

FACULTAD DE CIENCIAS



Proyecto Fin de Carrera

Departamento de Arquitectura de Computadores.

Proyecto de Fin de Carrera de Ingeniería Informática

Diseño, desarrollo y explotación de una solución basada en SBC (Raspberry Pi) para la sustitución del instrumental de los laboratorios de física de la Facultad de Ciencias.

Design, development and operation of an SBC-based solution (Raspberry Pi) for instrumental replacement in the Facultad de Ciencias laboratories.

Para acceder al título de

INGENIERO EN INFORMÁTICA

Autor: Diego Muñoz Callejo

Co-Directores: Rafael Menéndez de Llano Rozas

Esteban Stafford Fernández

Santander, 8 de septiembre de 2014

Agradecimientos

Agradecer en primer lugar a mi familia por el apoyo recibido, a mis co-directores Rafa y Esteban por la imprescindible ayuda y material prestado, y a mi compañero de proyecto paralelo Noel Díaz por poner todas las facilidades posibles estando ambos metidos en el mismo aprieto.

A Rafael Valiente Barroso y a Ángel Mañanes Pérez por la disposición de los laboratorios cuando hemos necesitado recabar información o hacer pruebas y a Mercedes Granda Miguel y Gustavo Ruíz Robredo por el material prestado para que esto pudiera llegar a alguna parte.

Resumen

Actualmente algunos laboratorios de Física de la Facultad de Ciencias tienen problemas de mantenimiento, bien porque se necesita instrumental obsoleto y en extinción por cuestiones de compatibilidad con elementos de difícil y costoso reemplazo o bien porque su coste es relativamente alto como para ser sustituido. Actualmente una manera de abordar este tipo de problemas es con SBC (single board computer). El más destacado miembro, junto con Arduino, de este nuevo club de ordenadores de placa única es el Raspberry Pi.

En este PFC se analizan las necesidades de estos laboratorios, se propone y diseña una solución software basada en Raspberry Pi apoyada en un hardware creado en un proyecto paralelo y se realizan las pruebas y evaluaciones necesarias así como un análisis de los requisitos y expectativas de explotar la solución de forma industrial.

Palabras clave.

Single board computer, SBC, Raspberry Pi, Python, Linux, ARM.

Abstract

Some laboratories in the Physics department at the Facultad de Ciencias have nowadays maintenance problems due to old and obsolete equipment, which is difficult to replace because of its high cost. One of the many ways to face this kind of problems is with the use of SBCs (single-board computers), the best known amongst them being the Raspberry Pi.

In this End of Degree project we analyse the needs of these laboratories, propose and design a software solution based on the Raspberry Pi and supported by a hardware created in a parallel project. The solution is tested and evaluated altogether, and analysed for potential commercial exploitation.

Keywords.

Single board computer, SBC, Raspberry Pi, Python, Linux, ARM.

Índice general

1. Introducción.	1
1.1. Introducción y objetivos.	1
1.1.1. Objetivos.	1
1.1.2. Proyecto de Fin de Carrera Paralelo	2
1.1.3. Estructura de la memoria.	2
1.2. Material y métodos.	3
1.2.1. Antecedentes.	3
1.2.2. Estado del arte.	4
1.2.3. Material y herramientas.	5
1.2.4. Metodología.	5
2. Presentación del problema y análisis de requisitos.	7
2.1. Problemas de la solución actual.	7
2.2. Análisis de requisitos.	8
2.3. Solución propuesta.	9
2.3.1. Sistema base.	9
2.3.2. Software y sistema operativo.	11
3. Diseño e implementación de la solución.	13
3.1. Punto de encuentro de ambos proyectos paralelos.	13
3.1.1. Especificación del interfaz de control de adquisición de datos piDA. 13	13
3.2. Primeras fases del desarrollo.	15
3.2.1. Creando el sistema base.	15

3.2.2. Eligiendo el entorno de desarrollo.	17
3.2.3. Eligiendo el framework gráfico.	18
3.2.4. Eligiendo una librería de representación gráfica.	19
3.3. Consideraciones a tener en cuenta antes de comenzar.	20
3.3.1. Dos soluciones muy distintas.	20
3.3.2. El método de trabajo a utilizar.	21
4. Diseño de la interfaz gráfica.	23
4.1. Disposición de los elementos en la ventana principal.	23
4.1.1. Los dos tercios superiores.	23
4.1.2. Tercio inferior de la ventana principal.	24
4.2. Resultado final.	28
5. Diseño e implementación del modelo.	29
5.1. Modulos que forman el programa.	29
5.1.1. Módulo <i>classes</i>	30
5.1.2. Módulo <i>GUI</i>	31
5.1.3. Módulo <i>utils</i>	32
5.2. Lógica de control de la gráfica.	33
5.2.1. Actualización de los datos.	33
5.2.2. Actualización automática.	35
5.3. El simulador.	36
5.3.1. Lógica de funcionamiento.	37
5.4. Los distintos hilos de ejecución.	38
5.5. Otras funciones integradas: Instalador.	38
6. Los prototipos y el interfaz piDA.	41
6.1. El interfaz piDA.	41
6.2. El primer prototipo.	42
6.2.1. Implementación en el programa.	43
6.2.2. Utilización del prototipo.	43
6.3. Segundo prototipo.	45

6.3.1. Implementación y utilización en el programa.	46
7. Evaluación y pruebas.	47
7.1. Descripción de las pruebas.	47
7.1.1. Carga del sistema.	49
7.1.2. ¿Cuándo una medida no es válida y el sistema ha tocado su límite?	49
7.1.3. ¿De dónde puede proceder el límite?	49
7.2. Pruebas con un solo canal.	50
7.2.1. Cargas del sistema en cada caso.	51
7.3. Pruebas con dos canales simultáneamente.	52
7.3.1. Cargas del sistema en cada caso.	53
7.4. Pruebas con cuatro canales simultáneamente.	54
7.4.1. Cargas del sistema en cada caso.	55
7.5. Puesta en común de los resultados y conclusión.	57
7.5.1. El límite.	57
7.5.2. El porqué del límite.	57
8. Conclusiones y trabajos futuros.	59
8.1. Conclusiones personales.	59
8.2. Conclusiones profesionales.	60
8.3. Trabajos futuros.	60
8.3.1. Nuevas funcionalidades..	60
8.3.2. Optimización de la solución.	60
8.3.3. Adaptación a otras SBCs.	61
8.3.4. Integración del producto final en los laboratorios de la Facultad.	61
8.3.5. Explotación comercial.	61

