
    I_resta = np.array(SAMPLES)
```

```
    j = 0
    # Tomamos la mitad de los puntos equiespaciando la tensión
    while j<(N_VECTOR/2):
        V_resta = np.absolute(Measure.V_final - (D_V*j))    #
Realizamos la resta del valor medido con el paso de la tensión
en valor absoluto
        k = np.argmin(V_resta)      # Nos quedamos con el índice
con menor magnitud
        # Almacenamos el nuevo par de valores
        Measure.V_n[j] = Measure.V_final[k]
        Measure.I_n[j] = Measure.I_final[k]
        j+=1

    # Tomamos la otra mitad de los puntos equiespaciando la
corriente
    while j<N_VECTOR:
        I_resta = np.absolute(Measure.I_final - (D_I*(j-
(N_VECTOR/2))))     # Realizamos la resta del valor medido con
el paso de la corriente en valor absoluto
        l = np.argmin(I_resta)      # Nos quedamos con el índice
de menor magnitud
        # Almacenamos el nuevo par de valores
        Measure.V_n[j] = Measure.V_final[l]
        Measure.I_n[j] = Measure.I_final[l]
        j+=1

    Measure.V_n.sort()  # Ordenamos las tensiones de forma
ascendente
    Measure.I_n[::-1].sort() # Ordenamos las intensidades de
forma descendente

    Measure.P_n = Measure.V_n * Measure.I_n # Potencia con n
puntos
```

Como hemos podido comprobar, en esta última función, se almacenan en dos vectores N/2 puntos con la tensión equiespaciada, y otros N/2 puntos con la corriente equiespaciada. La finalidad de esta función es reducir el número de puntos que se guardan de la medida, pero asegurándonos de que la forma de la curva se mantiene. En la figura 42 podemos ver gráficamente como funciona.

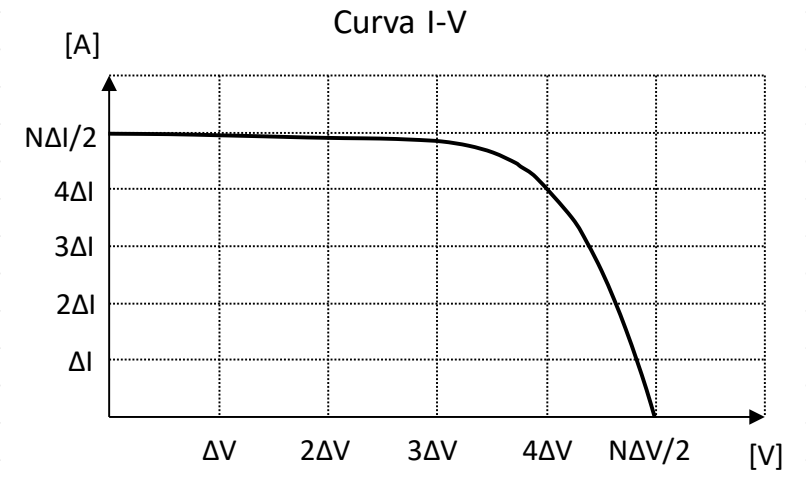


Figura 42. Gráfica de la curva I-V con los puntos equiespaciados

Para guardar todos los datos, existen dos funciones distintas. Por un lado, una versión extendida, la cual guardará más información de la medida y aumentará considerablemente el tiempo de la medida:

```
def Save_IV_values(Measure, foulder):
    # Primero almacenamos todos los puntos
    file = open("{}/datos_medida.txt".format(foulder),"w")
    s = str(round(Measure.V[0], 4)) + "\t"+
str(round(Measure.I[0], 4)) # Almacenamos la primera línea
    file.write(s)
    # Despues almacenamos las restantes añadiendo un salto de
carro
    i = 1
    while(i<SAMPLES):
        s = "\n"+ str(round(Measure.V[i], 4)) + "\t"+
str(round(Measure.I[i], 4))
        file.write(s)
        i += 1
    file.close()

    # En segundo lugar, almacenamos todos los puntos filtrados
    file = open("{}/datos_filtrados.txt".format(foulder),"w")
    s = str(round(Measure.V_valid[0], 4)) + "\t"+
str(round(Measure.I_valid[0], 4)) # Almacenamos la primera línea
    file.write(s)
    # Despues almacenamos las restantes añadiendo un salto de
carro
    i = 1
    while(i<np.size(Measure.V_valid)):
        s = "\n"+ str(round(Measure.V_valid[i], 4)) + "\t"+
str(round(Measure.I_valid[i], 4))
        file.write(s)
```

```
            i += 1
        file.close()

        # En tercer lugar, almacenamos todos los puntos filtrados
        file = open("{}/datos_corregidos.txt".format(foulder),"w")
        s = str(round(Measure.V_invalid[0], 4)) + "\t"+
str(round(Measure.I_invalid[0], 4)) # Almacenamos la primera
línea
        file.write(s)
        # Despues almacenamos las restantes añadiendo un salto de
carro
        i = 1
        while(i<np.size(Measure.V_invalid)):
            s = "\n"+ str(round(Measure.V_invalid[i], 4)) + "\t"+
str(round(Measure.I_invalid[i], 4))
            file.write(s)
            i += 1
        i = 0
        while(i<np.size(Measure.V_valid)):
            s = "\n"+ str(round(Measure.V_valid[i], 4)) + "\t"+
str(round(Measure.I_valid[i], 4))
            file.write(s)
            i += 1
        file.close()

        # En cuarto lugar, almacenamos N puntos
        file = open("{}/datos(N puntos).txt".format(foulder),"w")
        s = "\n"+ str(round(Measure.V_n[0], 4)) + "\t"+
str(round(Measure.I_n[0], 4)) # Almacenamos la primera línea
        file.write(s)
        # Despues almacenamos las restantes añadiendo un salto de
carro
        i = 1
        while(i<N_VECTOR):
            s = str(round(Measure.V_n[i], 4)) + "\t"+
str(round(Measure.I_n[i], 4))
            file.write(s)
            i += 1
        file.close()

        # Guardamos los parámetros
        file = open("{}/parametros.txt".format(foulder),"w")
        s = 'FECHA' + '\t' + 'HORA' + '\t' + 'Voc' + '\t' + 'Isc' +
'\t' + 'Pm' + '\t' + 'Vmp' + '\t' + 'Imp' + '\t' + 'FF' + '\t' +
'G' + '\t' + 'T_1' + '\t' + 'T_2'
        file.write(s)
        string = Measure.toString(Measure)
        s =
string.replace(",","@").replace(".",",").replace("@",".")
        file.write(s)
```

```
    file.close()
```

Y por otro lado, una versión reducida. Con esta se generará solo un fichero con N puntos de la medida. Se pierde información, pero se traduce en un menor tiempo de procesado:

```
def Save_IV_values_EG(Measure, foulder):

    # En tercer lugar, almacenamos todos los puntos filtrados
    file = open("{}/datos.txt".format(foulder),"w")
    s = str(round(Measure.V_invalid[0], 4)) + "\t"+
str(round(Measure.I_invalid[0], 4)) # Almacenamos la primera
línea
    file.write(s)
    # Despues almacenamos las restantes añadiendo un salto de
carro
    i = 1
    while(i<np.size(Measure.V_invalid)):
        s = "\n"+ str(round(Measure.V_invalid[i], 4)) + "\t"+
str(round(Measure.I_invalid[i], 4))
        file.write(s)
        i += 1
    i = 0
    while(i<np.size(Measure.V_n)):
        s = "\n"+ str(round(Measure.V_n[i], 4)) + "\t"+
str(round(Measure.I_n[i], 4))
        file.write(s)
        i += 1
    file.close()


    # Guardamos los parámetros
    file = open("{}/parametros.txt".format(foulder),"w")
    s = 'FECHA' + '\t' + 'HORA' + '\t' + 'Voc' + '\t' + 'Isc' +
'\t' + 'Pm' + '\t' + 'Vmp' + '\t' + 'Imp' + '\t' + 'FF' + '\t' +
'G' + '\t' + 'T_1' + '\t' + 'T_2'
    file.write(s)
    string = Measure.toString(Measure)
    s =
string.replace(",","@").replace(".",",").replace("@",".")
    file.write(s)
    file.close()
```

Al igual que con la parte del guardado de los datos, para el almacenado de las gráficas se han creado dos funciones distintas. Por un lado, una versión extendida con un mayor número de gráficas:

```
def Print_IV_curve(Measure, t_muestreo, foulder):
    t = np.linspace(0, t_muestreo, SAMPLES)     # Declaramos el
vector de tiempos

    a_minor = 0.4   # Constante para fijar los ejes secundarios
    a_major = 1     # Constante para fiar los ejes primarios
    iv_color = "#006db2"    # Color para la gráfica de IV
    pv_color = "#ef280f"    # Color para la gráfica de PV

    # Voltaje
    fig, ax1 = plt.subplots()
    plt.axis([0, t_muestreo, -5, Measure.Voc*1.1])
    ax1.plot(t,Measure.V, linewidth = 1)
    ax1.xaxis.set_minor_locator(AutoMinorLocator(4))
    ax1.yaxis.set_minor_locator(AutoMinorLocator(4))
    plt.grid(which = 'minor', alpha = a_minor)
    plt.grid(which = 'major', alpha = a_major)
    plt.title('VOLTAJE')
    plt.xlabel('Tiempo [s]')
    plt.ylabel('Voltaje [V]')

plt.savefig('{}/grafica_senal_1(tension).png'.format(foulder),
dpi = 300)

    # Corriente
    fig, ax2 = plt.subplots()
    plt.axis([0, t_muestreo, -1, Measure.Isc*1.1])
    ax2.plot(t,Measure.I, linewidth = 1)
    ax2.xaxis.set_minor_locator(AutoMinorLocator(4))
    ax2.yaxis.set_minor_locator(AutoMinorLocator(4))
    plt.grid(which = 'minor', alpha = a_minor)
    plt.grid(which = 'major', alpha = a_major)
    plt.title('CURRENT')
    plt.xlabel('Tiempo [s]')
    plt.ylabel('Current [A]')

plt.savefig('{}/grafica_senal_2(corriente).png'.format(foulder),
dpi = 300)

    # Curva IV con todos los puntos
    fig, ax3 = plt.subplots()
    plt.axis([0, Measure.Voc, 0, Measure.Isc*1.1])
    ax3.plot(Measure.V, Measure.I, linewidth = 1, color =
iv_color)
    ax3.xaxis.set_minor_locator(AutoMinorLocator(4))
    ax3.yaxis.set_minor_locator(AutoMinorLocator(4))
    plt.grid(which = 'minor', alpha = a_minor)
    plt.grid(which = 'major', alpha = a_major)
    plt.title('I-V CURVE')
    plt.xlabel('Voltaje [V]')
```

```
    plt.ylabel('Current [A]')
    plt.savefig('{}/Curva_IV.png'.format(foulder), dpi = 300)

    # Curva PV con todos los puntos
    fig, ax3 = plt.subplots()
    plt.axis([0, Measure.Voc, 0, Measure.Pmax*1.1])
    ax3.plot(Measure.V, Measure.P, linewidth = 1, color =
pv_color)
    ax3.xaxis.set_minor_locator(AutoMinorLocator(4))
    ax3.yaxis.set_minor_locator(AutoMinorLocator(4))
    plt.grid(which = 'minor', alpha = a_minor)
    plt.grid(which = 'major', alpha = a_major)
    plt.title('P-V CURVE')
    plt.xlabel('Voltaje [V]')
    plt.ylabel('Power [W]')
    plt.savefig('{}/Curva_PV.png'.format(foulder), dpi = 300)

    # Curva IV con n puntos
    fig, ax5 = plt.subplots()
    plt.axis([0, Measure.Voc, 0, Measure.Isc*1.1])
    ax5.plot(Measure.V_n, Measure.I_n, linewidth = 1, color =
'r')
    ax5.xaxis.set_minor_locator(AutoMinorLocator(4))
    ax5.yaxis.set_minor_locator(AutoMinorLocator(4))
    plt.grid(which = 'minor', alpha = a_minor)
    plt.grid(which = 'major', alpha = a_major)
    plt.title('I-V CURVE (n points)')
    plt.xlabel('Voltaje [V]')
    plt.ylabel('Current [A]')
    plt.savefig('{}/Curva_IV_n.png'.format(foulder), dpi = 300)

    #Curva PV con n puntos
    fig, ax6 = plt.subplots()
    plt.axis([0, Measure.Voc, 0, Measure.Pmax*1.1])
    ax6.plot(Measure.V_n, Measure.P_n, linewidth = 1, color =
'r')
    ax6.xaxis.set_minor_locator(AutoMinorLocator(4))
    ax6.yaxis.set_minor_locator(AutoMinorLocator(4))
    plt.grid(which = 'minor', alpha = a_minor)
    plt.grid(which = 'major', alpha = a_major)
    plt.title('P-V CURVE (n points)')
    plt.xlabel('Voltaje [V]')
    plt.ylabel('Power [W]')
    plt.savefig('{}/Curva_PV_n.png'.format(foulder), dpi = 300)


    # Curva IV con todos los puntos filtrada
    fig, ax7 = plt.subplots()
    plt.axis([0, Measure.Voc, 0, Measure.Isc*1.1])
    ax7.plot(Measure.V_valid, Measure.I_valid, linewidth = 1,
```

```
color = iv_color)
    ax7.plot(Measure.V_invalid, Measure.I_invalid, linewidth =
1.1, color = iv_color)
    ax7.xaxis.set_minor_locator(AutoMinorLocator(4))
    ax7.yaxis.set_minor_locator(AutoMinorLocator(4))
    plt.grid(which = 'minor', alpha = a_minor)
    plt.grid(which = 'major', alpha = a_major)
    plt.title('I-V CURVE')
    plt.xlabel('Voltaje [V]')
    plt.ylabel('Current [A]')
    plt.savefig('{}/Curva_IV_filtrada.png'.format(foulder), dpi
= 300)


    # Curva PV con todos los puntos filtrada
    fig, ax8 = plt.subplots()
    plt.axis([0, Measure.Voc, 0, Measure.Pmax*1.1])
    ax8.plot(Measure.V_valid, Measure.P_valid, linewidth = 1,
color = pv_color)
    ax8.plot(Measure.V_invalid, Measure.P_invalid, linewidth =
1.1, color = pv_color)
    ax8.xaxis.set_minor_locator(AutoMinorLocator(4))
    ax8.yaxis.set_minor_locator(AutoMinorLocator(4))
    plt.grid(which = 'minor', alpha = a_minor)
    plt.grid(which = 'major', alpha = a_major)
    plt.title('P-V CURVE')
    plt.xlabel('Voltaje [V]')
    plt.ylabel('Power [W]')
    plt.savefig('{}/Curva_PV_filtrada.png'.format(foulder), dpi
= 300)


    # Curva PV-IV con todos los puntos filtrada
    fig, ax9 = plt.subplots()
    plt.axis([0, Measure.Voc, 0, Measure.Isc*1.1])
    ax9.plot(Measure.V_valid, Measure.I_valid, linewidth = 1,
color = iv_color)
    ax9.plot(Measure.V_invalid, Measure.I_invalid, linewidth =
1.1, color = iv_color)
    ax9.xaxis.set_minor_locator(AutoMinorLocator(4))
    ax9.yaxis.set_minor_locator(AutoMinorLocator(4))
    plt.grid(which = 'minor', alpha = a_minor)
    plt.grid(which = 'major', alpha = a_major)
    plt.title('PV / IV CURVES')
    plt.ylabel('Current [A]')
    plt.xlabel('Voltaje [V]')

    ax10 = ax9.twinx()
    plt.axis([0, Measure.Voc, 0, Measure.Pmax*1.1])
    ax10.plot(Measure.V_valid, Measure.P_valid, linewidth = 1,
```

```
color = pv_color)
    ax10.plot(Measure.V_invalid, Measure.P_invalid, linewidth =
1.1, color = pv_color)
    plt.ylabel('Power [W]')
    plt.savefig('{}/Curva_PV_IV_filtrada.png'.format(foulder),
dpi = 300)

    plt.close()
```

Y por otro lado, una función reducida que genera el menor número de gráficas posibles para el correcto funcionamiento de la interfaz gráfica:

```
def Print_IV_curve_EG(Measure, t_muestreo, foulder):
    t = np.linspace(0, t_muestreo, SAMPLES)     # Declaramos el
vector de tiempos

    a_minor = 0.4   # Constante para fijar los ejes secundarios
    a_major = 1     # Constante para fiar los ejes primarios
    iv_color = "#006db2"    # Color para la gráfica de IV
    pv_color = "#ef280f"    # Color para la gráfica de PV

    # Voltaje
    fig, ax1 = plt.subplots()
    plt.axis([0, t_muestreo, -5, Measure.Voc*1.1])
    ax1.plot(t,Measure.V, linewidth = 1)
    ax1.xaxis.set_minor_locator(AutoMinorLocator(4))
    ax1.yaxis.set_minor_locator(AutoMinorLocator(4))
    plt.grid(which = 'minor', alpha = a_minor)
    plt.grid(which = 'major', alpha = a_major)
    plt.title('VOLTAJE')
    plt.xlabel('Tiempo [s]')
    plt.ylabel('Voltaje [V]')

plt.savefig('{}/grafica_senal_1(tension).png'.format(foulder),
dpi = 300)

    # Corriente
    fig, ax2 = plt.subplots()
    plt.axis([0, t_muestreo, -1, Measure.Isc*1.1])
    ax2.plot(t,Measure.I, linewidth = 1)
    ax2.xaxis.set_minor_locator(AutoMinorLocator(4))
    ax2.yaxis.set_minor_locator(AutoMinorLocator(4))
    plt.grid(which = 'minor', alpha = a_minor)
    plt.grid(which = 'major', alpha = a_major)
    plt.title('CURRENT')
    plt.xlabel('Tiempo [s]')
    plt.ylabel('Current [A]')

plt.savefig('{}/grafica_senal_2(corriente).png'.format(foulder),
```

```
dpi = 300)

    # Curva IV con todos los puntos filtrada
    fig, ax7 = plt.subplots()
    plt.axis([0, Measure.Voc, 0, Measure.Isc*1.1])
    ax7.plot(Measure.V_valid, Measure.I_valid, linewidth = 1,
color = iv_color)
    ax7.plot(Measure.V_invalid, Measure.I_invalid, linewidth =
1.1, color = iv_color)
    ax7.xaxis.set_minor_locator(AutoMinorLocator(4))
    ax7.yaxis.set_minor_locator(AutoMinorLocator(4))
    plt.grid(which = 'minor', alpha = a_minor)
    plt.grid(which = 'major', alpha = a_major)
    plt.title('I-V CURVE')
    plt.xlabel('Voltaje [V]')
    plt.ylabel('Current [A]')
    plt.savefig('{}/Curva_IV_filtrada.png'.format(foulder), dpi
= 300)

    # Curva PV con todos los puntos filtrada
    fig, ax8 = plt.subplots()
    plt.axis([0, Measure.Voc, 0, Measure.Pmax*1.1])
    ax8.plot(Measure.V_valid, Measure.P_valid, linewidth = 1,
color = pv_color)
    ax8.plot(Measure.V_invalid, Measure.P_invalid, linewidth =
1.1, color = pv_color)
    ax8.xaxis.set_minor_locator(AutoMinorLocator(4))
    ax8.yaxis.set_minor_locator(AutoMinorLocator(4))
    plt.grid(which = 'minor', alpha = a_minor)
    plt.grid(which = 'major', alpha = a_major)
    plt.title('P-V CURVE')
    plt.xlabel('Voltaje [V]')
    plt.ylabel('Power [W]')
    plt.savefig('{}/Curva_PV_filtrada.png'.format(foulder), dpi
= 300)

    # Curva PV-IV con todos los puntos filtrada
    fig, ax9 = plt.subplots()
    plt.axis([0, Measure.Voc, 0, Measure.Isc*1.1])
    ax9.plot(Measure.V_valid, Measure.I_valid, linewidth = 1,
color = iv_color)
    ax9.plot(Measure.V_invalid, Measure.I_invalid, linewidth =
1.1, color = iv_color)
    ax9.xaxis.set_minor_locator(AutoMinorLocator(4))
    ax9.yaxis.set_minor_locator(AutoMinorLocator(4))
    plt.grid(which = 'minor', alpha = a_minor)
    plt.grid(which = 'major', alpha = a_major)
    plt.title('PV / IV CURVES')
    plt.ylabel('Current [A]')
    plt.xlabel('Voltaje [V]')
```

```
    ax10 = ax9.twinx()
    plt.axis([0, Measure.Voc, 0, Measure.Pmax*1.1])
    ax10.plot(Measure.V_valid, Measure.P_valid, linewidth = 1,
color = pv_color)
    ax10.plot(Measure.V_invalid, Measure.P_invalid, linewidth =
1.1, color = pv_color)
    plt.ylabel('Power [W]')
    plt.savefig('{}/Curva_PV_IV_filtrada.png'.format(foulder),
dpi = 300)


    # Curva PV-IV con todos los puntos filtrada tamaño mini
    fig, ax9 = plt.subplots()
    plt.axis([0, Measure.Voc, 0, Measure.Isc*1.1])
    ax9.plot(Measure.V_valid, Measure.I_valid, linewidth = 1,
color = iv_color)
    ax9.plot(Measure.V_invalid, Measure.I_invalid, linewidth =
1.1, color = iv_color)
    ax9.xaxis.set_minor_locator(AutoMinorLocator(4))
    ax9.yaxis.set_minor_locator(AutoMinorLocator(4))
    plt.grid(which = 'minor', alpha = a_minor)
    plt.grid(which = 'major', alpha = a_major)
    plt.title('PV / IV CURVES', fontsize = 22)
    plt.ylabel('Current [A]', fontsize = 22)
    plt.xlabel('Voltaje [V]', fontsize = 22)

    ax10 = ax9.twinx()
    plt.axis([0, Measure.Voc, 0, Measure.Pmax*1.1])
    ax10.plot(Measure.V_valid, Measure.P_valid, linewidth = 1,
color = pv_color)
    ax10.plot(Measure.V_invalid, Measure.P_invalid, linewidth =
1.1, color = pv_color)
    plt.ylabel('Power [W]', fontsize = 22)
    plt.savefig(os.getcwd() + '/iconos/' +'/Last_measure.png',
dpi = 40)

    plt.close()
```

Esta función se encarga de guardar tanto los datos como las gráficas de la medida en versión extendida:

```
def Save_Measure(Measure, t_muestreo, foulder):
    # Creamos el nuevo nombre del directorio
    new_foulder = foulder + "/{}".format(Measure.filename)

    # Creamos el directorio
    os.mkdir(new_foulder)
```

```
    # Guardamos la medida
    Save_IV_values(Measure, new_foulder)
    Print_IV_curve(Measure, t_muestreo, new_foulder)
```

Esta función se encarga, al igual que la anterior, de guardar los datos y las gráficas de la medida pero en versión reducida (optimizada para la interfaz gráfica):

```
def Save_Measure_EG(Measure, t_muestreo, foulder):
    # Creamos el nuevo nombre del directorio
    new_foulder = foulder + "/{}".format(Measure.filename)

    # Creamos el directorio
    os.mkdir(new_foulder)

    # Guardamos la medida
    Save_IV_values_EG(Measure, new_foulder)
    Print_IV_curve_EG(Measure, t_muestreo, new_foulder)
```

Finalmente, está la función que realiza la medida y hace uso de varias de las funciones anteriores:

```
def Get_Measure(AD2, Measure, t_discharge, foulder):

    # 1.- Descargar el condensador
    str_action = 'Descarga condensador: '+ str(t_discharge) + '
s'
    print ('\n - ' + str_action)
    Discharge_capacitor()
    time.sleep(t_discharge)

    # 2.- Activamos el relé
    gpio.output(PIN_S1, True)
    gpio.output(PIN_S2, False)
    gpio.output(PIN_S3, False)

    time.sleep(0.027)   # Esperamos a que el relé no introduzca
ninguna perturbación en la medida

    # 3.- Iniciamos el barrido de valores I-V durante la carga
del condensador
    str_action = 'Barrido valores IV durante la carga'
    print ('\n - ' + str_action)
    gpio.output(PIN_S2, True)
    CH_1,CH_2 = Analog_In(AD2)
    Measure.V = np.array(CH_1) * Measure.Kv
    Measure.I = np.array(CH_2) * KI
```

```
    # 4.- Descarga del condensador
    str_action = 'Adquisición completada'
    print ('\n - ' + str_action)
    gpio.output(PIN_S2, False)  # Desconectamos primero el IGBT
para proteger el relé
    time.sleep(0.001)           # Esperamos el tiempo de
conmutación del IGBT
    gpio.output(PIN_S1, False)  # Desconectamos primero el IGBT
para proteger el relé
    time.sleep(0.03)            # Esperamos el tiempo de
conmutación del relé
    Discharge_capacitor()   # Descargamos el condensador
    Get_Parameters(Measure)
    # Elegimos la versión de guardado de la medida
    #Save_Measure(Measure, AD2.t_muestreo, foulder)
    Save_Measure_EG(Measure, AD2.t_muestreo, foulder)
    Measure.Print_Measure(Measure)

    str_action = 'Descarga condensador: '+ str(t_discharge) + '
s'
    print ('\n - ' + str_action)
    time.sleep(t_discharge)
    Disable_selector()

    str_action = ''
```


4.2.3. Script Python

Como se ha comentado al principio del capítulo, este dispositivo de medida puede utilizarse de varias maneras usando la librería que acabamos de exponer. En este apartado, mostraremos un ejemplo de script de Python donde se hace uso de esta librería.

Al comienzo del código, incluiremos en las dos primeras líneas el mismo par de comentarios que en la librería:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""TFG_program.py: Caracterización y modelado de paneles solares
en tiempo real."""

__author__ = "Pablo Casado Pérez"
```

Seguidamente incluimos las librerías necesarias para el correcto funcionamiento del script:

```
import TFG_library as pv
```

```
from ctypes import cdll, c_double
import sys
import time
import os

# Definimos la librería DWF DLL en función del sistema operativo
if sys.platform.startswith("win"):
    dwf = cdll.dwf    # Windows
else:
    dwf = cdll.LoadLibrary("libdwf.so")   # Linux
```

Declaramos las variables globales que podremos modificar antes de lanzar el script:

```
N_MEDIDAS = 1   # Número de medidas a realizar
T_MEDIDA = 1    # Tiempo entre medida y medida
T_DESCARGA = 1.32 # tao = R x C = 120 x 0.0022 = 0.264     -->
T_DESCARGA = 5 x tao = 1.32
```

En el caso de este ejemplo queremos guardar en un mismo fichero todos los parámetros de todas las medidas que se realicen, por lo que será necesaria la siguiente función:

```
def Guardar_archivo(Measure, path):
    #Creamos o abrimos el archivo parametros.txt
    file = open(path + '/parametros.txt', "a+")
    file.write(Measure.toString())
    file.close()
```

En este script, al no tratarse de una librería, es necesario incluir todas las acciones que queramos realizar en la función principal:

```
def main():

    # Inicializamos los pines de la Raspberry
    pv.Set_GPIO()
    # Ponemos el condensador a descargar por defecto
    pv.Discharge_capacitor()

    # Establecemos conexión con la AD2 e inicializamos sus
valores
    ad2 = pv.AD2()
```

```
    # Creamos el archivo de texto
    path = os.getcwd() + '/Medidas'
    file = open(path + '/parametros.txt', "a+")
    s = 'FECHA' + '\t' + 'HORA' + '\t' + 'Voc' + '\t' + 'Isc' +
'\t' + 'Pm' + '\t' + 'Vmp' + '\t' + 'Imp' + '\t' + 'FF' + '\t' +
'G' + '\t' + 'T_1' + '\t' + 'T_2'
    file.write(s)
    file.close()

    # Creamos una lista de Objetos Measure
    Medida = pv.Measure
    Medida.Kv = pv.KV_194
    i = 0
    for i in range(N_MEDIDAS - 1):
        print('MEDIDA ' + str(i) + '/' + str(N_MEDIDAS) + ':')
        Medida.Kv = pv.KV_194     # Fijamos la constante del
transconductor de tensión
        pv.Select_Resistor(ad2, Medida)
        pv.Get_Measure(ad2, Medida, T_DESCARGA, path)
        Guardar_archivo(Medida, path)
        time.sleep(T_MEDIDA)

    print('MEDIDA ' + str(N_MEDIDAS) + '/' + str(N_MEDIDAS) +
':')
    Medida.Kv = pv.KV_194     # Fijamos la constante del
transconductor de tensión
    pv.Select_Resistor(ad2, Medida)
    pv.Get_Measure(ad2, Medida, T_DESCARGA, path)
    Guardar_archivo(Medida, path)

    print('FIN')

    # Cerramos Analog Discovery y salimos
    dwf.FDwfDeviceCloseAll()
    exit()
```

Finalmente, indicamos que la función ‘main’ será la función principal:

```
if __name__ == "__main__":
    main()
```

4.3. Interfaz gráfica

Como se ha comentado al principio del capítulo, se ha elaborado una interfaz gráfica en Python. Esta se ha elaborado a partir del módulo tkinter de Python, ya que está considerado un estándar para la interfaz gráfica de usuario (GUI), el cual nos permite administrar ventanas. Gracias a la GUI podremos realizar y consultar medidas sin la necesidad de tener los conocimientos de Python necesarios para retocar el script que hemos expuesto anteriormente.

