Trabajo Fin de Grado
Ingeniería Electrónica, Robótica y Mecatrónica

Gestión y monitorización de instalaciones eléctricas trifásicas mediante Arduino y Raspberry Pi

Autor: Raúl Castilla Arquillo
Tutor: Juan Carlos de Pino López

Dpto. de Ingeniería Eléctrica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Proyecto Fin de Grado
Ingeniería Electrónica, Robótica y Mecatrónica

Gestión y monitorización de instalaciones eléctricas trifásicas mediante Arduino y Raspberry Pi

Autor:
Raúl Castilla Arquillo

Tutor:
Juan Carlos del Pino López
Profesor titular
Juan Carlos del Pino López

Dpto. de Ingeniería Eléctrica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2019

Trabajo Fin de Grado: Gestión y monitorización de instalaciones eléctricas trifásicas mediante Arduino y Raspberry Pi

Autor: Raúl Castilla Arquillo

Tutor: Juan Carlos del Pino López

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2019

El Secretario del Tribunal

Agradecimientos

A mi familia, por apoyarme en todo momento en cada decisión que he tomado, sabiendo que no todo iba a ser fácil. A mis amigos, por sacarme sonrisas en las interminables jornadas de estudio y bueno, por sacarme, aunque sea a tomar algo, y no dejar que me quede demasiado tiempo encerrado en casa estudiando. Finalmente, a mis profesores, por demostrarme que la Ingeniería no consiste solamente en un conjunto de fórmulas.

Raúl Castilla Arquillo

Sevilla, 2019

Índice

Notación

IoT	Internet de las cosas, del inglés: Internet of Things
HW	Hardware
SW	Software
ADC	Conversor analógico digital, del inglés: Analogue Digital Converter
CPU	Unidad central de procesamiento, del inglés: Central Processing Unit
FFT	Transformada rápida de Fourier, del inglés: Fast Fourier Transform
SQL	Lenguaje de consulta estructurada, del inglés: Structured Query Language
IDE	Entorno de desarrollo integrado, del inglés: Integrated Development Environment
DFT	Transformada discreta de Fourier, del inglés: Discrete Fourier Transform
RMS	Valor cuadrático medio, del inglés: Root Mean Square
IA	Inteligencia Artificial

Resumen

El presente trabajo tiene como objetivo la monitorización de sistemas trifásicos en tiempo real, permitiendo el análisis pormenorizado de las componentes de voltaje e intensidad de cada una de sus fases. En la realización del mismo se hace especial hincapié en el carácter de bajo coste de todos los componentes empleados. Al hacer uso de dispositivos ampliamente utilizados en el entorno "maker", a saber: un microcontrolador Arduino y una mini CPU Raspberry Pi, se busca también disminuir las complejidades inherentes que tienen muchos sistemas en el momento de configurar el entorno necesario, tanto para desarrollar en los mismos como para desplegar las aplicaciones desarrolladas. Se usará Software y Hardware Open Source, con las ventajas que ello conlleva.

1 INTRODUCCIÓN

No existe ninguna razón para que cualquier persona pueda tener un ordenador en casa.

- Ken Olsen, Presidente de Digital Equipment Corporation, 1977 -

Desde la creación del primer transistor en los Laboratorios Bell en 1947, la industria de la electrónica se ha visto sometida un crecimiento exponencial que ha revolucionado industria y sociedad hasta un punto inimaginable hasta ahora. Precedido por el desarrollo del primer microprocesador (el ampliamente usado Intel 4004), la creación del ordenador personal demostró que la electrónica no era algo destinado a quedarse sólo en el ámbito industrial o académico. El creciente aumento de interés en el mismo, llevó a a la industria a la conclusión de que cualquier persona debería poder utilizar aquellos sistemas sin tener que poseer acceso a un costoso equipo de fabricación o a extensos conocimientos de electrónica. Este interés quedó patente en la década de los 80, cuando ordenadores como el Apple II o el IBM PC generaron millones de dólares en ventas en el hasta ahora inexplorado mercado de la electrónica de consumo [1].

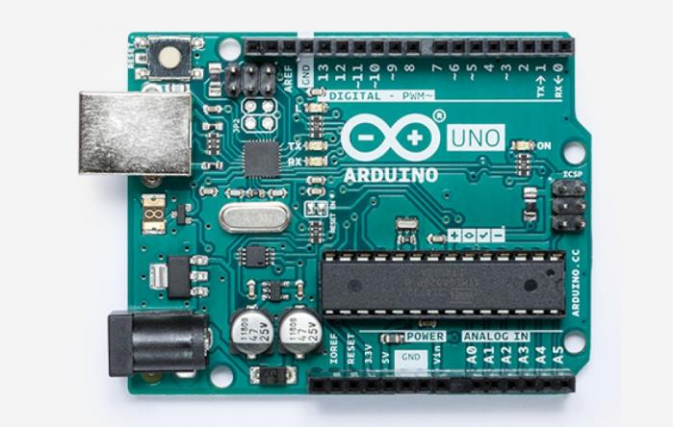

Figura 1-1 Arduino UNO Rev.3

En los últimos años, diversos avances en los procesos de producción, así como la masificación de Internet, han dado lugar a la aparición de proyectos colaborativos y libres, como la iniciativa Arduino [2], en el ámbito de los microcontroladores, y la fundación Raspberry [3], en lo que respecta a a ordenadores de bajo costo. Todos estos proyectos tienen como objetivo el acercamiento de la tecnología a las masas, facilitando que lleven a cabo sus proyectos.

Figura 1-2 Raspberry Pi 3 Model B+

Sumada a la anteriormente mencionada democratización de la tecnología, en la actualidad, conceptos como IoT [4] o Industria 4.0 [5] son cada vez más relevantes, poniendo en evidencia que ya no basta con hacer uso de las diferentes tecnologías existentes por separado, hay que integrarlas entre ellas, con especial énfasis en el intercambio fluido de información con el exterior. El presente trabajo no se plantea como algo limitado al ámbito académico o industrial, sino también como una solución fácilmente escalable para cualquier persona que tenga interés en disponer de la información permonerizada sobre el consumo eléctrico de su hogar al instante de forma sencilla y desde cualquier lugar.

1.1 Arquitectura del proyecto

El objetivo principal es el de realizar mediciones de voltaje e intensidad de un sistema trifásico en tiempo real, así como presentar estos datos de una forma intuitiva e interactiva a través de un servidor web. El sistema porpuesto se encuentra dividido en los siguientes bloques funcionales claramente diferenciados:

- Una etapa HW centrada en la recogida y adaptación de la señal analógica tanto de voltaje como de intensidad en valores aceptados por el ADC del sistema Arduino, en este caso un Arduino MKR 1010.
- El trabajo desarrollado por el microcontrolador, encargado de transformar en un valor digital cada uno de los valores análogicos recibidos en los seis canales del ADC utilizados y de transmitir esta información via puerto serie. Al estar este puerto vinculado a un puerto microusb a su salida, pueden realizarse lecturas por USB desde cualquier ordenador.
- Una etapa de recepción de los datos en la raspberry Pi a través del puerto serie y de manipulación de los mismos: guardado de los datos en hojas de cálculo y cómputo de valores de interés, a saber: valores eficaces y componentes armónicos de cada señal.
- Una fase final, especializada en la visualización de los datos obtenidos y los cálculos asociados. Consta de un servidor web con gráficas interactivas actualizadas en tiempo real y diversas herramientas para la visualización de FFTs y las hojas de cálculo.

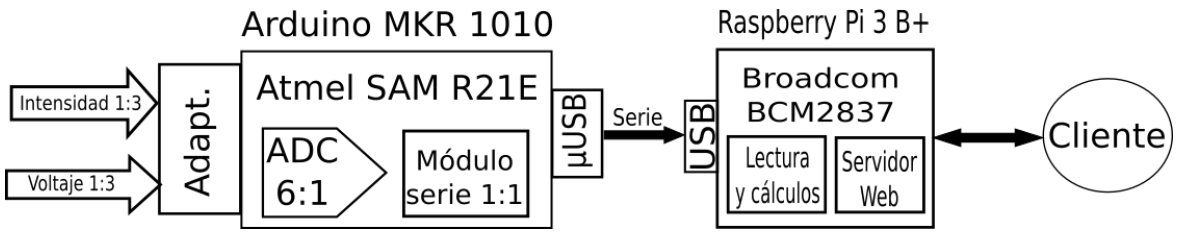

Figura 1-3 Esquema funcional del sistema

1.1.1 Microprocesador Arduino MKR 1010

Se trata de una placa de desarrollo que consta de dos módulos HW [6] principales:

- Un procesador ARM SAMD21 Cortex – M0+ 32 bits.
- Un módulo ESP32 que aporta conectividad tanto Bluetooth como Wifi 2.4GHz.

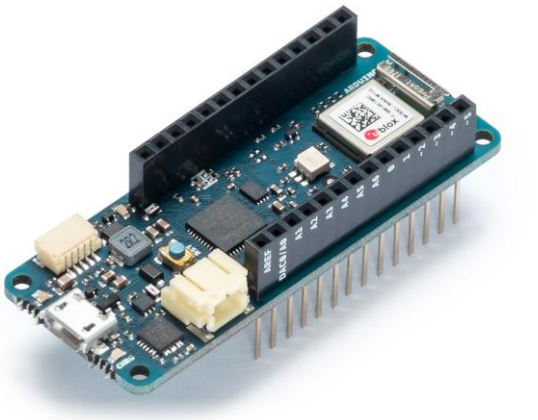

Figura 1-4 Arduino MKR1010

Nos encontramos ante un módulo por parte de la Fundación Arduino que aúna dos de las arquitecturas más relevantes en el mercado electrónico de los últimos años:

- La primera, un microprocesador embebido [7] en gran cantidad de productos de consumo, sobre todo gracias a su baja potencia de consumo. Recientemente, se está viendo desplazada por el ARM Cortex M4, aunque todavía le queda recorrido debido a su bajo precio.
- La segunda, una sucesora reciente del ESP8266 de Espressif, ampliamente utilizada en el entorno maker por su versatilidad y conectividad. El ESP32 [8] consta de mayor potencia en CPU y le añade conectividad Bluetooth a la conectividad Wifi que ya integraba su predecesora.

En lo que respecta a las características técnicas que más nos interesan en este proyecto, encontramos:

Alimentación USB	5V
Voltaje operativo	3.3V
Pins de E/S digitales	8
UART	1
Pines Analógicos	7 (ADC 8/10/12 bit)
Memoria Flash	256 KB
SRAM	32 KB

Tabla 1-1 Características técnicas de Arduino MKR1010

Nota: El máximo voltaje que cada GPIO es capaz de tolerar es de 3.3V.

1.1.2 CPU Raspberry Pi 3B+

Lo que se plantea como la última revisión de la Raspberry Pi 3 antes de la salida de Raspberry Pi 4 a lo largo de 2019, con algo más de velocidad en su procesador que su anterior revisión:

Procesador	Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit
Frecuencia de reloj	1.4 GHz
Memoria	1GB LPDDR2 SDRAM
Conectividad	Wifi 2.4/5 GHz, Bluetooth v4.2
Puertos	GPIO 40 pines, 4xUSB 2.0, MicroUSB de alimentación
Memoria Flash	256 KB
SRAM	32 KB

Tabla 1-2 Características técnicas de Raspberry Pi 3B+

1.2 Justificación de la arquitectura hardware empleada

1.2.1 Microprocesador-CPU

La elección del binomio microprocesador-CPU se ha visto motivada por diversas circunstancias:

- La necesidad de una alta velocidad de muestreo para poder realizar la toma de medidas de las 6 componentes de interés sin perder información durante el proceso: ya sea debido al cambio de canal en la lectura o a ármonicos de alta frecuencia en la señal que pasarían desapercibidos a velocidades menores. Este requisito ha promovido la elección de un microcontrolador para la toma de muestras, ya que posibilita un ciclo de ejecución de las tareas de forma repetible y precedible (tiempo real) debido a su arquitectura.
- El almacenamiento de la información recibida en forma de hojas de cálculo y su almacenamiento en memoria realentizarían en exceso la toma de datos por parte del microcontrolador, por lo que se hace evidente la necesidad de una CPU con una capacidad de computación algo mayor.
- Los cálculos necesarios sobre los datos obtenidos, especialmente para calcular FFTs, son demasiados exhaustivos computacionalmente para ser procesados en un microcontrolador.
- Puesto que se quieren presentar los datos desde un servidor web de forma dinámica, es necesario una CPU que corra un sistema operativo para la ejecución de todas las tareas pertinentes en segundo plano: despliegue del servidor, acceso a datos, presentación de páginas dinámicas al usuario, gestión de protocolos asociados a internet, etc.

1.2.2 Comunicación serie

En fases tempranas del proyecto, se planteó el envío de datos desde el microprocesador hasta la Raspberry haciendo uso de Wifi (sin necesidad de estar físicamente en el mismo lugar) mediante el envío de paquetes con protocolos de transporte como UDP (por su ligereza de datos) o TCP (por su seguridad en el envío de tramas). No obstante, las librerías encargadas de comunicar el procesador ARM con la antena wifi (ESP32) de la placa de desarrollo tardan un tiempo considerable en empaquetar los datos para su envío, provocado un pronunciado cuello de botella en el sistema.

Debido a lo anteriormente descrito, se optó por el uso del puerto serie del procesador, conectado al microusb de alimentación de la placa de desarrollo y facilitando así su conexión con otros dispositivos.

1.2.3 Hojas de cálculo vs bases de datos

Se llegó a la conclusión de que era necesario el uso de hojas de cálculo en lugar de bases de datos como SQL porque las primeras se almacenan en memoria como archivos en texto plano, mientras que una base de datos SQL necesita de cierta comunicación con la misma tanto como para insertar datos como para acceder a ellos, siendo esta más lenta en un contexto de repetidas lecturas y escrituras, en este caso, muestreos de altas frecuencias de un ADC más diversas lecturas por parte de un servidor web.

1.2.4 ADC integrado del Arduino MKR: ARM SAMD21

El procesador ARM SAM21D, que integra una arquitectura ARM Cortex M0+, puede ser encontrado en placas de desarrollo como el Arduino Zero, el MKR1000 y el MKR1010, este último nuestro caso. A diferencia de Arduinos más conocidos como el Arduino Uno, el MEGA, el Mini, etc, que poseen todos un procesador AVR (véase el ATMEGA328P [9]), la familia de ARM posee un ADC de mayor rapidez (ver pag. 2 del datasheet para más información sobre sus características [10]). Tiene las siguientes características:

- Una resolución de 12 bits.
- 8 canales de entrada.
- Un voltaje de referencia de hasta 3.3V.
- Una capacidad de muestreo de hasta 350 Ksps.
- Una ganancia desde 1/2x hasta 16x.

En lo que respecta a la velocidad de muestreo, el valor de 350 Ksps es su máximo teórico, en las siguientes secciones veremos que disminuye sensiblemente debido a diversas consideraciones.

2 HARDWARE

En lo que respecta al HW, este capítulo se dividirá en dos partes diferenciadas: la primera se encuentra reservada a los elementos utilizados para extraer las señales necesarias y la segunda hará referencia a la adaptación de las señales para poder ser leídas por el ADC. En este capítulo, también se podría hacer un análisis del ADC a nivel estructural, pero puesto que vamos a enfrentarnos al mismo en el capítulo 4 de programación, lo dejaremos para el mismo.

2.1. Medidores de la señal

2.1.1 Medidor de voltaje

Para la medición de voltaje, haremos uso de un trasformador Kreco [11] de 230V AC a 9V AC:

Figura 2-1 Trasformador de voltaje

2.1.2 Medidor de corriente

Para la medición de corriente se hará uso de una pinza amperimétrica modelo SCT013 [12]:

Figura 2-2 Pinza amperimétrica SCT013-000

El formato presentado tiene la ventaja de que es un método no invasivo para la medición de corriente debido a que usa su transformador interno para transformar el campo magnético creado por el conductor que lo atraviesa en una corriente eléctrica proporcional a su salida. Posee las siguientes características técnicas según datasheet:

Corriente de entrada	0-50A
Corriente de salida	0-50 mA
No linealidad	+-3 %
Material del núcleo	Ferrita

Tabla 2-1 Características técnicas SCT013-000

2.2. Adaptación de la señal

Esta adaptación se encuentra fuertemente condicionada por las características del ADC:

- Queremos realizar la medida de 6 señales de voltaje e intensidad alternas, es decir, señales que pueden tomar valores tanto positivos como negativos. Debido a ello, tenemos dos opciones: hacer uso de las entradas diferenciales del ADC o de entradas únicas, pero introduciendo un voltaje DC de bias u offset de un valor equivalente a la mitad del valor de pico de la señal, para que nunca alcance un valor negativo. Puesto a que sólo se dispone de 8 canales, nos vemos a obligados a escoger la última opción, ya que no habría entradas analógicas suficientes para el uso del modo diferencial. Pese a que, a primera vista, esto pueda parecer que supone una pérdida de la mitad del rango máximo de la señal, esto no es así, puesto que muchos ADC te obligan a limitar la entrada a Vref/2 y -Vref/2 para medidas diferenciales en comparación con un rango de entre 0 y Vref para una única entrada, con lo que el resultado final sigue siendo el mismo, siendo Vref el voltaje de referencia de entrada.
- Un ADC sólo esta capacitado para tomar un voltaje como su señal de entrada, por lo que es necesario modelar una resistencia de carga que transforme la intensidad creada por la pinza amperimétrica a un voltaje que pueda servir de entrada al ADC.