Índice de figuras

2.1. Escritorio de Raspbian.	12
3.1. Adaptador HDMI a VGA.	16
3.2. Ejemplos de Matplotlib.	20
4.1. El conjunto de la gráfica y sus controles en nuestro programa.	24
4.2. Pestaña para controlar las adquisiciones de datos.	25
4.3. Pestaña de visualización de datos.	26
4.4. Pestaña de exportación de datos.	27
4.5. El interfaz gráfico de nuestro programa.	28
5.1. Diagrama de clases y sus dependencias..	30
5.2. Diagrama de envío de datos a la gráfica.	34
5.3. Diagrama de refresco selectivo de la gráfica.	35
5.4. Función de encendido o apagado del refresco automático de la gráfica.	36
5.5. Función de encendido o apagado del refresco automático de la gráfica.	37
5.6. Diagrama de sincronización entre hilos.	38
6.1. El primer prototipo conectado a la Raspberry Pi.	42
6.2. Funcionamiento del primer prototipo de dos canales.	44
6.3. El segundo prototipo conectado a la Raspberry Pi.	45
6.4. Funcionamiento del segundo prototipo con cuatro canales.	46

Índice de tablas

2.1. Los diferentes SBC comparados.	10
3.1. Desglose de coste de la solución propuesta.	17
7.1. Cargas máximas en el sistema sin utilizar el interfaz de adquisición. . . .	50
7.2. Resultados de los tests utilizando un solo canal.	51
7.3. Comparación del error con la carga del sistema al utilizar un solo canal.	51
7.4. Resultados de los tests utilizando dos canales.	52
7.5. Comparación del error con la carga del sistema al utilizar dos canales. .	53
7.6. Resultados de los tests utilizando los cuatro canales.	55
7.7. Comparación del error con la carga del sistema al utilizar los cuatro canales.	56
7.8. Frecuencias máximas obtenidas por cada canal.	57

“Hay gente que cree que odia los ordenadores. Lo que odian en realidad son los malos programadores”

Larry Niven, escritor.

Capítulo 1

Introducción.

Como primer paso a la hora de abordar un proyecto se debe analizar el contexto en el que se va a trabajar y establecer unos objetivos que definan qué se quiere conseguir. En este capítulo se tratan estos temas fundamentales así como los materiales y métodos usados durante el desarrollo del mismo.

1.1. Introducción y objetivos.

El alto coste de reemplazo y/o actualización de material académico en estos tiempos de crisis económica es algo que no se puede permitir cualquier Universidad. La obsolescencia, desgaste y avería de los mismos suponen un reto económico que con este proyecto queremos reducir mediante el uso de hardware de bajo coste y una solución personalizada a los requerimientos de cada laboratorio. Este proyecto se centra en los laboratorios de radiactividad, y termodinámica y física aplicada, donde en el primero disponen de material obsoleto y de muy cara modernización y en el segundo material moderno pero de un coste relativamente desorbitado.

1.1.1. Objetivos.

El objetivo principal ha sido diseñar y construir una solución única para ambos laboratorios manteniendo un coste realmente bajo, demostrando así que no es necesario un desembolso exagerado para la docencia. Para ello es inevitable el recorte de las

características de los actuales equipos y diseñar algo menos ambicioso pero que cumpla con unos mínimos requisitos que no son ni más ni menos que los necesarios para realizar las diferentes prácticas de los alumnos durante el curso.

1.1.2. Proyecto de Fin de Carrera Paralelo

Complementario al software desarrollado en este proyecto, un hardware adecuado es necesario para recibir, procesar y digitalizar las señales analógicas que formarán la verdadera utilidad del software. Este hardware se desarrolla en otro Proyecto de Fin de Carrera paralelo a éste en mutua colaboración. De esta manera, mientras en este Proyecto se realizará el software necesario para el control de la adquisición de datos, almacenamiento, tratamiento y muestra de los mismos así como herramientas de análisis, otro proyecto se encarga de la solución hardware en la que se apoyará fundamentalmente este software recibiendo las distintas señales, adaptándolas y disponiéndolas adecuadamente para el software.

1.1.3. Estructura de la memoria.

A continuación se detalla la división de la memoria en capítulos y una breve descripción de cada uno de ellos:

- Capítulo 1: Sirve de introducción al proyecto, así como a los objetivos y peculiaridades del mismo. En un apartado dedicado se explica la metodología utilizada así como los materiales que se han dispuesto para el desarrollo.
- Capítulo 2: Presenta el problema que trata de solventar este proyecto, se realiza un análisis de requisitos y propone una solución.
- Capítulo 3: Se completa la especificación de la solución así como las herramientas utilizadas en el desarrollo de la misma.
- Capítulo 4: Detalla cómo se ha implementado el interfaz gráfico sección a sección.
- Capítulo 5: Detalla la implementación del modelo y el simulador desarrollado.

- Capítulo 6: El primer prototipo hardware, su integración y funcionamiento en la aplicación.
- Capítulo 7: Batalla de tests sobre el sistema final con objetivo de comprobar y evaluar su rendimiento.
- Capítulo 8: Conclusiones y trabajos futuros inspirados por este proyecto.
- Bibliografía y referencias: Contiene la lista de fuentes consultadas.
- Apéndices.

1.2. Material y métodos.

En esta sección se analiza el estado actual de las áreas a tratar y se detallan las herramientas de las que disponemos para el desarrollo del proyecto así como la metodología que se ha utilizado.

1.2.1. Antecedentes.

En el laboratorio de radiactividad el dispositivo utilizado para la espectroscopía de los elementos es la tarjeta de adquisición de datos *Personal Computer Analyzer* cuyos documentos datan de los 80 y actualmente soporta máximo hasta Windows 98 al estar el software ligado a MS-DOS. Dada a la antigüedad de las placas algunas se están estropeando y su sustitución implica un gasto de más de mil euros (ordenador no incluido). También se cuenta con la dificultad de encontrar ordenadores antiguos que puedan montar tarjetas ISA y la avería de los mismos por su alta antigüedad. Esta tarjeta se encarga de la obtención de pulsos eléctricos procedentes de un amplificador conectado a un fotodetector que detecta pulsos de radiación de una amplitud determinada. El software permite el manejo de los parámetros de obtención de datos y representa la información en un histograma de hasta 1024 canales de intervalo definible.