Todo el diseño se ha hecho adaptado a la pantalla de 7 pulgadas que se ha instalado en la carcasa del dispositivo. Esta posee una resolución de 800x480 pixeles, que marcarán el tamaño de la ventana principal. Dentro de esta ventana principal, se han creado dos frames distintos. Por un lado, el "control_frame", en el que estarán posicionados todos los botones del menú de la aplicación. Estos botones se mantendrán fijos para poder cambiar de sección siempre que se quiera. El tamaño de este marco será de 800x96 pixeles y estará en la parte inferior de la ventana principal. Por otro lado, el "board_frame", donde se mostrará la vista que hayamos seleccionado en los botones del menú. El tamaño de este marco es más grande, de 800x384 pixeles y estará posicionado en la parte superior de la ventana principal. En la figura 43 podemos ver una captura de pantalla con el programa de la interfaz gráfica tras ser iniciado. En esta se pueden distinguir los dos frames.

Figura 43. Captura de pantalla de la interfaz gráfica

En total habrá 4 secciones: Inicio, Resultados, Ajustes, Salir.

- Sección "Inicio". Esta será la sección donde se lanzarán las medidas. Esta vista tiene dos fases. En la figura 44 se muestra la primera de ellas, en la que se muestra la irradiancia y las temperaturas de ambos sensores en

tiempo real, además de la actual configuración del dispositivo. También existe un recuadro para poner el nombre de la carpeta donde queremos que se guarden las medidas que vayamos a realizar.

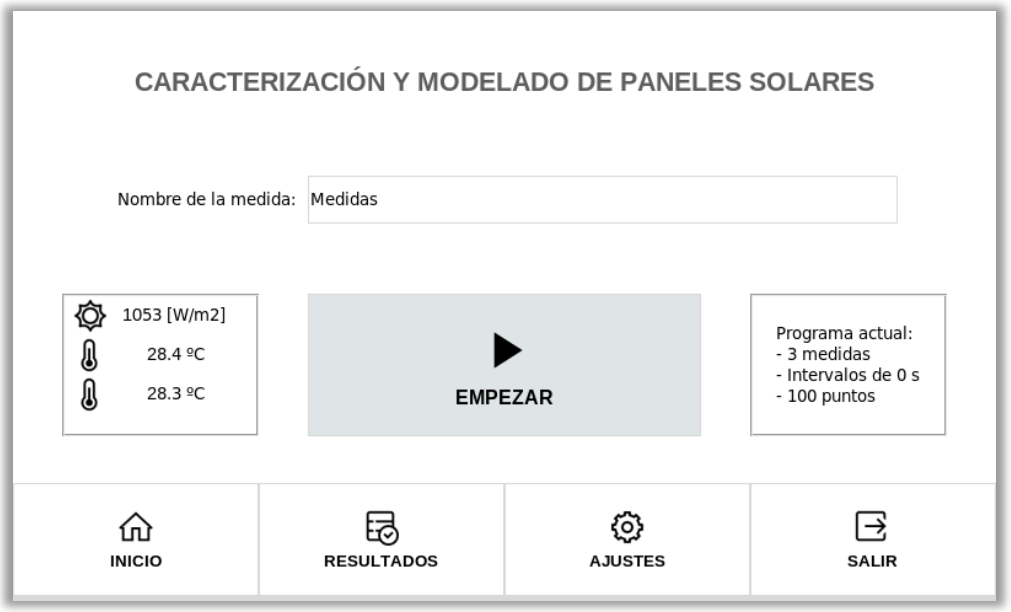


Figura 44. Captura de pantalla de la sección INICIO de la interfaz gráfica

Si pulsamos el botón "EMPEZAR", pasamos a la segunda fase de esta sección. En ella podemos encontrar una barra de progreso y el número de medidas realizadas. Además, también nos encontramos con un botón para parar la medición que nos devuelve a la primera fase de esta vista. En la figura 45 se muestra una captura de esta fase.

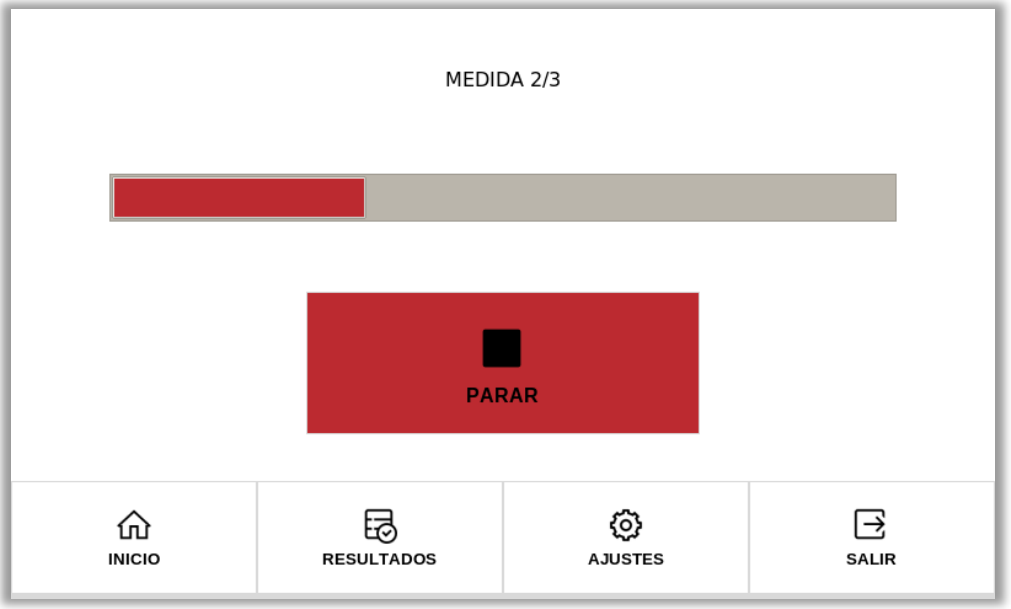


Figura 45. Captura de pantalla de la interfaz gráfica durante una tanda de medidas

- Sección “Resultados”. Dentro de esta vista tenemos 4 subsecciones. La primera de ellas es la de la figura 46. Es la que aparece cuando pulsamos en “Resultados” y nos muestra una previsualización de la última medida en caso de que hayamos realizado alguna. Para poder pasar a alguna de las otras subsecciones es importante que seleccionemos una medida en los “combobox” de la parte superior izquierda.

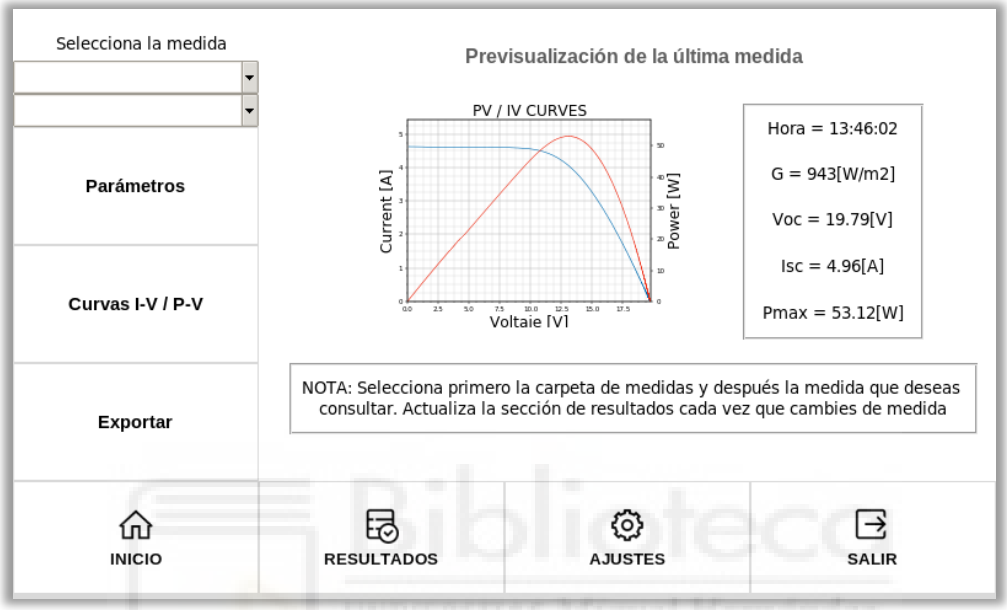


Figura 46. Captura de pantalla de la sección de RESULTADOS de la interfaz gráfica

Si seleccionamos una medida y pulsamos sobre el botón “parámetros”, accederemos a un listado con los parámetros de esa medida (figura 47).

Figura 47. Captura de pantalla de Parámetros en la sección de RESULTADOS de la interfaz gráfica

Estos parámetros son leídos de los distintos ficheros .txt que se generan con cada medida tal y como se ha explicado en la librería.

Si seleccionamos el botón de “curvas”, se mostrarán por pantalla las dos curvas características de la medida seleccionada. Para cambiar entre estas dos curvas tendremos que volver a pinchar en el botón “curvas”. En las figuras 48 y 49 se pueden ver las dos curvas distintas.

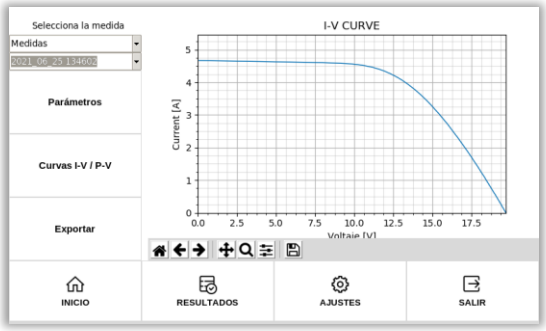


Figura 49. Captura de pantalla de una curva IV en la sección de RESULTADOS de la interfaz gráfica

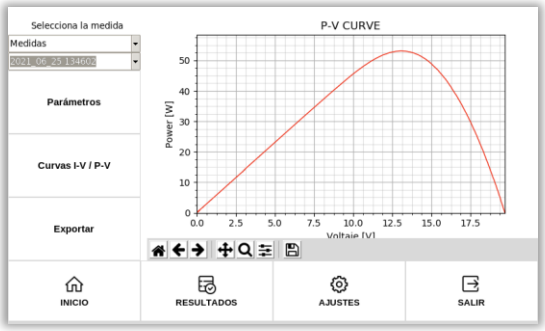


Figura 48. Captura de pantalla de una curva PV en la sección de RESULTADOS de la interfaz gráfica

Finalmente, si pulsamos en el botón “exportar”, se abrirá la vista de la figura 50. En ella podremos seleccionar el USB que tengamos conectado en alguno de los dos puertos de los que dispone el dispositivo. En raspbian es fácil acceder a los pendrives, pues están todos en la ruta /media/pi. Después, podemos elegir si guardar o borrar la carpeta o la medida que hayamos seleccionado.

Figura 50. Captura de pantalla de Exportar en la sección de RESULTADOS de la interfaz gráfica

- Sección “Opciones”. Esta sección es en la que se pueden ajustar la configuración de la medida, es decir, el número de medidas que queremos realizar, el tiempo entre estas medidas, y cuantos puntos queremos guardar de cada medida. Cada vez que hagamos alguna modificación, será necesario darle al botón de “Guardar cambios” para que estos tengan efecto. En la figura 51 se muestra la captura de esta vista.

Figura 51. Captura de pantalla de Parámetros en la sección de AJUSTES de la interfaz gráfica

- Sección “Salir”. Como su propio nombre indica, esta sección sirve para salir del programa. Tras pulsar en el botón “salir”, saldrá una ventana emergente preguntándonos si estamos seguros de cerrar el programa. De esta manera evitaremos posibles toques sin querer que interrumpan la medida que estemos realizando.

El script de Python con el código de la interfaz gráfica está adjunto en el ANEXO V al final de este documento.

4.4. Aplicación para Android

Además de la interfaz gráfica en Python para utilizar con la pantalla integrada en el dispositivo, se ha creado una aplicación móvil para Android. Esta se ha desarrollado como un trabajo para la asignatura de Aplicaciones Móviles Multimedia impartida por el profesor Miguel Onofre Martínez Rach del departamento de Ingeniería de Computadores. La programación se ha realizado con el lenguaje JAVA en el entorno Android Studio bajo la plantilla de Navigation Drawer. En el menú de esta plantilla hemos incluido 6 fragmentos que están asociados a un layout distinto. Estos son: Inicio, Instrucciones de uso, Medir, Resultados, Ajustes y Acerca de. En la figura 54 se muestra una captura del menú del Navigation Drawer de la aplicación. En las figuras 52 y 53 se muestran los fragmentos de Inicio e Instrucciones de uso respectivamente.

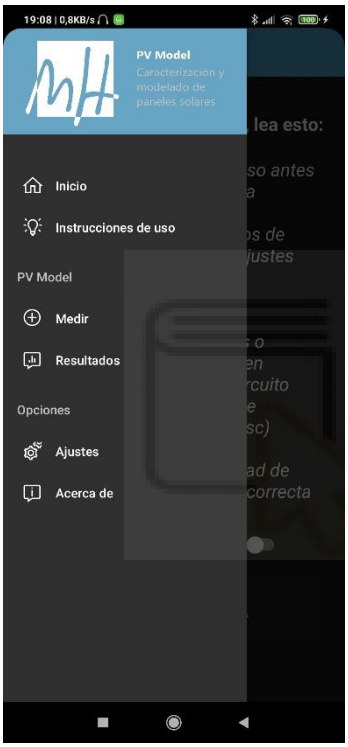


Figura 54. Captura de pantalla del menú

Figura 52. Captura de pantalla de Inicio

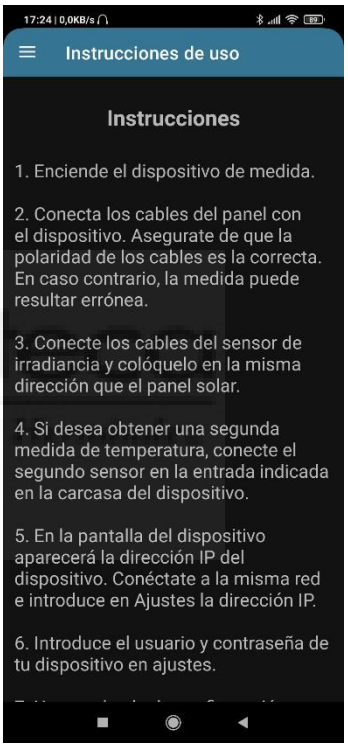


Figura 53. Captura de pantalla de Instrucciones de uso

Para poder comunicarnos con la RPI y lanzar el script de Python que hemos creado específicamente para esta aplicación se ha optado por utilizar una conexión SSH. La razón por la que se ha utilizado es porque esta conexión nos permite lanzar un comando en el terminal de Raspbian con bastante facilidad. Por ello, es muy importante introducir la dirección IP y las credenciales de acceso en el fragmento de Ajustes, de lo contrario no podremos realizar ninguna medida. En la figura 55 se muestra el layout del fragmento de Ajustes. En la figura 56 se muestra el de Medir, en el que será necesario que verifiquemos que hemos leído las instrucciones de uso para poder realizar una nueva medida. Si lanzamos una nueva medida, y no hemos configurado correctamente las credenciales y la IP, se nos indicará por pantalla como se muestra en la figura 57.

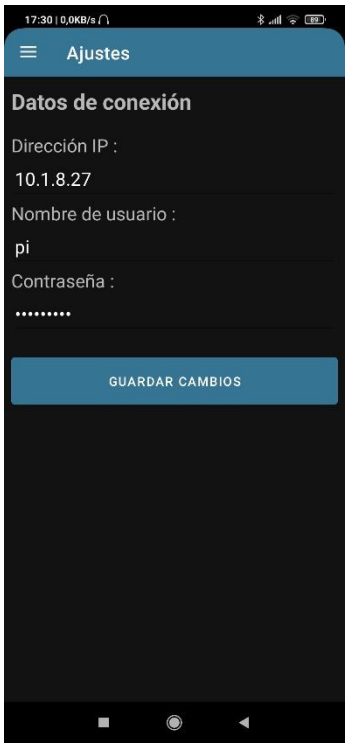


Figura 55. Captura de pantalla de Ajustes

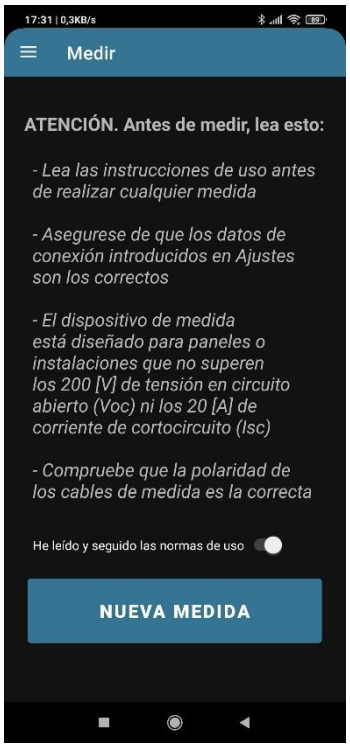


Figura 56. Captura de pantalla de Medir

Figura 57. Captura de pantalla de Resultados de la medida

Una vez introducidos los datos correctos, si realizamos una nueva medida, la aplicación iniciará una nueva actividad en la que se creará la conexión SSH y se ejecutará

Figura 60. Captura de pantalla de Realizando medida

Figura 58. Captura de pantalla de Resultado de la medida

Figura 59. Captura de pantalla del guardado de una medida

el comando en la RPI que lanza el script de Python. Para evitar que se congele la pantalla

del smartphone mientras se realiza la medida, se ha añadido un UI Thread, que es un nuevo hilo en el código de la aplicación. Además se ha añadido un progress dialog para indicar que se está realizando la medida (figura 60). Tras pasar 7 segundos, que es lo que tarda el dispositivo en realizar la medida, la RPI devuelve un string con todos los parámetros. Este string es interpretado por la app, que guarda en distintas variables locales los distintos valores. Seguidamente, como se muestra en la figura 58, se escriben por pantalla los resultados de la medida. Estos se pueden guardar en la base de datos con el botón “guardar medida”, el cual se deshabilitará una vez que sea pulsado (Figura 59). La base de datos se ha realizado con SQLite, cuyos paquetes están disponibles de m manera libre para Android.

Otra de las funcionalidades de esta aplicación es la de consultar anteriores medidas. Esto es posible en el layout del fragmento de Resultados, donde podremos seleccionar en base a qué parámetro queremos que se muestre la lista de medidas que hay guardadas en la base de datos. En la figura 62 se muestra la captura de la lista en función del parámetro seleccionado. Tras pulsar en uno, la lista se reinicia y nos muestra la medida seleccionada (Figura 61). También tenemos la opción de borrar la medida seleccionada.

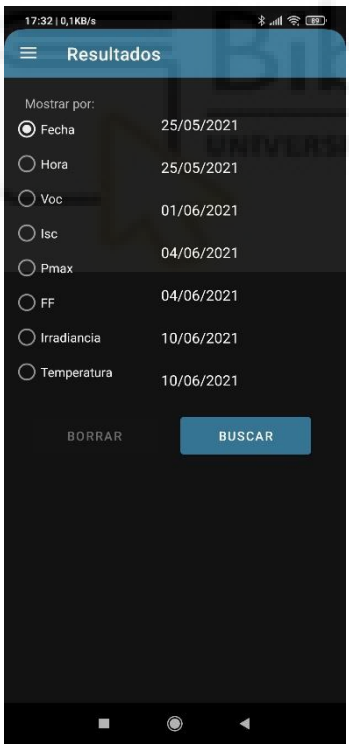


Figura 62. Captura de pantalla de la búsqueda de resultados

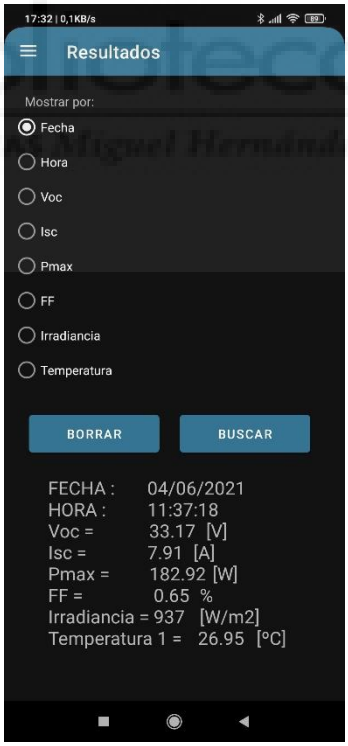


Figura 61. Captura de pantalla de resultados

CAPÍTULO 5. Resultados experimentales

A lo largo del desarrollo de este proyecto, algunas de las distintas pruebas que se han realizado han sido con los paneles BP380 de la marca BPSOLAR (Figura 63), situados en la azotea del edificio Torrevaillo del campus de Elche de la Universidad Miguel Hernández. Estos paneles, según su fabricante, tienen las características de la Tabla 11.

Figura 63. Paneles solares BP380 de la azotea del edificio Torrevaillo

	1000W/m2 (STC1)	800W/m2 (NOCT2)
Maximum power (Pmax)	80W	57.6W
Voltage at MPP (Vmpp)	17.6V	15.7V
Current at MPP (Impp)	4.5A	3.6A
Short circuit current (Isc)	4.8A	3.9A
Open circuit voltage (Voc)	22.1V	20.2V

Tabla 11. Características paneles solares BP380 según el fabricante

Además de utilizar los anteriores paneles solares, se han utilizado simuladores de paneles solares. En concreto el modelo E4351B Solar Array Simulator de la marca Keysight, que ofrece una potencia máxima de 480 W con unos rangos de salida de tensión de 0 a 120V y de salida de corriente de 0 a 4V.

Figura 64. Fotografía de uno de los simuladores de paneles solares

5.1. Ejemplos de medidas

En este apartado mostraremos algunos ejemplos de medidas realizadas con el dispositivo (Figura 65). En cada una de estas medidas se han generado 5 gráficas distintas (Curva IV, curva PV, ambas curvas juntas, tensión e intensidad) y dos .txt (uno con los datos de tensión y corriente, y otro con todos los parámetros).

Figura 65. Dispositivo realizando una medida

En primer lugar, mostraremos una medida realizada el día 26/06/2021 a las 11:21:09 con el simulador de paneles solares. En las figuras 67 se muestran la tensión y la intensidad con respecto al tiempo, mientras que en la figura 68 se muestra la gráfica de las dos curvas características. En la tabla 12 se muestran todos los parámetros asociados a la medida.

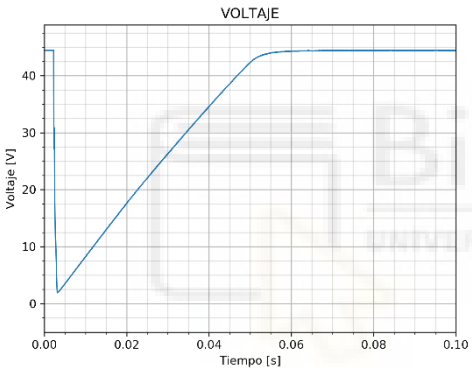


Figura 66. Gráfica de tensión

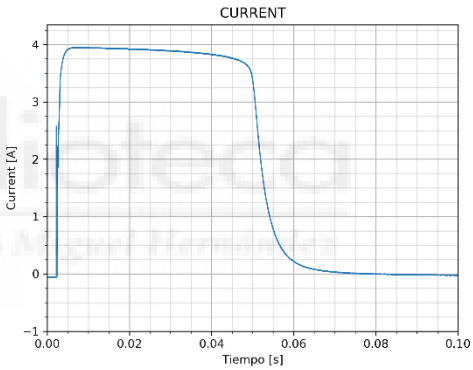


Figura 67. Gráfica de corriente

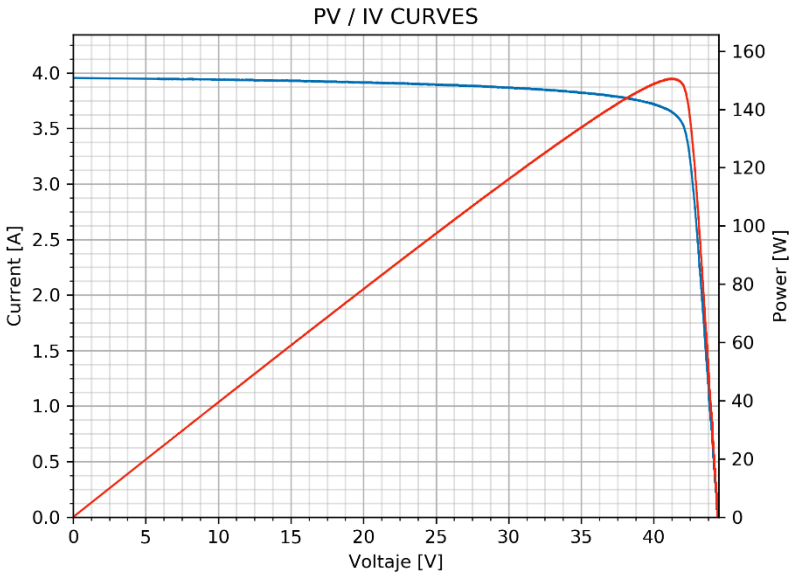


Figura 68. Gráfica con las curvas características

FECHA	HORA	Voc [V]	Isc [A]	Pm [W]	Vmp [V]	Imp [A]	FF	G [W/m2]	T_1 [ºC]	T_2 [ºC]
11/06/2021	11:21:09	44.47	3.95	150.61	35.34	3.82	0.9	352	28.7	28.7

Tabla 12. Resultados de la medida

En segundo lugar, mostraremos una medida realizada el día 11/06/2021 a las 12:01:24 con el simulador de paneles solares. En las figuras 69 y 70 se muestran la tensión y la intensidad con respecto al tiempo, y en la figura 71 se muestran las dos curvas características juntas. En la tabla 13 se muestran todos los parámetros asociados a la medida.