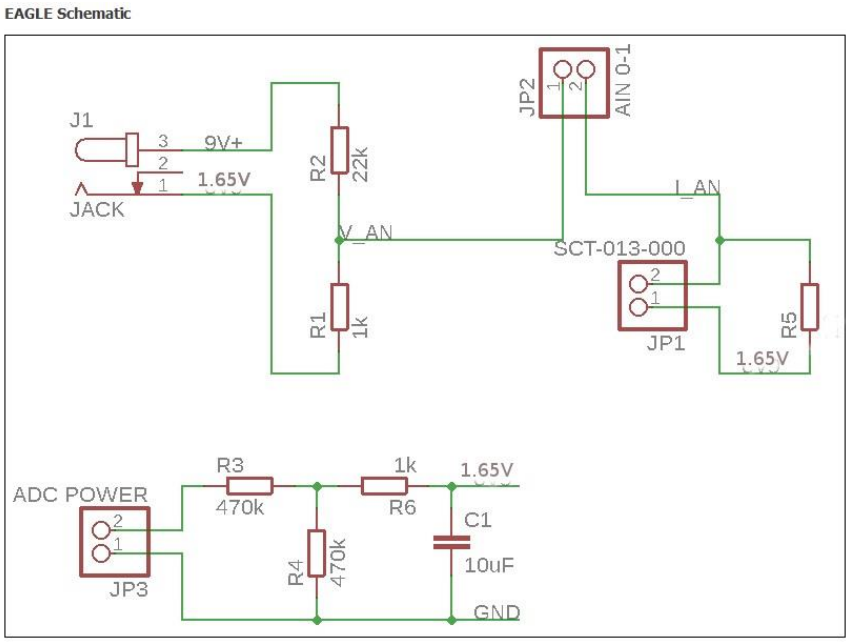

Figura 2-3 Adaptación de la señal

La parte superior del esquemático representa la adición de un bias de 1.65V, esto es, de la mitad del voltaje de referencia de 3.3V del ADC a las componentes de voltaje (parte izquierda) e intensidad (parte derecha). En la zona inferior del esquemático nos encontramos con un simple divisor resistivo utilizado para obtener el voltaje de bias y con un sencillo filtro RC a la salida para eliminar el ruido que pueda tener el voltaje de entrada. La resistencia de carga R5 es un potenciómetro que variará dependiendo del rango de la señal de intensidad a medir.

La adaptación de señal presentada dista mucho de ser perfecta y no es más que una primera aproximación para comprobar la efectividad del modelo presentado para la obtención de las medidas. Ciertas mejoras podrían introducirse en aspectos como la obtención del voltaje de bias (un integrado de regulación, referencia Zener, etc) o modelado de la resistencia de carga (resistencia controlada electrónicamente).

3 CONFIGURACIÓN DEL ENTORNO

A continuación, se presentarán los diversos entornos de desarrollo a utilizar, así como los lenguajes de programación y librerías implicadas en los diferentes módulos del proyecto.

3.1 Módulo Arduino

3.1.1 Entorno de desarrollo

Se usará la última versión de Arduino IDE [13] (v1.8.8) como entorno de desarrollo para el módulo Arduino que puede encontrarse tanto para Windows como para Mac y Linux.

Figura 3-1 Ventana de descarga de Arduino IDE

Una de las ventajas que tiene el uso de la citada IDE es que, a la hora de desarrollar código para la plataforma, se encarga de gestionar de forma automática la toolchain necesaria para compilar el código correspondiente a la arquitectura a utilizar, en este caso ARM, así como el flasheo del microcontrolador con el binario generado. Además, nos permite la descarga y uso de diferentes librerías de terceros de forma intuitiva.

No obstante, este entorno no viene con las herramientas necesarias para desarrollar código para el MKR1010, siendo necesario instalarlas. Para ello, una vez abierto el programa, accedemos a la sección Herramientas → Placa → Gestor de tarjetas y hacemos click sobre "Arduino SAMD Boards by Arduino" y versión 1.6.19 en la ventana desplegada.

Figura 3-2 Ventana del gestor de tarjetas

Para comprobar si todo se ha instalado correctamente, enchufamos nuestra placa al ordenador mediante USB y la seleccionamos en la ventana Herramientas → Arduino MKR Wifi 1010. Una vez hecho lo anterior, hacemos click en Herramientas → Puerto, escogiendo el puerto donde tenemos la placa conectada y pulsando el botón de Subir para flashear el código, un mensaje de éxito debería a aparecer en la parte inferior del programa.

3.1.2 Lenguaje y librería

Se usará C como lenguaje predeterminado para desarrollar, ya que es el lenguaje, junto a C++, en el que se encuentran gran parte de las librerías y el utilizado de forma predeterminada para los módulos Arduino. En cuanto a las librerías que se utilizarán, crearemos una propia llamada readfast.h, que se usará en la configuración y lectura del ADC y las correspondientes al HW del micro (Arduino.h) para el acceso a los diferentes pines y periféricos. El planteamiento utilizado en [14] se ha usado como inspiración para el mismo.

3.2 Raspberry Pi

3.2.1 Entorno

Como entorno en la Raspberry Pi se usará un sistema operativo basado en Linux, en concreto, la última versión de Raspbian Stretch Lite [15] en su release 2018-11-13 puesto que no necesitamos un entorno de escritorio en el mismo ya que supondría un gasto innecesario de recursos.

Figura 3-3 Ventana de descarga de Raspbian

Haremos uso de pyhton 3 como único lenguaje de desarrollo, por lo que es necesario instalar el intérprete y las librerías base necesarias en el sistema ejecutando los siguientes comandos desde la terminal:

```
apt-get install python3
apt-get install pip3
pip3 install virtualenv
virtualenv -p python3 venv
```

Nota: Virtualvenv nos permitirá iniciar un entorno de python3 totalmente aislado del entorno Python propio del sistema, con lo que nos evitaremos conflictos a la hora de instalar las librerías necesarías para los scripts desarrollados.

3.2.2 Librerías para el tratamiento de datos

Para la recogida de datos por puerto serie y su tratamiento, será necesario instalar las siguientes librerías desde la terminal:

```
. venv/bin/activate
Pip install pyserial           #Módulo serie
Pip install matplotlib         #Biblioteca de generación de gráficos
Pip install scipy              #Biblioteca de algoritmos matemáticos
Pip install pandas_datareader  #Biblioteca de manipulación de datos
```

Estas librerías se utilizarán por las siguientes razones:

- Pyserial es necesaria para leer los datos enviados por puerto serie a la Raspberry desde el Arduino.
- Matplotly es un framework que nos permitirá la creación de diversas gráficas, necesarias para una correcta visualización de los datos.
- Scipy contiene muchas funciones matemáticas útiles para cálculos complejos: las que usaremos en este proyecto serán las correspondientes al cálculo de FFTs, así como de enventanados y filtros.
- Pandas_datareader es una utilidad necesaria a la hora de manipular hojas de cálculos, tanto para la lectura como para la escritura de datasets.

3.2.3 Librerías para el servidor web

Es necesario instalar lo siguiente, todas ellas librerías relacionadas con Dash:

```
. venv/bin/activate
Pip install dash==0.35.1                    #Núcleo del dash
Pip install dash-html-components==0.13.4    #Componentes html
Pip install dash-core-components==0.42.1    #Componentes del dash
Pip install dash-table==3.1.11              #Tablas
```

Para la creación de un servidor web haremos uso de Dash de Ploty [16] que se trata de un framework útil en la elaboración de webs orientadas al análisis y visulización de datos. En concreto, nos interesan las funciones que posee para poder ver gráficas en tiempo real y manipular tablas.

Figura 3-4 Ejemplo de un dash orientado a la visualización de datos sobre la producción de petróleo

4 PROGRAMACIÓN DE RECOGIDA DE DATOS

Primero, nos centraremos en el código del Arduino encargado de recoger los datos manipulando el ADC y de enviarlos a través del puerto serie. Más tarde, profundizaremos en el fragmento de código ubicado en la Raspberry que recoge estos datos y realiza los cálculos asociados.

4.1 Programación en Arduino

4.1.1 Configuración del ADC

Para conseguir el máximo rendimiento por parte del ADC es necesario un correcto estudio del datasheet del procesador Atmel SAM21D [10], concretamente el apartado correspondiente al ADC, a partir de la página 844. Internamente, hay varios parámetros de su configuración interna que son necesarios manipular; una configuración completa puede verse en el Apéndice A1.2. En lo correspondiente a la función setupADC():

Primero se configura el voltaje de referencia del ADC a 3.3V, que es el de alimentación de la placa de desarrollo y, por tanto, el que se considera por defecto:

```
12. analogReference(AR_DEFAULT);
```

Un comando que se repite constantemente en el código es el referente a la espera del bit de sincronización del registro SYNCBUSY. El datasheet recomienda encarecidamente comprobar el estado de este bit a la espera de que las escrituras en los registros se realicen correctamente:

```
while (ADC->STATUS.bit.SYNCBUSY == 1);
```

Puesto que se van a realizar lecturas constantes de 6 entradas analógicas (desde el A1 hasta el A6), vamos a hacer uso de un modo de lectura denominado modo SCAN o escáner. Este modo, a diferencia del modo normal, no necesita que se le especifique al ADC mediante comandos que realice una conversión en un pin en concreto, sino que realiza la conversión del siguiente canal prefijado una vez se recoge el resultado de la conversión anterior en el ADC. Se ahorra mucho tiempo de configuración de registros con respecto al funcionamiento normal, puesto que la primera conversión realizada cada vez que se inicia una conversión tras una modificación de los registros devuelve un dato poco fiable, una desventaja que evadimos en este modo:

```
// 6 canales para escanear, desde A1 hasta A6 (INPUTSCAN + 1) según datasheet
36. ADC->INPUTCTRL.bit.INPUTSCAN=7;
```

El valor de registro INPUTOFFSET se incrementa una unidad tras cada conversión realizada por el ADC en modo SCAN, por ello es necesario ponerlo a 0 al inicio para empezar desde el valor más bajo:

```
ADC->INPUTCTRL.bit.INPUTOFFSET=0;
```

En cuanto a la configuración interna de los pines, disponemos de una tabla que podemos encontrar en [17] correspondiente al código fuente de nuestra placa de desarrollo, concretamente nos interesa la quinta columna; se observa que los pines de los canales del ADC no concuerdan con los de la placa:

Pin number	MKR Board pin	PIN	Peri.A	Peripheral B			
			EIC (EXTINT)	ADC (AIN)	AC (AIN)	PTC	DAC
15	A0 / DAC0	PA02	02	*00		Y00	OUT
16	A1	PB02	*02	*10		Y08	
17	A2	PB03	*03	*11		Y09	
18	A3	PA04	04	*04	00	Y02	
19	A4	PA05	05	*05	01	Y03	
20	A5	PA06	06	*06	02	Y04	
21	A6	PA07	07	*07	03	Y05	

Figura 4-1 Equivalencias internas de los pines

Los pines analógicos del A3 al A6 corresponden con los pines del 04 al 07 del periférico, mientras que los pines analógicos A1 y A2 se vinculan a los pines internos 10 y 11 respectivamente. El número de offset va de acuerdo a los pines del periférico, por lo que en un posterior fragmento de código se verá que es necesario saltarnos los valores correspondientes a los pines 8 y 9 en modo SCAN debido a que no tienen ninguna equivalencia con pines externos. El bit de MUXPOS es el que indicará el pin más bajo que se leerá, en este caso el A3, cómo se ha comprobado por la concordancia con los pines internos:

```
46. ADC->INPUTCTRL.bit.MUXPOS=g_APinDescription[A3].ulADCChannelNumber;
```

Tras configurar el reloj, iniciamos el módulo ADC y posteriormente, la conversión:

```
63. ADC->CTRLA.bit.ENABLE = 0x01;
64. while(ADC->STATUS.bit.SYNCBUSY == 1);
65.
66. // Iniciamos conversión
67. ADC->SWTRIG.bit.START = 1;
```

4.1.2 Recogida de lecturas en el ADC

Para recoger las lecturas hechas por el ADC siempre se sigue el mismo proceso especificado en la función readana(). Leemos el bit del registro de INPUTOFFSET para poder saber a qué canal corresponde la medida recogida:

```
   canal=ADC->INPUTCTRL.bit.INPUTOFFSET;
```

Como se ha explicado anteriormente, debido a la posición interna de los canales en HW, necesitamos saltarnos dos canales de los que no nos interesa la medida, esto se hace mirando si el valor del offset ha alcanzado el cuarto canal y modificándolo si procede:

```
89.   if(canal==4)
90.   {
91.       canal=canal+2;
92.       ADC->INPUTCTRL.bit.INPUTOFFSET=canal;
93.   }
94.   else if(canal==8)
95.   {
96.       canal=0;
97.       ADC->INPUTCTRL.bit.INPUTOFFSET=canal;
98.   }
```

Ahora sólo queda realizar la medida y recoger el resultado:

```
103.    ADC->SWTRIG.bit.START = 1;
104.    while (ADC->INTFLAG.bit.RESRDY == 0);
105.
106.    //Recogemos el resultado
107.    valueRead = ADC->RESULT.reg;
```

4.1.3 Bucle principal del programa

Tras configurar el puerto serie y el ADC al inicio, en el bucle principal se realiza lo siguiente:

```
29.  // Recogemosla medida
30.  uint16_t valueRead = readana();
31.  StartTime = micros();
32.
33.  // Enviamos la medida por el puerto serie
34.  sendBytes(valueRead);
35.  SecondTime = micros();
36.
37.  // El tiempo que se tarda en enviar un dato es variable,
38.  // por lo que esperamos 20 us para tener un codigo predecible
39.  while((SecondTime-StartTime) < 20)
40.      SecondTime = micros();
```

El objetivo principal del mismo es tener una frecuencia de muestreo constante, puesto que a la hora de enviar el dato por puerto serie (tres bytes: dos de medida y uno de canal medido) esta función varía entre 10us y 18us debido a las prioridades internas que posee el microcontrolador al gestionar los buffers del puerto serie.

4.2 Programación en Raspberry

4.2.1 Lectura y manipulación de los datos (data.py)

El siguiente código se puede encontrar a lo largo del apéndice A.2. El flujo principal del programa necesita la creación de dos hilos independientes: uno que se encargue de la lectura del puerto serie y vaya guardando los valores recogidos en una hoja de cálculo y otro que calcule las FFTs de los datos y genere las gráficas de las mismas.