El dispositivo utilizado en el laboratorio de termodinámica y física aplicada es el datalogger *500 Interface* de la compañía *PASCO*. Este dispositivo es más reciente, de

un coste aproximado de 600 euros ordenador no incluido. La versión para Windows del software tiene ciertas limitaciones, por lo que se utilizan ordenadores de Apple que eximen las medidas de estas limitaciones, lo cual escala el precio de la solución a un nivel mayor. Este datalogger dispone de cinco entradas para cinco sensores -dos digitales, tres analógicos- del mismo ensamblador, buffer interno, conversor analógico digital de 12 bits, una frecuencia de muestreo de 22kHz y es multiplataforma entre sus características más importantes.

1.2.2. Estado del arte.

El mundo de los SBCs es algo relativamente reciente, ya que no ha sido hasta 2012 cuando la Raspberry Pi hizo su aparición en público, revolucionando lo que hasta el momento había sido un monopolio del microcontrolador Arduino, y se dio entonces un nuevo punto de vista al concepto de SBC como el de un ordenador de propósito general.

La Raspberry Pi en principio fue diseñada para la educación en las escuelas británicas de las ciencias de computación pero pronto se convirtió rápidamente en un juguete para profesionales y entusiastas, ya que pese a tener el tamaño de una tarjeta de crédito era capaz de ejecutar eficientemente el sistema operativo Linux y además de disponer unas capacidades multimedia excepcionales, como la posibilidad de reproducir vídeo HD gracias a su decodificador H.264 por hardware.

Algo más de dos años después de su lanzamiento, la placa se sigue vendiendo sin haber recibido ni una mejora en el hardware que la compone y cada vez más gente prueba el mundo de linux, la programación o simplemente se monta su propio centro multimedia por poco más de 30€. La comunidad sigue creciendo día a día y cada vez hay más ingenios creados gracias a ella (sola o en combinación con Arduino).

En este proyecto queremos explorar las posibilidades de la Raspberry Pi en lo que es algo parecido a un desafío: Sustituir material docente del centro con un coste 20 veces superior por tan solo una caja alimentada por un simple cargador de móvil.

1.2.3. Material y herramientas.

Al tratarse de un desarrollo de software, las características de la solución final han sido escasas:

- Monitor Dell de 17" con resolución de hasta 1024x768@85Hz.
- Raspberry Pi Model B Rev.1.0 256MB RAM.
- Tarjeta SDHC Trascend Class 10 de 16GB.
- *HDMI to VGA adapter with power and audio*, adaptador HDMI a VGA de procedencia china.
- Teclado y ratón USB Dell
- Cargador de teléfono móvil Nokia.

Estos podrían ser los requisitos necesarios para el funcionamiento de la solución, muchos de ellos son “reciclados” por lo que el coste es muy bajo.

El desarrollo y testeo del software se ha realizado en distintos entornos por comodidad fuera de la Raspberry Pi: Mac OS X Mavericks, Windows 7 SP1, Debian 7.0 y Raspbian (el objetivo final), por lo que se ha asegurado que el mismo sea portable¹ a estas cuatro plataformas sin problemas al elegir como lenguaje de desarrollo Python 2.7.8.

1.2.4. Metodología.

Se ha usado una metodología estándar para el desarrollo de software:

- Se ha estudiado cual ha sido el contexto para el que se utilizará la nueva solución.
- Se ha observado a los usuarios utilizar las soluciones actuales intentando comprender qué hacían y por qué, preguntando cuando fuere necesario sin interferir en su procedimiento.

¹En las fases finales dónde ha sido necesario el uso del prototipo de interfaz de adquisición de datos el desarrollo se ha cerrado exclusivamente a la Raspberry Pi, aunque su portabilidad a otros sistemas sería cuestión de ajustar varios puntos del código.

- Se han realizado entrevistas con los mismos para recabar objetivos, mejoras y utilidades; es decir, requisitos del proyecto.
- Se realizó una investigación de soluciones, viabilidad y desarrollo de prototipos tanto software como hardware (en el proyecto paralelo).
- Se han llegado a conclusiones y posibles trabajos futuros.

El hecho de que el proyecto haya sido en paralelo con otro ha supuesto un desafío extra ya que esta parte del producto final se apoya fuertemente en la otra. Por tanto se realizó un desarrollo “en puente”, en el cual aquí se desarrolló una parte del mismo y en el otro proyecto la otra parte.

Se tuvieron que acordar modelos, ideas y especificaciones para que ambos proyectos llegaran a encontrarse y las piezas encajaran como un puzzle. La transición fue suave, apenas hubo que modificar código e incluso se añadieron ciertas funcionalidades al ver ambas partes funcionando al unísono, por lo que se puede hablar de éxito en el trabajo conjunto.

En el desarrollo del software se ha utilizado el sistema de versioning ofrecido gratuitamente por *Github.com*. Github ofrece un servicio gratuito de repositorios abiertos y software necesario para utilizarlos además de un útil acceso web. Es el servicio en alza desde hace varios años y para este desarrollo no se ha dudado un instante en utilizarlo dado a las facilidades ofrecidas y la naturaleza *OpenSource* del mismo.

Este proyecto se puede encontrar bajo la dirección: <https://github.com/dmcelectrico/PFC>

“Los problemas son oportunidades para demostrar lo que se sabe.”

Duke Ellington, compositor y músico.

Capítulo 2

Presentación del problema y análisis de requisitos.

En este capítulo se presentan los problemas de las soluciones actuales y la nueva propuesta para solventarlos. También se analizan los requisitos que ha de cumplir la nueva solución para que realmente sea una alternativa viable.