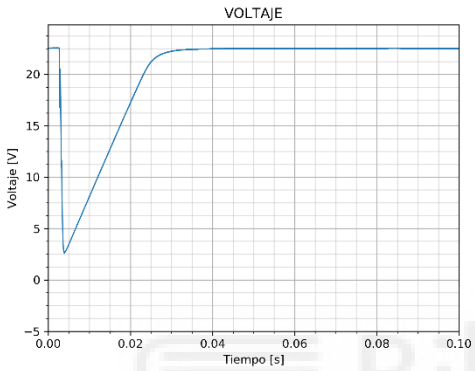


Figura 70. Gráfica de tensión

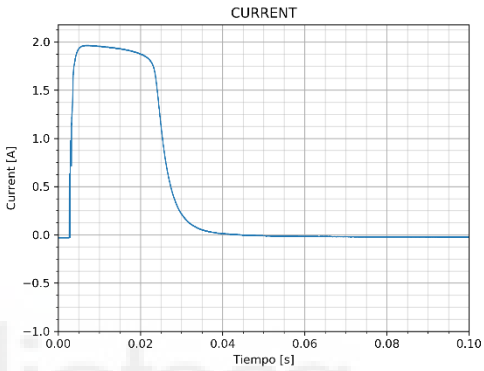


Figura 69. Gráfica de corriente

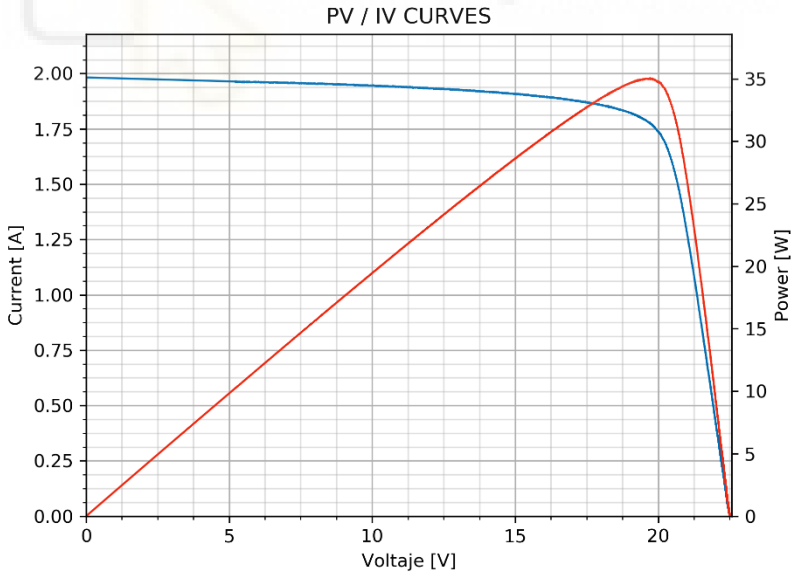


Figura 71. Gráfica con las curvas características

FECHA	HORA	Voc [V]	Isc [A]	Pm [W]	Vmp [V]	Imp [A]	FF	G [W/m2]	T_1 [ºC]	T_2 [ºC]

11/06/2021	12:01:24	22.57	1.98	35.07	13.55	1.92	0.8	351	28.6	28.5

Tabla 13. Resultados de la medida

En tercer lugar, mostraremos una medida realizada el día 28/06/2021 a las 11:47:12 con uno de los paneles de la azotea. En las figuras 72 y 73 se muestran la tensión y la intensidad con respecto al tiempo, mientras que en la figura 74 se muestra la gráfica de las dos curvas características. En la tabla 14 se muestran todos los parámetros asociados a la medida.

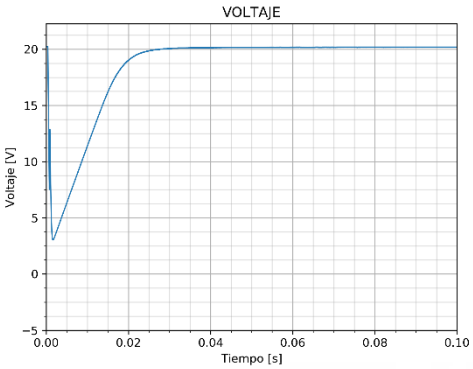


Figura 73. Gráfica de tensión

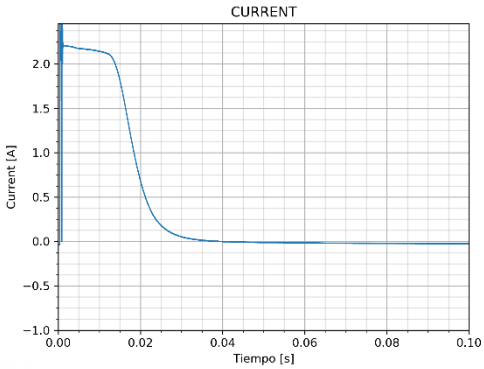


Figura 72. Gráfica de corriente

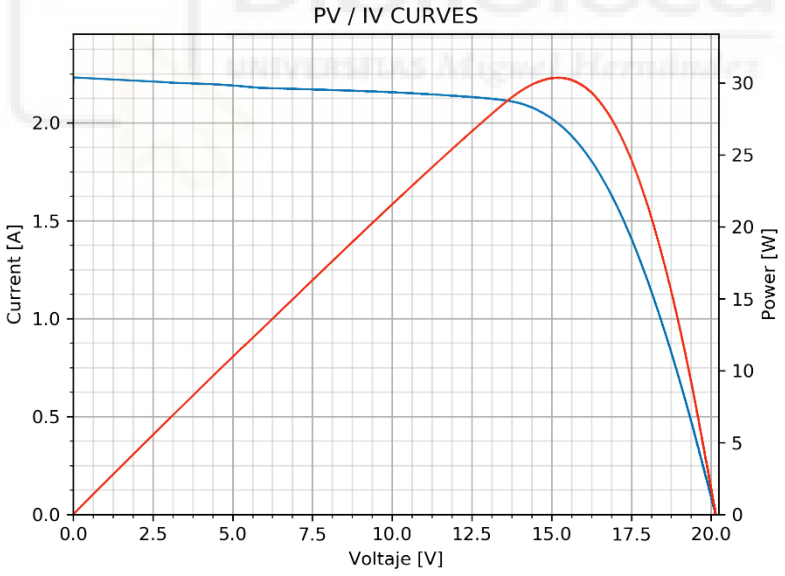


Figura 74. Gráfica con las curvas características

FECHA	HORA	Voc [V]	Isc [A]	Pm [W]	Vmp [V]	Imp [A]	FF	G [W/m2]	T_1 [ºC]	T_2 [ºC]
27/06/2021	11:46:00	20.24	2.23	30.37	13.67	2.11	0.7	371	29.5	29

Tabla 14. Resultados de la medida

En cuarto lugar, mostraremos una medida realizada el día 08/06/2021 a las 12:47:12 con dos paneles en serie de la azotea. En las figuras 75 y 76 se muestran la tensión y la intensidad con respecto al tiempo, mientras que en la figura 77 se muestra la gráfica de las dos curvas características. En la tabla 15 se muestran todos los parámetros asociados a la medida.

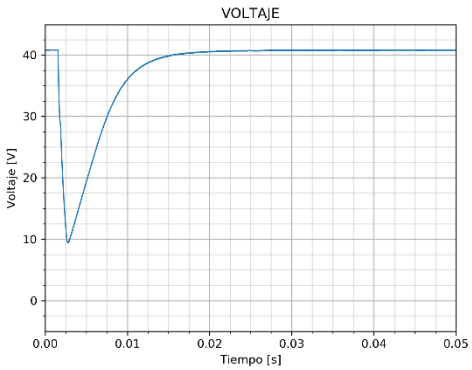


Figura 76. Gráfica de tensión

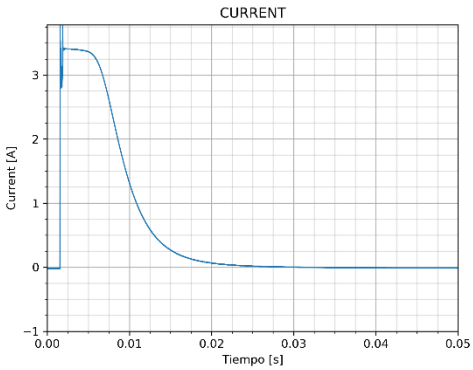


Figura 75. Gráfica de corriente

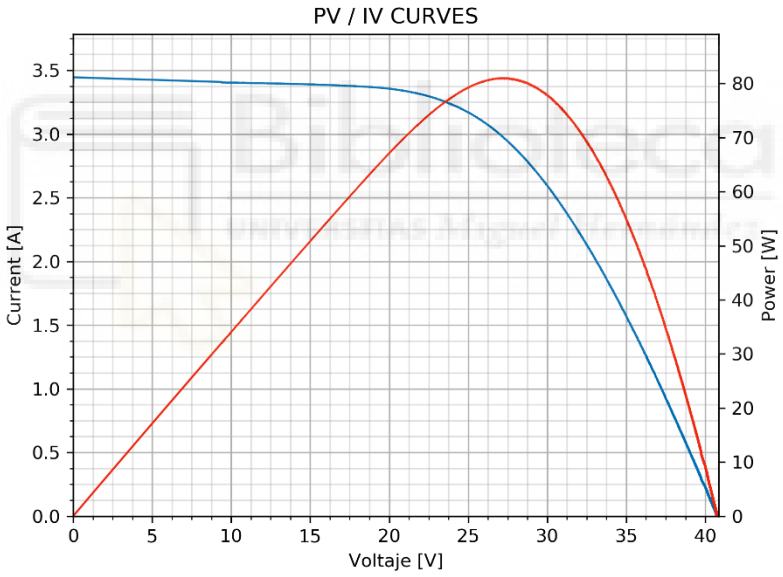


Figura 77. Gráfica con las curvas características

FECHA	HORA	Voc [V]	Isc [A]	Pm [W]	Vmp [V]	Imp [A]	FF	G [W/m2]	T_1 [ºC]	T_2 [ºC]
08/06/2021	12:47:12	40.85	3.44	81.03	14.8	3.39	0.6	350	28	28.2

Tabla 15. Resultados de la medida

Como se puede apreciar en las gráficas de la corriente, de las medidas realizadas con el simulador, cuando se activa el IGBT la corriente no parte desde la Isc. Además, en

las medidas realizadas con los paneles, se genera un pico de corriente al principio de la medida. Ambos sucesos podrían alterar la forma de las curvas, pero como se ha explicado en el anterior capítulo, el programa se ha preparado para que sea capaz de detectar estas perturbaciones y eliminar todos los puntos que hay desde el inicio hasta que se acaban. Después de retirar esta parte de los datos, el software realiza una regresión lineal con el 10% del nuevo inicio de los datos. Con la tensión sucede lo mismo y siempre que esta no parte desde 0V, se realiza una regresión lineal de la parte que no se ha podido medir.

También cabe destacar la opción del programa que permite realizar medidas periódicas. Para demostrar su funcionamiento, se ha realizado una tanda de 15 medidas y se ha exportado en Excel el fichero .txt que se genera con los parámetros de todas las medidas. En la tabla 16 se pueden ver los resultados.

HORA	Voc	Isc	Pm	Vmp	Imp	FF	G	T_1	T_2
11:24:35	34.28	8.05	183.22	25.62	7.15	0.66	920.2	20.84	21.42
11:25:32	34.35	8.07	183.66	25.55	7.19	0.66	921.8	21.00	21.60
11:26:30	34.35	8.09	184.45	25.62	7.20	0.66	924.1	20.81	21.41
11:27:27	34.35	8.10	185.38	25.98	7.13	0.67	928.1	20.59	21.00
11:28:25	34.42	8.12	185.81	25.62	7.25	0.67	932.3	19.60	19.98
11:29:22	34.56	8.13	187.13	25.77	7.26	0.67	932.8	20.00	20.61
11:30:19	34.56	8.15	187.41	25.77	7.27	0.67	934.1	19.98	20.40
11:31:17	34.49	8.20	188.21	25.84	7.28	0.67	938.1	20.40	20.84
11:32:15	34.49	8.20	188.81	26.06	7.25	0.67	943.3	20.63	21.20
11:33:12	34.56	8.23	188.90	25.84	7.31	0.66	947.3	20.30	20.61
11:34:10	34.49	8.20	188.67	25.70	7.34	0.67	952.4	20.62	21.16
11:35:07	34.49	8.25	189.33	25.77	7.35	0.66	955.4	20.20	20.79
11:36:04	34.49	8.29	189.78	25.62	7.41	0.66	955.5	19.99	20.43
11:37:02	34.56	8.32	190.31	25.70	7.41	0.66	957.2	19.99	20.40
11:37:59	34.49	8.35	191.20	25.98	7.36	0.66	962.1	20.22	20.80

Tabla 16. Resultados de la medida

5.2. Impacto del selector de resistencias

Como hemos comentado en el desarrollo de este trabajo, se ha incluido un selector de resistencias para aumentar la precisión con la que mide el transconductor de tensión. Para demostrar la efectividad de este, se ha dispuesto a medir uno de los paneles solares BP380 con distintas resistencias en el devanado primario del transconductor. Para apreciar mejor las diferencias entre los distintos casos, se han realizado medidas con 100 puntos equiespaciados. Recordemos que la tensión en circuito abierto de este panel es de 22.1[V], por lo que deberíamos de colocar una resistencia cercana a 2.2kΩ para tener la máxima precisión posible.

En primer lugar, se ha realizado una medida con una resistencia de 2kΩ. En la figura 78 se muestra la curva IV.

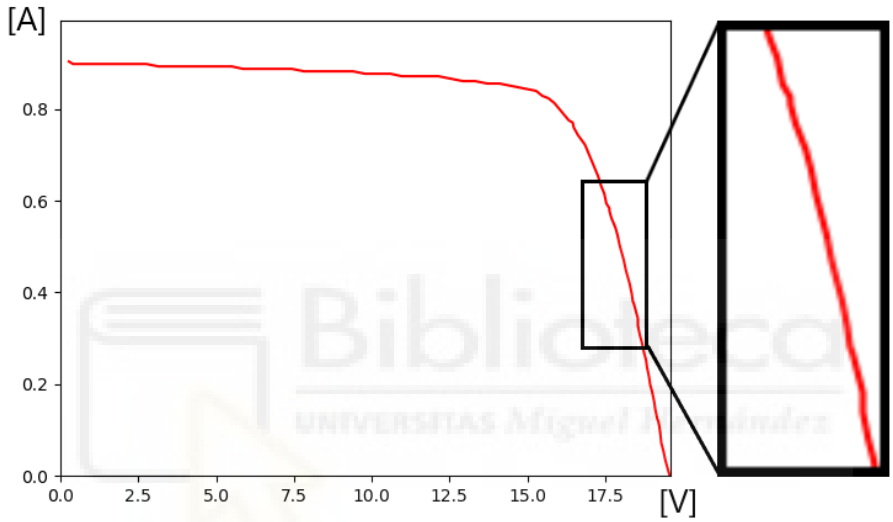


Figura 78. Curva resultante con la resistencia de 2kΩ

En segundo lugar, se ha realizado una medida con una resistencia de 4.7kΩ. En la figura 79 se muestra la curva IV.

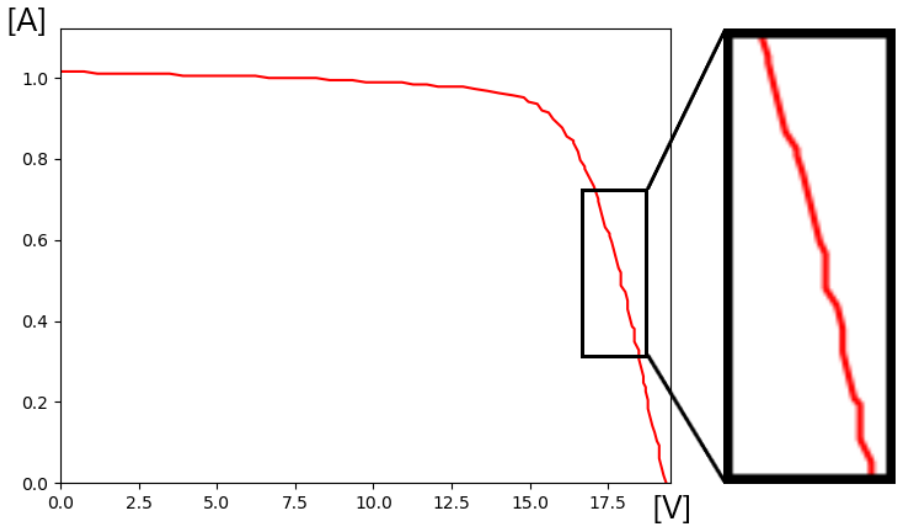


Figura 79. Curva resultante con la resistencia de 4.7kΩ

Y en tercer lugar, se ha realizado una medida con una resistencia de 10kΩ. En la figura 80 se muestra la curva IV.

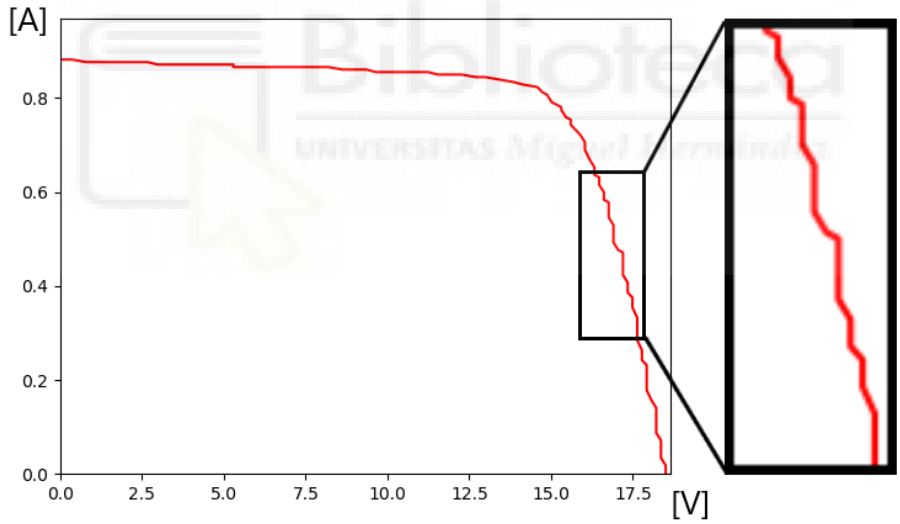


Figura 80. Curva resultante con la resistencia de 10kΩ

Como se puede apreciar en las tres gráficas, la que mayor precisión otorga es la más cercana a la tensión de circuito abierto del panel, que en este caso es la de 2kΩ. Y la que peor precisión tiene es la de la resistencia más lejana, que es la de 10kΩ. En definitiva, el selector de resistencias en el circuito de medida es clave para obtener una mayor precisión en la medida de la tensión.

5.3. Posibles aplicaciones para este dispositivo

A priori puede resultar que la energía producida por una instalación fotovoltaica es directamente proporcional a la radiación incidente en el plano del panel solar. Sin embargo, al igual que en otros procesos de generación eléctrica, las pérdidas son un factor a tener en cuenta como hemos comentado al principio de este documento. Es por ello, que el dispositivo de medida que se ha desarrollado en este trabajo puede ser de gran utilidad a la hora de evaluar el rendimiento de instalaciones.

En el Departamento de Ingeniería Mecánica y Energía de la Universidad Miguel Hernández se llevó a cabo el diseño de un sistema de refrigeración de paneles solares por enfriamiento evaporativo. Este sistema estaba basado en una chimenea solar dispuesta en la parte posterior de los módulos fotovoltaicos por donde se rociaba agua de manera descendente (Figura 81). El propósito de este proyecto era reducir las pérdidas por temperatura para aumentar la tensión de circuito abierto. Para verificar el funcionamiento de este sistema, los días 16 y 17 de marzo de 2021 se acudió al edificio Torrepinet del campus de Elche para realizar varias tandas de medidas con y sin refrigeración en un mismo panel solar.

Figura 81. Fotografía de las chimeneas de refrigeración

Cabe destacar que, debido a las fechas en las que se realizaron las medidas, el dispositivo de medida que se utilizó fue un primer prototipo del medidor (Figura 82). Este prototipo no tenía ajustada la precisión del Analog Discovery, por lo que los resultados no son los que se obtendrían con el dispositivo final.

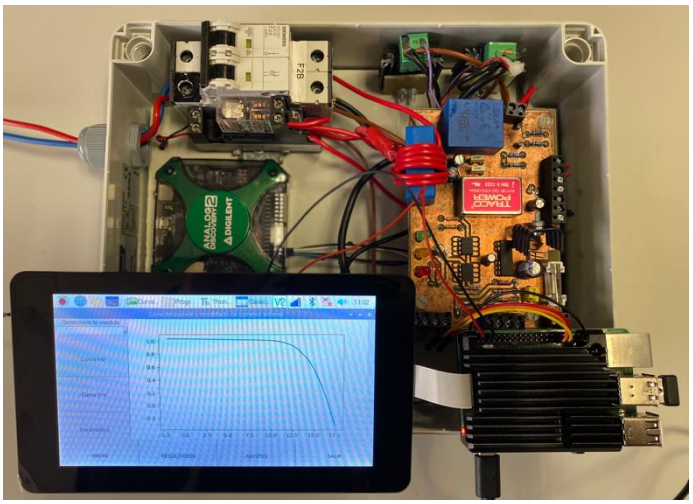


Figura 82. Fotografía del prototipo con el que se realizaron las medidas

Durante el día 16 se realizaron medidas con la chimenea encendida desde las 11:21 hasta las 12:20, y con la chimenea apagada desde las 12:30 hasta las 13:42. Mientras que el día 17 se realizaron medidas con el sistema de refrigeración apagado desde las 11:10 hasta las 12:07, y encendido desde las 12:08 hasta las 13:55. Ambos días las condiciones climáticas eran parecidas.

Para tener una vista general del resultado de las medidas, en la figura 83 se muestra una gráfica en la que se ha representado las potencias de todas las medidas en función de la irradiancia tanto para el panel refrigerado como el sin refrigerar. Y como se puede apreciar en ella, el sistema aumenta la potencia del panel entorno al 10% cuando está conectado.

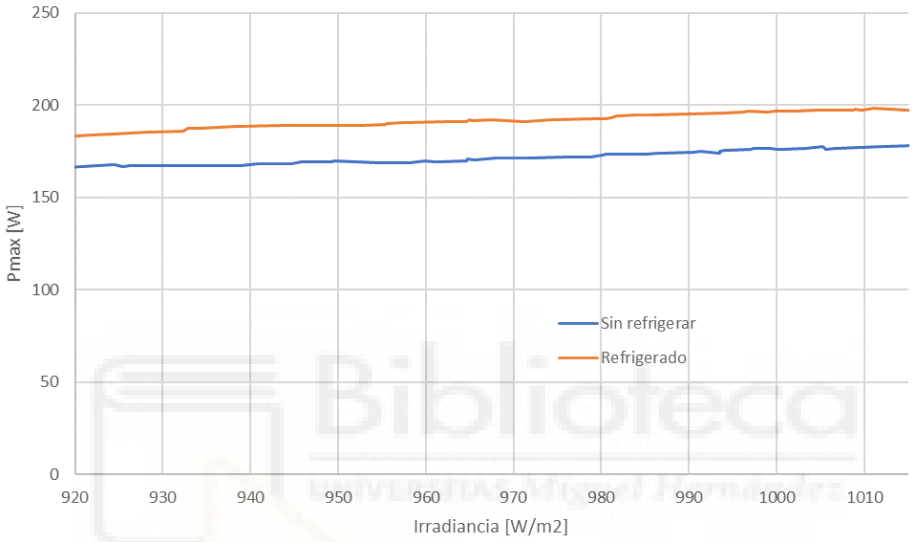


Figura 83. Gráfica de la potencia en función de la irradiancia en cada caso

Se han tomado dos medidas de cada caso con irradiancias y temperaturas ambiente similares para comparar sus curvas características. En las figuras 84 y 85 se muestran las temperaturas del panel en cada caso, mientras que en la figura 86 se muestra la comparativa de la curva P-V, y en la figura 87 la I-V.

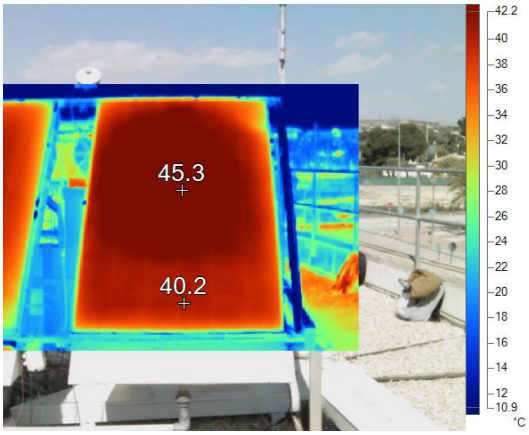


Figura 85. Captura térmica del panel sin refrigerar

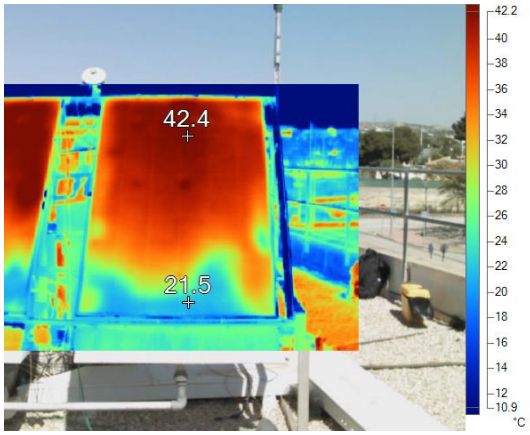


Figura 84. Captura térmica del panel refrigerado

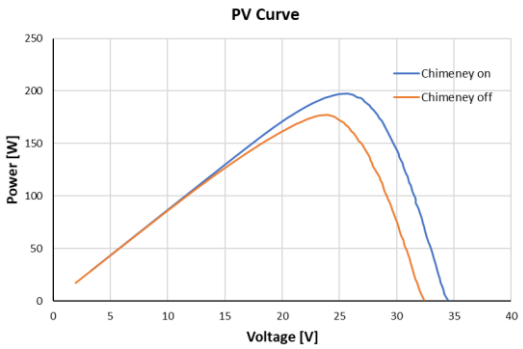


Figura 87. Comparativa curva P-V

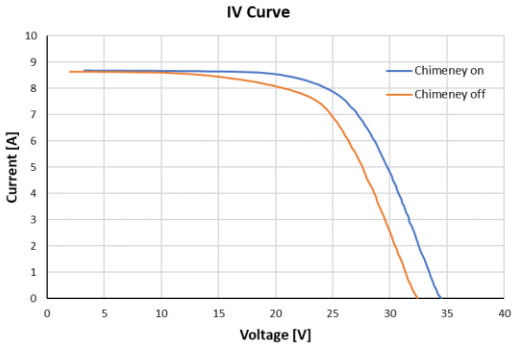


Figura 86. Comparativa curva I-V

Tras analizar ambas curvas observamos como aumenta la tensión en circuito abierto cuando la temperatura de las células del panel es menor, a pesar de que las condiciones climatológicas de ambas medidas son muy similares.

CAPÍTULO 6. Conclusiones y líneas abiertas

Tras realizar el diseño y la implementación del dispositivo electrónico portátil para la caracterización de paneles solares, se ha podido comprobar la inmediatez y precisión que se puede llegar a obtener con el método de la carga capacitiva. Además, gracias a las tandas de medidas que se han realizado, se ha podido constatar la importancia de la rapidez con la que se efectúa la medida, pues la variación de la irradiancia y la temperatura influyen en el resultado de esta.