4.2.2 Lectura de los datos

Al principio del código, nos encontramos con diversas definiciones que se usarán a lo largo del programa para poder realizar las conversiones a valores reales de los datos "brutos" medidos:

```
# Variables globales con los coeficientes correspondientes a las
16. # características del ADC
17. ADC_RESOLUTION = 1023
18. ADC_VREF = 3.3
19. ADC_BIAS = ADC_VREF/2
20. COEF_ADC = ADC_RESOLUTION/ADC_VREF
21.
22. # Coeficientes del voltaje
23. MAX_VALUE_VOLT = 9*np.sqrt(2)
24. REAL_VALUE_VOLT = 230*np.sqrt(2)
25.
26. # Coeficientes de la intensidad
27. RESISTENCIA_CARGA = 1000
28. COEF_PINZA = 2000  #100A en la realidad son 50mA (100/0.05)
```

Tras ello, creamos los directorios locales para guardar tanto las hojas de cálculo como las gráficas de las FFTs calculadas. Hecho esto, abrimos una nueva hoja de cálculo haciendo uso de la librería pandas_datareader y vamos guardando los datos obtenidos de leer en el puerto serie con la librería pyserial. Para abrir el puerto es necesario indicar la ubicación del fichero correspondiente a nuestro USB en Linux, en este caso se trata de /dev/ttyUSB0:

```
42. arduino = serial.Serial('/dev/ttyUSB0', 500000)
```

Para el cálculo de los valores reales medidos por la pinza y el transformador, se hace uso de las funciones calculatevoltADC(), calculatevoltage() y calculateintensity():

```
156. # Función usada para convertir el valor leído por el ADC a un valor
157. # de voltaje que se pueda usar
158. def calulatevoltADC(arg):
159.     volts=arg/COEF_ADC
160.     return volts

161. # Función para calcular el voltaje real que se esta midiendo del
transformador
162. def calculatevoltage(arg):
163.     volt=calulatevoltADC(arg)-ADC_BIAS
164.     div=2.4-ADC_BIAS  # 2.4 es el valor leído con 230*sqrt(2) voltios
165.     return "%.4f" % ((volt/div)*REAL_VALUE_VOLT)

166.# Función para calcular la intensidad real que se esta midiendo con la pinza
167. def calculateintensity(arg):
168.     volt=calulatevoltADC(arg)-ADC_BIAS  # Voltaje calculado
169.     value=(volt/RESISTENCIA_CARGA)*COEF_PINZA
170.     return "%.4f" % (value)
```

Se tiene en cuenta el voltaje de offset indicado anteriormente en el apartado de adaptación de la señal. En particular, para la intensidad hay que indicar el valor del potenciómetro que se usa como resistencia de carga. Para transformar el valor leído por el ADC a un voltaje real se utilizará la siguiente fórmula de uso común para un ADC:

$$voltaje\ entrada\ ADC = \frac{valor\ digital\ x\ Vref}{(bits\ de\ resolución\ ADC)^2}$$

4.2.3 Cálculo de la FFT

Para obtener los cálculos relativos a los armónicos de cada una de las señales es necesario calcular sus FFTs. Para cada unas de las componentes:

```
130.          # Aplicaremos un enventanado Kaiser con una beta igual a 100
131.          N = len(datos)
132.          datos -= np.mean(datos)    # Eliminamos el offset de DC
133.          datos_enventanados = datos * np.kaiser(N, 100)
134.          fft_enventanado = np.fft.rfft(datos_enventanados)
135.
```

Eliminando la media de los datos recogidos nos aseguramos de que están centrados en el cero, como debería ser teóricamente al tratarse de señales senoidales o casi senoidales. Antes de empezar con el cálculo de las transformadas hay que enventanarlos, escogiendo previamente una ventana apropiada. Para este caso se ha decidido hacer uso de una ventana Kaiser con una beta de 100 ajustada empíricamente; para más información sobre enventanados, acceder al apartado C.2. Una vez se han enventanado los datos, procedemos a calcular la parte real de su FFT, que es la componente que nos interesa y, a continuación, los representamos en una gráfica con la librería matplotlib. Una gráfica resultante de estos cálculos, teniendo como entrada un voltaje senoidal de una de las fases de un sistema trifásico de 50Hz muestreado a 6KHz:

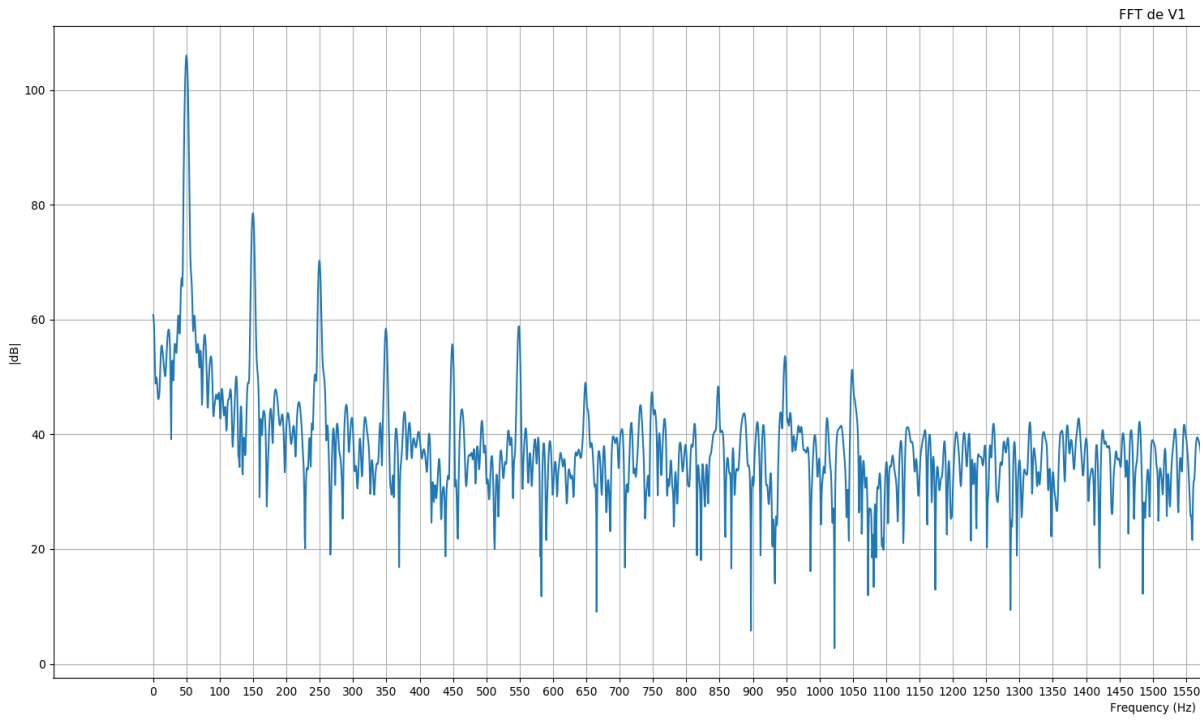

Figura 4-2 Fragmento de una gráfica de la FFT

5 PROGRAMACIÓN DEL SERVIDOR WEB

Para la programación del servidor web haremos uso de Dash, un framework de Plotly. El servidor que se montará consta de las siguientes partes:

- Un fichero app.py e index.py, que pertenecen al servidor propiamente dicho y se encargan de vincular los hipervínculos web a los diferentes scripts e interfaces.
- La página principal, que se encuentra en el script main_page.py y que nos redirige a cada una de las diferentes páginas.
- Las vistas de las gráficas de voltaje e intensidad en tiempo "real", así como los diversos datos de potencia, voltaje eficaz, etc. calculados a partir de los mismos. Estás páginas son servidas por los ficheros phase1, phase2 y phase3_live.py.
- Fft_viewer.py, que es la página que sirve como utilidad para tener acceso a las diferentes gráficas de las FFTs generadas.
- Table_viewer.py, para ver los datos guardados en las hojas de cálculo y las gráficas temporales de todas las componentes.

5.1 Visor en tiempo real

En esta página se observan las gráficas de voltaje e intensidad en la misma escala de tiempo, en este caso, según su número de muestra. A su vez, pueden verse diversos datos de interés en la parte superior de la misma.

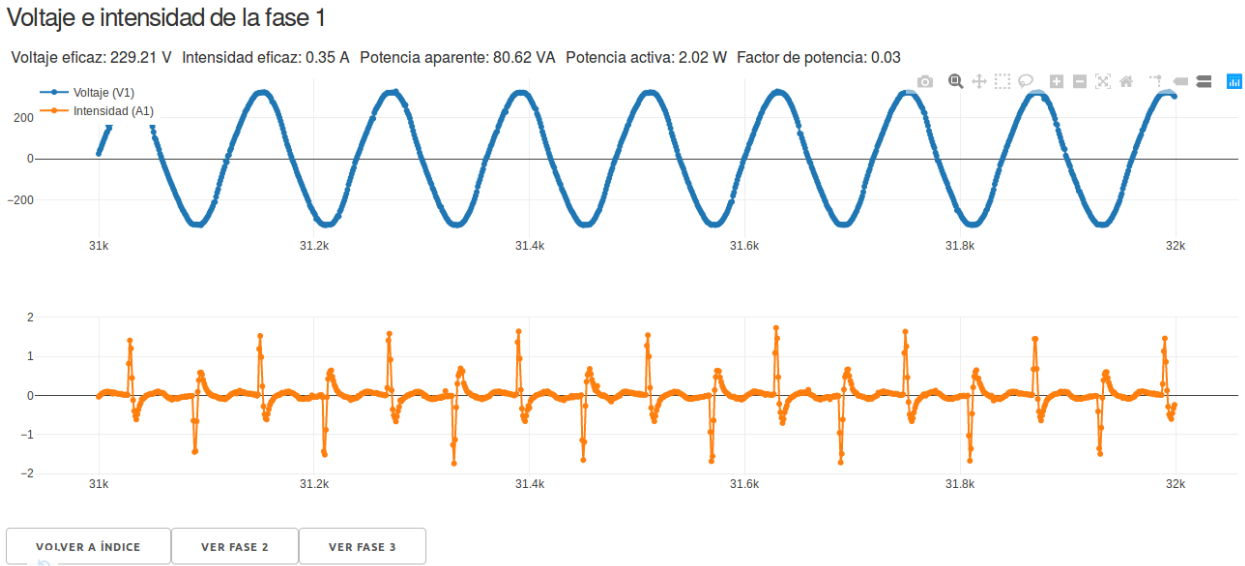

Figura 5-1 Interfaz correspondiente al visor en tiempo real

Puesto que la cantidad de datos recibidos por segundo es elevada, se ha decidido enseñar las muestras de mil en mil correspondientes a la hoja de cálculo que se está grabando en cada momento. Para el cálculo de los diversos datos, se computa el periodo de las muestras teniendo en cuenta el número de muestra en el que la señal de voltaje e intensidad pasan por el valor cero:

```
78.     # Recogemos los datos del fichero .csv correspondientes a V1 y A1
79.     for i in range(1000):  # Mostramos los valores de mil en mil muestras
80.         muestra=i+1000*n
81.         data['Muestra'].append(i+1000*n)

82.         # Vamos recogiendo datos de la columna V1 y los metemos en la graf.
83.         V1=df['V1'][i+1000*n]
84.         data['Voltaje'].append(V1)

85.         # Si el voltaje cruza el 0, guardamos ese numero de muestra para
compararlo
86.         # con el del amperaje y obtener el factor de potencia
87.         if V1>0 and lastV<0:
88.             muestrainitV=muestra
89.             cont=1
90.         lastV=V1

91.         # Vamos recogiendo datos de la columna A1 y los metemos en la graf.
92.         A1=df['A1'][i+1000*n]
93.         data['Intensidad'].append(A1)

94.         # Si el amperaje cruza el 0, guardamos ese número de muestra y
calculamos el
95.         # factor de potencia
96.         if A1>0 and lastA<0 and cont>0:
97.             muestrainitA=muestra
98.             difmuestra=int(muestrainitA)-int(muestrainitV)
99.             global factor_pot
100.            factor_pot=difmuestra/MUESTRAS_POR_PERIODO
101.        lastA=A1


102.        # Valor cuadrático medio
103.        # 1 - Elevamos al cuadrado el valor de las muestras
104.        sqV=float(V1)*float(V1)
105.        sqA=float(A1)*float(A1)
106.        # 2 - Sumamos
107.        sumV+=sqV
108.        sumA+=sqA
109.    #3 - Obtenemos el valor rms
110.    global Vrms, Arms
111.    Vrms=np.sqrt(sumV/1000)
112.    Arms=np.sqrt(sumA/1000)
```

5.2 Visor de FFT

Se trata de un visor de imágenes donde se nos habilita la opción de escoger la imagen a visualizar dependiendo de la componente que queremos estudiar:

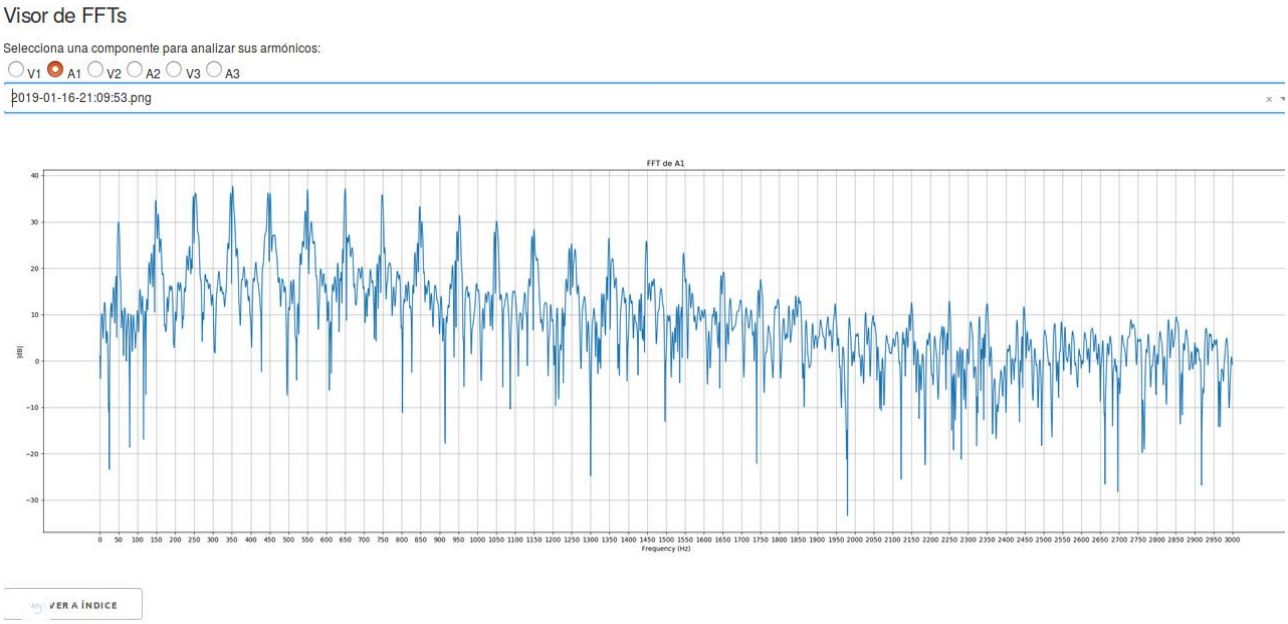

Figura 5-2 Interfaz correspondiente al visor FFT

5.3 Visor de tablas

En esta interfaz, podemos seleccionar la hoja de cálculo que queremos estudiar y tener una visión en la parte derecha de la página de las gráficas correspondientes a cada una de las columnas de interés:

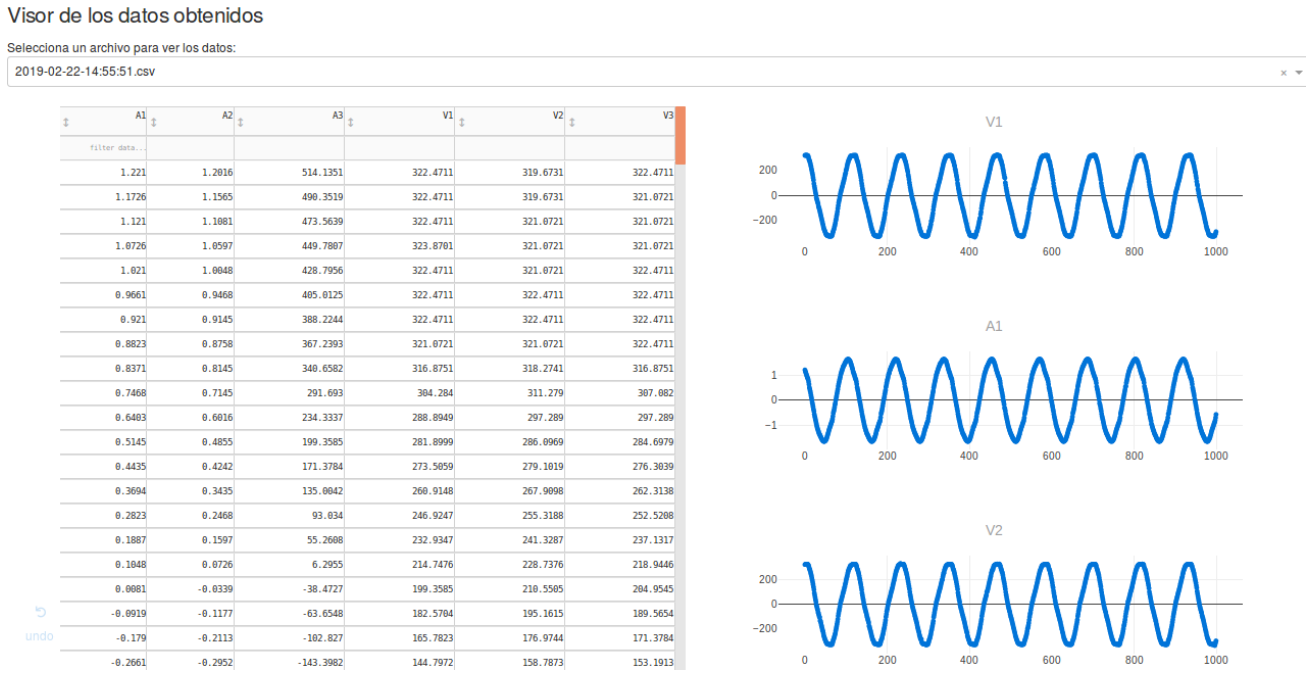

Figura 5-3 Interfaz correspondiente al visor de tablas

6 FUNCIONAMIENTO DEL SISTEMA

Se realizará una serie de pruebas para comprobar el funcionamiento del sistema. En la primera, se introducirán las mismas señales de voltaje e intensidad en las tres fases y se comprobará si hay desfase entre medidas. En la segunda, se cotejará la medida del sistema con la de un osciloscopio y se comprobará su similitud.