2.1. Problemas de la solución actual.

Ambas soluciones actuales son totalmente funcionales pero no libres de problemas e inconvenientes especialmente económicos que han motivado a la realización de este proyecto. A continuación se enumeran detalladamente para cada una de ellas:

Pasco 500 Interface

- El coste del interfaz es de unos 600€, aunque a esto habría que añadirle el coste del software para su manejo.
- Necesita de un equipo para poder usarse, puede ser un equipo con Windows pero el software para dicho sistema operativo impone ciertas limitaciones que hacen optar al departamento por adquirir equipos Apple, con el sobrecoste añadido.

Nucleus Personal Computer Analyzer II

- El material disponible actualmente está totalmente obsoleto (los documentos más recientes que disponemos datan de 1990).
- El reemplazo de dicho material bien por avería o por obsolescencia tiene un coste de aproximadamente 2000€ y un nuevo computador.
- Los computadores con los que puede funcionar la actual solución han de ser equipos que soporten MS-DOS y tengan puertos ISA para conectar las tarjetas de adquisición. Estos equipos en el mejor de los casos tienen diez años de antigüedad y la mayoría están alcanzando el final de su vida útil.

El principal desafío de este proyecto es realizar una solución cuya diferencia de coste sea muy inferior a las actuales.

2.2. Análisis de requisitos.

Heredado de las soluciones actuales, se pueden definir los siguientes requisitos mínimos para nuestro sistema final.

- Representación de medidas en tiempo real. El software debe proporcionar la posibilidad de leer en tiempo real el estado de las medidas, bien en una lectura digital o en una gráfica temporal.
- Manejo de las medidas. Para poder utilizar las medidas tomadas eficientemente, se deberá proveer la manera de combinar unas con otras, de manera que se puedan tomar medidas compuestas donde dos o más medidas se contrasten (por ejemplo, gráficas Temperatura-Presión). Estas gráficas y tablas deben ser posibles de almacenar para su posterior uso. Por supuesto, estas gráficas deben tener la posibilidad de ampliar, reducir y tomar captura de los datos representados.
- Exportación de datos. Los datos tomados con la aplicación deben poder ser exportados a otros formatos para su manipulación. Por ejemplo en formato de

datos separados por comas (*character-separated values* o CSV) para las tablas y *portable network graphics* (PNG) para las gráficas o histogramas.

- Guardar sesión. El programa debe permitir guardar el estado actual para poder continuar en otro momento o sesión, o incluso para que el alumno pueda en su casa revisar el trabajo hecho en el laboratorio. También es una manera de realizar plantillas para los diversos experimentos que se realizarán.
- Posibilidad de realizar medidas programadas. Esto es, el software se programa para que realice una medida a uno o más sensores durante un tiempo determinado.
- Posibilidad de realizar cálculos con las medidas tomadas. Para ciertos experimentos puede ser interesante realizar automáticamente ciertos cálculos matemáticos más o menos complejos. La automatización de esta tarea puede suponer una ayuda para el alumno y el profesor.

2.3. Solución propuesta.

2.3.1. Sistema base.

Desde que el proyecto era solo una idea, la intención ha sido utilizar una SBC (*Single-board computer*) *Raspberry Pi* para realizar la solución. No obstante, se investigaron otras posibilidades y su viabilidad: La *Raspberry Pi* es un SBC económico pero muy básico y poco potente, por lo que el objetivo del búsqueda era algo más completo y potente.

La *Intel Galileo* fue una de las placas, descartada por no tener conexión a un monitor y porque según opiniones encontradas en internet, no resultaba ser tan potente como *Intel* daba a entender. No obstante se barajó la posibilidad de poder usarse como equipo de procesamiento que realizara tareas complejas y liberara de carga al sistema principal ya que realmente su utilidad es más similar a la de las placas *Arduino*.

Éstas también fueron tomadas en cuenta y por la misma razón que las *Galileo* fueron descartadas. No obstante, se consideró igualmente la utilización de la misma como por ejemplo convertidor analógico digital y descargar a la *Raspberry Pi* de computación

extra (por ejemplo, descartando datos no válidos u operando con ellos para dar datos precocinados al software final).

Por otra parte están las placas que se lanzaron posteriormente como respuesta al éxito de la Raspberry Pi. Una de ellas, la Cubieboard con una potencia en general superior y varias versiones con diferentes características y, naturalmente, un precio superior partiendo de los 50€.

Otra de ellas es la Banana Pi, se describe como un clon de la Raspberry Pi pero más potente. En efecto, con un precio ajustado de solo 50€ supera en características incluso a la Cubieboard, ofreciendo un procesador con dos núcleos más moderno, 1GB de memoria RAM y misma conectividad que la Raspberry.

Nombre	Procesador	RAM	USB	Salida de vídeo	I/O
Raspberry Pi Model B	ARMv6 700MHz	512MB	Sí	HDMI, VGA, Composite	GPIO, I ² C, SPI
Arduino Uno ¹	ATmega328 16 MHz	2KB	No	No	Serial, SPI
Intel Galileo	Intel Quark X1000 (x86) 400 MHz	256MB	Sí	No	Serial, UART, ACPI, PCIe
Cubieboard1	Cortex-A8 1 GHz	1GB	Sí	HDMI	I ² C, SPI, LVDS
Banana Pi	Cortex-A7 Dual core	1GB	Sí	HDMI Composite	GPIO, I ² C, SPI

Tabla 2.1: Los diferentes SBC comparados.

Al final nos decidimos por mantener la idea original. Aunque las alternativas Cubieboard o Banana Pi son interesantes por su potencia extra, la comunidad tras de ellas es prácticamente inexistente, justo al contrario que con la *Raspberry Pi*, la cual goza de una cantidad de soporte en internet enorme y eso hace más sencilla la tarea de solucionar problemas, buscar librerías o recabar experiencias. Además del reducido

¹En combinación con otro SBC.

coste de la Raspberry Pi, factor a tener en cuenta dados los objetivos del proyecto.

2.3.2. Software y sistema operativo.

La Raspberry pi ofrece como sistemas operativos las siguientes opciones: *GNU/Linux* (varias distribuciones) o *RISC OS*. La decisión es una distribución *GNU/Linux*, en este caso *Raspbian*(Figura 2.1), una versión portada de *Debian 7.0 Wheezy* a la *Raspberry Pi* con mayor comunidad y desarrollo detrás respecto a otras opciones como *Arch Linux* o *pidora* (versión portada de *Fedora* a *Raspberry Pi*).