Otro punto por destacar es la incorporación de la pantalla táctil con la interfaz gráfica, ya que nos ha permitido tratar los datos de manera instantánea. Además permite al usuario controlar el dispositivo sin necesidad de conocimientos previos de Python o del funcionamiento de este.

Además, gracias al desarrollo de este dispositivo, se ha podido elaborar un artículo para la conferencia internacional HEFAT (Heat Transfer, Fluid Mechanics and Thermodynamics). En él se ha verificado el desempeño del sistema de refrigeración de paneles solares por enfriamiento evaporativo diseñado por el Departamento de Ingeniería Mecánica y Energía de la Universidad Miguel Hernández. Este documento se ha dejado adjunto en el ANEXO IV.

En cuanto a posibles líneas futuras de este proyecto, podemos plantear dos distintas. Por un lado, la mejora de la parte de software del dispositivo, como por ejemplo:

- Incluir una nueva función para representar múltiples medidas en función de un parámetro (Hora, irradiancia, potencia máxima, …) en la sección de resultados de la interfaz gráfica.
- Incluir un servidor con una base de datos para guardar todas las medidas, de manera que no sea necesario encender el dispositivo o exportarlas a una memoria USB.
- Seguir testeando la interfaz gráfica en busca de situaciones que puedan dar errores que no se hayan contemplado.
- Incluir una base de datos con las características técnicas de distintos modelos de paneles solares para poder contrastarlos con las medidas que se realicen en esos paneles. Está función la llevan incluida la mayoría de trazadores de curvas del mercado.

Por otro lado, podrían realizarse mejoras en el apartado de hardware del dispositivo. Algunos ejemplos serían:

- Aumentar la tensión máxima que se puede medir con el dispositivo para que se acerque a los dispositivos comerciales.
- Incluir más resistencias en el selector para aumentar la precisión en un mayor rango de tensiones.

Bibliogafía

[1] F. Spertino, J. Ahmad, A. Ciocia, P. D. Leo y A. F. Murtaza, «Capacitor charging method for I–V curve tracer and MPPT,» *Elsevier,* pp. 1-14, 2015.

[2] J. P. Alcocer, «SISTEMA AUTOMÁTICO DE MEDIDA DE CURVAS I-V PARA APLICACIONES FOTOVOLTAICAS,» Universidad Miguel Hernández, Elche, 2016.

[3] «Analog Discovery 2 Reference Manual,» Digilent, [En línea]. Available: https://reference.digilentinc.com/test-and-measurement/analog-discovery-2/reference-manual. [Último acceso: 2021].

[4] C. J. Kaiser, The capacitor handbook, Springer, 1990.

[5] M. H. Rashid, Electrónica de potencia, circuitos, dispositivos y aplicaciones, Prentice Hall Hispanoamericana, S.A., 1993.

[6] «ADS1015 / ADS1115,» Adafruit, [En línea]. Available: https://learn.adafruit.com/raspberry-pi-analog-to-digital-converters/ads1015-slash-ads1115. [Último acceso: 2021].

[7] J. M. Blanes, P. Casado, V. Galiano, J. Ruiz, A. Garrigós, D. Marroquí, J. Toledo, C. Torres y M. Lucas, «PHOTOVOLTAIC EVAPORATIVE CHIMNEY I-V MEASUREMENT SYSTEM,» *HEFAT,* pp. 1-6, 2021.

[8] «Interfaces gráficas de usuario con Tk,» [En línea]. Available: https://docs.python.org/es/3/library/tk.html. [Último acceso: 2021].

[9] «Documentation GPIO raspberry pi,» [En línea]. Available: https://www.raspberrypi.org/documentation/usage/gpio/. [Último acceso: 2021].

[10] «Trazador de curva I-V,» mediaprec, [En línea]. Available: https://mediprec.es/trazador-de-curva-i-v-39.html. [Último acceso: 15 06 2021].

[11] «Solar Cells,» [En línea]. Available: http://www.chemistryexplained.com/Ru-Sp/Solar-Cells.html. [Último acceso: 2021].

[12] J. Toledo, J. M. Blanes, V. Galiano y A. Laudani, «Photovoltaic Modelling,» [En línea]. Available: https://pvmodel.umh.es/pvmodel/viewmodel/1916. [Último acceso: 2021].

[13] «HTI 0254 Trazador de curvas I-V hasta 1500V / 15A,» amazon, [En línea]. Available: https://www.amazon.es/HTI-Trazador-curvas-hasta-1500V/dp/B07N8961X9. [Último acceso: 15 06 2021].

Anexos

I. Presupuesto

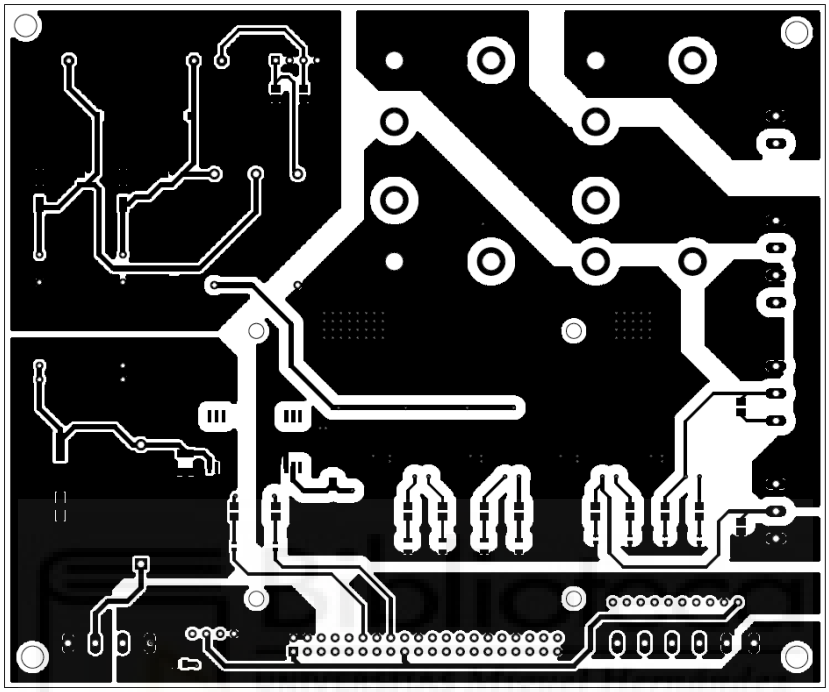
En este anexo se ha dejado adjunto el presupuesto del proyecto. Todos los componentes han sido comprados en la tienda online Digi-Key. Se han incluido los componentes necesarios para fabricar un solo dispositivo. Los precios de la tabla están actualizados al día 20 de junio de 2021 y tienen el IVA incluido.

Componente	Referencia	Cantidad	Precio unitario	Precio total
Raspberry pi 4 8GB	RASPBERRY PI 4 MODEL B 8G	1	62.82 €	62.82 €
Carcasa de refrigeración	4341	1	20.90 €	20.90 €
Pantalla 7'' Rpi	8997466	1	50.26 €	50.26 €
Analog Discovery 2	410-321	1	334.20 €	334.20 €
Transductor de corriente	LV 25-P	1	22.81 €	22.81 €
Transductor de tensión	LA 100-P	1	51.31 €	51.31 €
Optoacoplador	HCPL-2231-500E	4	5.77 €	23.08 €
Convertidor DC DC 12V	DCP021212U	1	8.04 €	8.04 €
Convertidor AC/DC 5V 12V	TMP 30252C	1	65.67 €	65.67 €
Convertidor DC/DC +/-15V	TEN 5-1223	1	21.70 €	21.70 €
Relé 20A 12V	G7L-2A-X-L DC12	2	72.94 €	145.88 €
Transistor BJT NPN 350V	BST39TA	2	0.33 €	0.66 €
Transistor MOSFET N 240V	BSS87H6327FTSA1	4	0.38 €	1.51 €
Transistor IGBT 1200V	IHW20N120R5XKSA1	1	2.78 €	2.78 €
Portafusibles	4527	1	0.57 €	0.57 €
Cubierta portafusibles	4527C	1	0.27 €	0.27 €
Fusible	5SF 1-R	1	0.20 €	0.20 €
Condensador 2200uF	LNX2G222MSEG	1	43.56 €	43.56 €
Condensador 22uF	EEE-FK1E220R	2	0.35 €	0.70 €
Condensador 33uF	EEE-FN1J330P	1	6.10 €	6.10 €
Condensador 100uF	EEE-TK1J101AQ	1	1.59 €	1.59 €
Condensador 1uF	EEE-HD1H1R0R	1	0.30 €	0.30 €
Condensador 47nF	CL21B473KCFNNNE	1	0.08 €	0.08 €
Condensador 100nF	CL21B104KCFNNNE	6	0.08 €	0.48 €
Resistencia 2k 1W	RMCF2512JT2K00	1	0.25 €	0.25 €
Resistencia 10k 1W	RMCF2512JT10K0	1	0.25 €	0.25 €
Resistencia 4.7k 1W	RMCF2512JT4K70	4	0.25 €	1.00 €
Led blanco SMD	EAST2012WA1	4	0.21 €	0.85 €
Led rojo SMD	SML-D12U1WT86	2	0.10 €	0.19 €
Led verde SMD	SML-D12M1WT86	4	0.10 €	0.38 €
Cabecera 40 pines	5103308-8	1	2.27 €	2.27 €
Conector Rpi	1988	1	2.47 €	2.47 €
Cable HDMI 2.0	BC-HH003F	1	5.85 €	5.85 €
Cable USB 3.0	A-USB30AM-30AM-050	2	4.63 €	9.26 €
Cable RJ45	AMJG0808-0050-BKB-26	1	2.06 €	2.06 €

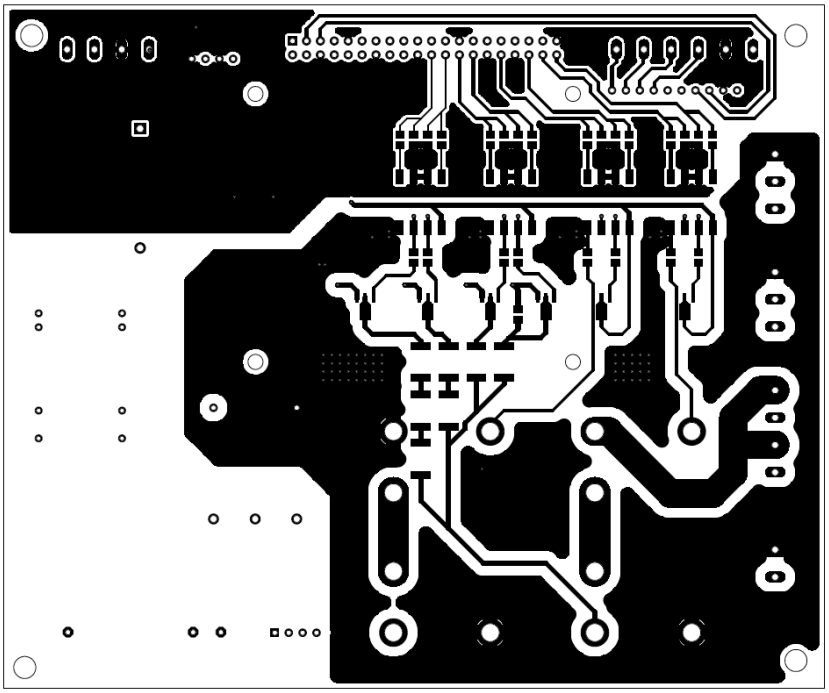
Caja aluminio 250x250	1550WNBK	1	41.92 €	41.92 €
Conector RJ45	EHRJ45P5ES	1	10.20 €	10.20 €
Conector USB 3.0	AC-USB3-AAB	2	12.25 €	24.50 €
Conector HDMI	AC-HDMI-RRB	1	10.26 €	10.26 €
Conector IEC320-C14	1-1609112-3	1	9.62 €	9.62 €
			TOTAL	986.80 €

II. Diseño PCB

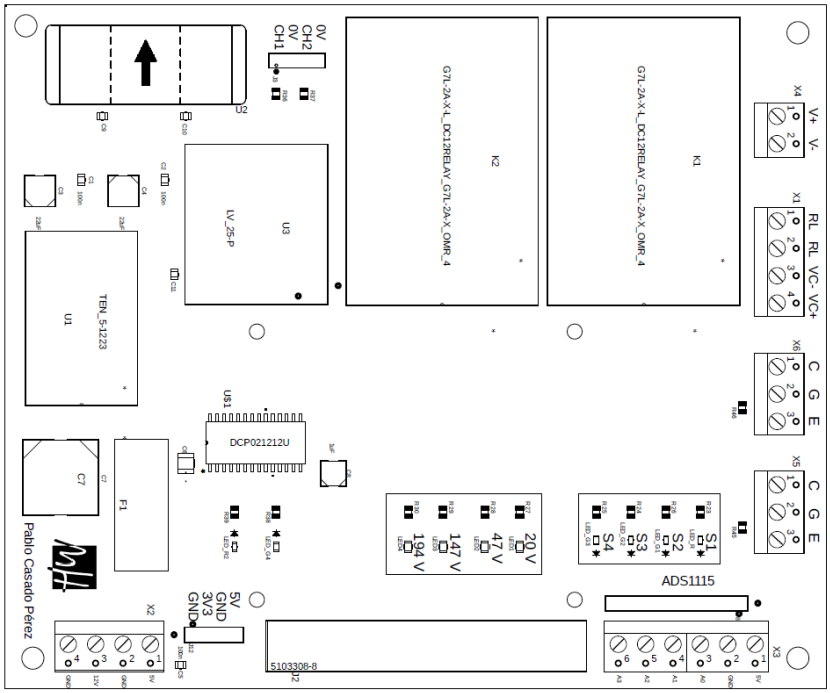
El diseño se ha realizado con el programa EAGLE de Autodesk. A continuación, mostraremos algunas de las vistas de la placa del proyecto. En primer lugar, el gerber de la capa de cobre de la cara superior:



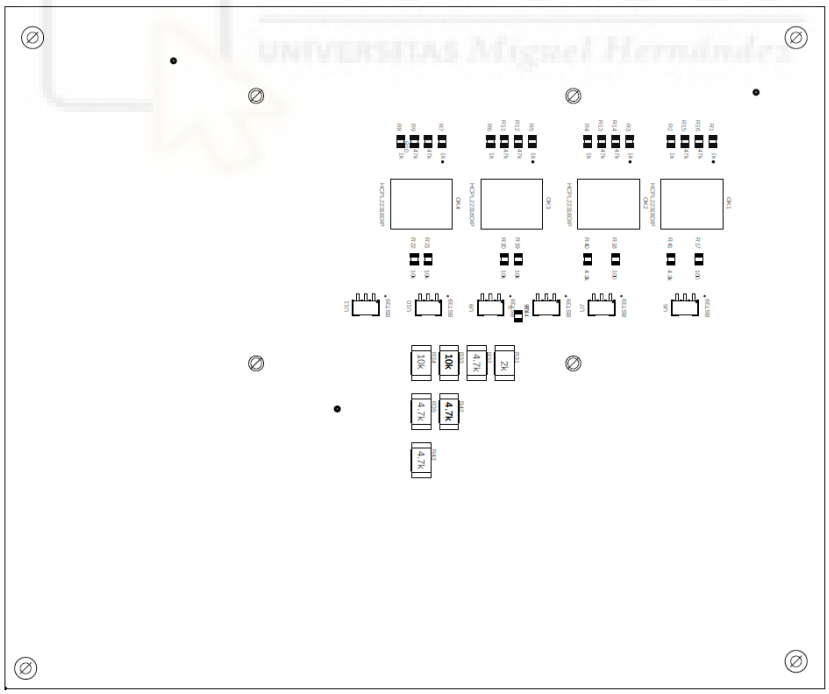
En segundo lugar, el gerber de la capa de cobre de la cara inferior de la PCB:



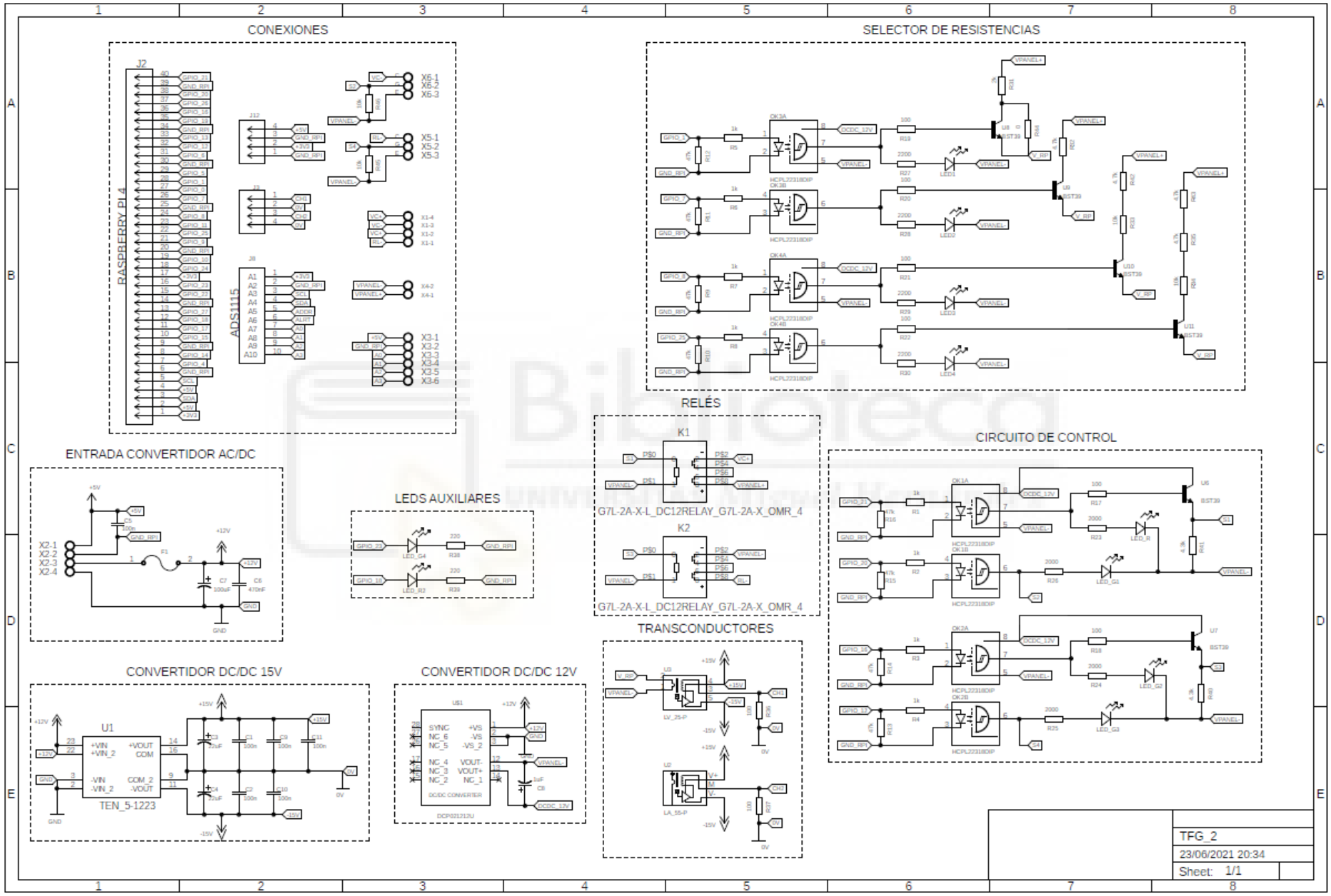
Seguidamente, el gerber de la serigrafía de la cara superior de la placa:



Y finalmente, el gerber de la serigrafía de la cara inferior de la placa:

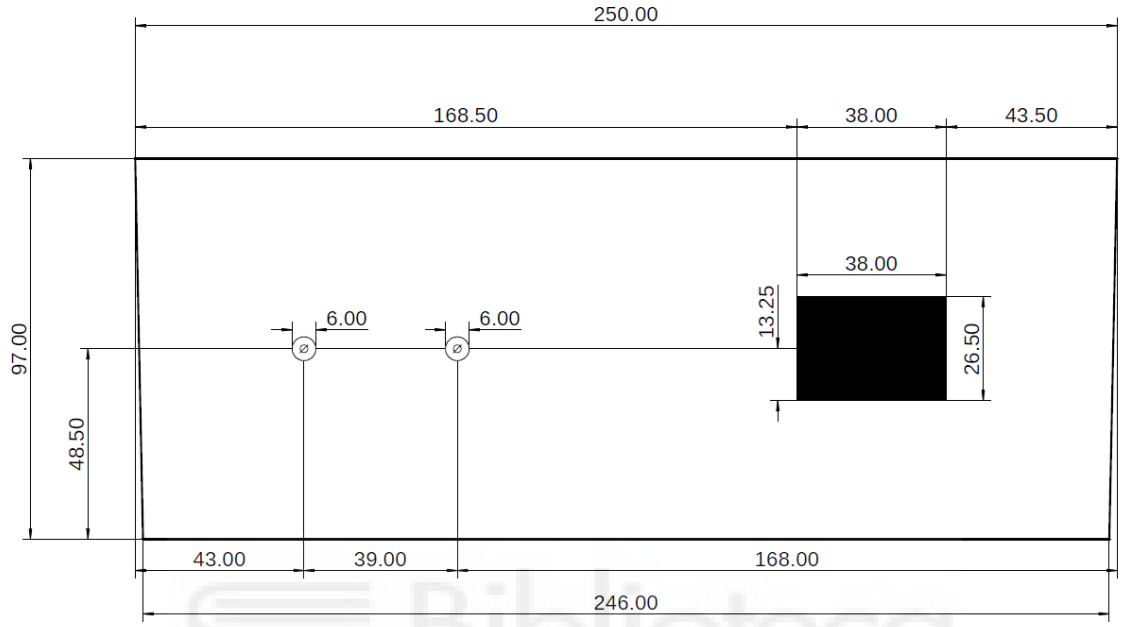


En la siguiente hoja se ha dejado adjunto el esquemático del proyecto de EAGLE.

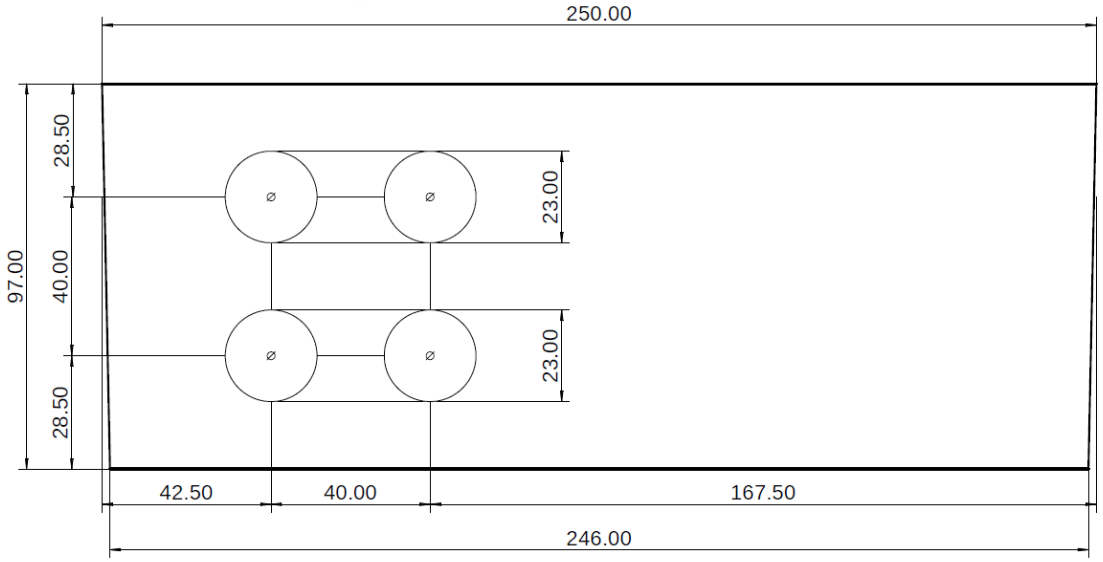

CONEXIONES
SELECTOR DE RESISTENCIAS
RELÉS
ENTRADA CONVERTIDOR AC/DC
LEDS AUXILIARES
CIRCUITO DE CONTROL
CONVERTIDOR DC/DC 15V
CONVERTIDOR DC/DC 12V
TRANSCONDUCTORES
RASPBERRY PI 4
ADS1115
G7L-2A-X-L_DC12RELAY_G7L-2A-X_OMR_4
TEN_5-1223
TFG_2
23/06/2021 20:34
Sheet: 1/1

III. Planos para el mecanizado de la caja

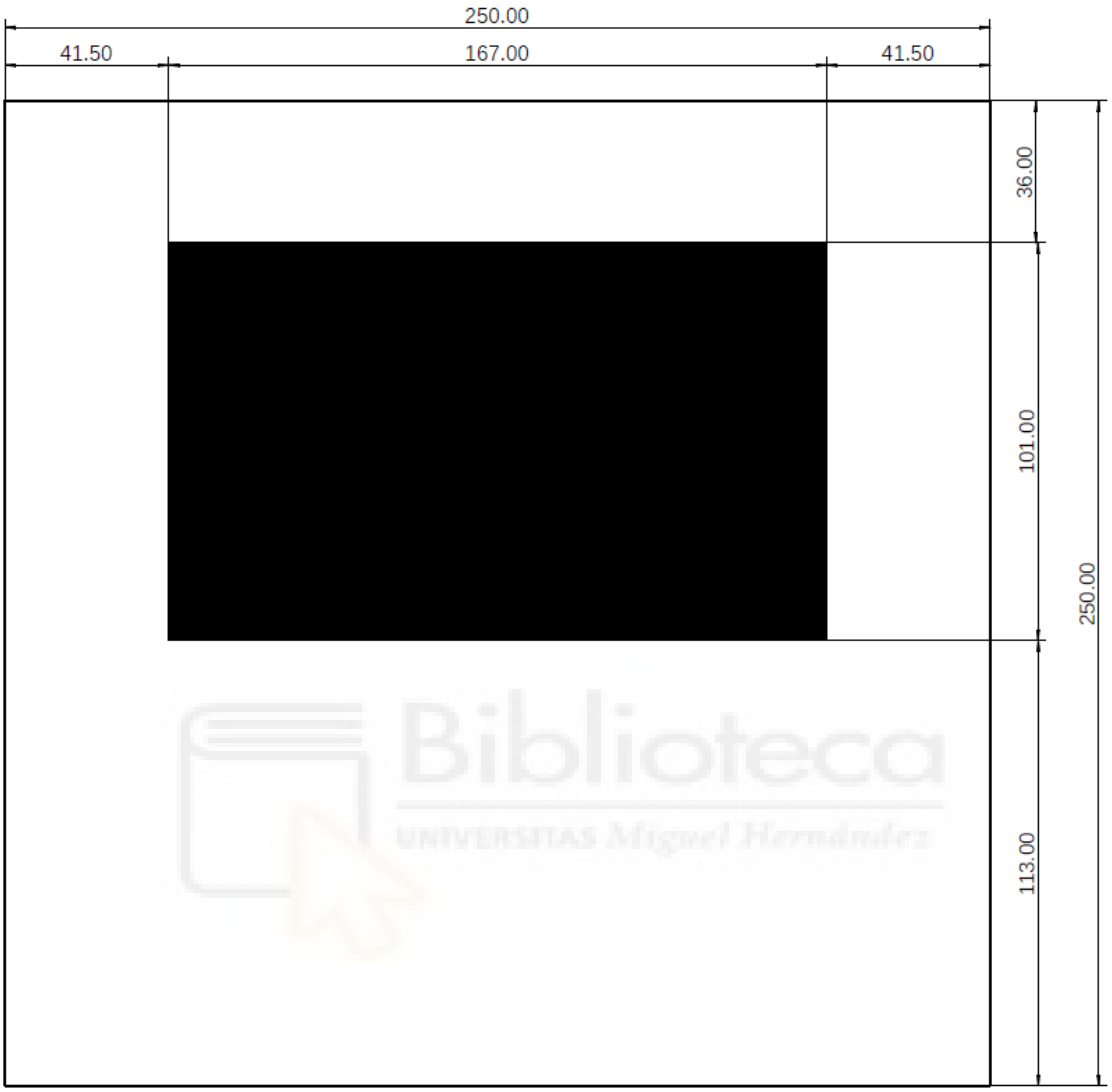
En primer lugar, se muestra el plano del lateral superior de la caja, en el que se posicionarán los conectores de potencia del dispositivo.