6.1 Retardo en la medida

6.1.1 Fase A

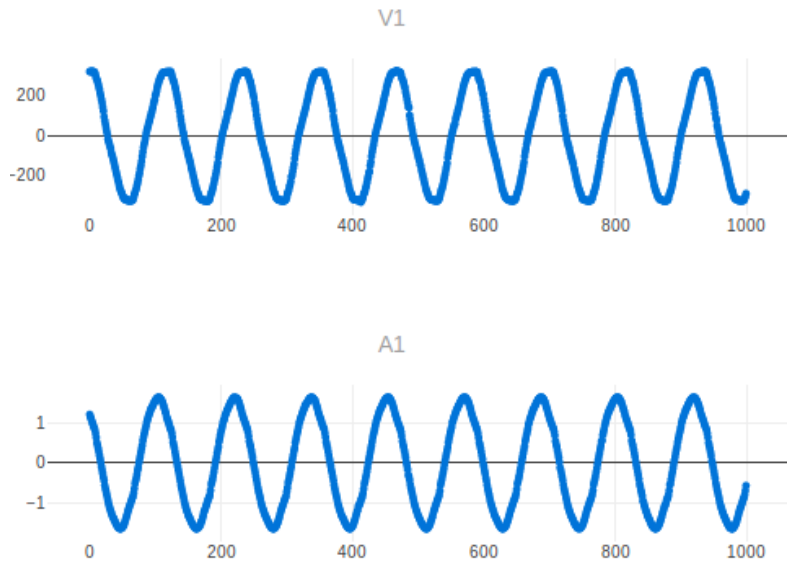

Figura 6-1 Gráfica de la fase A

6.1.2 Fase B

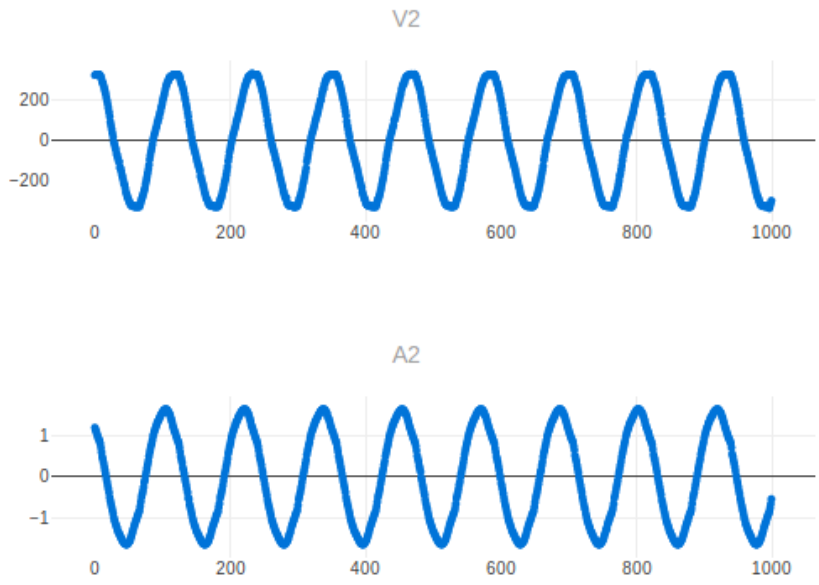

Figura 6-2 Gráfica de la fase B

6.1.3 Fase C

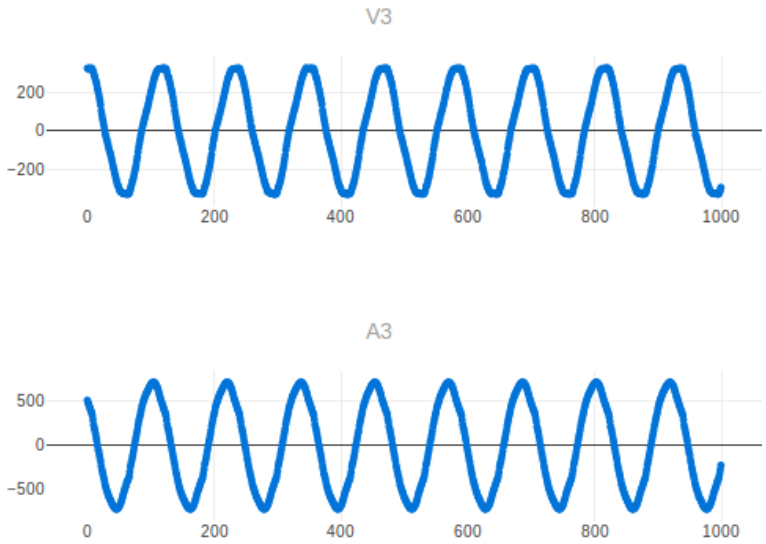

Figura 6-3 Gráfica de la fase C

6.2 Resultados según elemento de carga

6.2.1 Carga reactiva

Para el primero se probó a conectar el cargador de un móvil con un amperaje de entrada de 0.5A como máximo y con una resistencia R5 en el adaptador del apartado 2.2 con R5 = 4.85KΩ.

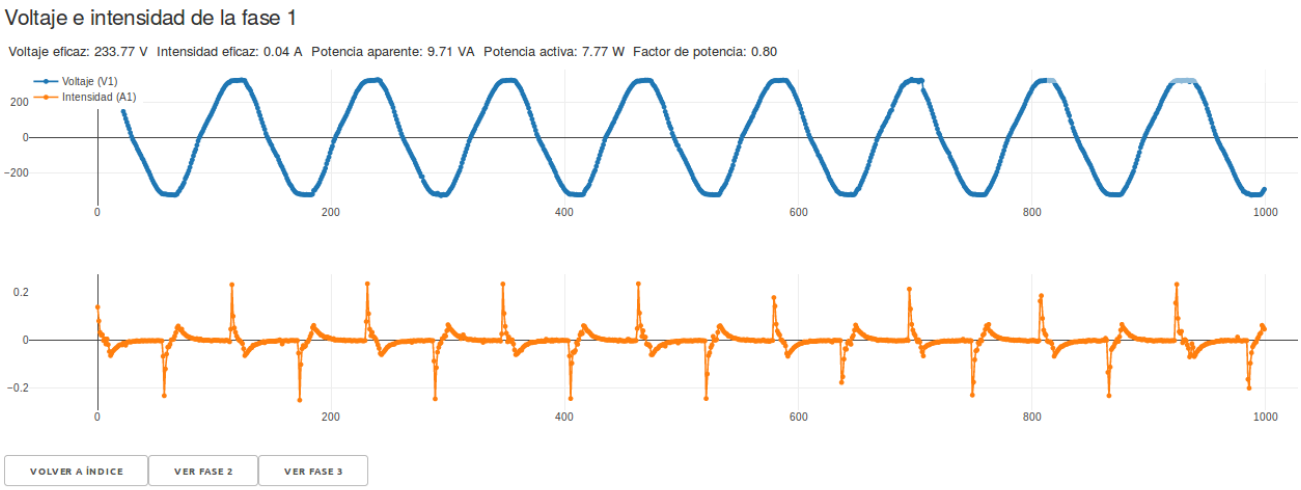

Figura 6-4 Gráfica del sistema con un cargador de móvil como carga

Esta intensidad medida en el osciloscopio nos da la siguiente forma:

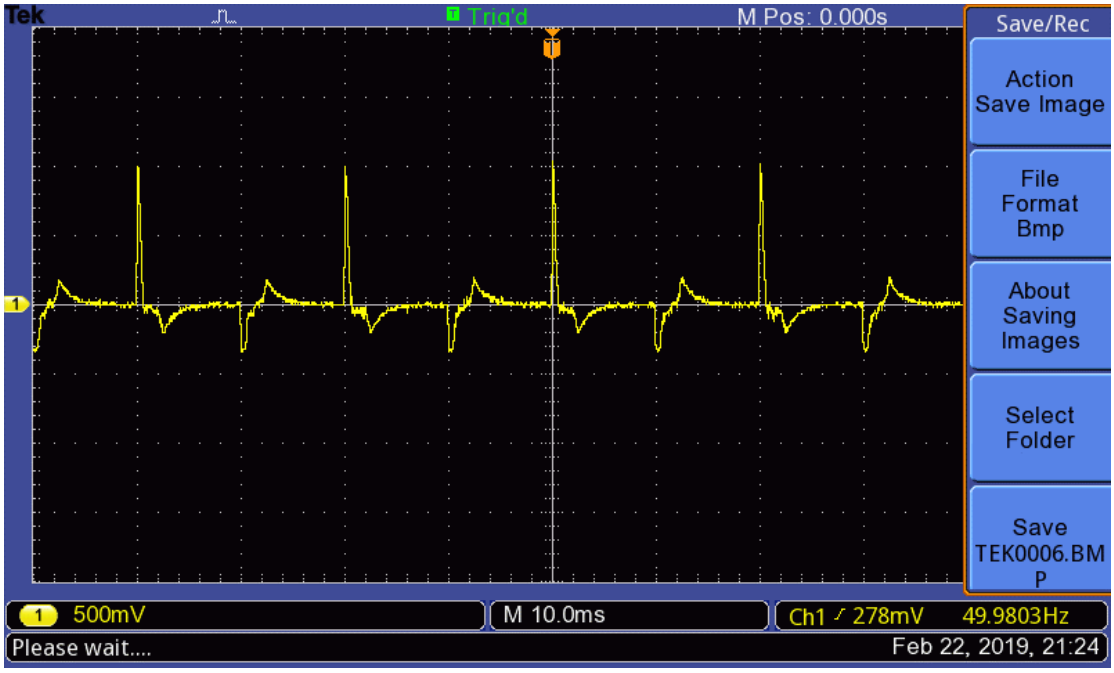

Figura 6-5 Gráfica del osciloscopio con un cargador de móvil como carga

En cuanto a su FFT de la intensidad:

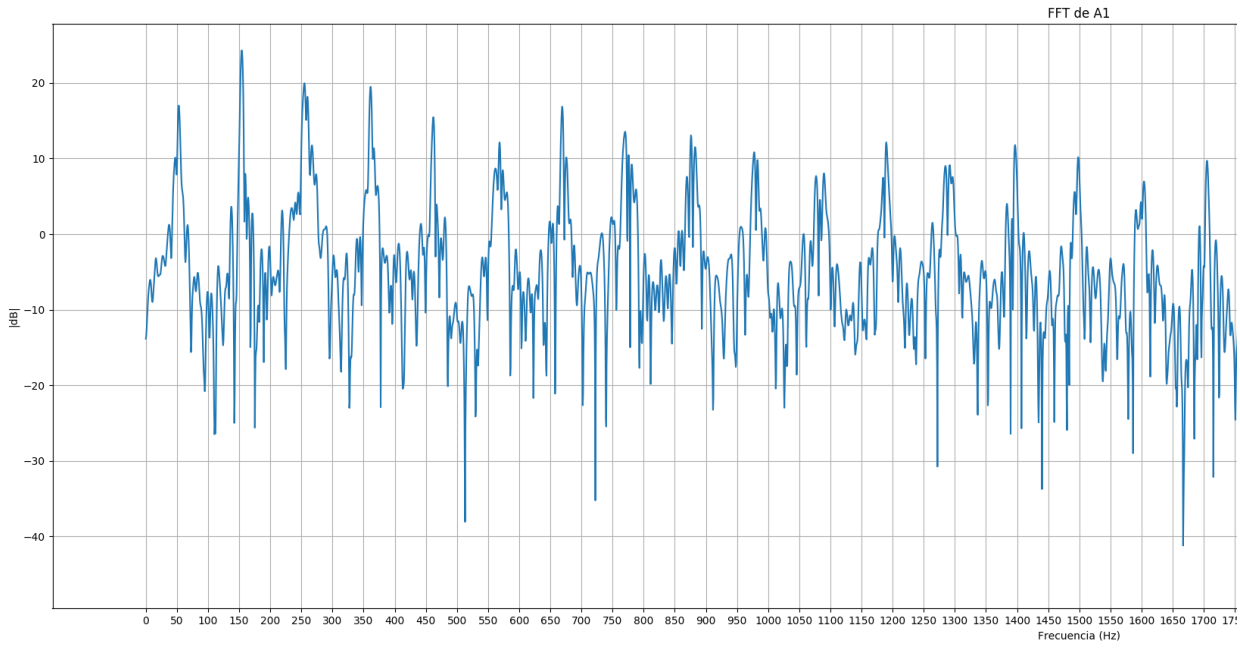

Figura 6-6 Gráfica de la FFT de la intensidad con un cargador de móvil como carga

6.2.2 Carga resistiva

En esta segunda medida se conectó un radiador resistivo de 200W de potencia y una resistencia R5 en el adaptador del apartado 2.2 con R5 = 2KΩ.

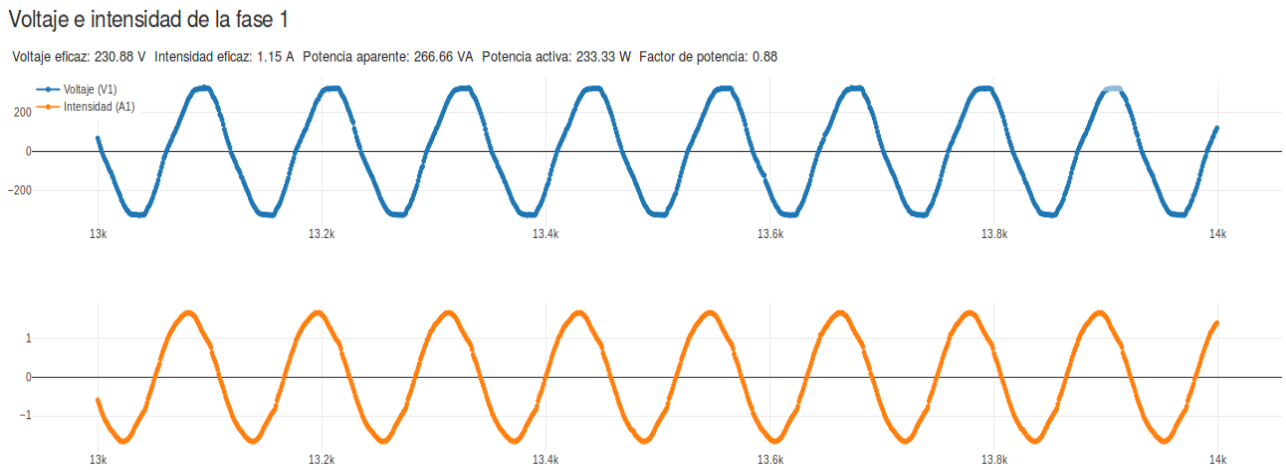

Figura 6-7 Gráfica del sistema con un calefactor como carga

Esta intensidad medida en el osciloscopio nos da la siguiente forma:

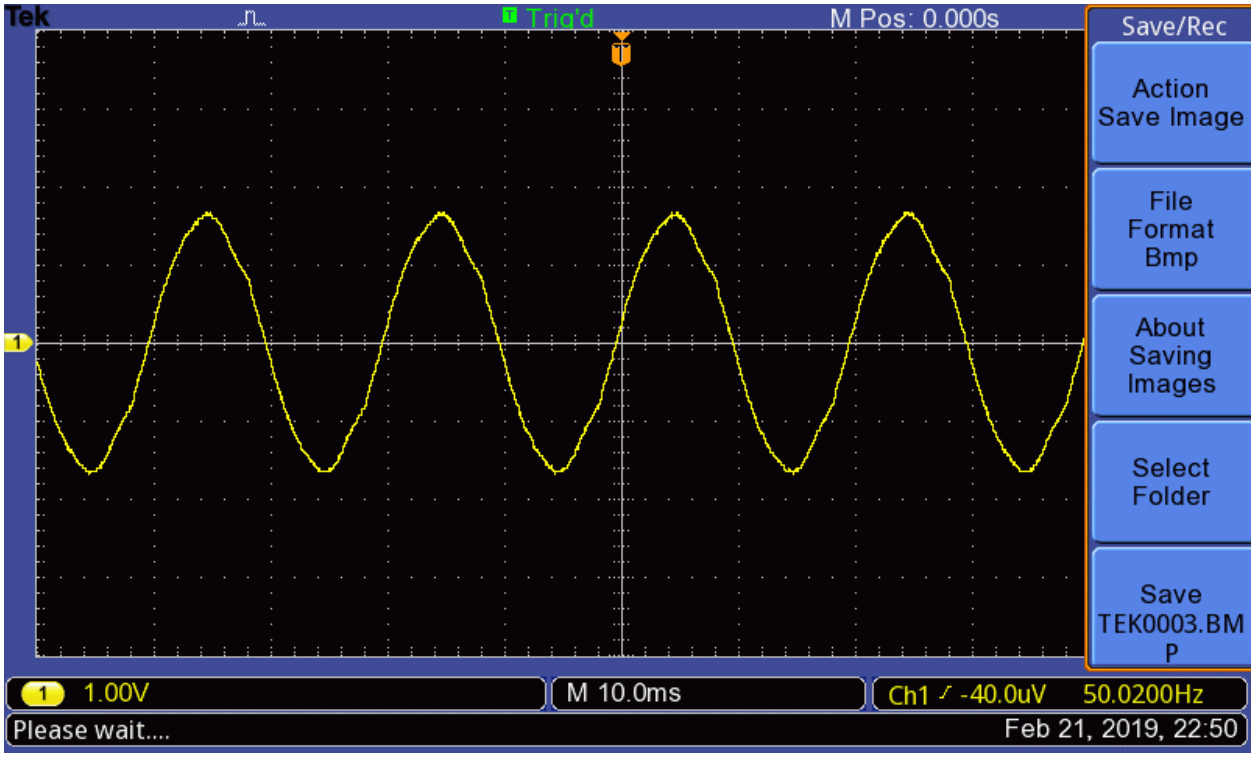

Figura 6-8 Gráfica del osciloscopio con un calefactor como carga

En cuanto a su FFT de la intensidad:

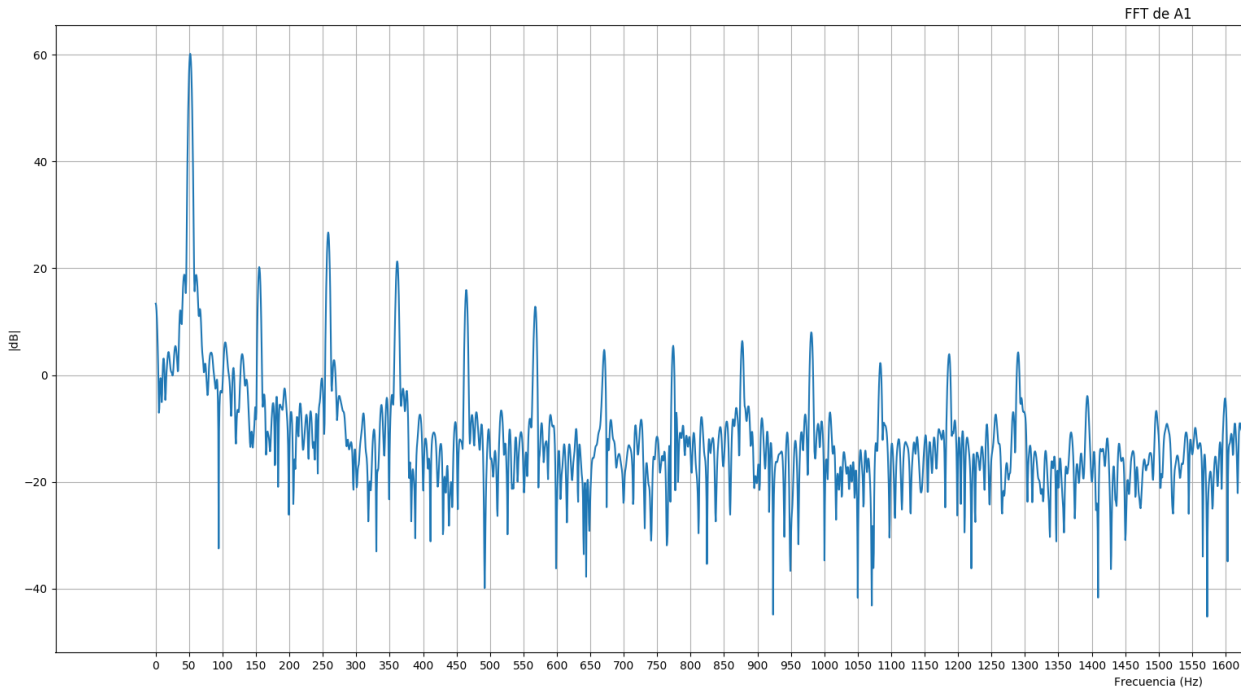

Figura 6-9 Gráfica de la FFT de la intensidad con un calefactor como carga

6.2.3 Valores del voltaje

Puesto que la alimentación siempre es la misma, su gráfica en el osciloscopio es de la forma:

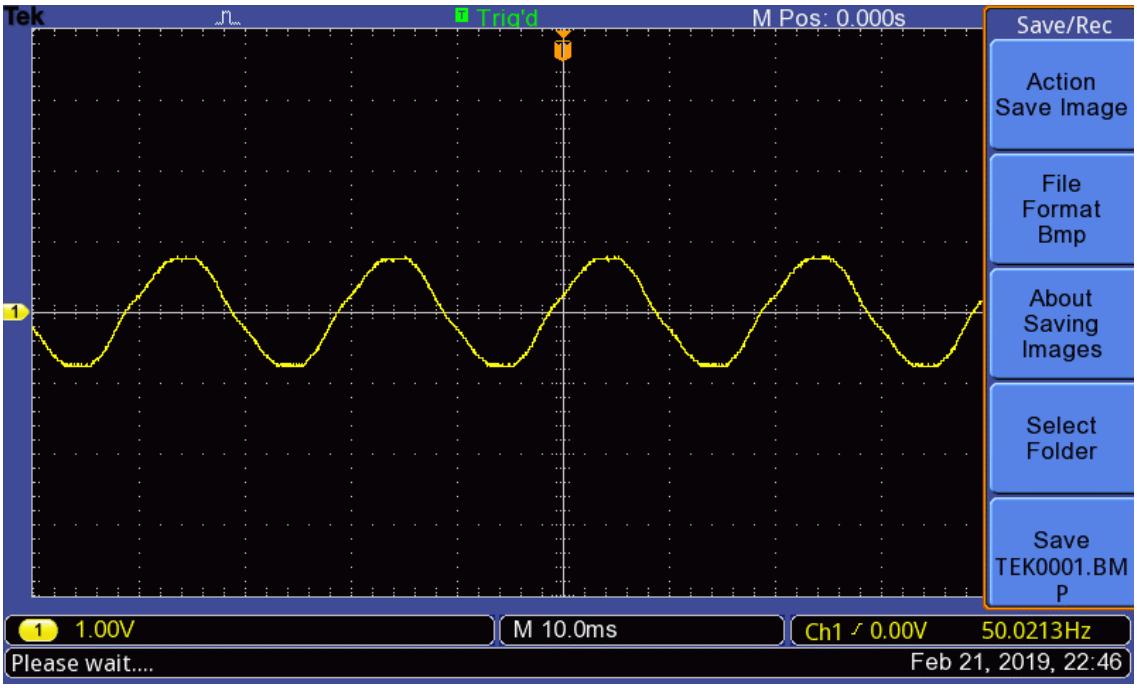

Figura 6-10 Gráfica del osciloscopio

En cuanto a su FFT:

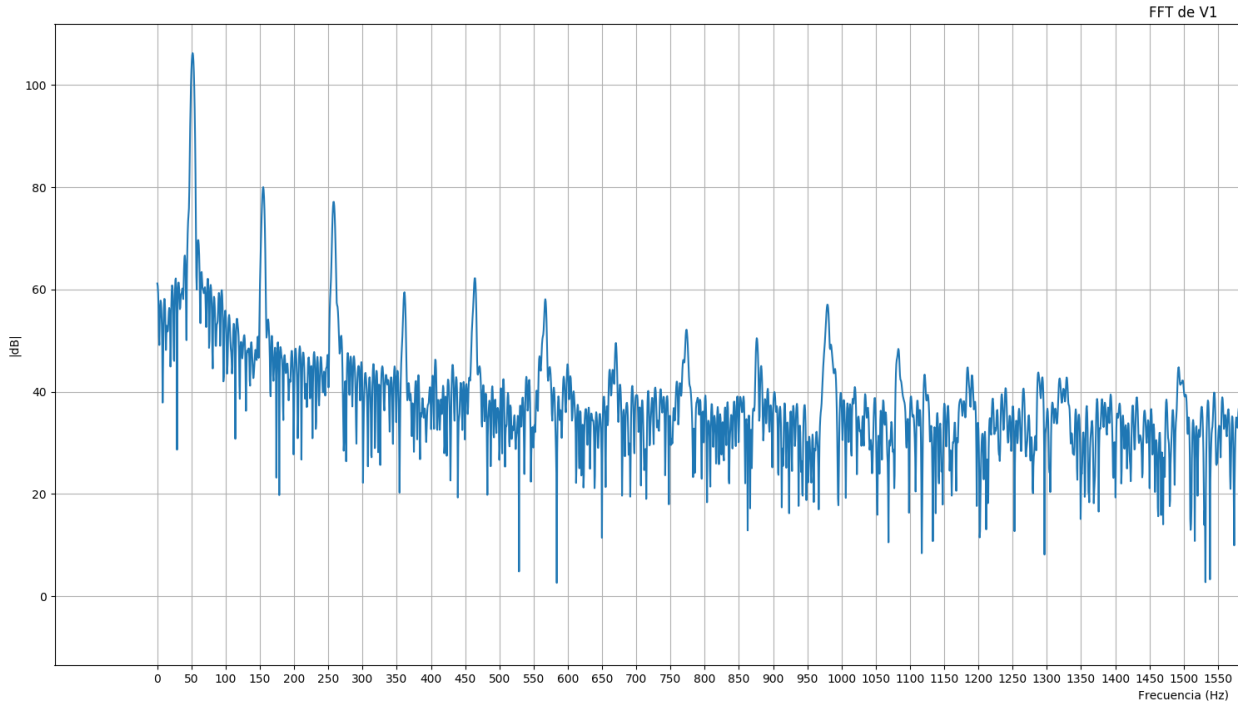

Figura 6-11 Gráfica de la FFT

6.3 Análisis de los resultados

Si primero nos fijamos en los valores de voltaje, puede observarse que tanto el valor eficaz como su FFT, el primero con un valor eficaz cercano a 230V y la segunda con valores de pico en los múltiplos impares de 50Hz (puesto que es un sistema trifásico), son muy cercanos a la realidad. En general, el cálculo de las FFTs en todas las ocasiones da valores muy satisfactorios, ya que, aunque se aprecia cierto ruido, los valores picos suelen ser cercanos a frecuencias múltiplos de 50Hz, esto es gracias al enventanado realizado, que elimina gran parte de los ruidos no deseables.

La intensidad en el caso de la carga reactiva no posee su armónico fundamental en los 50Hz, lo que era de esperar, puesto que su forma de onda dista mucho de una forma senoidal. No obstante, en el caso de la carga resistiva sí que nos encontramos con un armónico fundamental de 50Hz y una forma de onda casi sinusoidal. En el caso de la carga reactiva se hace difícil un cálculo de desfase y potencia activa verosímil, esto es debido a que el método utilizado para el cálculo no es útil para formas de onda que suelen tomar muchos valores cercanos a cero, que, sumado al ruido en la medida, hace complejo calcular el periodo de la onda. Un acercamiento para paliar este defecto podría ser el de descomponer la onda en su componente fundamental y calcular el periodo de la misma.

Como se especificó en el apartado de adaptación de señal, un voltaje de offset más estable podría suponer una mejora en la medida. Además, un amplificador buffer a la entrada del ADC podría mejorar cierto ruido creado internamente en el ADC debido a las capacitancias parásitas ocasionadas por el constante cambio de canal de medida. En la toma de medidas per se, según se puede ver en el apartado 6.1, no parece registrarse un retardo significativo ocasionado por una lenta velocidad de muestreo o de cambio de canal de medida.

6.4 Conclusiones

A la vista de los resultados obtenidos podemos concluir que el objetivo planteado, a saber, la creación de un sistema de bajo coste para la monitorización de sistemas trifásicos, ha sido alcanzado con creces. A continuación, se listan algunas de sus ventajas e inconvenientes en adición a las ya mencionadas a lo largo del presente trabajo.

6.4.1 Ventajas del sistema

- Puesto que se tiene un control total del proceso de medida, desde la toma del dato en bruto hasta su representación en las diferentes interfaces, se abre la posibilidad de modificar e incluso de añadir toda utilidad necesaria a partir del sistema base.
- Su bajo coste y la facilidad de obtención y prototipado de los diferentes componentes empleados.
- La facilidad para calibrar los diferentes parámetros de medida y adaptarlos al sistema que se quiere analizar.
- Se pueden obtener diversos datos referentes al sistema de forma sencilla en tiempo real: tanto valores RMS como armónicos fundamentales del mismo.
- La parte del servidor no tiene por qué quedarse limitada a una raspberry, sino que puede ser implementada en cualquier CPU con un sistema operativo basado en Linux que conste de un puerto USB o serie, con la mejora en recursos computacionales que ello pueda conllevar.
- Posibilidad de monitorizar tanto sistemas trifásicos como monofásicos.

6.4.2 Desventajas del sistema

- Pese a que el modo SCAN del ADC del Arduino empleado no supone un déficit significativo en la capacidad de muestreo, varios ADCs simultáneos podrían mejorarla e incluso reducir el ruido provocado internamente debido al cambio constante del canal de medida.
- Haciendo uso de un sistema operativo en tiempo real en la parte correspondiente a la CPU, se podría mejorar la lectura de medidas por el puerto serie al no compartir los recursos asociados a la tarea de lectura con otras tareas del sistema que no hayan sido debidamente planificadas.

6.5 Futuras líneas de investigación

Existe un concepto denominado Desagregación de la Potencia (del inglés: Power Disaggregation [18]) que se basa en hacer uso de la huella característica de consumo de cada uno de los dispositivos que podemos encontrar en un hogar: tanto de la huella de la potencia calculada a partir los valores RMS como la de la forma característica de la intensidad consumida en un periodo. A partir de esta huella, el usuario puede conocer en cada momento qué factores influyen en el consumo eléctrico de su hogar, pudiendo actuar en consecuencia. Este concepto no quedaría limitado solamente al de una acción pasiva por parte del usuario, un sistema domótico también podría beneficiarse enormemente de un reconocimiento mediante IA de los patrones en el consumo, pudiendo accionar diferentes actuadores en consecuencia o dar un feedback específico al usuario. Un futuro sistema podría tener en su base de datos información relativa a las diferentes tarifas de consumo que ofertan las compañías eléctricas y aconsejar al usuario la más económica dependiendo de sus hábitos de consumo.

ÍNDICE DE CONCEPTOS

C.1 FFT

En el tratamiento digital de la señal, existe un concepto denominado DFT: dada una secuencia discreta y de duración finita, suponiendo que se analiza un único periodo de una señal periódica extendida de forma infinita, aunque la propia señal no lo sea, la DFT es una función matemática que nos permite representar dicha señal en el dominio de la frecuencia, en contraste con la habitual representación con respecto al tiempo. Esto significa que toda señal puede ser descompuesta en un sumatorio de senos y/o cosenos con diferente periodo [19].

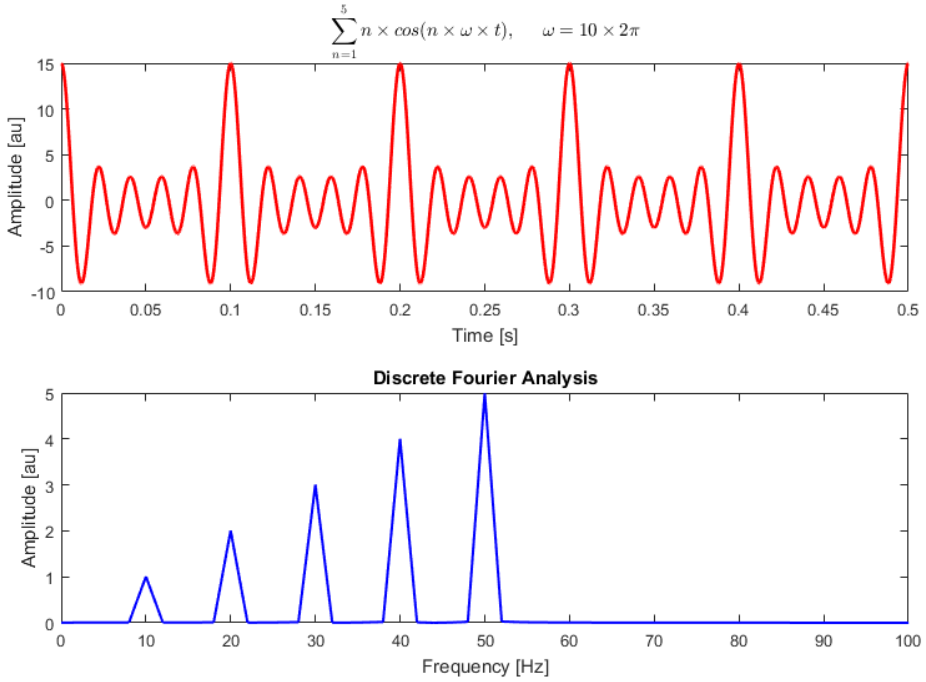

Figura 0-1 DFT del sumatorio de 5 cosenos de frecuencia variable

Consecuentemente, cuando hablamos de FFT [20], nos referimos a un algoritmo que, de forma computacionalmente eficiente en lo que respecta a número y complejidad de operaciones, se utiliza para calcular la DFT.