La siguiente decisión es qué lenguaje o lenguajes de programación se utilizarán para desarrollar el software necesario para hacer que la solución funcione. Después de cursar toda la carrera a uno siempre le viene a la cabeza el mismo lenguaje de programación: Java. Es el lenguaje que más se estudia y más a fondo en todo el programa académico, por tanto la familiarización con el mismo es muy alta, pero nuestra SBC tiene una potencia relativamente limitada y operar la máquina virtual de Java sobre ella traería más quebraderos de cabeza que beneficios. *C* es una opción relativamente viable, su bajo nivel y rendimiento es adecuado para lo que se desea desarrollar, pero es un lenguaje especialmente complicado y en el que sin experiencia adecuada el proceso se puede complicar rápidamente.

La comunidad de la *Raspberry Pi* tiene especial predilección por *Python*, siendo uno de los lenguajes que te invitan a aprender -junto a otros- con tu nueva unidad e instalación de *Raspbian*. Pese a tener alguna noción básica del lenguaje procedentes de la asignatura *Procesadores del Lenguaje*, realizar el proyecto ha sido llevar mis conocimientos de este lenguaje a otro nivel completamente distinto, añadiendo dificultad extra ya que el aprendizaje del mismo se ha realizado “*al vuelo*” a la vez que se ha desarrollado, y esto ha inducido a errores sobre todo en las estructuras internas y funciones de *Python* de programación.

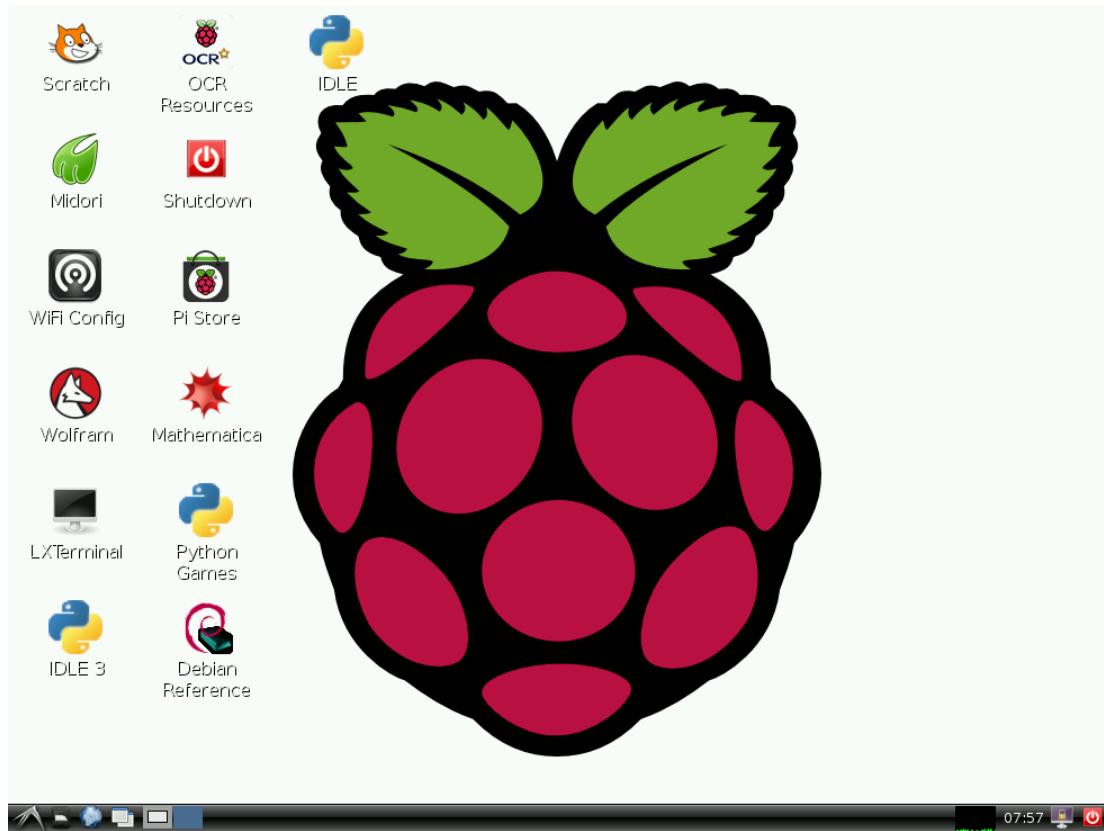


Figura 2.1: Raspbian.

“Programar sin una arquitectura o diseño en mente es como explorar una gruta sólo con una linterna: no sabes dónde estás, dónde has estado ni hacia dónde vas.”

Danny Thorpe, programador.

Capítulo 3

Diseño e implementación de la solución.

En esté capítulo se describe cómo fue el diseño y comienzo del desarrollo del proyecto a partir de la idea redactada en los capítulos anteriores.

3.1. Punto de encuentro de ambos proyectos paralelos.

El punto de encuentro acordado entre ambos proyectos para su posterior unión se fijó en el interfaz donde el estado de la adquisición es controlado y los datos adquiridos pueden ser obtenidos. Se especificó el siguiente diseño del interfaz para que ambas partes se desarrollen en torno al mismo:

3.1.1. Especificación del interfaz de control de adquisición de datos piDA.

- `__init__()`: Constructor de la clase para gestionar una adquisición con los parámetros por defecto (Frecuencia de muestreo 0 -la máxima posible-, canal de entrada 0, máximo número de muestras 0, sin límite).
- `__init__(sampling rate)`: Constructor de la clase para gestionar una adquisición a la frecuencia de muestreo que se pasa como argumento. Una frecuencia de muestreo

0 no fuerza ningún tiempo de espera entre muestras consecutivas, con lo que las muestras se toman a la mayor tasa posible. El resto de parámetros toman sus valores por defecto.