En segundo lugar, el plano del lateral inferior de la caja, en el que se posicionaran los conectores de Ethernet, HDMI y USB 3.0 del dispositivo.



Y por último, el plano de la tapa de la caja, en el que únicamente está el recorte cuadrado para la pantalla de 7 pulgadas.



IV. Artículo HEFAT 2021

PHOTOVOLTAIC EVAPORATIVE CHIMNEY I-V MEASUREMENT SYSTEM

Casado P.[1], Blanes J.M.*[1], Galiano V.[2], Ruiz J.[3], Garrigós A.[1], Marroquí D.[1], Toledo J.[4], Torres C.[1] and Lucas M.[3]
*Author for correspondence
[1] Industrial Electronics Group
[2] Department of Physics and Computer Architecture
[3] Department of Mechanical Engineering and Energy
[4] Center of Operation Research
University Miguel Hernández of Elche
Elche, 03202, Spain
E-mail: jmblanes@umh.es

ABSTRACT

This paper deals with the design and implementation of an *I-V* curve measurement system to be used in the analysis of a photovoltaic evaporative chimney system. A photovoltaic evaporative chimney is a novel system that aims to increase the efficiency of solar photovoltaic modules by evaporative cooling. Till the date, the experimental tests presented to assess the advantages of this system have been always performed on a system where the photovoltaic modules are directly connected to a grid-tied inverter and where only the working point fixed by the inverter is measured. The aim of this work is to implement an electronic system to measure the full *I-V* curve of the photovoltaic panels in real time. The *I-V* curve tracer presented is based on the use of a capacitive load controlled by a single board computer. The final design includes protections, capacitor charging/discharging power electronics, remote commands inputs and current, voltage, irradiance and temperature sensors. Finally, some photovoltaic chimney curve measurements are presented.

NOMENCLATURE

DC	[-]	Direct Current
G	[W/m^2]	Irradiance
I2C	[-]	Inter Integrated Circuit
IGBT	[-]	Insulated Gate Bipolar Transistor
I_{MPP}	[A]	Maximum Power Point Current
I_{PH}	[A]	Single diode model photocurrent
I_S	[A]	Single diode model diode saturation current
I_{SC}	[A]	Short Circuit Current
I-V	[-]	Current-Voltage
MPP	[-]	Maximum Power Point
MPPT	[-]	Maximum Power Point Tracker
n	[A]	Single diode model diode ideality factor
N_s	[-]	Number of solar cells in series
P_{MPP}	[W]	Maximum Power Point Power
PV	[-]	Photovoltaic
P-V	[-]	Power-Voltage
R_s	[Ω]	Single diode model diode series resistance
R_s	[Ω]	Single diode model diode series resistance
R_{sh}	[Ω]	Single diode model diode shunt resistance
T	[K]	Temperature
V_{MPP}	[V]	Maximum Power Point Voltage
V_{OC}	[V]	Open Circuit Voltage

INTRODUCTION

A photovoltaic evaporative chimney is a novel system that increase the efficiency of solar photovoltaic modules by evaporative cooling. It is based on a solar chimney attached to the rear side of the PV modules that enhances its performance by cooling down the panel due to the buoyancy-driven flow induced in the chimney [1,2]. Moreover, in the so-called evaporative area, water is sprayed parallel to the downward airflow by a series of nozzles. As the water descends, a small part of it evaporates, cooling the remaining water. This zone works as a small-scale cooling tower. The air that has been in contact with water may have reduced its temperature (it will depend on ambient conditions), enhancing the cooling effect in the panel. As the water used for cooling the modules will be available to be used for the condensation of a refrigeration cycle, the system also increases the efficiency of the heat pump (water-cooled system). Hence, the benefits of the photovoltaic evaporative chimney are two-fold. Figure 1, reproduced from [2], shows the schematic arrangement of the *PV* evaporative solar chimney system integrated in an air conditioning scheme.

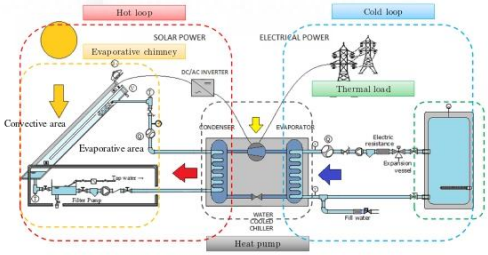


Figure 1 Schematic of the *PV* evaporative chimney system

The results presented in [1] show that the use of the photovoltaic chimney increases the solar module efficiency about 4.9% on average. Nevertheless, in the same work, it was verified that, with the use of the evaporative chimney, the *PV* module temperature is no longer constant, and a temperature

stratification appears. The cooling effect of the chimney is very evident in the lower part of the module (more than 10ºC), less evident in the middle (3.5ºC) and almost ineffective in the upper part of the panel.

Nowadays, most *PV* modules use bypass diodes to prevent output current reduction and hot-spots due to shading effects [3]. In a *PV* module, all the cells connected in series carry the same current. When there is a mismatch in the cells working conditions, and some cells that produce more current than others, the current flow will find the bypass diodes path reducing the output voltage. Usually there is not one bypass diode per cell, instead the cells are grouped in strings and one bypass diode is used in each string. The activation of bypass diodes has one undesired effect, it creates local maximum power points (*MPP*), and the maximum power tracking (*MPPT*) circuit precision can be compromised.

Figure 2 shows the configuration of the solar modules used in the *experimental* setup presented in [1] and [2], model Sunrise SR-P660255. These solar modules have 60 cells in series divided in three strings of 20 cells, each string has two bypass diodes in parallel.

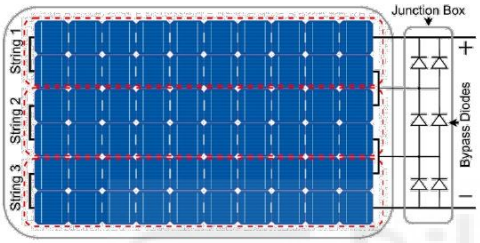


Figure 2 SR-P660255 Bypass Diode Configuration

In case that one of the strings is not able to generate the same current that the other ones, its respective bypass diode will be activated and the resulting *I-V* and *P-V* curves will have the shape of the curves shown in Figure 3. The situation could be even worse if there is a mismatch between the three strings, in this case three zones will appear (two diodes activated, one diode activated, and no diode activated, with three different local *MPP*).

In the literature, these electrical mismatch conditions are usually related with shading, but, if non uniform artificial cooling is used, cell temperature mismatches could also trigger this effect. Under same irradiance conditions, cells at lower temperature can produce more power because their open circuit voltage is higher and so the *MPP* voltage, but, on the other hand, their short circuit current is slightly decreased, so there are some cells that produce more current than others appearing a potential option to activate the bypass diodes.

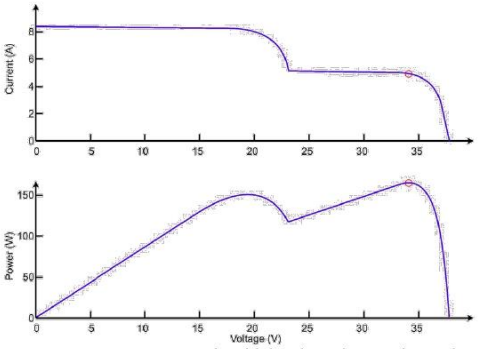


Figure 3 *I-V* Curve example with local maximums due to the bypass diodes activation

Till the date, the experimental tests presented to assess the efficiency of the evaporative chimney have been always performed on a system where the photovoltaic panels are directly connected to a grid-tied inverter. It is supposed that the inverter maximum power tracking (*MPPT*) system ensures the extraction of the maximum power available although if local maximums are presented it could not be the case.

The aim of this work is to design and implement an electronic system to measure the full *I-V* of the *PV* modules in real time, in order to ensure that the evaporative cooling system is not producing enough cell temperature mismatching conditions to activate the string bypass diodes and that the power extracted by the inverter is the maximum.

The rest of the paper is organized as follows. The proposed *I-V* measurement system design is detailed in Section II. Next, experimental results are presented in Section III and Section IV concludes this work.

I-V MEASUREMENT SYSTEM DESIGN

To measure *PV* modules, *I-V* curves different methods can be used but the principle of operation of all of them is the same, they are based on a controlled sweep of the current provided by the panel, from the short circuit point to the open circuit point. The methods described in the literature to perform the curve sweep are:

1. Variable Resistance.

The simplest way to measure the *I-V* curve is by using a variable resistor connected directly to the panel. The value of the resistor is increased in small steps from zero to open circuit, measuring the current and voltage provided by the panel in each step. These variations can be carried out manually [4], making the process very slow or automated [5,6] (for example connecting different resistors in parallel and using switches controlled by a computer and automating the measurements). This method is only valid for low power installations because

during the measurement all the generated power by the *PV* modules are dissipated in the resistors.

2. Electronic Load

The electronic load method is based on using a transistor as a variable load connected to the panel [7-10]. To sweep the curve, the transistor works in the three possible modes of operation (on, linear zone and off), controlled by the voltage applied to the gate. The great advantage of this method, with respect to the variable resistance, is its simpler control and the fast and accurate variation of the transistor equivalent resistance. This method can only be used for medium / low power measurements since the transistor must be able to dissipate all the power supplied by the photovoltaic installation.

3. DC-DC converter

The *DC-DC* converters ability to emulate a resistor has been used in many applications. This ability can also be applied to obtain the *I-V* curves in *PV* applications [11]. The value of the emulated resistance can be varied by modifying the converter duty cycle and therefore, if a duty cycle sweep is performed, a characteristic curve sweep will also be performed. Buck-Boost type converters and their variations (SEPIC and Cük) are the only ones that allow a complete sweep of the curve. Buck type converters do not allow the curve to be captured near the short circuit point and Boost type ones do not allow to capture the open circuit point. The main problem of this methodology is the current/voltage ripple caused by the commutations, which makes difficult to accurately measure the curve.

4. Capacitive load.

This measurement method is based on connecting the *PV* modules to a high-capacity discharged capacitor, the capacitor will charge from zero volts up to the module open circuit voltage of the panel, sweeping the entire characteristic curve [12, 13]. The capacitance needed depends on the speed of the measurement equipment, the short circuit current and the open circuit voltage of the modules.

The implemented curve tracer is based on this last method because its simplicity and accuracy. For the connection and disconnection of the installation to the measurement equipment, a relay with four poles is used while the charging and discharging of the capacitor is controlled by two *IGBTs* transistors. The capacitor current and voltage during the sweep are captured through an Analog Discovery 2 USB oscilloscope connected to a single board computer, Raspberry Pi 4 model B, that controls all the process and also acquire the data from the temperature and irradiance sensors that are connected through an analog-digital converter via *I2C* protocol. The full *I-V* curve acquisition process has been automated with a python application and a user-friendly interface created. Figure 4 shows a simplified block diagram of the *I-V* curve tracer designed.

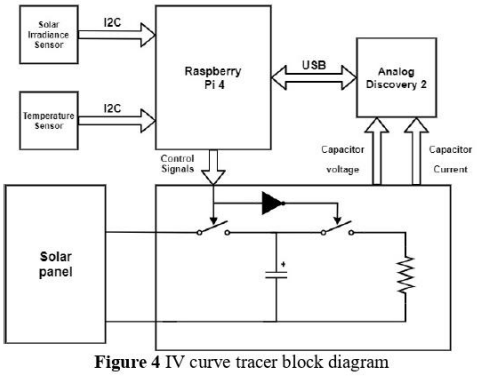


Figure 4 IV curve tracer block diagram

Next more a detailed description of three main parts of the system is given.

Power Electronics Circuit

The main element is the capacitor with a capacitance enough, so the charging process is not excessively fast and allowing the acquisition of enough sensed points to characterize the curve. It is worth noting that the distribution of points acquired on the curve is not uniform, since initially (when the capacitor is discharged) the photovoltaic module will provide its maximum current and the capacitor will charge quickly and at the end of the acquisition the photovoltaic module approaches its open circuit providing low current and charging the capacitor slowly. Taking into account that the acquisition system maximum sample rate is 100 MS/s, the capacitor has been dimensioned so it can acquire more than one thousand points of the curve in the worst-case scenario (maximum short circuit current 22 A, minimum open circuit voltage 10 V), so a 2200 µF capacitance is needed. Finally, a 2200 µF / 3000 V Vishay electrolytic capacitor has been selected. The rest of the power circuit consists in a relay that is used to isolate the photovoltaic panel from system when no measurements are needed. Besides, the *IGBTs* transistors have the function of connecting and disconnecting the capacitor to charge it through the module or to discharge it through a resistor.

Measurement Circuit

The data to be measured during each curve capture are: The capacitor, voltage and current, the ambient temperature and the irradiance. For voltage and current measurements, the LV25-P and LA55-P sensors from the LEM manufacturer will be used. The outputs of these sensors are directly connected to an Analog Discovery 2 USB oscilloscope that is controlled by the Raspberry Pi 4. The temperature is measured with a Texas Instruments LM35 sensor and the irradiance with a sensor model 6450 from the manufacturer Davis. The temperature and irradiance sensors are connected to an analog-digital converter ADS1015 from Adafruit that communicates with the Raspberry Pi using the *I2C* communication protocol.

Control circuit

The control circuit is governed by a single board computer, Rapsberry Pi 4 model B, that controls the power electronics and the measurements circuits. All control and monitoring software has been programmed in Python. A 7″ touchscreen is used to program, export and represent the measurements. The system has also wireless connection so it can be used remotely if a wireless network is available. All the system is powered by a 12V DC input, this input can be a battery or a commercial *AC/DC* adapter.

Figure 5 shows the *I-V* curve tracer system, it is no assembled so the different parts can be shown.

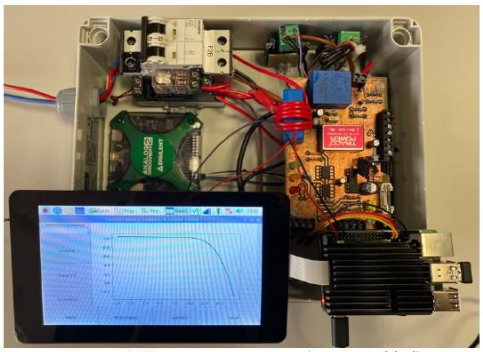


Figure 5 IV curve tracer system (not assembled)

EXPERIMENTAL RESULTS

The Photovoltaic Evaporative Chimney system under test is installed on a laboratory roof at the Universidad Miguel Hernandez of Elche (38°16'N), Spain. The basis of the solar installation consists of two photovoltaic modules Sunrise, SR-P660255, see specifications on Table 1. The orientation for the PV modules is true south (Azimuth angle 0°) and 45° tilt angle. The detailed configuration of the hydraulic circuit can be found in [1]

Magnitude	Units	Value
P_{MPP}	W	255
Dimensions	mm	1637x992x40
Number of cells	-	60
Number of diodes	-	6
Cell Type	-	Poly-Cristalline
I_{SC}	A	9.11
V_{OC}	V	37.49
V_{MPP}	V	30.24
I_{MPP}	A	8.44
Temp. coef I_{SC}	(%/ºC)	0.55
Temp. coef V_{OC}	(%/ºC)	-0.33
Temp. coef P_{MPP}	(%/ºC)	-0.44

Table 1 Sunrise SR-660255 specifications

In Figures 6 and 7 it is shown two thermographic photographs of one of the *PV* modules. Both photographs were taken under same irradiance and ambient temperature conditions, but in Figure 6 the evaporative chimneys is not activated and in Figure 7 it is activated. It is evident the cooling effect of the evaporative chimney and also the temperature stratification with a more of 20ºC difference between the lower and upper part of the PV.

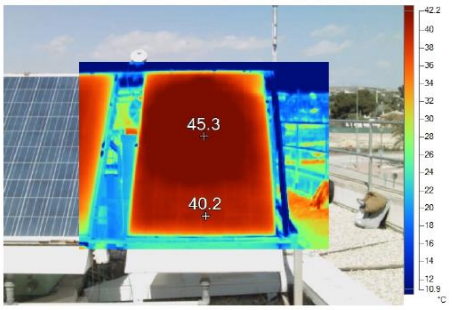


Figure 6 PV Chimney Not Activated - Module Temperature Distribution

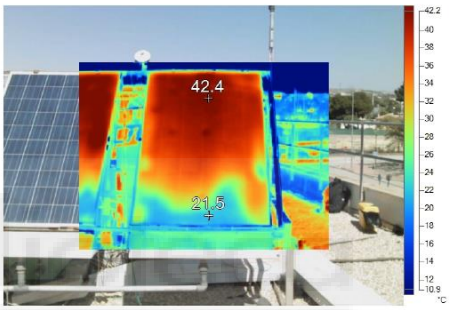


Figure 7 PV Chimney Activated - Module Temperature Distribution

Figures 8 and 9 show two examples of curves measured with the designed *I-V* curve tracer. Both curves were measured under same irradiance and ambient temperature conditions, the difference between both measurements is the solar chimney system activation. It can be seen that, although the *PV* module temperature is not uniform when the chimney is activated, the bypass diodes of the panel are not activated, and no local maximums appears in the *P-V* curve. Besides it is clear the advantage of using the evaporative chimney, it can be seen in the figures that although the short circuit current is slightly reduced due to the cooling effect, the open circuit voltage increases what also deals in an increase of the maximum power that can be extracted from the module. In this example, the maximum power increase was around 11% with the use of the chimney (197 W versus 177 W)

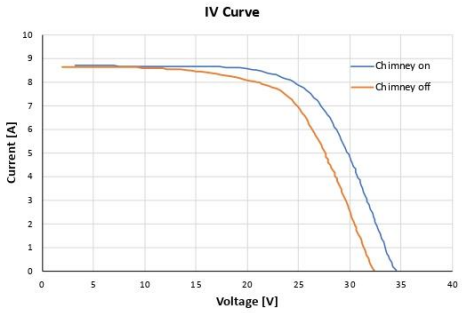


Figure 8 IV Curves comparison with chimney activated and deactivated

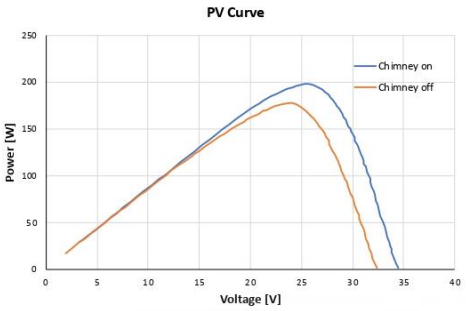


Figure 9 PV Curves comparison with chimney activated and deactivated

Besides, in order to assure that the temperature stratification caused by the chimney cooling does activate the bypass diodes causing local maximums, a periodic measurement of the curves was been programmed, performing a curve sweep each minute. Some of the curves extracted at different times are depicted in Figure 10. No local maximums were found.

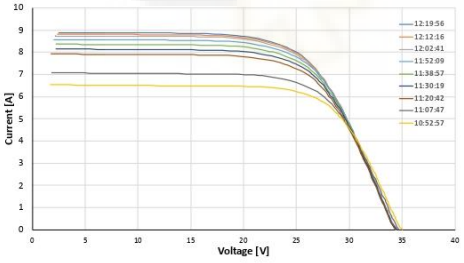


Figure 10 IV Curves extracted with chimney activated

Finally, a comparison between the maximum power extracted with the chimney activated and deactivated for difference irradiance levels it is shown in Figure 11. It is worth noting that the tests were performed during two consecutive days, at the same hours and the ambient temperature was almost the same both days. The whole *I-V* curve was measured each minute using the designed curve measurement system, and the maximum power was extracted later just for the comparison. The benefits of evaporative chimney cooling effects are obvious increasing the power extracted from the *PV* modules at the same working conditions.

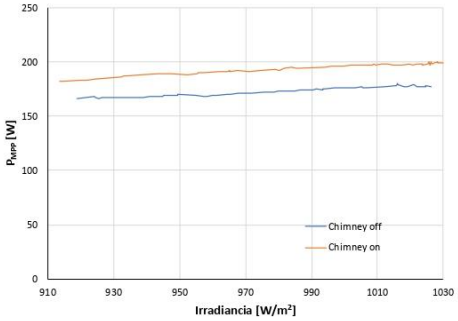


Figure 11 Maximum power available for different irradiance levels with the chimney activated and deactivated

CONCLUSION

In this paper an *I-V* curve measurement system has been presented. The aim of this system is to analyze the behavior of a photovoltaic evaporative chimney system measuring the full *I-V* curve in real time. Experimental results have shown that, although a temperature stratification appears in the *PV* modules due to the chimney cooling effect, the modules bypass diodes are not activated, and no local maximum power points appears. Besides, the curves measured show the benefits the photovoltaic chimney system. The cooling effect increases the power extracted from the photovoltaic installation around 10%.

REFERENCES

[1] M. Lucas, F.J. Aguilar, J. Ruiz, C.G. Cutillas, A.S. Kaiser, P.G. Vicente, Photovoltaic Evaporative Chimney as a new alternative to enhance solar cooling, *Renewable Energy*, vol 111, pp 26-37, October 2017.

[2] J. Ruiz, P. Martínez, H. Sadafi, F.J. Aguilar, F.J. Toledo, J.M. Blanes, M. Lucas, Analytical optimization of a solar driven cooling system enhanced with a photovoltaic evaporative chimney, *Proceedings of the 14th International Conference on Heat Transfer. Fluid Mechanics, and Thermodynamics, Wicklow, Ireland, June 2019.*

[3] Vieira, R.G.; de Araújo, F.M.U.; Dhimish, M.; Guerra, M.I.S. A Comprehensive Review on Bypass Diode Application on

Photovoltaic Modules. *Energies* **2020**, *13*, 2472. [2] A.Q. Malik, S.J. BinHajiDamit, "Outdoor testing of single crystal silicon solar cells ", Renewable Energy, 28, 2003,pp.1433-1445.
[4] E.E. Van Dyk, A.R. Gxasheka, E.L. Meyer, "Monitoring current-voltage characteristics of photovoltaic modules", IEEE PVSC, 2002, pp. 1516-1519.
[5] E.E. Van Dyk, A.R. Gxasheka, E.L. Meyer, "Monitoring current-voltage characteristics and energy output of silicon photovoltaic modules", RenewableEnergy, 30, 2005, pp. 399-411
[6] F. Recart, H. et aI., "Simple data acquisition of the current-voltage and illumination-voltage curves of solar cells ", IEEE WCPEC, 2006, pp. 1215-1218.
[7] Y . Kuai, S. Y uvarajan, "An electronic load for testing photovoltaic panels", Journal of Power Sources, 154, 2006, pp. 308-313.
[8] N. Forero, J. Hernandez, G. Gordillo, "Development of a monitoring system for a PV solar plant", Energy Conversion & Management, 47, 2006, pp. 2329-2336.
[9] Salmon, R. Phelps et al., "Solar cell measurement system for NPS Spacecraft Architecture and Technology Demonstration Satellite NPSAT1", *17th* Annual AIAAl USU Conference on Small Satellites, 2003.
[10] A. Garrigos, J. M. Blanes, "Power Mosfet is core of regulated DC load", EDN 2005, pp 92-93.
[11] E.Duran,J.A.Gomez,M.Sidrach,J.M.Andujar,"Measuring the I-V Curve of PV Generators", IEEE Industrial Electronics Magazine, September 2009.
[12] F. Recart, H. et aI., "Simple data acquisition of the current-voltage and illumination-voltage curves of solar cells ", IEEE WCPEC, 2006, pp. 1215-1218.
[13] J. Munoz, E. Lorenzo, "Capacitive load based on IGBTs for on-site characterization of PV arrays ", Solar Energy, 80, 2006, pp. 1489-1497.