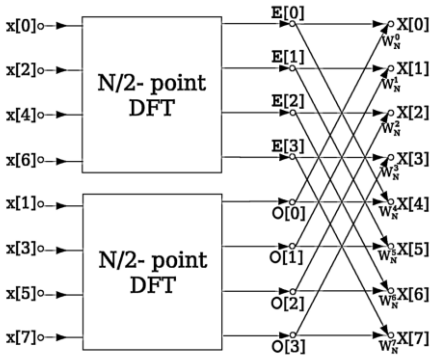

Figura 0-2 FFT en mariposa de una señal

C.2 Ventana de Kaiser

Cuando queremos calcular la FFT de una señal, se suele suponer que se ha adquirido un número entero de periodos, sin embargo, esta suposición dista de la realidad, puesto que, además de no ser siempre señales periódicas, podemos encontrarnos con ciertas discontinuidades en los extremos e incluso a lo largo de la señal por diversos factores como ruido en la medida, lentitud de procesamiento, digitalización de la señal, etc. Estos efectos se introducen en la señal en forma de componentes de alta frecuencia. Para eliminarlos, es necesario el uso de funciones ventana [21], que realizan un prefiltrado de los datos antes de calcular su FFT correspondiente. Existen muchos tipos de ventanas: Hamming, Blackman, Gauss, etc. Cada una de estas ventanas tiene sus particularidades, sobre todo en lo que respecta al rizado del propio enventanado, ya sea en la banda de paso o en la de rechazo. Entre ellas, la ventana de Kaiser suele ser ampliamente utilizada, puesto que es una de las ventanas más cercanas a una ventana ideal. La FFT de una ventana Kaiser [22] puede tener la forma:

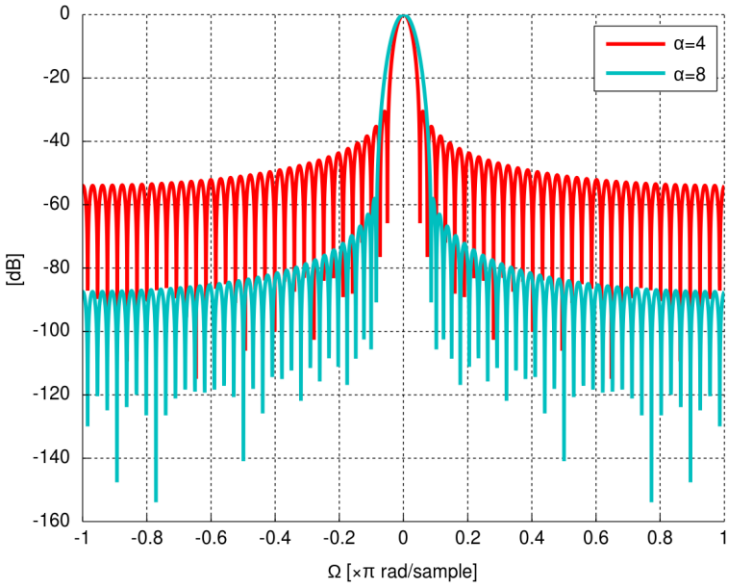

Figura 0-3 FFT de una ventana kaiser para α 4 y 8

Matemáticamente, una ventana de Kaiser se puede definir de la siguiente forma [23]:

$$w(n) = I_0\left(\beta\sqrt{1-\frac{4n^2}{(M-1)^2}}\right)/I_0(\beta)$$

Para un número de muestras:

$$-\frac{M-1}{2} \leq n \leq \frac{M-1}{2},$$

Siendo I_0 una función Bessel modificada de primer tipo. Al aplicar esta fórmula necesitamos definir tanto el número de muestras como el parámetro β; cuanto mayor es β, más estrecho es el lóbulo central.

C.3 ADC

Cuando se quiere recoger datos de una fuente externa, por ejemplo, sensores, desde un sistema informático o CPU, los datos pueden ser tanto análogicos como digitales. En el caso de datos que adoptan valores analógicos, es necesario transformarlos a un valor digital antes de que puedan ser procesados por el sistema. Un ADC [24] es un dispositivo electrónico que se encarga de transformar un voltaje analógico de entrada a un valor digital según parámetros internos del mismo. Puede estar definido por gran cantidad de parámetros, aunque en el presente estudio sólo algunos resultan de nuestro interés:

- Resolución: dada en n número de bits. Nos indica que un dato analógico sólo puede adoptar un valor discreto entre 0 y $2^n - 1$ como equivalente digital.
- Máximo valor de referencia: Sería el máximo valor de voltaje analógico que se puede convertir y cuyo valor digital equivaldría a $2^n - 1$, esto es, el mayor valor posible.
- Frecuencia de muestreo: cantidad máxima de valores análogicos que el ADC puede convertir en un segundo.

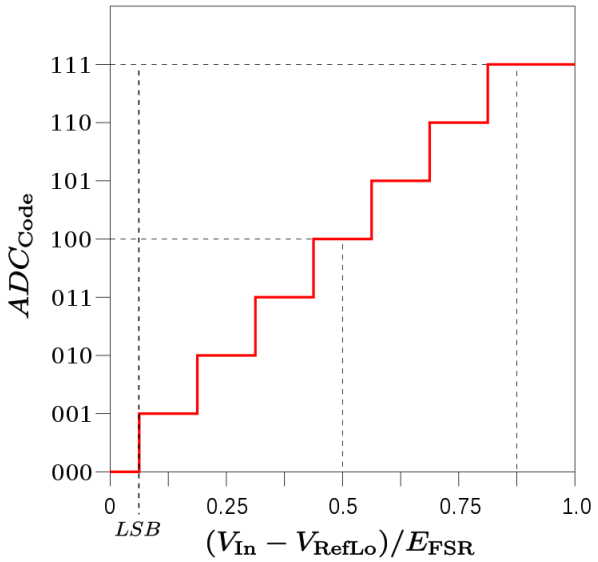

Figura 0-4 Cuantización de los valores de un ADC

En la anterior tabla puede verse a qué valor digital (eje y) se convertiría cualquier valor analógico entre 0 y 1V (eje x) de entrada suponiendo un ADC de 3 bits de resolución al que se le introduce un voltaje de referencia de 1V.

APÉNDICE: CÓDIGOS

A continuación se adjuntarán los diferentes códigos que dan forma a este proyecto para facilitar su referencia y acceso:

A.1 Códigos de Arduino

A.1.1 read_adc.ino

```
1. // Código correpondiente a Arduino SAMD Board
2. // Arduino MKR Wifi 1010
3. // Con chip Atmel SAM R21D
4.

5. #include "readfast.h"
6.
7. uint32_t StartTime, SecondTime, ElapsedTime=0;
8.
9. void setup()
10. {
11.
12.  // Inicio del puerto serie a 500K baudios
13.  Serial.begin(500000);
14.
15.  canal=-1;  //Variable ya declarada en readfast.h
16.
17. // Configuramos el ADC escribiendo en sus registros
18. setupADC();
19.
20. // Nos aseguramos de realizar un primer ciclo de lectura
21. // que es el mas propenso a fallo
22. while(canal!=0)
23.    readana();
24.
25. }
26.
27. void loop()
28. {
29.  // Recogemos la medida
30.  uint16_t valueRead =readana();
31.  StartTime = micros();
32.
33.  // Enviamos la medida por el puerto serie
34.  sendBytes(valueRead);
```

```
35.  SecondTime = micros();
36.
37.  // El tiempo que se tarda en enviar un dato es variable,
38.  // por lo que esperamos 20 us para tener un codigo predecible
39.  while((SecondTime-StartTime) < 20)
40.      SecondTime=micros();
41. }
42.

43. // Función que envía tres bytes por serie
44. void sendBytes(uint16_t value)
45. {
46.   uint8_t array[3];
47.
48.   // El primer byte enviado es el canal leido
49.   // El segundo y el tercero son los 10 bits divididos
50.   // en dos bytes
51.   array[0]=canal;
52.   array[1]=(highByte(value));
53.   array[2]=(lowByte(value));
54.
55.   Serial.write(array,3);
56.
57. }
```

A.1.2 readfast.h

```
1. #include <Arduino.h>
2. #include <wiring_private.h>
3.
4. void inline setupADC(void);
5. uint16_t inline readana(void);
6. uint8_t canal;
7.
8. void inline setupADC(void)
9. {
10.
11. // Establecemos el voltaje de ref. en el ADC a 3.3V
12. analogReference(AR_DEFAULT);
13.
14. // Establecemos cada uno de los pines físicos que
16. // vamos a utilizar como Inputs Analógicos
17. pinPeripheral(A1, PIO_ANALOG);
18. pinPeripheral(A2, PIO_ANALOG);
19. pinPeripheral(A3, PIO_ANALOG);
20. pinPeripheral(A4, PIO_ANALOG);
```

```
21. pinPeripheral(A5, PIO_ANALOG);
22. pinPeripheral(A6, PIO_ANALOG);
23.
24. // Tras cada escritura en un registro del ADC o llamada a una función
25. // que escribe en el registro, esperaremos a que el bit SYNCBUSY
26. // del registro de STATUS se ponga a 0, siguiendo las recomendaciones
27. // del datasheet para evitar corrupciones
28. while (ADC->STATUS.bit.SYNCBUSY == 1);
29.
30. // Deshabilitamos el ADC para cambiar sus registros
31. ADC->CTRLA.bit.ENABLE = 0x00;
32. while (ADC->STATUS.bit.SYNCBUSY == 1);
33.
34. // 6 canales para escanear, desde A1 hasta A6 (INPUTSCAN + 1)
35. // segun datasheet
36. ADC->INPUTCTRL.bit.INPUTSCAN=7;
37. while (ADC->STATUS.bit.SYNCBUSY == 1);
38.
39. // Para empezar el escáner desde lo establecido en el próximo
40. // campo MUXPOS
41. ADC->INPUTCTRL.bit.INPUTOFFSET=0;
42. while (ADC->STATUS.bit.SYNCBUSY == 1);
43.
44. // Empezamos el escaneo en el pin mas bajo posible, A1 en este caso
45. // que equivale a g_APinDescription[A3].ulADCChannelNumber
46. ADC->INPUTCTRL.bit.MUXPOS=g_APinDescription[A3].ulADCChannelNumber;
47. while (ADC->STATUS.bit.SYNCBUSY == 1);
48.
49. // Máscara de bits para el PRESCALER
50. ADC->CTRLB.reg &= 0b1111100011111111;
51.
52. // Divisor de reloj de 128, uno menor a 64 no funciona experimentalmente
53. ADC->CTRLB.reg |= ADC_CTRLB_PRESCALER_DIV128;
54.
56. // Tomamos una muestra por petición de medida y ajustamos el resultado
57. ADC->AVGCTRL.reg = ADC_AVGCTRL_SAMPLENUM_1 | ADC_AVGCTRL_ADJRES(0x00ul);
58.
59. // Tiempo de toma de muestra igual a 0
60. ADC->SAMPCTRL.reg = 0x00;
61.
62. // Habilitamos el ADC
63. ADC->CTRLA.bit.ENABLE = 0x01;
64. while(ADC->STATUS.bit.SYNCBUSY == 1);
65.
66. // Iniciamos conversión
67. ADC->SWTRIG.bit.START = 1;
68.
69. // Limpiamos el flag que indica que hay un dato en memoria
70. // para recoger
```

```
71. ADC->INTFLAG.reg = ADC_INTFLAG_RESRDY;
72.
73. }
74.
75.

76. uint16_t inline readana(void)
77. {
78.   uint16_t valueRead = 0;
79.
80.   // El siguiente registro va aumentando conforme se va
81.   // tomando una medida, nos indica de que pin ha leído el ADC
82.   canal=ADC->INPUTCTRL.bit.INPUTOFFSET;
83.   while (ADC->STATUS.bit.SYNCBUSY == 1);
84.
85.   // Este condicional hace que se salten el canal 5 y 6
86.   // en la toma de datos y se vuelva al primero al leer todos
87.   // los pines, esto es necesario debido a la distribución
88.   // interna de pines en el microcontrolador
89.   if(canal==4)
90.   {
91.       canal=canal+2;
92.       ADC->INPUTCTRL.bit.INPUTOFFSET=canal;
93.   }
94.   else if(canal==8)
95.   {
96.       canal=0;
97.       ADC->INPUTCTRL.bit.INPUTOFFSET=canal;
98.   }
99.   while (ADC->STATUS.bit.SYNCBUSY == 1);
100.
101.    // Realizamos una medida en el canal indicado con INPUTOFFSET
102.    // y esperamos a que la conversión se complete
103.    ADC->SWTRIG.bit.START = 1;
104.    while (ADC->INTFLAG.bit.RESRDY == 0);
105.
106.    //Recogemos el resultado
107.    valueRead = ADC->RESULT.reg;
108.
109.
110.    return valueRead;   //La resolución es de 10 bits
111.
112. }
```

A.2 Código de adquisición de datos

A.2.1 data.py

```
1. # Script principal al que hay que llamar para iniciar la toma
2. # de datos y generación de FFTs
3.
4. import datetime, matplotlib, serial, time, os
5. matplotlib.use('agg')  # Solución de bug para hacer gráficas en thread
6. import numpy as np
7. from app import app
8. import pandas as pd
9. from threading import Thread
10. import pylab as pl
11.
12. SECONDS_FILE = 20        # En segundos
13. FREC_MUESTREO = 6000     # En Hertzios
14.
15. # Variables globales con los coeficientes correspondientes a las
16. # características del ADC
17. ADC_RESOLUTION = 1023
18. ADC_VREF = 3.3
19. ADC_BIAS = ADC_VREF/2
20. COEF_ADC = ADC_RESOLUTION/ADC_VREF
21.
22. # Coeficientes del voltaje
23. MAX_VALUE_VOLT = 9*np.sqrt(2)
24. REAL_VALUE_VOLT = 230*np.sqrt(2)
25.
26. # Coeficientes de la intensidad
27. RESISTENCIA_CARGA = 1000
28. COEF_PINZA = 2000  #100A en la realidad son 50mA (100/0.05)
29.
30. # Creamos los directorios donde guardar los .png de los FFT si no existen
31. directories = ['fft/V1','fft/A1','fft/V2','fft/A2','fft/V3','fft/A3']
32. for i in range(len(directories)):
33.     if not os.path.exists(directories[i]):
34.         os.makedirs(directories[i])
35.
36. # Creamos el directorio log si no existe
37. if not os.path.exists('log/'):
38.     os.makedirs('log/')
39.
40. # Inicializamos el puerto serie y las variables encargadas de guardar
41. # los datos recibidos
42. arduino = serial.Serial('/dev/ttyACM0', 500000)
43. word = np.zeros(8, dtype=np.int)
44. fila = ""
45.
46. def main():
```

```
47.
48.     while True:
49.
50.         # Creamos un nuevo .csv
51.         savefile=newfile()
52.
53.         # Mientras no pase el tiempo de la var. SECONDS_FILE se van
54.         # rellenando filas con los datos recibidos por el puerto serie
55.         while minutes_passed(savefile.startime):
56.             # Según el codigo en el arduino read es una variable de
57.             # tres bytes, en read[0] se guarda el canal leído y en
58.             # read[1] y read[2] el higbyte y lowbyte de la medida
59.             read = arduino.read(3)
60.
61.             # Si se han leído 0 bytes es que no nos ha llegado nada
62.             if len(read) == 0:
63.                 break
64.
65.             # En info guardamos el valor de la medida manipulando los bits
66.             info = (read[1] << 8) | read[2]
67.
68.             # Dependiendo del valor de canal guardamos el dato en diversas
69.             # posiciones del array word
70.             word[read[0]-1]=info
71.
71.             #Si el canal leído es el 7 significa que ya hemos leído los 6
72. canales
73.             if read[0]==7:
74.
75.fila=(calculatevoltage(word[6])+','+calculateintensity(word[7])+','+calculate
76.voltage(word[0])+','
77.
78.+calculateintensity(word[1])+','+calculatevoltage(word[2])+','+calculatevolta
79.ge(word[5])+'\n')
80.                 savefile.file_descriptor.write(fila)
81.
82.         # Al salir del bucle cerramos el descriptor del fichero e iniciamos
83. un thread
84.         # que llame a la funcion processdata(), encargada de generar las FFT
85. en segundo plano
86.         savefile.file_descriptor.close()
87.         thread = Thread(target = processdata, args = (savefile, ))
88.         thread.start()
89.

90. # Función encargada de crear un fichero .csv con la fecha de creación
91. # y poner en la primera fila los nombres de las columnas
92. def newfile():
93.     oldepoch = time.time()
```

```
94.     st = datetime.datetime.fromtimestamp(oldepoch).strftime('%Y-%m-%d-
95.%H:%M:%S')
96.     file = open('log/'+st+'.csv','w')
97.     savefile=record_file(oldepoch,file,st)
98.     file.write("V1,A1,V2,A2,V3,A3\n")
99.     return savefile
100.
101. # Clase que rellena los descriptores del fichero .csv
102. class record_file:
103.     def __init__(self, startime, file_descriptor,file_name):
104.         self.startime = startime
105.         self.file_descriptor = file_descriptor
106.         self.file_name= file_name
107.
108. # Función que comprueba si ha pasado el tiempo establecido para crear un
109. # nuevo .csv
110. def minutes_passed(oldepoch):
111.     return time.time() - oldepoch <= SECONDS_FILE # segundos
112.
113. # Función que lee cada una de las columnas generads en el .csv y crea un
.png
114. # con las FFT (Transformadas rapidas de Fourier) de las componentes V e I
116. def processdata(arg):
117.     field = ['V1','A1','V2','A2','V3','A3']
118.     for j in range(0,6):  #Iteramos por las 6 columnas
119.         datos = np.zeros(10000)
120.         i,h,n = 0,0,0
121.
122.         # Abrimos el fichero creado anteriormente
123.         # Solo analizaremos los primeros 10000 datos
124.         csv_reader = pd.read_csv('log/'+arg.file_name+'.csv')
125.        for i in range(10000):
126.             datos[i]=csv_reader[field[j]][i]
127.
128.
129.         # Iniciamos los cálculos para obtener la FFT de cada una de las
componentes
130.         # Aplicaremos un enventanado Kaiser con una beta igual a 100
131.         N = len(datos)
132.         datos -= np.mean(datos)    # Eliminamos el offset de DC
133.         datos_enventanados = datos * np.kaiser(N, 100)
134.         fft_enventanado = np.fft.rfft(datos_enventanados)
135.
136.         # Procedemos a crear la gráfica con los datos necesarios
137.         fig = pl.figure(figsize=(35,10))
138.         ejey = 20*np.log10(np.abs(fft_enventanado))
139.         ejex = np.linspace(0, FREC_MUESTREO/2, len(ejey))
```

```
140.         # Modificamos el estilo de la gráfica: título, ejes, etc
141.         ax = fig.add_subplot(111)
142.         ax.plot(ejex,ejey)
143.         rangex = np.zeros(61)
144.         for h in range(0, 3050, 50):
145.             rangex[n]=h
146.             n = n+1
147.         ax.xaxis.set_ticks(rangex)
148.         ax.grid(True)
149.         ax.set_title('FFT de ' + field[j])
150.         ax.set_xlabel('Frecuencia (Hz)')
151.         ax.set_ylabel('|dB|')
152.
153.         # Guardamos el archivo .png de la gráfica con un nombre determinado
154.         pl.savefig(directories[j]+'/'+arg.file_name+'.png')
155.         pl.close(fig)

156. # Función usada para convertir el valor leído por el ADC a un valor
157. # de voltaje que se pueda usar
158. def calulatevoltADC(arg):
159.     volts=arg/COEF_ADC
160.     return volts

161. # Función para calcular el voltaje real que se está midiendo del
transformador
162. def calculatevoltage(arg):
163.     volt=calulatevoltADC(arg)-ADC_BIAS
164.     div=2.4-ADC_BIAS  # 2.4 es el valor leído con 230*sqrt(2) voltios
165.     return "%.4f" % ((volt/div)*REAL_VALUE_VOLT)

166.# Función para calcular la intensidad real que se está midiendo con la pinza
167. def calculateintensity(arg):
168.     volt=calulatevoltADC(arg)-ADC_BIAS  # Voltaje calculado
169.     value=(volt/RESISTENCIA_CARGA)*COEF_PINZA
170.     return "%.4f" % (value)

171. if __name__ == "__main__":
172.     main()
```