- `__init__(sampling rate, channel)`: Constructor de la clase para gestionar una adquisición a la frecuencia de muestreo y del canal de entrada dados con el número máximo de muestras por defecto.
- `__init__(sampling rate, channel, max count)`: Constructor de la clase para gestionar una adquisición a la frecuencia de muestreo, del canal de entrada y con el máximo número de muestras a adquirir (0 si no hay límite) dados.
- `set_sampling_rate(sampling rate)`: fija la frecuencia de muestreo de la adquisición. Una frecuencia de muestreo 0 no fuerza ningún tiempo de espera entre muestras consecutivas, con lo que las muestras se toman a la mayor tasa posible.
- `get_sampling_rate()`: devuelve la frecuencia de muestreo que se ha fijado para la adquisición.
- `set_channel(channel)`: fija el canal de entrada del que se toman las muestras.
- `get_channel()`: devuelve el canal de entrada que se ha fijado para la adquisición.
- `set_max_count(max count)`: fija el número máximo de muestras que se pueden tomar. 0 no fija ningún límite.
- `get_max_count()`: devuelve el número máximo de muestras fijado para la adquisición.
- `start()`: comienza la adquisición.
- `stop()`: detiene la adquisición.
- `get_data()`: devuelve una lista con todos los datos adquiridos hasta ese instante. Cada miembro de esa lista es a su vez una lista de dos elementos: el primero es el tiempo de adquisición (`float`) y el segundo el valor de la adquisición (`int`).

- `get_data(n_count)`: devuelve una lista con los últimos `n_count` datos adquiridos.
- `get_status()`: devuelve el estado en el que se encuentra la adquisición: '`'waiting'`', '`'running'`', '`'stopped'`'.
- `get_elapsed_time()`: devuelve la duración del periodo de ejecución de la adquisición.
- `print_data()`: escribe en pantalla un listado de todos los datos adquiridos hasta ese instante.
- `print_data(n_count)`: escribe en pantalla un listado de los últimos `n_count` datos adquiridos.

3.2. Primeras fases del desarrollo.

3.2.1. Creando el sistema base.

Dado que la solución a desarrollar no se trata solo del software, sino de un equipo informático completo similar a un ordenador personal, lo primero que se ha investigado es cómo combinar el SBC con los distintos periféricos -teclado, ratón, pantalla- para conformar el producto final.

El teclado y el ratón ha sido lo más sencillo de elegir, ya que la única exigencia es que sean USB para poder ser enchufados a la Raspberry Pi y su coste es muy reducido (menos de 10€ ambos periféricos).

No ha sido el caso de la pantalla: La Raspberry Pi dispone de dos tipos de salida de vídeo: el estándar digital relativamente nuevo HDMI capaz de una resolución de hasta 1080p¹, y el modo analógico totalmente obsoleto Composite (mediante un conector RCA, como las antiguas conexiones de los DVDs o videoconsolas a los televisores de tubo) capaz de una resolución máxima de 576i²)

¹1920x1080 píxeles proyectados progresivamente (un fotograma cada vez)

²576 líneas entrelazadas, se muestra la mitad de la imagen cada vez proyectando las líneas pares o impares alternadamente.

Esto dota a la Raspberry Pi de cierta flexibilidad al contemplar un estándar compatible con prácticamente todos los televisores del mercado en los últimos 30 años y también disfrutar de una calidad de vídeo digital acorde con los tiempos con HDMI, pero el interés se centra en poder conectar el SBC a los monitores disponibles en los laboratorios y despachos de la Facultad. Actualmente la práctica totalidad de los monitores disponibles para los alumnos en la facultad funcionan bajo el estándar VGA, no HDMI. Se investigó la posibilidad de convertir ese HDMI digital en VGA analógico. Una vez conseguido un viejo monitor Dell de 17" se probaron diferentes adaptadores y convertidores económicos.

Una de las comunidades en internet sobre la Raspberry Pi mantiene una *Wiki* donde se han comparado ya distintos adaptadores y cables[1]. Se eligieron de la lista dos modelos de coste contenido (menos de 6€) y comentarios positivos. Desgraciadamente uno de los mismos no funcionó como se esperaba (solo 640x480 de resolución en la pantalla, muy insuficiente para una aplicación actual) pero el otro adaptador (Figura 3.1) en cambio funcionó a la perfección, adaptándose automáticamente a la pantalla a la que fuera conectado. En el caso del monitor Dell del que se dispone: 1024x768, en otras pruebas se alcanzaron 1280x1024 y según especificaciones del producto, podrían obtenerse resoluciones de hasta 1920x1080.



Figura 3.1: Adaptador HDMI a VGA.

Con esto ya se dispone de un equipo completo y funcional. El desglose de elementos y su precio es el siguiente:

Elemento	Cantidad	Precio
Raspberry Pi Model B	1	£24,35 (30,50€)
Tarjeta de memoria SD de 4GB clase 10[2]	1	5,90€
Adaptador HDMI-VGA[3]	1	\$7,82 (5,85€)
Teclado USB[4]	1	6,99€
Ratón USB[5]	1	4,90€
Cargador de móvil 5v 700mAh[6]	1	7,49€ ³

Tabla 3.1: Desglose de coste de la solución propuesta.

El total asciende a 54,14€, un precio muy atractivo comparado con lo que cuesta cualquier ordenador personal hoy en día. El precio no incluye el coste del monitor, puesto que se reciclaría alguno de los existentes en el centro. De no ser así, el precio ascendería unos 70€ por un modelo estándar y se seguiría requiriendo el adaptador HDMI-VGA, puesto que el coste de un monitor VGA sigue siendo aún inferior al de uno digital con HDMI.

Estos son los requerimientos técnicos y económicos para el sistema base en el que correrá el software que se ha desarrollado en este proyecto. Esto no incluye los elementos electrónicos y hardware necesarios para poder adquirir datos de los sensores, ese es el propósito del proyecto paralelo, ejecutado bajo las mismas premisas que éste.

3.2.2. Eligiendo el entorno de desarrollo.

Es básico antes de comenzar el desarrollo de una aplicación informática elegir un entorno donde realizar la programación del mismo, para ello existen los IDEs (entornos de desarrollo) y Python dispone de gran cantidad de ellos[7]: desde entornos ya conocidos a estas alturas como Netbeans hasta grandes productos corporativos como

³Aunque los hay más económicos, en este aspecto se prefiere calidad ante precio ya que muchos de los disponibles en el mercado a precios económicos resultan no ofrecer el rendimiento esperado, y en un dispositivo como la Raspberry Pi induce al malfuncionamiento o la avería de la misma.

son Visual Studio o XCode. Un IDE permite simplificar la tarea del desarrollador al proveer herramientas como depuradores, sintaxis de colores para su mejor lectura, compiladores integrados, etc. Además algunos incluyen su propio constructor de interfaces gráficas (*GUI builder*) con el que la tarea de diseño se hace intuitivamente y de forma visual en lugar de tener que “dibujarla” escribiendo con el teclado.