V. Código librería TFG_library.py

```
#!/usr/bin/env python

# -*- coding: utf-8 -*-
"""TFG_library.py: Caracterización y modelado de paneles solares
en tiempo real."""

__author__ = "Pablo Casado Pérez"



# ======================== Librerías
=======================================

from ctypes import cdll, c_int, byref, create_string_buffer,
c_double
import sys
import RPi.GPIO as gpio
import time
import numpy as np
import Adafruit_ADS1x15
from datetime import datetime
import matplotlib.pyplot as plt
from matplotlib.ticker import AutoMinorLocator
import os
import warnings

# Definimos la librería DWF DLL
if sys.platform.startswith("win"):
    dwf = cdll.dwf    # Windows
else:
    dwf = cdll.LoadLibrary("libdwf.so")   # Linux

# ======= Constantes =======

PIN_S1 = 40
PIN_S2 = 38
PIN_S3 = 36
PIN_S4 = 32

PIN_R20 = 29
PIN_R47 = 26
PIN_R147 = 24
```

```
PIN_R194 = 22

C = 0.0022
SAMPLES = 16000

# Conversion ratio transconductor 2500:1000
KV_194 = 78.548
KV_147 = 59.764
KV_47 = 19.712
KV_20 = 9.04128

KI = 2  # 7 vueltas y resistencia de 100R

FS = 4.096/32767
KG = 1000*1/1.67
KT = 100

N_VECTOR = 100

str_action = ''

gpio.setwarnings(False)

# ====== Clases y objetos =======

# Clase para la medida
class Measure:
    # Tensión e intensidad
    V = np.array(SAMPLES)
    I = np.array(SAMPLES)
    P = np.array(SAMPLES)

    # Tensión e intensidad con N puntos
    V_n = np.zeros(N_VECTOR)
    I_n = np.zeros(N_VECTOR)
    P_n = 0

    # Tensión e intensidad filtrada
    V_valid = np.array(1)
    I_valid = np.array(1)
    P_valid = np.array(1)

    V_invalid = np.array(1)
    I_invalid = np.array(1)
```

```
    P_invalid = np.array(1)

    # Parámetros
    filename = ""
    date = ""
    hour = ""
    Voc = 0
    Isc = 0
    Pmax = 0
    Vmp = 0
    Imp = 0
    FF = 0
    G = 0
    Eff = 0
    T1 = 0
    T2 = 0

    # Constantes
    Kv = KV_194 # Factor del transconductor de tensión para la
medida

    # Función para imprimir por pantalla los parametros de la
medida
    def Print_Measure(self):
        print ("\n - Parametros:")
        print('\t\tHora : '+ self.hour)
        print('\t\tVoc = ' + str(self.Voc) + ' [V]')
        print('\t\tIsc = ' + str(self.Isc) + ' [A]')
        print('\t\tPmax = ' + str(self.Pmax) + ' [W]\t\t(Vmp,
Imp) = ('+ str(self.Vmp) + ',' + str(self.Imp) + ')')
        print('\t\tFF = ' + str(self.FF))
        print('\t\tG = ' + str(self.G) + ' [W/m2]')
        print('\t\tEficiencia = ' + str(self.Eff * 100) + ' %')
        print('\t\tTemperatura 1 = ' + str(self.T1) + ' ºC')
        print('\t\tTemperatura 2 = ' + str(self.T2) + ' ºC')

    # Función para generar un string con los parametros de la
medida
    def toString(self):
        S_Measure =  '\n' + self.date + '\t' + self.hour + '\t'
+ '{}'.format(self.Voc) + '\t' + '{}'.format(self.Isc) + '\t' +
'{}'.format(self.Pmax) + '\t' +
'{}'.format(self.Vmp).replace("[","").replace("]","") + '\t' +
'{}'.format(self.Imp).replace("[","").replace("]","") + '\t' +
'{}'.format(self.FF) + '\t' + str(self.G) + '\t' +
```

```
'{}'.format(self.T1) + '\t' + '{}'.format(self.T2)#+
'{}'.format(self.Eff) + '\t'
        return S_Measure

# Clase para el AD2
class AD2():

    hdev = c_int()
    t_muestreo = 0.05
    Sample_rate = 1000000

    def __init__(self):

        version = create_string_buffer(16)
        dwf.FDwfGetVersion(version)

        # Consultar el número de AD2 conectados a la RPi
        cdevices = c_int()
        dwf.FDwfEnum(c_int(0), byref(cdevices))

        # Abrimos el AD2
        self.hdev = c_int()
        #dwf.FDwfDeviceOpen(c_int(0), byref(self.hdev))     #
Configuración para hasta 8k muestras por canal
        dwf.FDwfDeviceConfigOpen(c_int(0), c_int(1),
byref(self.hdev))     # Configuración para hasta 16k muestras
por canal

        # Configuramos las entradas analógicas
        dwf.FDwfAnalogInFrequencySet(self.hdev,
c_double(self.Sample_rate))
        dwf.FDwfAnalogInChannelFilterSet(self.hdev, c_int(-1),
c_int(1))

        # Fijamos los dos canales con la mayor precisión posible
        dwf.FDwfAnalogInChannelRangeSet(self.hdev, c_int(-1),
c_double(4))

    # Función para cambiar la frecuencia de muestreo
    def Set_samplerate(self, Voc, Isc):

        t_muestreo = C*Voc/Isc
        Sample_rate = int(SAMPLES/t_muestreo)
```

```
        dwf.FDwfAnalogInFrequencySet(self.hdev,
c_double(Sample_rate))


# ======================== Funciones
=======================================

# Función para fijar los pines de la Raspberry Pi
def Set_GPIO():
    gpio.setmode(gpio.BOARD)
    gpio.setup(PIN_S1, gpio.OUT)
    gpio.setup(PIN_S2, gpio.OUT)
    gpio.setup(PIN_S3, gpio.OUT)

    gpio.setwarnings(False)
    gpio.setup(PIN_R20, gpio.OUT)
    gpio.setup(PIN_R47, gpio.OUT)
    gpio.setup(PIN_R147, gpio.OUT)
    gpio.setup(PIN_R194, gpio.OUT)

    # Inicializamos el selector de resistencias para que no pase
corriente por ninguna
    gpio.output(PIN_R20, False)
    gpio.output(PIN_R47, False)
    gpio.output(PIN_R147, False)
    gpio.output(PIN_R194, False)

# Función para poner el condensador a descargar
def Discharge_capacitor():
    gpio.output(PIN_S1, False)
    gpio.output(PIN_S2, True)
    gpio.output(PIN_S3, True)

# Función para deshabilitar el selector de resistencias
def Disable_selector():
    gpio.output(PIN_R20, False)
    gpio.output(PIN_R47, False)
    gpio.output(PIN_R147, False)
    gpio.output(PIN_R194, False)


# Función para la adquisición de datos
def Analog_In(AD2):
    # Comienza la adquisición
```

```
    dwf.FDwfAnalogInConfigure(AD2.hdev, c_int(0), c_int(1))


    sts = c_int()
    while True:
      dwf.FDwfAnalogInStatus(AD2.hdev, c_int(1), byref(sts))
      if sts.value == 2 :
            break
      time.sleep(0.1)
    # Finaliza la adquisición

    # Leemos las muestras de datos almacenadas en el buffer
    CH_1 = (c_double*SAMPLES)()
    CH_2 = (c_double*SAMPLES)()
    dwf.FDwfAnalogInStatusData(AD2.hdev, c_int(0), CH_1 ,
len(CH_1))
    dwf.FDwfAnalogInStatusData(AD2.hdev, c_int(1), CH_2 ,
len(CH_2))

    return CH_1, CH_2


# Función para medir tensión en el panel solar
def Get_Panel_Voltaje(AD2, Kv):
    # Activamos y desactivamos las señales de control necesarias
(Descargar el condensador por seguridad)
    gpio.output(PIN_S1, False)
    gpio.output(PIN_S2, True)
    gpio.output(PIN_S3, True)

    # Aumentamos la frecuencia de muestreo del AD2
    dwf.FDwfAnalogInFrequencySet(AD2.hdev, c_double(1000000))

    # Obtenemos el voltaje sin ponderar
    V, I = Analog_In(AD2)

    # Calculamos el voltaje medio
    AV = sum(V)/len(V)*Kv

    # Volvemos a fijar la frecuencia de muestreo a la anterior
    dwf.FDwfAnalogInFrequencySet(AD2.hdev,
c_double(AD2.Sample_rate))

    print(" - Tensión del panel = ", AV)
```

```
    return AV

# Función para seleccionar la resistencia que más precisión
aporte al transconductor de tensión
def Select_Resistor(AD2, Measure):
    # Seleccionamos la resistencia más alta para medir la
tensión del panel por seguridad
    gpio.output(PIN_R20, False)
    gpio.output(PIN_R47, False)
    gpio.output(PIN_R147, False)
    gpio.output(PIN_R194, True)

    # Obtenemos el voltaje del panel
    Vp = Get_Panel_Voltaje(AD2, KV_194)

    # Ponemos el condensador a descargar por defecto
    Discharge_capacitor()

    # Creamos un array con las diferencias del voltaje del panel
con el asociado a las distintas resistencias del transconductor
    Difference = np.array([abs(Vp - 20), abs(Vp - 47), abs(Vp -
147), abs(Vp - 194)])

    # Creamos un diccionario con los valores de Kv para los
distintos índices del vector diferencia
    Resistor = {
        0: KV_20,
        1: KV_47,
        2: KV_147,
        3: KV_194
    }

    # Le asignamos a  Kv el valor que más se ajusta al panel
    Measure.Kv = Resistor.get(np.argmin(Difference))

    gpio.output(PIN_R194, False)
    time.sleep(0.01)

    # Saturamos el transistor asociado a la resistencia elegida
    if(Measure.Kv == KV_20):
        gpio.output(PIN_R20, True)
        gpio.output(PIN_R47, False)
        gpio.output(PIN_R147, False)
```

```
        gpio.output(PIN_R194, False)

    elif(Measure.Kv == KV_47):
        gpio.output(PIN_R20, False)
        gpio.output(PIN_R47, True)
        gpio.output(PIN_R147, False)
        gpio.output(PIN_R194, False)

    elif(Measure.Kv == KV_147):
        gpio.output(PIN_R20, False)
        gpio.output(PIN_R47, False)
        gpio.output(PIN_R147, True)
        gpio.output(PIN_R194, False)

    elif(Measure.Kv == KV_194):
        gpio.output(PIN_R20, False)
        gpio.output(PIN_R47, False)
        gpio.output(PIN_R147, False)
        gpio.output(PIN_R194, True)

    else: print("ERROR: No se puede seleccionar ninguna
resistencia")

    print(" - Resistencia seleccionada")


# Función para obtener la irradiancia
def Get_Irradiance():
    # Creamos el objeto del modelo del convertidor ADC con la
librería su librería
    adc = Adafruit_ADS1x15.ADS1115()

    # Cogemos varios valores de irradiancia y hacemos la media
    i = 0
    values = [0]*20
    for i in range (len(values)):
        values[i] = adc.read_adc(0, gain = 1)

    # Calculamos la irradiancia
    G = FS*KG*sum(values)/len(values)

    return G
```

```
# Función para obtener las temperaturas
def Get_Temperature():
    # Creamos el objeto del modelo del convertidor ADC con la
librería su librería
    adc = Adafruit_ADS1x15.ADS1115()

    # Cogemos varios valores de temperatura para el primer y
segundo sensor y hacemos la media
    temp_1 = [0]*20
    temp_2 = [0]*20
    i = 0
    for i in range (len(temp_1)):
        temp_1[i] = adc.read_adc(1, gain = 1)
        temp_2[i] = adc.read_adc(2, gain = 1)

    T_1 = round(FS*KT*sum(temp_1)/len(temp_1),1)
    T_2 = round(FS*KT*sum(temp_2)/len(temp_2),1)

    return T_1, T_2


# Función para guardar todos los parámetros en el objeto Medida
def Get_Parameters(Measure):
    global SAMPLES

    # Guardamos la fecha y hora
    Measure.filename = datetime.now().strftime('%Y_%m_%d
%H%M%S')
    Measure.date = datetime.now().strftime('%d/%m/%Y')
    Measure.hour = datetime.now().strftime('%H:%M:%S')

    # Obtenemos la irradiancia y la 'eficiencia'(Habrá que
añadir una opción de introducir el área del panel a medir, de
momento no se ha habilitado la opción)
    Measure.G = int(Get_Irradiance())
    Measure.Eff = 0

    # Obtenemos las temperaturas
    Measure.T1, Measure.T2 = Get_Temperature()

    # Tensión e intensidad filtrada
    m_v = np.argmin(Measure.V)
    m_i = np.argmax(Measure.I)
```

```
    # Elegimos como corte el elemento que esté más a la derecha
de nuestra gráfica
    if m_v > m_i :
        m = m_v
    else:
        m = m_i

    # Nos quedamos con la parte que nos interesa
    Measure.V_valid = Measure.V[m:SAMPLES-1]
    Measure.I_valid = Measure.I[m:SAMPLES-1]

    # Inicializamos dos vectores para la regresion lineal
    V_rs = Measure.V_valid[0:(int)((SAMPLES-1-m)*0.05)]
    I_rs = Measure.I_valid[0: (int)((SAMPLES-1-m)*0.05)]

    with warnings.catch_warnings():
        warnings.simplefilter('ignore', np.RankWarning)
        p = np.poly1d(np.polyfit(V_rs, I_rs, 1))

    Measure.V_invalid = np.linspace(0, Measure.V[m], m+1)
    Measure.I_invalid = np.full((m+1), Measure.I[m])
    i = 1
    while i < m :
        Measure.I_invalid[i] = p(Measure.V_invalid[i])
        i = i + 1

    # Calculamos la potencia válida y la no válida
    Measure.P_valid = Measure.V_valid * Measure.I_valid
    Measure.P_invalid = Measure.V_invalid * Measure.I_invalid

    # Concatenamos la parte válida con la parte de la regresión
lineal
    Measure.V_final = np.concatenate((Measure.V_invalid,
Measure.V_valid),axis = None)
    Measure.I_final = np.concatenate((Measure.I_invalid,
Measure.I_valid),axis = None)
    Measure.P_final = np.concatenate((Measure.P_invalid,
Measure.P_valid),axis = None)


    # Calculamos la potencia
    Measure.P_final = Measure.V_final * Measure.I_final
```

```
    # Calculamos la tensión en circuito abierto y la corriente
de cortocircuito
    Measure.Voc = round(np.amax(Measure.V),2)
    Measure.Isc = round(np.amax(Measure.I_final),2)

    # Calculamos la potencia máxima y el punto de máxima
potencia
    Measure.Pmax = round(np.amax(Measure.P_final),2)
    i = np.argmax(Measure.P_valid)
    Measure.Vmp = round(Measure.V_final[i],2)
    Measure.Imp = round(Measure.I_final[i],2)

    # Calculamos el factor de forma
    Measure.FF = round(Measure.Pmax / (Measure.Voc *
Measure.Isc),1)



    # Tensión e intensidad con N medidas
    D_I = Measure.Isc/(N_VECTOR/2)
    D_V = Measure.Voc/(N_VECTOR/2)

    V_resta = np.array(SAMPLES)
    I_resta = np.array(SAMPLES)


    j = 0
    # Tomamos la mitad de los puntos equiespaciando la tensión
    while j<(N_VECTOR/2):
        V_resta = np.absolute(Measure.V_final - (D_V*j))
        k = np.argmin(V_resta)
        # Almacenamos el nuevo par de valores
        Measure.V_n[j] = Measure.V_final[k]
        Measure.I_n[j] = Measure.I_final[k]
        j+=1

    # Tomamos la otra mitad de los puntos equiespaciando la
corriente
    while j<N_VECTOR:
        I_resta = np.absolute(Measure.I_final - (D_I*(j-
(N_VECTOR/2))))
        l = np.argmin(I_resta)
        # Almacenamos el nuevo par de valores
        Measure.V_n[j] = Measure.V_final[l]
        Measure.I_n[j] = Measure.I_final[l]
```

```
        j+=1

    Measure.V_n.sort()
    Measure.I_n[::-1].sort()

    Measure.P_n = Measure.V_n * Measure.I_n


# Función para guardar IV (versión extendida, puede añadir más
tiempo entre medida y medida)
def Save_IV_values(Measure, foulder):

    file = open("{}/datos_medida.txt".format(foulder),"w")
    s = str(round(Measure.V[0], 4)) + "\t"+
str(round(Measure.I[0], 4))
    file.write(s)

    i = 1
    while(i<SAMPLES):
        s = "\n"+ str(round(Measure.V[i], 4)) + "\t"+
str(round(Measure.I[i], 4))
        file.write(s)
        i += 1
    file.close()

    file = open("{}/datos_filtrados.txt".format(foulder),"w")
    s = str(round(Measure.V_valid[0], 4)) + "\t"+
str(round(Measure.I_valid[0], 4))
    file.write(s)

    i = 1
    while(i<np.size(Measure.V_valid)):
        s = "\n"+ str(round(Measure.V_valid[i], 4)) + "\t"+
str(round(Measure.I_valid[i], 4))
        file.write(s)
        i += 1
    file.close()

    file = open("{}/datos_corregidos.txt".format(foulder),"w")
    s = str(round(Measure.V_invalid[0], 4)) + "\t"+
str(round(Measure.I_invalid[0], 4))
    file.write(s)

    i = 1
```

```
    while(i<np.size(Measure.V_invalid)):
        s = "\n"+ str(round(Measure.V_invalid[i], 4)) + "\t"+
str(round(Measure.I_invalid[i], 4))
        file.write(s)
        i += 1
    i = 0
    while(i<np.size(Measure.V_valid)):
        s = "\n"+ str(round(Measure.V_valid[i], 4)) + "\t"+
str(round(Measure.I_valid[i], 4))
        file.write(s)
        i += 1
    file.close()

    file = open("{}/datos(N puntos).txt".format(foulder),"w")
    s = "\n"+ str(round(Measure.V_n[0], 4)) + "\t"+
str(round(Measure.I_n[0], 4))
    file.write(s)

    i = 1
    while(i<N_VECTOR):
        s = str(round(Measure.V_n[i], 4)) + "\t"+
str(round(Measure.I_n[i], 4))
        file.write(s)
        i += 1
    file.close()

    # Guardamos los parámetros
    file = open("{}/parametros.txt".format(foulder),"w")
    s = 'FECHA' + '\t' + 'HORA' + '\t' + 'Voc' + '\t' + 'Isc' +
'\t' + 'Pm' + '\t' + 'Vmp' + '\t' + 'Imp' + '\t' + 'FF' + '\t' +
'G' + '\t' + 'T_1' + '\t' + 'T_2'
    file.write(s)
    string = Measure.toString(Measure)
    s =
string.replace(",","@").replace(".",",").replace("@",".")
    file.write(s)
    file.close()


# Función para guardar IV (versión lite, genera solo un fichero)
def Save_IV_values_EG(Measure, foulder):


    file = open("{}/datos.txt".format(foulder),"w")
```

```
    s = str(round(Measure.V_invalid[0], 4)) + "\t"+
str(round(Measure.I_invalid[0], 4))
    file.write(s)

    i = 1
    while(i<np.size(Measure.V_invalid)):
        s = "\n"+ str(round(Measure.V_invalid[i], 4)) + "\t"+
str(round(Measure.I_invalid[i], 4))
        file.write(s)
        i += 1
    i = 0
    while(i<np.size(Measure.V_n)):
        s = "\n"+ str(round(Measure.V_n[i], 4)) + "\t"+
str(round(Measure.I_n[i], 4))
        file.write(s)
        i += 1
    file.close()


    # Guardamos los parámetros
    file = open("{}/parametros.txt".format(foulder),"w")
    s = 'FECHA' + '\t' + 'HORA' + '\t' + 'Voc' + '\t' + 'Isc' +
'\t' + 'Pm' + '\t' + 'Vmp' + '\t' + 'Imp' + '\t' + 'FF' + '\t' +
'G' + '\t' + 'T_1' + '\t' + 'T_2'
    file.write(s)
    string = Measure.toString(Measure)
    s =
string.replace(",","@").replace(".",",").replace("@",".")
    file.write(s)
    file.close()


# Función para guardar las gráficas (versión extendida, puede
añadir más tiempo entre medida y medida)
def Print_IV_curve(Measure, t_muestreo, foulder):
    t = np.linspace(0, t_muestreo, SAMPLES)

    a_minor = 0.4
    a_major = 1
    iv_color = "#006db2"
    pv_color = "#ef280f"

    # Voltaje
    fig, ax1 = plt.subplots()
    plt.axis([0, t_muestreo, -5, Measure.Voc*1.1])
```

```
    ax1.plot(t,Measure.V, linewidth = 1)
    ax1.xaxis.set_minor_locator(AutoMinorLocator(4))
    ax1.yaxis.set_minor_locator(AutoMinorLocator(4))
    plt.grid(which = 'minor', alpha = a_minor)
    plt.grid(which = 'major', alpha = a_major)
    plt.title('VOLTAJE')
    plt.xlabel('Tiempo [s]')
    plt.ylabel('Voltaje [V]')

plt.savefig('{}/grafica_senal_1(tension).png'.format(foulder),
dpi = 300)

    # Corriente
    fig, ax2 = plt.subplots()
    plt.axis([0, t_muestreo, -1, Measure.Isc*1.1])
    ax2.plot(t,Measure.I, linewidth = 1)
    ax2.xaxis.set_minor_locator(AutoMinorLocator(4))
    ax2.yaxis.set_minor_locator(AutoMinorLocator(4))
    plt.grid(which = 'minor', alpha = a_minor)
    plt.grid(which = 'major', alpha = a_major)
    plt.title('CURRENT')
    plt.xlabel('Tiempo [s]')
    plt.ylabel('Current [A]')

plt.savefig('{}/grafica_senal_2(corriente).png'.format(foulder),
dpi = 300)

    # Curva IV con todos los puntos
    fig, ax3 = plt.subplots()
    plt.axis([0, Measure.Voc, 0, Measure.Isc*1.1])
    ax3.plot(Measure.V, Measure.I, linewidth = 1, color =
iv_color)
    ax3.xaxis.set_minor_locator(AutoMinorLocator(4))
    ax3.yaxis.set_minor_locator(AutoMinorLocator(4))
    plt.grid(which = 'minor', alpha = a_minor)
    plt.grid(which = 'major', alpha = a_major)
    plt.title('I-V CURVE')
    plt.xlabel('Voltaje [V]')
    plt.ylabel('Current [A]')
    plt.savefig('{}/Curva_IV.png'.format(foulder), dpi = 300)

    # Curva PV con todos los puntos
    fig, ax3 = plt.subplots()
    plt.axis([0, Measure.Voc, 0, Measure.Pmax*1.1])
```

```
    ax3.plot(Measure.V, Measure.P, linewidth = 1, color =
pv_color)
    ax3.xaxis.set_minor_locator(AutoMinorLocator(4))
    ax3.yaxis.set_minor_locator(AutoMinorLocator(4))
    plt.grid(which = 'minor', alpha = a_minor)
    plt.grid(which = 'major', alpha = a_major)
    plt.title('P-V CURVE')
    plt.xlabel('Voltaje [V]')
    plt.ylabel('Power [W]')
    plt.savefig('{}/Curva_PV.png'.format(foulder), dpi = 300)

    # Curva IV con n puntos
    fig, ax5 = plt.subplots()
    plt.axis([0, Measure.Voc, 0, Measure.Isc*1.1])
    ax5.plot(Measure.V_n, Measure.I_n, linewidth = 1, color =
'r')
    ax5.xaxis.set_minor_locator(AutoMinorLocator(4))
    ax5.yaxis.set_minor_locator(AutoMinorLocator(4))
    plt.grid(which = 'minor', alpha = a_minor)
    plt.grid(which = 'major', alpha = a_major)
    plt.title('I-V CURVE (n points)')
    plt.xlabel('Voltaje [V]')
    plt.ylabel('Current [A]')
    plt.savefig('{}/Curva_IV_n.png'.format(foulder), dpi = 300)

    #Curva PV con n puntos
    fig, ax6 = plt.subplots()
    plt.axis([0, Measure.Voc, 0, Measure.Pmax*1.1])
    ax6.plot(Measure.V_n, Measure.P_n, linewidth = 1, color =
'r')
    ax6.xaxis.set_minor_locator(AutoMinorLocator(4))
    ax6.yaxis.set_minor_locator(AutoMinorLocator(4))
    plt.grid(which = 'minor', alpha = a_minor)
    plt.grid(which = 'major', alpha = a_major)
    plt.title('P-V CURVE (n points)')
    plt.xlabel('Voltaje [V]')
    plt.ylabel('Power [W]')
    plt.savefig('{}/Curva_PV_n.png'.format(foulder), dpi = 300)


    # Curva IV con todos los puntos filtrada
    fig, ax7 = plt.subplots()
    plt.axis([0, Measure.Voc, 0, Measure.Isc*1.1])
```

```
    ax7.plot(Measure.V_valid, Measure.I_valid, linewidth = 1,
color = iv_color)
    ax7.plot(Measure.V_invalid, Measure.I_invalid, linewidth =
1.1, color = iv_color)
    ax7.xaxis.set_minor_locator(AutoMinorLocator(4))
    ax7.yaxis.set_minor_locator(AutoMinorLocator(4))
    plt.grid(which = 'minor', alpha = a_minor)
    plt.grid(which = 'major', alpha = a_major)
    plt.title('I-V CURVE')
    plt.xlabel('Voltaje [V]')
    plt.ylabel('Current [A]')
    plt.savefig('{}/Curva_IV_filtrada.png'.format(foulder), dpi
= 300)


    # Curva PV con todos los puntos filtrada
    fig, ax8 = plt.subplots()
    plt.axis([0, Measure.Voc, 0, Measure.Pmax*1.1])
    ax8.plot(Measure.V_valid, Measure.P_valid, linewidth = 1,
color = pv_color)
    ax8.plot(Measure.V_invalid, Measure.P_invalid, linewidth =
1.1, color = pv_color)
    ax8.xaxis.set_minor_locator(AutoMinorLocator(4))
    ax8.yaxis.set_minor_locator(AutoMinorLocator(4))
    plt.grid(which = 'minor', alpha = a_minor)
    plt.grid(which = 'major', alpha = a_major)
    plt.title('P-V CURVE')
    plt.xlabel('Voltaje [V]')
    plt.ylabel('Power [W]')
    plt.savefig('{}/Curva_PV_filtrada.png'.format(foulder), dpi
= 300)


    # Curva PV-IV con todos los puntos filtrada
    fig, ax9 = plt.subplots()
    plt.axis([0, Measure.Voc, 0, Measure.Isc*1.1])
    ax9.plot(Measure.V_valid, Measure.I_valid, linewidth = 1,
color = iv_color)
    ax9.plot(Measure.V_invalid, Measure.I_invalid, linewidth =
1.1, color = iv_color)
    ax9.xaxis.set_minor_locator(AutoMinorLocator(4))
    ax9.yaxis.set_minor_locator(AutoMinorLocator(4))
    plt.grid(which = 'minor', alpha = a_minor)
    plt.grid(which = 'major', alpha = a_major)
    plt.title('PV / IV CURVES')
```

```
    plt.ylabel('Current [A]')
    plt.xlabel('Voltaje [V]')

    ax10 = ax9.twinx()
    plt.axis([0, Measure.Voc, 0, Measure.Pmax*1.1])
    ax10.plot(Measure.V_valid, Measure.P_valid, linewidth = 1,
color = pv_color)
    ax10.plot(Measure.V_invalid, Measure.P_invalid, linewidth =
1.1, color = pv_color)
    plt.ylabel('Power [W]')
    plt.savefig('{}/Curva_PV_IV_filtrada.png'.format(foulder),
dpi = 300)

    plt.close()

# Función para guardar las gráficas (versión lite, se generan
las mínimas posibles)
def Print_IV_curve_EG(Measure, t_muestreo, foulder):
    t = np.linspace(0, t_muestreo, SAMPLES)

    a_minor = 0.4
    a_major = 1
    iv_color = "#006db2"
    pv_color = "#ef280f"

    # Voltaje
    fig, ax1 = plt.subplots()
    plt.axis([0, t_muestreo, -5, Measure.Voc*1.1])
    ax1.plot(t,Measure.V, linewidth = 1)
    ax1.xaxis.set_minor_locator(AutoMinorLocator(4))
    ax1.yaxis.set_minor_locator(AutoMinorLocator(4))
    plt.grid(which = 'minor', alpha = a_minor)
    plt.grid(which = 'major', alpha = a_major)
    plt.title('VOLTAJE')
    plt.xlabel('Tiempo [s]')
    plt.ylabel('Voltaje [V]')

plt.savefig('{}/grafica_senal_1(tension).png'.format(foulder),
dpi = 300)

    # Corriente
    fig, ax2 = plt.subplots()
    plt.axis([0, t_muestreo, -1, Measure.Isc*1.1])
    ax2.plot(t,Measure.I, linewidth = 1)
```

```
    ax2.xaxis.set_minor_locator(AutoMinorLocator(4))
    ax2.yaxis.set_minor_locator(AutoMinorLocator(4))
    plt.grid(which = 'minor', alpha = a_minor)
    plt.grid(which = 'major', alpha = a_major)
    plt.title('CURRENT')
    plt.xlabel('Tiempo [s]')
    plt.ylabel('Current [A]')

plt.savefig('{}/grafica_senal_2(corriente).png'.format(foulder),
dpi = 300)

    # Curva IV con todos los puntos filtrada
    fig, ax7 = plt.subplots()
    plt.axis([0, Measure.Voc, 0, Measure.Isc*1.1])
    ax7.plot(Measure.V_valid, Measure.I_valid, linewidth = 1,
color = iv_color)
    ax7.plot(Measure.V_invalid, Measure.I_invalid, linewidth =
1.1, color = iv_color)
    ax7.xaxis.set_minor_locator(AutoMinorLocator(4))
    ax7.yaxis.set_minor_locator(AutoMinorLocator(4))
    plt.grid(which = 'minor', alpha = a_minor)
    plt.grid(which = 'major', alpha = a_major)
    plt.title('I-V CURVE')
    plt.xlabel('Voltaje [V]')
    plt.ylabel('Current [A]')
    plt.savefig('{}/Curva_IV_filtrada.png'.format(foulder), dpi
= 300)

    # Curva PV con todos los puntos filtrada
    fig, ax8 = plt.subplots()
    plt.axis([0, Measure.Voc, 0, Measure.Pmax*1.1])
    ax8.plot(Measure.V_valid, Measure.P_valid, linewidth = 1,
color = pv_color)
    ax8.plot(Measure.V_invalid, Measure.P_invalid, linewidth =
1.1, color = pv_color)
    ax8.xaxis.set_minor_locator(AutoMinorLocator(4))
    ax8.yaxis.set_minor_locator(AutoMinorLocator(4))
    plt.grid(which = 'minor', alpha = a_minor)
    plt.grid(which = 'major', alpha = a_major)
    plt.title('P-V CURVE')
    plt.xlabel('Voltaje [V]')
    plt.ylabel('Power [W]')
    plt.savefig('{}/Curva_PV_filtrada.png'.format(foulder), dpi
= 300)
```

```
    # Curva PV-IV con todos los puntos filtrada
    fig, ax9 = plt.subplots()
    plt.axis([0, Measure.Voc, 0, Measure.Isc*1.1])
    ax9.plot(Measure.V_valid, Measure.I_valid, linewidth = 1,
color = iv_color)
    ax9.plot(Measure.V_invalid, Measure.I_invalid, linewidth =
1.1, color = iv_color)
    ax9.xaxis.set_minor_locator(AutoMinorLocator(4))
    ax9.yaxis.set_minor_locator(AutoMinorLocator(4))
    plt.grid(which = 'minor', alpha = a_minor)
    plt.grid(which = 'major', alpha = a_major)
    plt.title('PV / IV CURVES')
    plt.ylabel('Current [A]')
    plt.xlabel('Voltaje [V]')

    ax10 = ax9.twinx()
    plt.axis([0, Measure.Voc, 0, Measure.Pmax*1.1])
    ax10.plot(Measure.V_valid, Measure.P_valid, linewidth = 1,
color = pv_color)
    ax10.plot(Measure.V_invalid, Measure.P_invalid, linewidth =
1.1, color = pv_color)
    plt.ylabel('Power [W]')
    plt.savefig('{}/Curva_PV_IV_filtrada.png'.format(foulder),
dpi = 300)


    # Curva PV-IV con todos los puntos filtrada tamaño mini
    fig, ax9 = plt.subplots()
    plt.axis([0, Measure.Voc, 0, Measure.Isc*1.1])
    ax9.plot(Measure.V_valid, Measure.I_valid, linewidth = 1,
color = iv_color)
    ax9.plot(Measure.V_invalid, Measure.I_invalid, linewidth =
1.1, color = iv_color)
    ax9.xaxis.set_minor_locator(AutoMinorLocator(4))
    ax9.yaxis.set_minor_locator(AutoMinorLocator(4))
    plt.grid(which = 'minor', alpha = a_minor)
    plt.grid(which = 'major', alpha = a_major)
    plt.title('PV / IV CURVES', fontsize = 22)
    plt.ylabel('Current [A]', fontsize = 22)
    plt.xlabel('Voltaje [V]', fontsize = 22)

    ax10 = ax9.twinx()
    plt.axis([0, Measure.Voc, 0, Measure.Pmax*1.1])
    ax10.plot(Measure.V_valid, Measure.P_valid, linewidth = 1,
color = pv_color)
```

```
    ax10.plot(Measure.V_invalid, Measure.P_invalid, linewidth =
1.1, color = pv_color)
    plt.ylabel('Power [W]', fontsize = 22)
    plt.savefig(os.getcwd() + '/iconos/' +'/Last_measure.png',
dpi = 40)

    plt.close()


# Función para guardar la medida (versión extendida, tarda más
tiempo)
def Save_Measure(Measure, t_muestreo, foulder):
    # Creamos el nuevo nombre del directorio
    new_foulder = foulder + "/{}".format(Measure.filename)

    # Creamos el directorio
    os.mkdir(new_foulder)

    # Guardamos la medida
    Save_IV_values(Measure, new_foulder)
    Print_IV_curve(Measure, t_muestreo, new_foulder)


# Función para guardar la medida (versión lite, tarda menos
tiempo)
def Save_Measure_EG(Measure, t_muestreo, foulder):
    # Creamos el nuevo nombre del directorio
    new_foulder = foulder + "/{}".format(Measure.filename)

    # Creamos el directorio
    os.mkdir(new_foulder)

    # Guardamos la medida
    Save_IV_values_EG(Measure, new_foulder)
    Print_IV_curve_EG(Measure, t_muestreo, new_foulder)


# Función para realizar una medida
def Get_Measure(AD2, Measure, t_discharge, foulder):

    # 1.- Descargar el condensador
    str_action = 'Descarga condensador: '+ str(t_discharge) + '
s'
    print ('\n - ' + str_action)
```

```
        Discharge_capacitor()
        time.sleep(t_discharge)

        # 2.- Activamos el relé
        gpio.output(PIN_S1, True)
        gpio.output(PIN_S2, False)
        gpio.output(PIN_S3, False)

        time.sleep(0.027)

        # 3.- Iniciamos el barrido de valoers I-V durante la carga
    del condensador
        str_action = 'Barrido valores IV durante la carga'
        print ('\n - ' + str_action)
        gpio.output(PIN_S2, True)
        CH_1,CH_2 = Analog_In(AD2)
        Measure.V = np.array(CH_1) * Measure.Kv
        Measure.I = np.array(CH_2) * KI

        # 4.- Descarga del condensador
        str_action = 'Adquisicion completada'
        print ('\n - ' + str_action)
        gpio.output(PIN_S2, False)
        time.sleep(0.001)
        gpio.output(PIN_S1, False)
        time.sleep(0.03)
        Discharge_capacitor()
        Get_Parameters(Measure)
        # Elegimos la versión de guardado de la medida
        #Save_Measure(Measure, AD2.t_muestreo, foulder)
        Save_Measure_EG(Measure, AD2.t_muestreo, foulder)
        Measure.Print_Measure(Measure)

        str_action = 'Descarga condensador: '+ str(t_discharge) + '
    s'
        print ('\n - ' + str_action)
        time.sleep(t_discharge)
        Disable_selector()

str_action = ''
```