A.3 Códigos del servidor web

A.3.1 index.py

```
1. # Script principal al que hay que llamar para iniciar el servidor

2. import dash_core_components as dcc
3. import dash_html_components as html
4. from dash.dependencies import Input, Output
5. from app import app
6. from apps import phase1_live, phase2_live, phase3_live
7. from apps import  main_page, fft_viewer, table_viewer

8. app.layout = html.Div([
9.     dcc.Location(id='url', refresh=False),
10.     html.Div(id='page-content')
11. ])

12. # Callback encargado de enlazar cada página con los scripts en el
13. # directorio /apps
14. @app.callback(Output('page-content', 'children'),
15.               [Input('url', 'pathname')])
16. def display_page(pathname):
17.      if pathname == '/':
18.           return main_page.layout
19.      elif pathname == '/phase1_live':
20.           return phase1_live.layout
21.      elif pathname == '/phase2_live':
22.           return phase2_live.layout
23.      elif pathname == '/phase3_live':
24.           return  phase3_live.layout
25.      elif pathname == '/fft_viewer':
26.           return  fft_viewer.layout
27.      elif pathname == '/table_viewer':
28.           return  table_viewer.layout
29.      else:
30.           return '404'

31. def main():
32.     app.run_server(debug=True)

33. if __name__ == '__main__':
34.     main()
```

A.3.2 app.py

```
1. # Script al que se llama desde index.py para iniciar el servidor
2. import dash

3.app = dash.Dash(__name__)
4.server = app.server
5.app.config.suppress_callback_exceptions = True
```

A.3.3 apps/main_page.py

```
1. import dash_core_components as dcc
2. import dash_html_components as html
3. from dash.dependencies import Input, Output
4. from app import app

5. # Página de índice con los hipervínculos a las distintas secciones
6. layout = html.Div([
7.     html.H3('Monitorización de sistemas trifásicos'),
8.    dcc.Link('Gráfica en tiempo real de fase 1 (V1 y A1)',
href='/phase1_live'),
9.     html.Br(),
10.    dcc.Link('Gráfica en tiempo real de fase 2 (V2 y A2)',
href='/phase2_live'),
11.     html.Br(),
12.    dcc.Link('Gráfica en tiempo real de fase 3 (V3 Y A3)',
href='/phase3_live'),
13.     html.Br(),
14.     dcc.Link('Visualizar FFT de cada componente', href='/fft_viewer'),
15.     html.Br(),
16.     dcc.Link('Cargar tablas de datos', href='/table_viewer')
17. ])
```

A.3.4 apps/phase1_live.py

```
1. import datetime, dash, plotly, glob, os
2. from dash.dependencies import Input, Output
3. import dash_core_components as dcc
4. import dash_html_components as html
5. from app import app
6. import pandas as pd
7. import numpy as np

8. # Definiciones
```

```
FREC_MUESTREO = 6000
FREC_VOLTAJE = 50
MUESTRAS_POR_PERIODO = FREC_MUESTREO/FREC_VOLTAJE

# Variables globales
factor_pot,Vrms,Arms = 0,0,0

layout = html.Div(

    html.Div([
        html.H4('Voltaje e intensidad de la fase 1'),
        html.Div(id='live-update-text-phase1'),
       dcc.Graph(id='live-update-graph-phase1'),
       dcc.Interval(
            id='interval-component-phase1',
            interval=1*1000,    # En milisegundos
            n_intervals=0
        ),

        html.Br(),

        # Botones con hipervínculos
        html.A(html.Button('Volver a índice'),href='/'),
        html.A(html.Button('Ver fase 2'),href='/phase2_live'),
        html.A(html.Button('Ver fase 3'),href='/phase3_live'),

    ]),
)

# Callback encargado de actualizar el texto con las variables de interés
@app.callback(Output('live-update-text-phase1', 'children'),
              [Input('interval-component-phase1', 'n_intervals')])
def update_metrics(n):
    Volt=Vrms
    Amp=Arms
    PotApa=Vrms*Arms
    PotAct=PotApa*factor_pot
    Fact=factor_pot

    style = {'padding': '5px', 'fontSize': '16px'}

    return [html.Span('Voltaje eficaz: {0:.2f} V'.format(Volt),
style=style),
    html.Span('Intensidad eficaz: {0:.2f} A'.format(Amp), style=style),
    html.Span('Potencia aparente: {0:0.2f} VA'.format(PotApa), style=style),
    html.Span('Potencia activa: {0:0.2f} W'.format(PotAct), style=style),
    html.Span('Factor de potencia: {0:0.2f}'.format(Fact), style=style)]
```

```
54.# Callback encargado de actualizar las gráficas y calcular los datos de
interés
55. # a partir de los datos guardados en *.csv
56. @app.callback(Output('live-update-graph-phase1', 'figure'),
57.               [Input('interval-component-phase1', 'n_intervals')])
58. def update_graph_live(n):
59.     n=n+1
60.     sumV,sumA=0,0
61.     lastA,lastV=0,0
62.     cont=0

63.     data = {
64.         'Muestra': [],
65.         'Voltaje': [],
66.         'Intensidad': []
67.     }

68.     # Abrimos el último archivo creado en la carpeta log y lo enlazamos a la
69.     # variable df
70.     list_of_files = glob.glob('log/*')
71.     latest_file = max(list_of_files, key=os.path.getctime)
72.     df = pd.read_csv(latest_file)
73.     last_row=len(df.index)

74.     # Si vemos que alcanzamos el final del fichero y no hay uno nuevo
volvemos
75.     # a la primera muestra
76.     if last_row<(1000+1000*n):
77.         n=0

78.     # Recogemos los datos del fichero .csv correspondientes a V1 y A1
79.     for i in range(1000):  # Mostramos los valores de mil en mil muestras
80.         muestra=i+1000*n
81.         data['Muestra'].append(i+1000*n)

82.         # Vamos recogiendo datos de la columna V1 y los metemos en la graf.
83.         V1=df['V1'][i+1000*n]
84.         data['Voltaje'].append(V1)

85.         # Si el voltaje cruza el 0, guardamos ese numero de muestra para
compararlo
86.         # con el del amperaje y obtener el factor de potencia
87.         if V1>0 and lastV<0:
88.             muestrainitV=muestra
89.             cont=1
90.         lastV=V1

91.         # Vamos recogiendo datos de la columna A1 y los metemos en la graf.
92.         A1=df['A1'][i+1000*n]
```

```
93.         data['Intensidad'].append(A1)

94.         # Si el amperaje cruza el 0, guardamos ese numero de muestra y
calculamos el
95.         # factor de potencia
96.         if A1>0 and lastA<0 and cont>0:
97.             muestrainitA=muestra
98.             difmuestra=int(muestrainitA)-int(muestrainitV)
99.             global factor_pot
100.            factor_pot=difmuestra/MUESTRAS_POR_PERIODO
101.        lastA=A1


102.        # Valor cuadrático medio
103.        # 1 - Elevamos al cuadrado el valor de las muestras
104.        sqV=float(V1)*float(V1)
105.        sqA=float(A1)*float(A1)
106.        # 2 - Sumamos
107.        sumV+=sqV
108.        sumA+=sqA

109.    #3 - Obtenemos el valor rms
110.    global Vrms, Arms
111.    Vrms=np.sqrt(sumV/1000)
112.    Arms=np.sqrt(sumA/1000)

113.    # Creamos una gráfica con dos subplots (subgráficas)
114.    fig = plotly.tools.make_subplots(rows=2, cols=1, vertical_spacing=0.2)
115.    fig['layout']['margin'] = {
116.        'l': 30, 'r': 10, 'b': 30, 't': 10
117.    }
118.    fig['layout']['legend'] = {'x': 0, 'y': 1, 'xanchor': 'left'}

119.    # En la primera mostramos en el eje x el número de muestra
120.    # y en el eje y el valor del voltaje
121.    fig.append_trace({
122.        'x': data['Muestra'],
123.        'y': data['Voltaje'],
124.        'name': 'Voltaje (V1)',
125.        'mode': 'lines+markers',
126.        'type': 'scatter'
127.    }, 1, 1)

128.    # En la segunda mostramos en el eje x el número de muestra
129.    # y en el eje y el valor del amperaje
130.    fig.append_trace({
131.        'x': data['Muestra'],
132.        'y': data['Intensidad'],
133.        'name': 'Intensidad (A1)',
134.        'mode': 'lines+markers',
```

```
        'type': 'scatter'
    }, 2, 1)

    return fig
```

A.3.5 apps/phase2_live.py

```
import datetime, dash, plotly, glob, os
from dash.dependencies import Input, Output
import dash_core_components as dcc
import dash_html_components as html
from app import app
import pandas as pd
import numpy as np

# Definiciones
FREC_MUESTREO = 6000
FREC_VOLTAJE = 50
MUESTRAS_POR_PERIODO = FREC_MUESTREO/FREC_VOLTAJE

# Variables globales
factor_pot,Vrms,Arms = 0,0,0

layout = html.Div(

    html.Div([
        html.H4('Voltaje e intensidad de la fase 1'),
        html.Div(id='live-update-text-phase1'),
       dcc.Graph(id='live-update-graph-phase1'),
       dcc.Interval(
            id='interval-component-phase1',
            interval=1*1000,    # En milisegundos
            n_intervals=0
        ),

        html.Br(),

        # Botones con hipervínculos
        html.A(html.Button('Volver a índice'),href='/'),
        html.A(html.Button('Ver fase 2'),href='/phase2_live'),
        html.A(html.Button('Ver fase 3'),href='/phase3_live'),

    ]),
)

# Callback encargado de actualizar el texto con las variables de interés
```

```
@app.callback(Output('live-update-text-phase1', 'children'),
              [Input('interval-component-phase1', 'n_intervals')])
def update_metrics(n):
    Volt=Vrms
    Amp=Arms
    PotApa=Vrms*Arms
    PotAct=PotApa*factor_pot
    Fact=factor_pot

    style = {'padding': '5px', 'fontSize': '16px'}

    return [html.Span('Voltaje eficaz: {0:.2f} V'.format(Volt),
style=style),
    html.Span('Intensidad eficaz: {0:.2f} A'.format(Amp), style=style),
    html.Span('Potencia aparente: {0:0.2f} VA'.format(PotApa), style=style),
    html.Span('Potencia activa: {0:0.2f} W'.format(PotAct), style=style),
    html.Span('Factor de potencia: {0:0.2f}'.format(Fact), style=style)]


# Callback encargado de actulizar las gráficas y calcular los datos de
interés
# a partir de los datos guardados en *.csv
@app.callback(Output('live-update-graph-phase1', 'figure'),
              [Input('interval-component-phase1', 'n_intervals')])
def update_graph_live(n):
    n=n+1
    sumV,sumA=0,0
    lastA,lastV=0,0
    cont=0

    data = {
        'Muestra': [],
        'Voltaje': [],
        'Intensidad': []
    }

    # Abrimos el último archivo creado en la carpeta log y lo enlazamos a la
    # variable df
    list_of_files = glob.glob('log/*')
    latest_file = max(list_of_files, key=os.path.getctime)
    df = pd.read_csv(latest_file)
    last_row=len(df.index)

    # Si vemos que alcanzamos el final del fichero y no hay uno nuevo
volvemos
    # a la primera muestra
    if last_row<(1000+1000*n):
        n=0

    # Recogemos los datos del fichero .csv correspondientes a V1 y A1
```

```
79.     for i in range(1000):  # Mostramos los valores de mil en mil muestras
80.         muestra=i+1000*n
81.         data['Muestra'].append(i+1000*n)

82.         # Vamos recogiendo datos de la columna V1 y los metemos en la graf.
83.         V1=df['V1'][i+1000*n]
84.         data['Voltaje'].append(V1)

85.         # Si el voltaje cruza el 0, guardamos ese número de muestra para
compararlo
86.         # con el del amperaje y obtener el factor de potencia
87.         if V1>0 and lastV<0:
88.             muestrainitV=muestra
89.             cont=1
90.         lastV=V1

91.         # Vamos recogiendo datos de la columna A1 y los metemos en la graf.
92.         A1=df['A1'][i+1000*n]
93.         data['Intensidad'].append(A1)

94.         # Si el amperaje cruza el 0, guardamos ese número de muestra y
calculamos el
95.         # factor de potencia
96.         if A1>0 and lastA<0 and cont>0:
97.             muestrainitA=muestra
98.             difmuestra=int(muestrainitA)-int(muestrainitV)
99.             global factor_pot
100.            factor_pot=difmuestra/MUESTRAS_POR_PERIODO
101.        lastA=A1


102.        # Valor cuadrático medio
103.        # 1 - Elevamos al cuadrado el valor de las muestras
104.        sqV=float(V1)*float(V1)
105.        sqA=float(A1)*float(A1)
106.        # 2 - Sumamos
107.        sumV+=sqV
108.        sumA+=sqA

109.    #3 - Obtenemos el valor rms
110.    global Vrms, Arms
111.    Vrms=np.sqrt(sumV/1000)
112.    Arms=np.sqrt(sumA/1000)

113.    # Creamos una gráfica con dos subplots (subgráficas)
114.    fig = plotly.tools.make_subplots(rows=2, cols=1, vertical_spacing=0.2)
115.    fig['layout']['margin'] = {
116.        'l': 30, 'r': 10, 'b': 30, 't': 10
117.    }
```

```
    fig['layout']['legend'] = {'x': 0, 'y': 1, 'xanchor': 'left'}

    # En la primera mostramos en el eje x el número de muestra
    # y en el eje y el valor del voltaje
    fig.append_trace({
        'x': data['Muestra'],
        'y': data['Voltaje'],
        'name': 'Voltaje (V1)',
        'mode': 'lines+markers',
        'type': 'scatter'
    }, 1, 1)

    # En la segunda mostramos en el eje x el número de muestra
    # y en el eje y el valor del amperaje
    fig.append_trace({
        'x': data['Muestra'],
        'y': data['Intensidad'],
        'name': 'Intensidad (A1)',
        'mode': 'lines+markers',
        'type': 'scatter'
    }, 2, 1)

    return fig
```