Sin embargo, se ha preferido prescindir de un IDE propiamente dicho por dos razones en concreto:

1. Se carece de la experiencia en Python necesaria para realizar un proyecto de estas dimensiones, por tanto para realizar un aprendizaje sólido en el mismo la mejor solución es programar todo a mano y encontrarse con los problemas que todo novato ha de encontrarse.
2. Por norma general, *programar* un interfaz gráfico a través de una herramienta gráfica suele ser irónicamente más costoso que escribiéndolo a mano y conociendo las herramientas disponibles para la ordenación de los elementos, así como creando esquemas en bolígrafo y papel de qué se quiere crear.

Por estas razones, se ha decidido utilizar el editor de texto *Sublime Text 2* para el desarrollo de todo el proyecto. Además, con respecto al segundo punto y el código innecesario añadido por los constructores de interfaces gráficos, ha resultado mucho más práctico saber en todo momento qué es lo que está computando el procesador en cuanto a la GUI, pues la Raspberry Pi es conocida por ralentizarse a la hora de procesar gráficos.

3.2.3. Eligiendo el framework gráfico.

Python dispone de múltiples frameworks gráficos[8] libres para elegir en el desarrollo de nuestra aplicación. En total se ha probado con dos posibilidades multiplataforma: Tkinter y WxPython.

Tkinter.

Este framework tiene como punto a favor que está integrado de serie con Python y evita al programa de depender de librerías externas. Además la programación en Tkinter[9] resulta muy sencilla puesto que no hay una cantidad exagerada de controles o *widgets*⁴ ni muchas opciones de personalizar los mismos.

No obstante, ésta es la misma razón por la que se desiste su uso: tras probar diversas implementaciones de lo que podría ser la solución final se descubrió que es una librería poco potente y más orientada a realizar pequeños diálogos o ventanas que a realizar un programa relativamente grande con multitud de controles.

WxPython.

WxPython es una versión para Python del famoso *WxWidgets*[10] en C++ con toda la versatilidad y calidad de su predecesor. Ésta ha sido la librería elegida para nuestro programa ya que su API, aunque esté pobemente documentada[11] y muchas veces directamente copiada de la versión en C++, ha resultado suficientemente potente.

No obstante, en otras versiones[12] de la documentación, ésta es más explícita, está escrita para Python e incluso muestra la apariencia de los widgets en las distintas plataformas.

En su contra además se puede añadir que es un framework que requiere de más *artesanía* pero cuyos resultados saltan a la vista rápidamente y necesita ser instalado aparte. Este no es un problema realmente ya que el gestor de paquetes *Aptitude* (más conocido por su comando *apt-get*) de nuestro sistema operativo *Raspbian* lo dispone en sus repositorios oficiales y su instalación se ha automatizado en un script que acompaña al código del programa.

3.2.4. Eligiendo una librería de representación gráfica.

Una de las funciones fundamentales de la aplicación será mostrar los datos adquiridos mediante una gráfica con respecto al tiempo, por tanto se necesita una

⁴Cada uno de los elementos con los que un interfaz gráfico interactúa con el usuario: botón, caja de texto, menú, etc.

librería que permita realizarlo o por otra parte programar una librería propia adaptada al uso. La *wiki* de Python[13] ofrece muchas alternativas. La primera de ellas y elegida en este proyecto es Matplotlib, compatible con el framework gráfico WxPython (o *WxWindows*).

Matplotlib dispone en su página web[14] de multitud de ejemplos (Figura 3.2) de todo lo que es capaz de hacer y llama la atención que integra la posibilidad de exportar los datos dibujados en multitud de formatos como PNG, PDF o SVG así como su potente herramienta de zoom. Además dispone de un extenso API con todas las posibilidades que ofrece.

Otra alternativa que podría haberse elegido podría haber sido Plotly, pero su uso requiere de registro en su página web y conexión a internet para verificar el mismo cada vez que se utilice. Esto resulta muy molesto e innecesario, especialmente si se desea que esta solución se implemente en varias estaciones de trabajo y se desconoce si dispondrán conexión a internet. Como alternativa *más libre* está Veusz, pero no resulta especialmente atractivo visualmente.

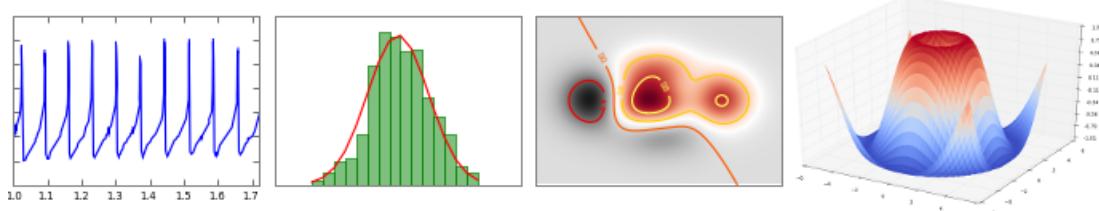


Figura 3.2: Ejemplos de Matplotlib.

3.3. Consideraciones a tener en cuenta antes de comenzar.

3.3.1. Dos soluciones muy distintas.

Las dos interfaces que se pretenden sustituir con este proyecto son muy diferentes entre sí: si bien ambas usan gráficas o histogramas para la representación de los datos, la mecánica interna de cómo obtener los datos y procesarlos es muy distinta. Por acuerdo mutuo se decidió realizar primero la solución que se ve más simple tanto a

nivel electrónico como a nivel de procesado: la del laboratorio de termodinámica y física aplicada y su dispositivo *PASCO 500 Interface*.

Es el más sencillo porque la adquisición de datos no necesita tanta frecuencia de muestreo⁵ y se pueden procesar fácilmente. En cambio para el laboratorio de radiactividad se necesitarían varios miles de hercios, algo que se ve muy inviable para hacer directamente en la Raspberry Pi sin la necesidad de un hardware externo complejo con el que no se ha contado durante el desarrollo de este proyecto.