VI. Código programa de medida TFG_program.py

```
#!/usr/bin/env python

# -*- coding: utf-8 -*-
"""TFG_program.py: Caracterización y modelado de paneles solares
en tiempo real."""

__author__ = "Pablo Casado Pérez"


import TFG_library as pv
from ctypes import cdll, c_double
import sys
import time
import os

# Definimos la librería DWF DLL en funcion del sistema operativo
if sys.platform.startswith("win"):
    dwf = cdll.dwf    # Windows
else:
    dwf = cdll.LoadLibrary("libdwf.so")   # Linux


N_MEDIDAS = 1
T_MEDIDA = 1
T_DESCARGA = 1.32

def Guardar_archivo(Measure, path):
    #Creamos o abrimos el archivo parametros.txt
    file = open(path + '/parametros.txt', "a+")
    file.write(str(Measure))
    file.close()

def main():

    # Inicializamos los pines de la Raspberry
    pv.Set_GPIO()

    # Ponemos el condensador a descargar por defecto
    pv.Discharge_capacitor()
```

```
        # Establecemos conexión con la AD2 e inicializamos sus
    valores
        ad2 = pv.AD2()

        # Creamos el archivo de texto
        path = os.getcwd() + '/Medidas'
        file = open(path + '/parametros.txt', "a+")
        s = 'FECHA' + '\t' + 'HORA' + '\t' + 'Voc' + '\t' + 'Isc' +
    '\t' + 'Pm' + '\t' + 'Vmp' + '\t' + 'Imp' + '\t' + 'FF' + '\t' +
    'G' + '\t' + 'T_1' + '\t' + 'T_2'
        file.write(s)
        file.close()

        # Creamos una lista de Objetos Measure
        Medida = pv.Measure
        Medida.Kv = pv.KV_194
        i = 0
        for i in range(N_MEDIDAS - 1):
            print('MEDIDA ' + str(i) + '/' + str(N_MEDIDAS) + ':')
            Medida.Kv = pv.KV_194
            pv.Select_Resistor(ad2, Medida)
            pv.Get_Measure(ad2, Medida, T_DESCARGA, path)
            Guardar_archivo(Medida, path)
            time.sleep(T_MEDIDA)

        print('MEDIDA ' + str(N_MEDIDAS) + '/' + str(N_MEDIDAS) +
    ':')
        Medida.Kv = pv.KV_194
        pv.Select_Resistor(ad2, Medida)
        pv.Get_Measure(ad2, Medida, T_DESCARGA, path)
        Guardar_archivo(Medida, path)

        print('FIN')

        # Cerramos Analog Discovery y salimos
        dwf.FDwfDeviceCloseAll()
        exit()

    if __name__ == "__main__":
main()
```