A.3.6 apps/phase3_live.py

```
import datetime, dash, plotly, glob, os
from dash.dependencies import Input, Output
import dash_core_components as dcc
import dash_html_components as html
from app import app
import pandas as pd
import numpy as np

# Definiciones
FREC_MUESTREO = 6000
FREC_VOLTAJE = 50
MUESTRAS_POR_PERIODO = FREC_MUESTREO/FREC_VOLTAJE

# Variables globales
factor_pot,Vrms,Arms = 0,0,0

layout = html.Div(

    html.Div([
        html.H4('Voltaje e intensidad de la fase 1'),
        html.Div(id='live-update-text-phase1'),
```

```
        dcc.Graph(id='live-update-graph-phase1'),
        dcc.Interval(
             id='interval-component-phase1',
             interval=1*1000,    # En milisegundos
             n_intervals=0
         ),

         html.Br(),

         # Botones con hipervínculos
         html.A(html.Button('Volver a índice'),href='/'),
         html.A(html.Button('Ver fase 2'),href='/phase2_live'),
         html.A(html.Button('Ver fase 3'),href='/phase3_live'),

    ]),
)

# Callback encargado de actualizar el texto con las variables de interés
@app.callback(Output('live-update-text-phase1', 'children'),
              [Input('interval-component-phase1', 'n_intervals')])
def update_metrics(n):
    Volt=Vrms
    Amp=Arms
    PotApa=Vrms*Arms
    PotAct=PotApa*factor_pot
    Fact=factor_pot

    style = {'padding': '5px', 'fontSize': '16px'}

    return [html.Span('Voltaje eficaz: {0:.2f} V'.format(Volt),
style=style),
    html.Span('Intensidad eficaz: {0:.2f} A'.format(Amp), style=style),
    html.Span('Potencia aparente: {0:0.2f} VA'.format(PotApa), style=style),
    html.Span('Potencia activa: {0:0.2f} W'.format(PotAct), style=style),
    html.Span('Factor de potencia: {0:0.2f}'.format(Fact), style=style)]


# Callback encargado de actualizar las gráficas y calcular los datos de
interés
# a partir de los datos guardados en *.csv
@app.callback(Output('live-update-graph-phase1', 'figure'),
              [Input('interval-component-phase1', 'n_intervals')])
def update_graph_live(n):
    n=n+1
    sumV,sumA=0,0
    lastA,lastV=0,0
    cont=0

    data = {
```

```
64.         'Muestra': [],
65.         'Voltaje': [],
66.         'Intensidad': []
67.     }

68.     # Abrimos el ultimo archivo creado en la carpeta log y lo enlazamos a la
69.     # variable df
70.     list_of_files = glob.glob('log/*')
71.     latest_file = max(list_of_files, key=os.path.getctime)
72.     df = pd.read_csv(latest_file)
73.     last_row=len(df.index)

74.     # Si vemos que alcanzamos el final del fichero y no hay uno nuevo
volvemos
75.     # a la primera muestra
76.     if last_row<(1000+1000*n):
77.         n=0

78.     # Recogemos los datos del fichero .csv correspondientes a V1 y A1
79.     for i in range(1000):  # Mostramos los valores de mil en mil muestras
80.         muestra=i+1000*n
81.         data['Muestra'].append(i+1000*n)

82.         # Vamos recogiendo datos de la columna V1 y los metemos en la graf.
83.         V1=df['V1'][i+1000*n]
84.         data['Voltaje'].append(V1)

85.         # Si el voltaje cruza el 0, guardamos ese número de muestra para
compararlo
86.         # con el del amperaje y obtener el factor de potencia
87.         if V1>0 and lastV<0:
88.             muestrainitV=muestra
89.             cont=1
90.         lastV=V1

91.         # Vamos recogiendo datos de la columna A1 y los metemos en la graf.
92.         A1=df['A1'][i+1000*n]
93.         data['Intensidad'].append(A1)

94.         # Si el amperaje cruza el 0, guardamos ese número de muestra y
calculamos el
95.         # factor de potencia
96.         if A1>0 and lastA<0 and cont>0:
97.             muestrainitA=muestra
98.             difmuestra=int(muestrainitA)-int(muestrainitV)
99.             global factor_pot
100.            factor_pot=difmuestra/MUESTRAS_POR_PERIODO
101.        lastA=A1
```

```
102.        # Valor cuadrático medio
103.        # 1 - Elevamos al cuadrado el valor de las muestras
104.        sqV=float(V1)*float(V1)
105.        sqA=float(A1)*float(A1)
106.        # 2 - Sumamos
107.        sumV+=sqV
108.        sumA+=sqA

109.    #3 - Obtenemos el valor rms
110.    global Vrms, Arms
111.    Vrms=np.sqrt(sumV/1000)
112.    Arms=np.sqrt(sumA/1000)

113.    # Creamos una gráfica con dos subplots (subgráficas)
114.    fig = plotly.tools.make_subplots(rows=2, cols=1, vertical_spacing=0.2)
115.    fig['layout']['margin'] = {
116.        'l': 30, 'r': 10, 'b': 30, 't': 10
117.    }
118.    fig['layout']['legend'] = {'x': 0, 'y': 1, 'xanchor': 'left'}

119.    # En la primera mostramos en el eje x el numero de muestra
120.    # y en el eje y el valor del voltaje
121.    fig.append_trace({
122.        'x': data['Muestra'],
123.        'y': data['Voltaje'],
124.        'name': 'Voltaje (V1)',
125.        'mode': 'lines+markers',
126.        'type': 'scatter'
127.    }, 1, 1)

128.    # En la segunda mostramos en el eje x el número de muestra
129.    # y en el eje y el valor del amperaje
130.    fig.append_trace({
131.        'x': data['Muestra'],
132.        'y': data['Intensidad'],
133.        'name': 'Intensidad (A1)',
134.        'mode': 'lines+markers',
135.        'type': 'scatter'
136.    }, 2, 1)

137.    return fig
```

A.3.7 apps/fft_viewer.py

```
import dash, flask, glob, os
import dash_core_components as dcc
import dash_html_components as html
from app import app

# Directorios donde se guardan los .png con las FFTs para acceder a ellas
directories = ['fft/V1/','fft/A1/','fft/V2/','fft/A2/','fft/V3/','fft/A3/']
static_image_route = '/static/'

layout = html.Div([

    html.H4('Visor de FFTs'),
    html.Div('Selecciona una componente para analizar sus armónicos:'),

    # Checkboxes que habilitan los correspondientes .pngs en la
    # lista desplegable
    dcc.RadioItems(
    id='radio',
    options=[
        {'label': 'V1', 'value': '0'},
        {'label': 'A1', 'value': '1'},
        {'label': 'V2', 'value': '2'},
        {'label': 'A2', 'value': '3'},
        {'label': 'V3', 'value': '4'},
        {'label': 'A3', 'value': '5'}
    ],
    value='0',
    labelStyle={'display': 'inline-block'}
    ),

    # Lista desplegable con las imágenes a mostrar
    dcc.Dropdown(
        id='image-dropdown'
    ),

    # Imagen a mostrar
    html.Img(id='image',style={'width': '2000px','margin':' 0px 0px 0px -
200px'}),

    html.Br(),

    # Botón que redirige a pagina principal
    html.A(html.Button('Volver a índice'),href='/')

])

# Callback que se encarga de mostrar la imagen cuando se selecciona una
# en la lista desplegable
```

```
@app.callback(
    dash.dependencies.Output('image', 'src'),
    [dash.dependencies.Input('image-dropdown', 'value')])
def update_image_src(value):
    return static_image_route + value



# Muestra imagen usando ruta estática
@app.server.route('{}<image_path>.png'.format(static_image_route))
def serve_image(image_path):
    image_name = '{}.png'.format(image_path)
    if image_name not in list_of_images:
        raise Exception('"{}" is excluded from the allowed static
files'.format(image_path))
    return flask.send_from_directory(image_directory, image_name)


# Callback que muestra los distintos ficheros a seleccionar
# en la lista desplegable dependiendo del checkbox
@app.callback(
    dash.dependencies.Output('image-dropdown', 'options'),
    [dash.dependencies.Input('radio', 'value')])
def set_cities_options(value):
    global image_directory, list_of_images
    image_directory = directories[int(value)]
    list_of_images = [os.path.basename(x) for x in
sorted(glob.glob('{}*.png'.format(image_directory)), key=os.path.getmtime)]
    return  [{'label': i, 'value': i} for i in list_of_images]

# Callback que ayuda al callback anterior al mostrar los ficheros
# a elegir
@app.callback(
    dash.dependencies.Output('image-dropdown', 'value'),
    [dash.dependencies.Input('image-dropdown', 'options')])
def set_cities_value(available_options):
    global list_of_images
    return list_of_images[0]
```

A.3.8 apps/table_viewer.py

```
import datetime, glob, os
from app import app
import dash, dash_table
from dash.dependencies import Input, Output
import dash_core_components as dcc
import dash_html_components as html
import pandas as pd

 # Directorio donde se guardan los .csv
 table_directory = 'log/'
list_of_tables = [os.path.basename(x) for x in
sorted(glob.glob('{}*.csv'.format(table_directory)))]


layout = html.Div(

    html.Div(

     className="row",

     children=[

         html.H4('Visor de los datos obtenidos'),
         html.Div('Selecciona un archivo para ver los datos:'),

         # Lista desplegable con los diferentes archivos
         dcc.Dropdown(
         id='my-dropdown',
         options=[{'label': i, 'value': i} for i in list_of_tables],
         value=list_of_tables[0]),

         html.Div(id='output-container'),
         html.Br(),

         html.Div(

             # Tabla que se muestra en la zona izquierda de la pantalla
             dash_table.DataTable(
                 id='table-paging-with-graph',
                 pagination_settings={
                     'current_page': 0,
                     'page_size': 1000   #Max. de datos mostrados por página
en la tabla
                 },

                 # Opciones de personalización de la tabla
                 pagination_mode='be',
                 filtering='be',
```

```
                filtering_settings='',
                sorting='be',
                sorting_type='multi',
                sorting_settings=[]
            ),

            # Altura en px. que ocupa la tabla: 1500
            style={'height': 1500, 'overflowY': 'scroll'},
            className='six columns' # Ancho equivalente de la tabla
        ),

        html.Div(

            # Gráficos a mostrar en la parte derecha de la pantalla
            id='table-paging-with-graph-container',
            className="five columns" # Ancho equivalente del campo
        ),

        html.Br(),

    # Botón que redirige al indice
    html.A(html.Button('Volver a índice'), href='/')
    ]),
)


# Callback que se encarga de mostrar los datos en la tabla de la izquierda
# y de realizar las funciones disponibles de ordenar de mayor a menor o
viceversa
# y filtrar por dato
@app.callback(
    Output('table-paging-with-graph', "data"),
    [Input('table-paging-with-graph', "pagination_settings"),
    Input('table-paging-with-graph', "sorting_settings"),
     Input('table-paging-with-graph', "filtering_settings"),
     Input('table-paging-with-graph', 'columns')])
def update_table(pagination_settings, sorting_settings, filtering_settings,
empty):

    filtering_expressions = filtering_settings.split(' && ')
    dff = df

    for filter in filtering_expressions:
        if ' eq ' in filter:
            col_name = filter.split(' eq ')[0]
            filter_value = filter.split(' eq ')[1]
            dff = dff.loc[dff[col_name] == filter_value]
        if ' > ' in filter:
            col_name = filter.split(' > ')[0]
```

```
            filter_value = float(filter.split(' > ')[1])
            dff = dff.loc[dff[col_name] > filter_value]
        if ' < ' in filter:
            col_name = filter.split(' < ')[0]
            filter_value = float(filter.split(' < ')[1])
            dff = dff.loc[dff[col_name] < filter_value]

    if len(sorting_settings):
        dff = dff.sort_values(
            [col['column_id'] for col in sorting_settings],
            ascending=[
                col['direction'] == 'asc'
                for col in sorting_settings
            ],
            inplace=False
        )

    return dff.iloc[
    pagination_settings['current_page']*pagination_settings['page_size']:
        (pagination_settings['current_page'] +
1)*pagination_settings['page_size']
    ].to_dict('rows')


# Callback que actualiza las gráficas de la parte derecha de la pantalla
@app.callback(
    Output('table-paging-with-graph-container', "children"),
    [Input('table-paging-with-graph', "data"),
    Input('output-container', 'children')])
def update_graph(rows, empty):

    dff = pd.DataFrame(rows)

    return html.Div(
        [
            dcc.Graph(
                id=column,
                figure={
                    "data": [
                        {
                           # El eje x se entiende como número de muestra
                            # por defecto
                            "y": dff[column] if column in dff else [],
                            "type": "bar",
                            'mode': 'lines+markers',
                            'type': 'scatter',
                            "marker": {"color": "#0074D9"},
                        }
                    ],

```

```
                    # Aquí ponemos las opciones de visualización: título,
altura, etc
                    # de cada una de las gráficas
                    "layout": {
                    'title': column,
                    "xaxis": {"automargin": True},
                    "yaxis": {"automargin": True},
                    "height": 250,
                    "margin": {"t": 50, "l": 50, "r": 10},
                    },

                },
            )
            # Recoge los datos de los campos aqui designados en el .csv
            for column in ["V1", "A1", "V2", "A2", "V3", "A3"]
        ]
    )

# Callback que se encarga de mostrar los diferentes ficheros en
# la lista desplegable y guardar el descriptor de fichero del archivo
# seleccionado en la variable global df
@app.callback(
    dash.dependencies.Output('table-paging-with-graph', 'columns'),
    [dash.dependencies.Input('my-dropdown', 'value')])
def update_output(value):
    file_to_open=table_directory+value
    global df
    df = pd.read_csv(file_to_open)
    return [{"name": i, "id": i} for i in sorted(df.columns)]
```

A.4 Script de inicio

```
# Ejecuta el servidor dash y el script de toma de
# toma de datos del Arduino en background
. env/bin/activate
python3 data.py &
python3 index.py &
```

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

REFERENCIAS

[1] "History of computing," [Online]. Available: https://en.wikipedia.org/wiki/History_of_computing.

[2] "What is Arduino," [Online]. Available: https://www.arduino.cc/en/Guide/Introduction.

[3] «Raspberry Pi Foundation,» [En línea]. Available: https://www.raspberrypi.org/about/.

[4] «El internet de las cosas,» [En línea]. Available: https://www.cerem.es/blog/cambio-de-paradigma-el-internet-de-las-cosas-iot.

[5] «Industria 4.0,» [En línea]. Available: https://www2.deloitte.com/es/es/pages/manufacturing/articles/que-es-la-industria-4.0.html.

[6] "MKR 1010," [Online]. Available: https://store.arduino.cc/arduino-mkr-wifi-1010.

[7] «ARM Cortex M0+,» [En línea]. Available: https://developer.arm.com/products/processors/cortex-m/cortex-m0-plus.

[8] "ESP32," [Online]. Available: https://www.espressif.com/en/products/hardware/esp32/overview.

[9] "ATmega328p," [Online]. Available: https://www.microchip.com/wwwproducts/en/ATmega328p.

[10] «SAM21D datasheet,» [En línea]. Available: https://cdn.sparkfun.com/datasheets/Dev/Arduino/Boards/Atmel-42181-SAM-D21_Datasheet.pdf.

[11] "Transformador AC-AC," [Online]. Available: https://www.kreco.com.cn/products/9VAC_1000mA_AC-AC_adapter.html.

[12] «SCT013 datasheet,» [En línea]. Available: https://www.mcielectronics.cl/website_MCI/static/documents/Datasheet_SCT013.pdf.

[13] «Arduino software,» [En línea]. Available: https://www.arduino.cc/en/main/software.

[14] "Fast 10 bit ADC," [Online]. Available: https://www.avdweb.nl/arduino/adc-dac/fast-10-bit-adc.

[15] «Descarga Raspbian,» [En línea]. Available: https://www.raspberrypi.org/downloads/raspbian/.

[16] «Dash by plotly,» [En línea]. Available: https://plot.ly/products/dash/.

[17] «Arduino pinout,» [En línea]. Available: https://github.com/arduino/ArduinoCore-samd/blob/master/variants/mkrwifi1010/variant.cpp.

[18] «Power Disaggregation,» [En línea]. Available: https://lids.mit.edu/research/research-highlights/power-disaggregation.

[19] «DFT,» [En línea]. Available: https://en.wikipedia.org/wiki/Discrete_Fourier_transform.

[20] «FFT,» [En línea]. Available: https://en.wikipedia.org/wiki/Fast_Fourier_transform.

[21] «Ventajas del enventanado,» [En línea]. Available: http://www.ni.com/white-paper/4844/es/.

[22] «Kaiser,» [En línea]. Available: https://en.wikipedia.org/wiki/Kaiser_window.

[23] «Numpy's Kaiser reference,» [En línea]. Available: https://het.as.utexas.edu/HET/Software/Numpy/reference/generated/numpy.kaiser.html.

[24] «ADC explained,» [En línea]. Available: https://en.wikipedia.org/wiki/Analog-to-digital_converter.