VII. Código interfaz gráfica TFG_EG_program.py

```
#!/usr/bin/env python

# -*- coding: utf-8 -*-
"""TFG_EG_program.py: Caracterización y modelado de paneles
solares en tiempo real."""

__author__ = "Pablo Casado Pérez"


import sys
import matplotlib
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg,
NavigationToolbar2Tk
matplotlib.use('Agg')
from matplotlib.ticker import AutoMinorLocator
import numpy as np
PYTHON_VERSION = sys.version_info.major
from tkinter import ttk
import os
import TFG_library as pv
from ctypes import cdll, c_double
import time
import threading
import datetime
from PIL import ImageTk, Image
import shutil

if PYTHON_VERSION < 3:
    import Tkinter as tk   #Python 2.x
else:
    import tkinter as tk   #Python 3.x

# Definimos la librería DWF DLL
if sys.platform.startswith("win"):
    dwf = cdll.dwf    # Windows
else:
    dwf = cdll.LoadLibrary("libdwf.so")   # Linux

Medida = pv.Measure
```

```
RUN_Status = 0  # 0 parado, 1 empezando, 2 midiendo, 3 y 4 parar

N_measures = 3

t_measure = 0

t_discharge = 1.32

N_points = 100

progress = 0

lst_files = ["Medidas",""]
lst_measures = [None]*2
lst_USB = [None]

file_selected = ""
measure_selected = ""
USB_selected = ""

str_N_measure = ""

name_dir = "Medidas"

menu_FLAG = -1 #0->INICIO ; 1->RESULTADOS ; 2 ->AJUSTES ; 3 ->
Exportar
Curve_FLAG = 0

class Aplicacion:


    def __init__(self, window):
        # Inicializamos los pines de la Raspberry
        pv.Set_GPIO()

        # Ponemos el condensador a descargar por defecto
        pv.Discharge_capacitor()

        # Creamos la ventana
        self.wind = window
        self.wind.attributes('-fullscreen', True)
        self.wind.bind("<F11>", lambda event:
self.window.attributes("-fullscreen", not
self.window.attributes("-fullscreen")))
```

```
        self.wind.bind("<Escape>", lambda event:
self.window.attributes("-fullscreen", False))
        self.wind.title('Caracterización y modelado de paneles
solares')
        self.wind.geometry('800x480')

        s = tk.ttk.Style()
        s.theme_use('clam')



        #-- Frames & Containers--
        self.board_frame = tk.Frame(self.wind, width=800,
height=384, bg = "white")
        self.board_frame.grid(row=0, sticky="nsew")

        control_frame = tk.Frame(self.wind, width=800,
height=96, bg = "white")
        control_frame.grid(row=1, sticky="nsew")

        #-- Variables para las opciones --
        str_N_measures = tk.StringVar()
        str_t_measure = tk.StringVar()
        str_t_discharge = tk.StringVar()
        str_name_dir = tk.StringVar()
        str_action = tk.StringVar()
        str_N_points = tk.StringVar()
        str_info = tk.StringVar()

        str_N_measures.set(str(N_measures))
        str_t_measure.set(str(t_measure))
        str_t_discharge.set(str(t_discharge))
        str_name_dir.set(name_dir)
        str_action.set(pv.str_action)
        str_N_points.set(str(N_points))
        str_info.set('')



        #-- Functions --

        def Save_foulder(self):
            global USB_selected, file_selected, menu_FLAG
```

```
            USB_path = '/media/pi/'+ USB_selected +'/PV_MODEL/'+
file_selected

            try:
                if menu_FLAG == 3:
                    str_info = 'Guardando...'
                    self.info_label["text"] = str_info
                    self.board_frame.update()
                i = 0
                while os.path.exists(USB_path):
                    i += 1
                    USB_path = '/media/pi/'+ USB_selected
+'/PV_MODEL/'+ file_selected + '(' + str(i) + ')'

                shutil.copytree(os.getcwd() + '/Medidas/' +
file_selected, USB_path)
                if menu_FLAG == 3:
                    str_info = '\'' + file_selected + '\'' + '
guardado con éxito'
                    self.info_label["text"] = str_info

            except:
                tk.messagebox.showerror("Error","Ha ocurrido un
error durante el guardado")
                str_info = 'Error al guardar'
                self.info_label["text"] = str_info

            self.board_frame.update()

            print('- Medidas guardadas')


        def Save_measure(self):
            global USB_selected, measure_selected,
file_selected, menu_FLAG

            USB_path = '/media/pi/'+ USB_selected +'/PV_MODEL/'+
measure_selected

            try:
                if menu_FLAG == 3:
                    str_info = 'Guardando...'
                    self.info_label["text"] = str_info
                    self.board_frame.update()
```

```
                i = 0
                while os.path.exists(USB_path):
                    i += 1
                    USB_path = '/media/pi/'+ USB_selected
+'/PV_MODEL/'+ measure_selected + '(' + str(i) + ')'

                shutil.copytree(os.getcwd() + '/Medidas/' +
file_selected + '/' + measure_selected, USB_path)

                if menu_FLAG == 3:
                    str_info = '\'' + measure_selected + '\'' +
' guardado con éxito'
                    self.info_label["text"] = str_info
            except:
                tk.messagebox.showerror("Error","Ha ocurrido un
error durante el guardado")
                str_info = 'Error al guardar'
                self.info_label["text"] = str_info

            self.board_frame.update()

            print('- Medida guardada')


        def Delete_foulder(self):
            global file_selected, menu_FLAG

            try:
                resultado =
tk.messagebox.askquestion("Borrar","¿Está seguro de que desea
borrar la carpeta:"+file_selected+"?")
                if resultado == "yes":
                    if menu_FLAG == 3:
                        str_info = 'Borrando...'
                        self.info_label["text"] = str_info
                        self.board_frame.update()

                    shutil.rmtree(os.getcwd() + '/Medidas/' +
file_selected)

                    if menu_FLAG == 3:
                        str_info = '\'' + file_selected + '\'' +
' ha sido borrado'
                        self.info_label["text"] = str_info
```

```
            except:
                tk.messagebox.showerror("Error","Ha ocurrido un
error al borrar")
                str_info = 'Error al borrar'
                self.info_label["text"] = str_info

            self.board_frame.update()
            file_selected = ''

            print('- Medidas borradas')


        def Delete_measure(self):
            global measure_selected, file_selected, menu_FLAG

            try:
                resultado =
tk.messagebox.askquestion("Borrar","¿Está seguro de que desea
borrar la medida:"+measure_selected+"?")
                if resultado == "yes":
                    if menu_FLAG == 3:
                        str_info = 'Borrando...'
                        self.info_label["text"] = str_info
                        self.board_frame.update()

                    shutil.rmtree(os.getcwd() + '/Medidas/' +
file_selected + '/' + measure_selected)

                    if menu_FLAG == 3:
                        str_info = '\'' + measure_selected +
'\'' + ' ha sido borrado'
                        self.info_label["text"] = str_info

            except:
                tk.messagebox.showerror("Error","Ha ocurrido un
error al borrar")
                str_info = 'Error al borrar'
                self.info_label["text"] = str_info
            self.board_frame.update()
            measure_selected = ''

            print('- Medida borrada')
```

```
        def Export_function():
            global menu_FLAG

            menu_FLAG = 3

            USB_path = "/media/pi"

            lst_USB  = []
            print(USB_path)
            with os.scandir(USB_path) as ficheros:
                 lst_USB = [fichero.name for fichero  in
ficheros if fichero.is_dir()]

            def Selected_USB(event=None):
                global USB_selected

                USB_selected = self.measure_combo_USB.get()
                print('usb : '+ USB_selected)
                self.board_frame.R.update()


            # Borramos el board_frame.R
            self.board_frame.R.pack_forget()
            self.board_frame.R.destroy()

            # Creamos el board_frame.R
            self.board_frame.R = tk.Frame(self.board_frame,
width=590, height=384, bg = "white")
            self.board_frame.R.grid(row = 0, column = 1,
sticky="nsew")

            # Creamos el board_frame.R_r
            self.board_frame.R_r = tk.Frame(self.board_frame.R,
width=200, height=384, bg = "white")
            self.board_frame.R_r.grid(row = 0, column = 1,
sticky="nsew")

            # Creamos el board_frame.R_l
            self.board_frame.R_l = tk.Frame(self.board_frame.R,
width=390, height=384, bg = "white")
            self.board_frame.R_l.grid(row = 0, column = 0,
sticky="nsew")

            # Creamos los botones de la parte derecha
```

```
            self.save_icon = tk.PhotoImage(file  =
"/home/pi/Desktop/Programa_EG_Placa_nueva/version_2.0/iconos/ic_
guardar.png")
            self.save_folder_button =
tk.Button(self.board_frame.R_r, image = self.save_icon, text =
'Guardar carpeta', font = ("Arial",11, "bold"), height = 84,
width = 200, compound="top", command = lambda:
Save_foulder(self), bg = "white", bd = 0)
            self.save_folder_button.pack(side = "top", expand =
True)
            self.save_measure_button =
tk.Button(self.board_frame.R_r, image = self.save_icon, text =
'Guardar medida', font = ("Arial",11, "bold"), height = 84,
width = 200, compound="top", command = lambda:
Save_measure(self), bg = "white", bd = 0)
            self.save_measure_button.pack(side = "top", expand =
True)

            self.delete_icon = tk.PhotoImage(file  =
"/home/pi/Desktop/Programa_EG_Placa_nueva/version_2.0/iconos/ic_
borrar.png")
            self.delete_folder_button =
tk.Button(self.board_frame.R_r, image = self.delete_icon, text =
'Borrar carpeta', font = ("Arial",11, "bold"), height = 84,
width = 200, compound="top", command = lambda:
Delete_foulder(self), bg = "white", bd = 0)
            self.delete_folder_button.pack(side = "top", expand
= True)
            self.delete_measure_button =
tk.Button(self.board_frame.R_r, image = self.delete_icon, text =
'Borrar medida', font = ("Arial",11, "bold"), height = 84, width
= 200, compound="top", command = lambda: Delete_measure(self),
bg = "white", bd = 0)
            self.delete_measure_button.pack(side = "top", expand
= True, fill = "both")

            # Creamos el combobox para seleccionar el USB
            self.tittle = tk.Label(self.board_frame.R_l, text =
'GESTIÓN DE MEDIDAS', font = ("Arial", 12, "bold"), bg =
"white", fg = "#5e5e5e")
            self.tittle.place(relx=0.2, rely=0.05, relwidth=0.6,
relheight=0.1)

            self.USB_list = tk.Label(self.board_frame.R_l, text
= 'Selecciona la memoria USB', height = 1, width = 19, bg =
"white")
            self.USB_list.place(relx=0.25, rely=0.2,
relwidth=0.5, relheight=0.1)
```

```
            self.measure_combo_USB =
tk.ttk.Combobox(self.board_frame.R_l, values = lst_USB)
            self.measure_combo_USB.place(relx=0.25, rely=0.3,
relwidth=0.5, relheight=0.1)

self.measure_combo_USB.bind("<<ComboboxSelected>>",Selected_USB)

            self.info_labelframe =
tk.LabelFrame(self.board_frame.R_l, bg = 'white')
            self.info_labelframe.place(relx=0.1, rely=0.55,
relwidth=0.8, relheight=0.2)

            self.info_folder_label =
tk.Label(self.info_labelframe, text = 'Carpeta:', font =
("Arial",11), justify = "left", bg = 'white')
            self.info_folder_label.grid(row = 0, column = 0,
sticky="nsew", padx = 5, pady = 5)
            self.info_folder_label_2 =
tk.Label(self.info_labelframe, text = file_selected, font =
("Arial",11), justify = "left", bg = 'white')
            self.info_folder_label_2.grid(row = 0, column = 1,
sticky="nsew", padx = 5, pady = 5)

            self.info_measure_label =
tk.Label(self.info_labelframe, text = 'Medida:', font =
("Arial",11), justify = "left", bg = 'white')
            self.info_measure_label.grid(row = 1, column = 0,
sticky="nsew", padx = 5, pady = 5)
            self.info_measure_label_2 =
tk.Label(self.info_labelframe, text = measure_selected, font =
("Arial",11), justify = "left", bg = 'white')
            self.info_measure_label_2.grid(row = 1, column = 1,
sticky="nsew", padx = 5, pady = 5)

            self.info_label = tk.Label(self.board_frame.R_l,
height = 1, width = 19, bg = "white")
            self.info_label.place(relx=0.1, rely=0.8,
relwidth=0.8, relheight=0.1)


        def Print_IV_Curve():

            try:
                path = os.getcwd() + '/Medidas/' + file_selected
+ '/' + measure_selected + '/datos.txt'
```

```
                data = np.loadtxt(path, delimiter='\t', skiprows
= 0, usecols=[0,1])

                a_minor = 0.4
                a_major = 1

                fig, ax1 = plt.subplots(figsize = (6, 3), dpi =
100)
                ax1.plot(data[:,0], data[:,1], linewidth = 1,
color = "#006db2")
                plt.axis([0, np.amax(data[:,0]), 0,
1.1*np.amax(data[:,1])])
                ax1.xaxis.set_minor_locator(AutoMinorLocator(4))
                ax1.yaxis.set_minor_locator(AutoMinorLocator(4))
                plt.grid(which = 'minor', alpha = a_minor)
                plt.grid(which = 'major', alpha = a_major)
                plt.title('I-V CURVE')
                plt.xlabel('Voltaje [V]')
                plt.ylabel('Current [A]')

                # Borramos el board_frame.R
                self.board_frame.R.pack_forget()
                self.board_frame.R.destroy()

                # Creamos el board_frame.R
                self.board_frame.R = tk.Frame(self.board_frame,
width=600, height=384)
                self.board_frame.R.grid(row = 0, column = 1,
sticky="nsew")


                line = FigureCanvasTkAgg(fig,
self.board_frame.R)
                line.get_tk_widget().pack(side = tk.TOP, fill =
"x", expand = 1)
                line.draw()

                toolbar = NavigationToolbar2Tk(line,
self.board_frame.R)
                toolbar.update()
                line.get_tk_widget().pack(side = tk.TOP, fill =
tk.BOTH, expand = 1)
            except FileNotFoundError:
```

```
                tk.messagebox.showwarning("Alerta","Actualice la
pestaña de 'RESULTADOS' y seleccione una medida")



        def Print_PV_Curve():
            try:
                path = os.getcwd() + '/Medidas/' + file_selected
+ '/' + measure_selected + '/datos.txt'
                data = np.loadtxt(path, delimiter='\t', skiprows
= 0, usecols=[0,1])

                a_minor = 0.4
                a_major = 1

                fig, ax2 = plt.subplots(figsize = (6, 3), dpi =
100)
                ax2.plot(data[:,0], data[:,0]*data[:,1],
linewidth = 1, color = "#ef280f")
                plt.axis([0, np.amax(data[:,0]), 0,
1.1*np.amax(data[:,0]*data[:,1])])
                ax2.xaxis.set_minor_locator(AutoMinorLocator(4))
                ax2.yaxis.set_minor_locator(AutoMinorLocator(4))
                plt.grid(which = 'minor', alpha = a_minor)
                plt.grid(which = 'major', alpha = a_major)
                plt.title('P-V CURVE')
                plt.xlabel('Voltaje [V]')
                plt.ylabel('Power [W]')

                # Borramos el board_frame.R
                self.board_frame.R.pack_forget()
                self.board_frame.R.destroy()

                # Creamos el board_frame.R
                self.board_frame.R = tk.Frame(self.board_frame,
width=600, height=384, bg = "white")
                self.board_frame.R.grid(row = 0, column = 1,
sticky="nsew")

                line = FigureCanvasTkAgg(fig,
self.board_frame.R)
                line.get_tk_widget().pack(side = "top", fill =
"both", expand = True)
                line.draw()
```

```
                toolbar = NavigationToolbar2Tk(line,
self.board_frame.R)
                toolbar.update()
                line.get_tk_widget().pack(side = "top", fill =
"both", expand = True)
            except FileNotFoundError:
                tk.messagebox.showwarning("Alerta","Actualice la
pestaña de 'RESULTADOS' y seleccione una medida")

        def Print_Curves():
            global Curve_FLAG

            try:
                if Curve_FLAG == 0:
                    Print_IV_Curve()
                    Curve_FLAG = 1
                else :
                    Print_PV_Curve()
                    Curve_FLAG = 0

            except :
                tk.messagebox.showwarning("Alerta","Actualice la
pestaña de 'RESULTADOS' y seleccione una medida")


        def Print_Parameters():
            path = os.getcwd() + '/Medidas/' + file_selected +
'/' + measure_selected + '/parametros.txt'

            try:
                file = open(path, 'r')

                for linea in file.readlines():
                    columna = str(linea).split("\t")
                    data0 = columna[0]
                    data1 = columna[1]
                    data2 = columna[2]
                    data3 = columna[3]
                    data4 = columna[4]
                    data5 = columna[5]
                    data6 = columna[6]
                    data7 = columna[7]
                    data8 = columna[8]
                    data9 = columna[9]
```

```
                    data10 = columna[10]
                file.close()
                ancho_entry = 15

                # Borramos el board_frame.R
                self.board_frame.R.pack_forget()
                self.board_frame.R.destroy()

                # Creamos el board_frame.R
                self.board_frame.R = tk.Frame(self.board_frame,
width=600, height=384, bg = "white")
                self.board_frame.R.grid_propagate(False)
                self.board_frame.R.grid(row = 0, column = 1,
sticky="nsew")


                self.board_frame.R_u =
tk.Frame(self.board_frame.R, width=600, height=80, bg = "white")
                #self.board_frame.R_u.grid_propagate(False)
                self.board_frame.R_u.place(relx=0, rely=0,
relwidth=1, relheight=0.2)

                self.board_frame.R_d =
tk.Frame(self.board_frame.R, width=600, height=304, bg =
"white")
                #self.board_frame.R_d.grid_propagate(False)
                self.board_frame.R_d.place(relx=0, rely=0.2,
relwidth=1, relheight=0.8)


                #Imprimimos todos los parámetros
                tk.Label(self.board_frame.R_u, text = '', bg =
"white").pack(side = "top")
                tk.Label(self.board_frame.R_u, text = '', bg =
"white").pack(side = "top")
                tk.Label(self.board_frame.R_u, text =
'PARÁMETROS', font = ("Arial", 12, "bold"), bg = "white", fg =
"#5e5e5e").pack(side = "top")
                tk.Label(self.board_frame.R_u, text = '', bg =
"white").pack(side = "top")


                #Imprimimos todos los parámetros
                tk.Label(self.board_frame.R_d, text = '', bg =
"white").grid(row = 1, column = 1, padx = 20, pady = 5)
```

```
                tk.Label(self.board_frame.R_d, text = 'Fecha: ',
bg = "white").grid(row = 2, column = 1, padx = 10, pady = 5)
                self.date_entry = tk.Entry(self.board_frame.R_d,
width = ancho_entry, justify = "center", bd = 0)
                self.date_entry.grid(row = 2, column = 2, padx =
10, pady = 5)

                self.date_entry.insert(0,data0)

                tk.Label(self.board_frame.R_d, text = 'Hour: ',
bg = "white").grid(row = 3, column = 1, padx = 10, pady = 5)
                self.hour_entry = tk.Entry(self.board_frame.R_d,
width = ancho_entry, justify = "center", bd = 0)
                self.hour_entry.grid(row = 3, column = 2, padx =
10, pady = 5)
                self.hour_entry.insert(0,data1)

                tk.Label(self.board_frame.R_d, text = 'Voc: ',
bg = "white").grid(row = 4, column = 1, padx = 10, pady = 5)
                self.Voc_entry = tk.Entry(self.board_frame.R_d,
width = ancho_entry, justify = "center", bd = 0)
                self.Voc_entry.grid(row = 4, column = 2, padx =
10, pady = 5)
                self.Voc_entry.insert(0,data2[:10])

                tk.Label(self.board_frame.R_d, text = 'Isc: ',
bg = "white").grid(row = 4, column = 3, padx = 10, pady = 5)
                self.Isc_entry = tk.Entry(self.board_frame.R_d,
width = ancho_entry, justify = "center", bd = 0)
                self.Isc_entry.grid(row = 4, column = 4, padx =
10, pady = 5)
                self.Isc_entry.insert(0,data3[:10])

                tk.Label(self.board_frame.R_d, text = 'Pmax: ',
bg = "white").grid(row = 5, column = 1, padx = 10, pady = 5)
                self.Pmax_entry = tk.Entry(self.board_frame.R_d,
width = ancho_entry, justify = "center", bd = 0)
                self.Pmax_entry.grid(row = 5, column = 2, padx =
10, pady = 5)
                self.Pmax_entry.insert(0,data4[:10])

                tk.Label(self.board_frame.R_d, text = 'Vmp: ',
bg = "white").grid(row = 6, column = 1, padx = 10, pady = 5)
                self.Vmp_entry = tk.Entry(self.board_frame.R_d,
width = ancho_entry, justify = "center", bd = 0)
```

```
                self.Vmp_entry.grid(row = 6, column = 2, padx =
10, pady = 5)
                self.Vmp_entry.insert(0,data5[:10])

                tk.Label(self.board_frame.R_d, text = 'Imp: ',
bg = "white").grid(row = 6, column = 3, padx = 10, pady = 5)
                self.Imp_entry = tk.Entry(self.board_frame.R_d,
width = ancho_entry, justify = "center", bd = 0)
                self.Imp_entry.grid(row = 6, column = 4, padx =
10, pady = 5)
                self.Imp_entry.insert(0,data6[:10])

                tk.Label(self.board_frame.R_d, text = 'FF: ', bg
= "white").grid(row = 7, column = 1, padx = 10, pady = 5)
                self.FF_entry = tk.Entry(self.board_frame.R_d,
width = ancho_entry, justify = "center", bd = 0)
                self.FF_entry.grid(row = 7, column = 2, padx =
10, pady = 5)
                self.FF_entry.insert(0,data7[:10])

                tk.Label(self.board_frame.R_d, text = 'G: ', bg
= "white").grid(row = 8, column = 1, padx = 10, pady = 5)
                self.G_entry = tk.Entry(self.board_frame.R_d,
width = ancho_entry, justify = "center", bd = 0)
                self.G_entry.grid(row = 8, column = 2, padx =
10, pady = 5)
                self.G_entry.insert(0,data8[:10])

                tk.Label(self.board_frame.R_d, text =
'Temperatura 1: ', bg = "white").grid(row = 10, column = 1, padx
= 10, pady = 5)
                self.T1_entry = tk.Entry(self.board_frame.R_d,
width = ancho_entry, justify = "center", bd = 0)
                self.T1_entry.grid(row = 10, column = 2, padx =
10, pady = 5)
                self.T1_entry.insert(0,data9[:10])

                tk.Label(self.board_frame.R_d, text =
'Temperatura 2: ', bg = "white").grid(row = 10, column = 3, padx
= 10, pady = 5)
                self.T2_entry = tk.Entry(self.board_frame.R_d,
width = ancho_entry, justify = "center", bd = 0)
                self.T2_entry.grid(row = 10, column = 4, padx =
10, pady = 5)
                self.T2_entry.insert(0,data10[:10])
```

```
            except:
                tk.messagebox.showwarning("Alerta","Actualice la
pestaña de 'RESULTADOS' y seleccione una medida")


        def Guardar_archivo(Measure, path):
            file = open(path + '/parametros.txt', "a+")
            file.write(Measure.toString(Measure))
            file.close()


        def Start_measure(self):
            global RUN_Status, N_measures, t_discharge,
t_measure, name_dir, progress, str_N_measure, menu_FLAG, Medida
            str_N_measure = ''
            RUN_Status = 1
            function_home()
            self.board_frame.update()
            time.sleep(1)

            name_dir = str_name_dir.get()
            path = os.getcwd() + '/Medidas/' + name_dir
            os.makedirs(path, exist_ok=True)
            file = open(path + '/parametros.txt', "a+")
            s = 'FECHA' + '\t' + 'HORA' + '\t' + 'Voc' + '\t' +
'Isc' + '\t' + 'Pm' + '\t' + 'Vmp' + '\t' + 'Imp' + '\t' + 'FF'
+ '\t' + 'G' + '\t' + 'T_1' + '\t' + 'T_2'
            file.write(s)
            file.close()

            progress = 0
            if menu_FLAG == 0:
                    self.measure_progressbar["value"] = progress

            i = 0
            for i in range(N_measures - 1):

                if (RUN_Status == 1) or (RUN_Status == 2):
                    print(i)
                    str_N_measure = 'MEDIDA ' + str(i+1) + '/' +
str(N_measures)
                    if menu_FLAG == 0:
                        self.name_label_medida["text"] =
str_N_measure
```

```
                    print(str_N_measure + ':')

                    Medida.Kv = pv.KV_194     # Fijamos la
constante del transconductor de tensión
                    pv.Select_Resistor(ad2, Medida)
                    pv.Get_Measure(ad2, Medida, t_discharge,
path)
                    Guardar_archivo(Medida, path)

                    progress = int((i+1)*(100/N_measures))
                    print(progress)
                    if menu_FLAG == 0:
                        self.measure_progressbar["value"] =
progress
                    self.board_frame.update()

                    time.sleep(t_measure)
                else :
                    break


            if (RUN_Status == 1) or (RUN_Status == 2):

                str_N_measure = 'MEDIDA ' + str(N_measures) +
'/' + str(N_measures)
                if menu_FLAG == 0:
                    self.name_label_medida["text"] =
str_N_measure
                print(str_N_measure + ':')
                Medida.Kv = pv.KV_194     # Fijamos la constante
del transconductor de tensión
                pv.Select_Resistor(ad2, Medida)
                pv.Get_Measure(ad2, Medida, t_discharge, path)
                Guardar_archivo(Medida, path)
                progress = 100
                if menu_FLAG == 0:
                    self.measure_progressbar["value"] = progress
                self.board_frame.update()


            time.sleep(2)
            if menu_FLAG == 0:
                function_home()
            progress = 0
```

```
            if menu_FLAG == 0:
                self.measure_progressbar["value"] = progress
            RUN_Status = 0
            if menu_FLAG == 0:
                self.board_frame.update()
                function_home()

            str_N_measure = ''



        def Check(self):
            global RUN_Status

            while RUN_Status == 1:
                self.board_frame.update()
                time.sleep(1)



        def Start(self):
            global w, t

            t = threading.Thread(target=Start_measure, args =
[self])
            w = threading.Thread(target=Check, args = [self])
            t.start()
            w.start()


        def Stop_measure(self):
            global RUN_Status

            RUN_Status = 0
            function_home()


        def Show_results():
            print('Combobox')

        def Show_files():
            global lst_files

            lst_files.clear()
```

```
            path = os.getcwd() + '/Medidas/'
            print(path)
            with os.scandir(path) as ficheros:
                lst_files = [fichero.name for fichero  in
ficheros if fichero.is_dir()]
            self.board_frame.update()


        def Show_measures():
            global lst_measures, lst_files, file_selected
            print('Measures actualizada')
            self.board_frame.update()
            file_selected = self.measure_combo_file.get()
            lst_measures.clear()
            path = os.getcwd() + '/Medidas/' + file_selected
            print(path)
            with os.scandir(path) as ficheros:
                lst_measures = [fichero.name for fichero  in
ficheros if fichero.is_dir()]
            print(lst_measures)
            self.board_frame.update()


        def Save_settings(self):
            global N_measures, t_discharge, t_measure

            function_settings()
            self.board_frame.update()


            t_discharge = float(str_t_discharge.get())
            N_measures = int(str_N_measures.get())
            t_measure = float(str_t_measure.get())
            N_points = int(str_N_points.get())

            if N_points > 16000:
                N_points = 16000
                str_N_points.set('16000')

            print('T. descarga del condensador :
'+str(t_discharge))
            print('N de medidas : '+str(N_measures))
            print('T. entre medidas : '+str(t_measure))
            print('N de puntos : '+str(N_points))
```

```
        def function_home():
            global progress, menu_FLAG, N_measures, t_measure,
N_points

            menu_FLAG = 0
            print("INICIO")
            # Borramos el board_frame
            self.board_frame.pack_forget()
            self.board_frame.destroy()

            # Creamos el board_frame
            self.board_frame = tk.Frame(self.wind, width=800,
height=384, bg = "white")
            self.board_frame.grid_propagate(False)
            self.board_frame.grid(row=0, sticky="nsew")

            if(RUN_Status == 0):
                # Creamos el board_frame
                self.title_label = tk.Label(self.board_frame,
text = 'CARACTERIZACIÓN Y MODELADO DE PANELES SOLARES', font =
("Arial", 16, "bold"), bg = "white", fg = "#5e5e5e")
                self.title_label.place(relx=0.1, rely=0.1,
relwidth=0.8, relheight=0.1)

                self.name_label = tk.Label(self.board_frame,
text = 'Nombre de la medida: ', bg = "white")
                self.name_label.place(relx=0.1, rely=0.35,
relwidth=0.2, relheight=0.1)
                self.name_entry = tk.Entry(self.board_frame,
textvariable = str_name_dir, bd = 0)
                self.name_entry.place(relx=0.3, rely=0.35,
relwidth=0.6, relheight=0.1)

                self.start_icon = tk.PhotoImage(file  =
"/home/pi/Desktop/Programa_EG_Placa_nueva/version_2.0/iconos/ic_
empezar.png")
                self.start_button = tk.Button(self.board_frame,
image = self.start_icon, text = 'EMPEZAR', font = ("Arial", 12,
"bold"), compound="top", command = lambda: Start(self), bg =
"#dfe5e6", bd = 0)
                self.start_button.place(relx=0.3, rely=0.6,
relwidth=0.4, relheight=0.3)

                self.info_labelframe_2 =
tk.LabelFrame(self.board_frame, bg = 'white')
```

```
                self.info_labelframe_2.place(relx=0.75, rely =
0.6, relwidth=0.2, relheight=0.30)
                str_info_2 = 'Programa actual:\n- ' +
str(N_measures) + ' medidas\n- Intervalos de ' + str(t_measure)
+ ' s\n- ' + str(N_points) + ' puntos'
                self.info_label_2 =
tk.Label(self.info_labelframe_2, text = str_info_2, justify =
"left", bg = 'white').pack(side = "top", fill = "both", expand =
True)

                def Get_info(self):
                    self.info_labelframe =
tk.LabelFrame(self.board_frame, bg = 'white')
                    self.info_labelframe.place(relx=0.05, rely =
0.6, relwidth=0.2, relheight=0.30)

                    self.irradiance_label =
tk.Label(self.info_labelframe, text = '{}
[W/m2]'.format(int(pv.Get_Irradiance())), justify = "left", bg =
'white')
                    self.irradiance_label.grid(row = 0, column =
1, sticky="nsew", padx = 5)
                    self.irradiance_icon = tk.PhotoImage(file  =
"/home/pi/Desktop/Programa_EG_Placa_nueva/version_2.0/iconos/ic_
irradiancia.png")
                    self.irradiance_icon_label =
tk.Label(self.info_labelframe, image = self.irradiance_icon,
justify = "left", bg = 'white')
                    self.irradiance_icon_label.grid(row = 0,
column = 0, sticky="nsew", padx = 5)

                    self.temperature_label =
tk.Label(self.info_labelframe, text = '{:,.1f}
ºC'.format(pv.Get_Irradiance()), justify = "left", bg = 'white')
                    self.temperature_label.grid(row = 1, column
= 1, sticky="nsew", padx = 0)
                    self.temperature_icon = tk.PhotoImage(file
=
"/home/pi/Desktop/Programa_EG_Placa_nueva/version_2.0/iconos/ic_
temperatura.png")
                    self.temperature_icon_label =
tk.Label(self.info_labelframe, image = self.temperature_icon,
justify = "left", bg = 'white')
                    self.temperature_icon_label.grid(row = 1,
column = 0, sticky="nsew", padx = 5)
```

```
                    self.temperature_2_label =
tk.Label(self.info_labelframe, text = '{:,.1f}
ºC'.format(pv.Get_Irradiance()), justify = "left", bg = 'white')
                    self.temperature_2_label.grid(row = 2,
column = 1, sticky="nsew", padx = 0)
                    self.temperature_2_icon = tk.PhotoImage(file
=
"/home/pi/Desktop/Programa_EG_Placa_nueva/version_2.0/iconos/ic_
temperatura.png")
                    self.temperature_2_icon_label =
tk.Label(self.info_labelframe, image = self.temperature_icon,
justify = "left", bg = 'white')
                    self.temperature_2_icon_label.grid(row = 2,
column = 0, sticky="nsew", padx = 5)

                x = threading.Thread(target=Get_info, args =
[self])
                x.start()

            else :
                self.name_label_medida =
tk.Label(self.board_frame, text = str_N_measure, bg = "white",
font = (24))
                self.name_label_medida.place(relx=0.3, rely=0.1,
relwidth=0.4, relheight=0.1)

                self.name_label_action =
tk.Label(self.board_frame, textvariable = str_action, bg =
"white")
                self.name_label_action.place(relx=0.3, rely=0.4,
relwidth=0.6, relheight=0.3)

                self.stop_icon = tk.PhotoImage(file  =
"/home/pi/Desktop/Programa_EG_Placa_nueva/version_2.0/iconos/ic_
pausa.png")
                self.stop_button = tk.Button(self.board_frame,
image = self.stop_icon, text = 'PARAR', font = ("Arial", 12,
"bold"), compound = "top", command = lambda:Stop_measure(self),
bg = "#bc2a30", fg = "#000000", bd = 0)
                self.stop_button.place(relx=0.3, rely=0.6,
relwidth=0.4, relheight=0.3)



                s.configure("blue.Horizontal.TProgressbar",
background = "#bc2a30", foreground = "#f5f5f5", bd = 0)
                self.measure_progressbar =
ttk.Progressbar(self.board_frame, style =
```

```
"blue.Horizontal.TProgressbar", length = 100, mode =
'determinate')
                self.measure_progressbar.place(relx=0.1,
rely=0.35, relwidth=0.8, relheight=0.1)
                self.measure_progressbar["value"] = progress


        def function_results():
            global file_selected, menu_FLAG, Medida
            menu_FLAG = 1
            print("RESULTADOS")

            def Selected_file(event=None):
                global file_selected
                file_selected = self.measure_combo_file.get()
                print('file : '+file_selected)
                Show_measures()
                self.measure_combo_measure["values"] =
lst_measures
                self.board_frame.update()

            def Selected_measure(event=None):
                global measure_selected
                measure_selected =
self.measure_combo_measure.get()
                print('measure : '+measure_selected)
                self.board_frame.update()

            # Borramos el board_frame
            self.board_frame.pack_forget()
            self.board_frame.destroy()

            # Creamos el board_frame
            self.board_frame = tk.Frame(self.wind, width=800,
height=384, bg = "white")
            self.board_frame.grid_propagate(False)
            self.board_frame.grid(row=0, sticky="nsew")

            self.board_frame.L = tk.Frame(self.board_frame,
width=210, height=384, bg = "white")
            self.board_frame.L.grid(row = 0, column = 0)
            self.board_frame.R = tk.Frame(self.board_frame,
width=590, height=384, bg = "white")
            self.board_frame.R.grid(row = 0, column = 1)
```

```
            # Barra lateral
            self.measure_list_1 = tk.Label(self.board_frame.L,
text = 'Selecciona la medida', height = 1, width = 200, bg =
"white")
            self.measure_list_1.place(relx=0.025, rely=0.04,
relwidth=0.95, relheight=0.07)
            Show_files()
            self.measure_combo_file =
tk.ttk.Combobox(self.board_frame.L, values = lst_files,
postcommand = Show_results)
            self.measure_combo_file.place(relx=0, rely=0.11,
relwidth=0.95, relheight=0.07)

self.measure_combo_file.bind("<<ComboboxSelected>>",Selected_fil
e)


            self.measure_combo_measure =
tk.ttk.Combobox(self.board_frame.L, values = lst_measures,
postcommand = Show_results)
            self.measure_combo_measure.place(relx=0, rely=0.18,
relwidth=0.95, relheight=0.07)

self.measure_combo_measure.bind("<<ComboboxSelected>>",Selected_
measure)


            self.parameters_button =
tk.Button(self.board_frame.L, text = 'Parámetros', font =
("Arial",11, "bold"), height = 96, width = 200, command =
Print_Parameters, bg = "white", bd = 0)
            self.parameters_button.place(relx=0, rely=0.25,
relwidth=0.95, relheight=0.25)


            self.Curves_button = tk.Button(self.board_frame.L,
text = 'Curvas I-V / P-V', font = ("Arial",11, "bold"), height =
96, width = 200, command = Print_Curves, bg = "white", bd = 0)
            self.Curves_button.place(relx=0, rely=0.5,
relwidth=0.95, relheight=0.25)


            self.Export_button = tk.Button(self.board_frame.L,
text = 'Exportar', font = ("Arial",11, "bold"), height = 96,
width = 200, command = Export_function, bg = "white", bd = 0)
            self.Export_button.place(relx=0, rely=0.75,
relwidth=0.95, relheight=0.25)


            # Cuadro de gestión
            tk.Label(self.board_frame.R, text =
'Previsualización de la última medida', font = ("Arial", 12,
```

```
"bold"), bg = "white", fg = "#5e5e5e").place(relx=0.05,
rely=0.05, relwidth=0.9, relheight=0.1)

            try:
                self.results_frame =
tk.LabelFrame(self.board_frame.R, bg = "white", bd = 2)
                self.results_frame.place(relx=0.65, rely=0.2,
relwidth=0.25, relheight=0.5)

                str_results_label_1 = 'Hora = ' + Medida.hour
                self.results_label_1 =
tk.Label(self.results_frame, text = str_results_label_1, justify
= "left", bg = "white")
                self.results_label_1.pack(side = tk.TOP, fill =
"both", expand = True)

                str_results_label_2 = 'G = ' + str(Medida.G) +
'[W/m2]'
                self.results_label_2 =
tk.Label(self.results_frame, text = str_results_label_2, justify
= "left", bg = "white")
                self.results_label_2.pack(side = tk.TOP, fill =
"both", expand = True)

                str_results_label_3 = 'Voc = ' + str(Medida.Voc)
+ '[V]'
                self.results_label_3 =
tk.Label(self.results_frame, text = str_results_label_3, justify
= "left", bg = "white")
                self.results_label_3.pack(side = tk.TOP, fill =
"both", expand = True)

                str_results_label_4 = 'Isc = ' + str(Medida.Isc)
+ '[A]'
                self.results_label_4 =
tk.Label(self.results_frame, text = str_results_label_4, justify
= "left", bg = "white")
                self.results_label_4.pack(side = tk.TOP, fill =
"both", expand = True)

                str_results_label_5 = 'Pmax = ' +
str(Medida.Pmax) + '[W]'
                self.results_label_5 =
tk.Label(self.results_frame, text = str_results_label_5, justify
= "left", bg = "white")
```

```
                self.results_label_5.pack(side = tk.TOP, fill =
"both", expand = True)



                self.last_measure = tk.PhotoImage(file  =
"/home/pi/Desktop/Programa_EG_Placa_nueva/version_2.0/iconos/Las
t_measure.png")
                self.last_measure_label =
tk.Label(self.board_frame.R, image = self.last_measure, bg =
'white')
                self.last_measure_label.place(relx=0.05,
rely=0.15, relwidth=0.6, relheight=0.55)
            except:
                tk.Label(self.board_frame.R, text = 'No se ha
realizado \nninguna medida todavía', font = ("Arial", 12), bg =
"white").place(relx=0.05, rely=0.35, relwidth=0.55,
relheight=0.1)



            self.instructions_results_frame =
tk.LabelFrame(self.board_frame.R, bg = "white", bd = 2)
            self.instructions_results_frame.place(relx=0.025,
rely=0.75, relwidth=0.95, relheight=0.15)

            str_instructions = 'NOTA: Selecciona primero la
carpeta de medidas y después la medida que deseas \nconsultar.
Actualiza la sección de resultados cada vez que cambies de
medida'
            self.instructions_results_label_2 =
tk.Label(self.instructions_results_frame, text =
str_instructions, justify = "center", bg = "white")
            self.instructions_results_label_2.pack(side =
tk.TOP, fill = "both", expand = True)



        def function_settings():
            global menu_FLAG
            menu_FLAG = 2

            print("AJUSTES")
            # Borramos el board_frame
            self.board_frame.pack_forget()
            self.board_frame.destroy()
```

```
            # Creamos el board_frame
            self.board_frame = tk.Frame(self.wind, width=800,
height=384, bg = "white")
            self.board_frame.grid_propagate(False)
            self.board_frame.grid(row=0, sticky="nsew")

            #Imprimimos todos los parámetros
            tk.Label(self.board_frame, text = '', bg =
"white").grid(row=0, sticky="nsew")
            tk.Label(self.board_frame, text = 'AJUSTES DEL
DISPOSITIVO', font = ("Arial", 12, "bold"), bg = "white", fg =
"#5e5e5e").grid(row=1, sticky="nsew")
            tk.Label(self.board_frame, text = '', bg =
"white").grid(row=2, sticky="nsew")

            # Creamos un Frame Container para los ajustes del
condensador
            settings_frame_1 = tk.LabelFrame(self.board_frame,
text = 'Ajustes del condensador', width=800, height=60, bg =
"white", bd = 2)
            settings_frame_1.grid_propagate(False)
            settings_frame_1.grid(row=3, sticky="nsew")

            tk.Label(settings_frame_1, text = 'Tiempo de
descarga del condensador [s]: ', bg = "white").grid(row = 2,
column = 1, padx = 20, pady = 5)
            self.t_discharge_entry = tk.Entry(settings_frame_1,
textvariable = str_t_discharge, bd = 0).grid(row = 2, column =
2, padx = 20, pady = 5)

            # Creamos un Frame Container para los ajustes de la
medida
            settings_frame_2 = tk.LabelFrame(self.board_frame,
text = 'Ajustes de la medida', width=800, height=120, bg =
"white", bd = 2)
            settings_frame_2.grid_propagate(False)
            settings_frame_2.grid(row=4, sticky="nsew")

            tk.Label(settings_frame_2, text = 'Nº de medidas: ',
bg = "white").grid(row = 3, column = 1, padx = 20, pady = 5)
            self.n_measures_entry = tk.Entry(settings_frame_2,
textvariable = str_N_measures, bd = 0).grid(row = 3, column = 2,
padx = 20, pady = 5)
```

```
            tk.Label(settings_frame_2, text = 'Tiempo entre
medidas [s]: ', bg = "white").grid(row = 4, column = 1, padx =
20, pady = 5)
            self.t_measures_entry = tk.Entry(settings_frame_2,
textvariable = str_t_measure, bd = 0)
            self.t_measures_entry.grid(row = 4, column = 2, padx
= 20, pady = 5)

            tk.Label(settings_frame_2, text = 'Nº de puntos: ',
bg = "white").grid(row = 5, column = 1, padx = 20, pady = 5)
            self.n_points_entry = tk.Entry(settings_frame_2,
textvariable = str_N_points, bd = 0)
            self.n_points_entry.grid(row = 5, column = 2, padx =
20, pady = 5)
            tk.Label(settings_frame_2, text = '(máx. 16.000
puntos) ', bg = "white").grid(row = 5, column = 3, padx = 20,
pady = 5)

            tk.Label(self.board_frame, text = '', bg =
"white").grid(row=6, sticky="nsew")

            self.save_button = tk.Button(self.board_frame, text
= 'Guardar cambios', font = ("Arial", 12, "bold"), height = 2,
width = 40, compound="c", command = lambda :
Save_settings(self), bg = "#dfe5e6", bd = 0, fg =
"black").grid(row=7, sticky="ns")

            tk.Label(self.board_frame, text = '', bg =
"white").grid(row=8, sticky="nsew")


        def function_close():
            print("SALIR")

            # Borramos el board_frame
            self.board_frame.pack_forget()
            self.board_frame.destroy()

            # Creamos el board_frame
            self.board_frame = tk.Frame(self.wind, width=800,
height=384, bg = "white")
            self.board_frame.grid_propagate(False)
            self.board_frame.grid(row=0, sticky="nsew")
            self.logo = tk.PhotoImage(file  =
"/home/pi/Desktop/Programa_EG_Placa_nueva/version_2.0/iconos/Log
o_programa.png")
```

```
            self.logo_label = tk.Label(self.board_frame, image =
self.logo, bg = 'white')
            self.logo_label.place(relx=0, rely=0.05, relwidth=1,
relheight=0.9)

            resultado = tk.messagebox.askquestion("Salir","¿Está
seguro que desea salir del programa?")

            if resultado == "yes":
                # Cerramos Analog Discovery y salimos
                try:
                    os.remove(os.getcwd() +
'/iconos/Last_measure.png')
                except:
                    print('Error al borrar la última medida')
                dwf.FDwfDeviceCloseAll()
                exit()



        #-- Buttons --
        self.home_icon = tk.PhotoImage(file  =
"/home/pi/Desktop/Programa_EG_Placa_nueva/version_2.0/iconos/ic_
INICIO.png")
        self.home_button = tk.Button(control_frame, text =
'INICIO', font = ("Arial", 10, "bold"), image = self.home_icon,
height = 80, width = 19, compound="top", command =
function_home, bg = "#ffffff", bd = 0).pack(side = "left", fill
= "both", expand = True)

        self.results_icon = tk.PhotoImage(file  =
"/home/pi/Desktop/Programa_EG_Placa_nueva/version_2.0/iconos/ic_
RESULTADOS.png")
        self.results_button = tk.Button(control_frame, text =
'RESULTADOS', font = ("Arial", 10, "bold"), image =
self.results_icon, height = 80, width = 19, compound="top",
command = function_results, bg = "#ffffff", bd = 0).pack(side =
"left", fill = "both", expand = True)

        self.settings_icon = tk.PhotoImage(file  =
"/home/pi/Desktop/Programa_EG_Placa_nueva/version_2.0/iconos/ic_
AJUSTES.png")
        self.settings_button = tk.Button(control_frame, text =
'AJUSTES', font = ("Arial", 10, "bold"), image =
self.settings_icon, height = 80, width = 19, compound="top",
```

```
command = function_settings, bg = "#ffffff", bd = 0).pack(side =
"left", fill = "both", expand = True)

        self.terminal_icon = tk.PhotoImage(file  =
"/home/pi/Desktop/Programa_EG_Placa_nueva/version_2.0/iconos/ic_
SALIR.png")
        self.terminal_button = tk.Button(control_frame, text =
'SALIR', font = ("Arial", 10, "bold"), image =
self.terminal_icon, height = 80, width = 19, compound="top",
command = function_close, bg = "#ffffff", bd = 0).pack(side =
"left", fill = "both", expand = True)

        self.logo = tk.PhotoImage(file  =
"/home/pi/Desktop/Programa_EG_Placa_nueva/version_2.0/iconos/Log
o_programa.png")
        self.logo_label = tk.Label(self.board_frame, image =
self.logo, bg = 'white')
        self.logo_label.place(relx=0, rely=0.05, relwidth=1,
relheight=0.9)


if __name__ == "__main__":
    window = tk.Tk()
    window.iconphoto(True, tk.PhotoImage(file  =
"/home/pi/Desktop/Programa_EG_Placa_nueva/version_2.0/iconos/Log
o_ventana.png"))
    App = Aplicacion(window)
    window.mainloop()
```