3.3.2. El método de trabajo a utilizar.

Solo hay un único desarrollador, por lo que ha habido flexibilidad total a la hora de elegir el método de trabajo. En este caso se ha efectuado el desarrollo mediante pequeños hitos diarios desarrollando cada parte del programa. Como se ha utilizado Git como repositorio de software, se ha adoptado la norma de que todo lo que se envíe al repositorio ha de funcionar. Por tanto al llegar a cada hito se ha procedido a testar el programa contra fallos, se han corregido los más críticos y documentado los más leves en cada envío al repositorio. Por tanto el cumplimiento del método ha sido estricto al no poder dejar un objetivo a medias, que dejaría el programa no utilizable o parcialmente inutilizable.

Al haber ofrecido el grupo un despacho donde trabajar a ambos proyectos paralelos, la comunicación ha sido bidireccional y regular, proponiendo cambios y comentando la implementación de las diversas partes diariamente. Además durante gran parte del desarrollo se han realizado reuniones semanales en una sala de juntas con los codirectores de ambos proyectos y los proyectandos en orden de clarificar dudas, preguntar alternativas de desarrollo o consejos en general.

⁵Velocidad a la que se obtienen los datos.

“Desde el punto de vista de un programador, el usuario no es más que un periférico que teclea cuando se le envía una petición de lectura”

Peter Williams.

Capítulo 4

Diseño de la interfaz gráfica.

El extremo final del “puente” es la interfaz gráfica donde el usuario final operará con el programa. En este capítulo se detalla el diseño de la misma.

4.1. Disposición de los elementos en la ventana principal.

Se ha diseñado una ventana principal con barras de herramientas y de estado donde colocar los elementos. La gráfica donde se visualizarán los datos es una parte fundamental del programa, por lo que se decidió que ocupara dos tercios del espacio disponible verticalmente y el tercio restante lo ocuparán los controles para la adquisición de datos, visualización de los mismos y otras opciones no relacionadas con la gráfica. La resolución con la que se ha creado el diseño es 1024x768; una resolución modesta pero segura, pues cualquier monitor antiguo es capaz de mostrarla. Pese a que algunos test se han realizado bajo 1280x1024 o incluso panorámico a 1920x1080 se ha asegurado que la visualización de los componentes sea óptima a 1024x768.

4.1.1. Los dos tercios superiores.

Debajo de la gráfica se ha incluido una pequeña barra de herramientas, a la izquierda para controlar elementos de la gráfica como el zoom, mover la misma o exportar el dibujo, mientras que a la derecha se han incluido otros controles como una barra deslizante para elegir la tasa de refresco automático de los datos, un botón para

desactivarla, un botón para comenzar y otro para detener la adquisición de todos los canales disponibles, y un botón para actualizar los datos de la gráfica manualmente.

El resultado final se puede apreciar en la Figura 4.1.

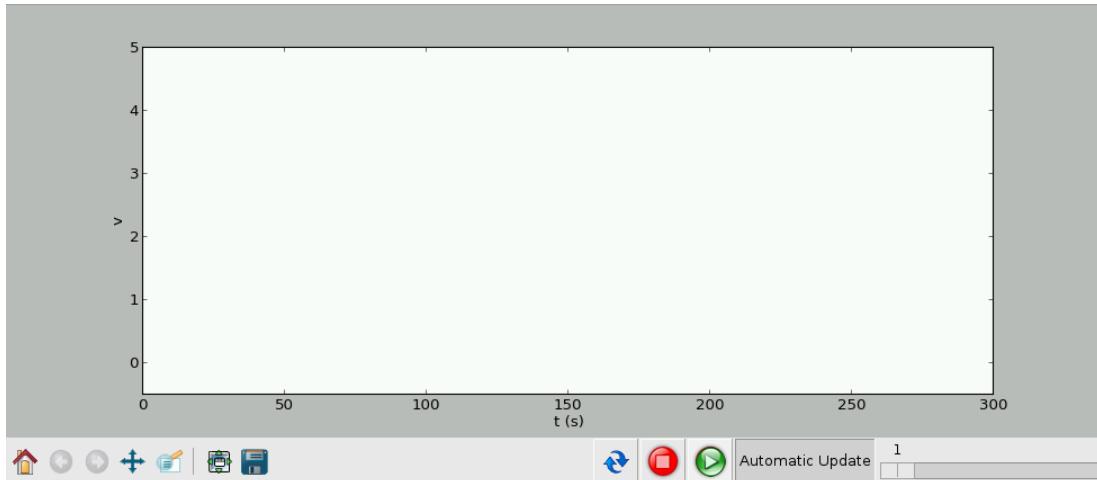


Figura 4.1: El conjunto de la gráfica y sus controles.

4.1.2. Tercio inferior de la ventana principal.

Para dividir los controles de la adquisición, la visualización de los datos y la exportación de los mismos (estas serán las opciones por el momento) se han creado tres pestañas:

Control de adquisición de datos.

La primera pestaña, correspondiente a los controles de adquisición de datos se ha dividido horizontalmente en cuatro columnas, una por cada canal.

La implementación se ha realizado de manera que cada columna contiene un objeto AcquisitionGUI representando un panel con los siguientes controles:

- Un texto informativo de estado del módulo de adquisición de datos junto con un indicador de colores.
- Un desplegable que permite elegir el color de la línea que se dibujará en la gráfica.

- Un cuadro de texto donde especificar la frecuencia de muestreo a la que se desea que trabaje el canal.
- Un botón para “Stop” para detener la adquisición de datos y otro “Start” para comenzar la misma.
- Un botón “Simulation” para cargar en ese canal el módulo de simulación en lugar del módulo de adquisición de datos en tiempo real.

Cada uno de estos paneles está relacionado con un módulo de adquisición de datos si está disponible y el control del mismo así como la disposición de los datos para la gráfica se realiza directamente desde aquí.

El constructor de la clase incluye el número de canal al que va a representar y que inicializará a través del interfaz piDA. De esta manera es posible aumentar la cantidad de canales a los que da soporte el programa tan solo creando más paneles, ya que todas las funciones relacionadas están dentro del mismo.

El resultado final se puede apreciar en la Figura 4.2.

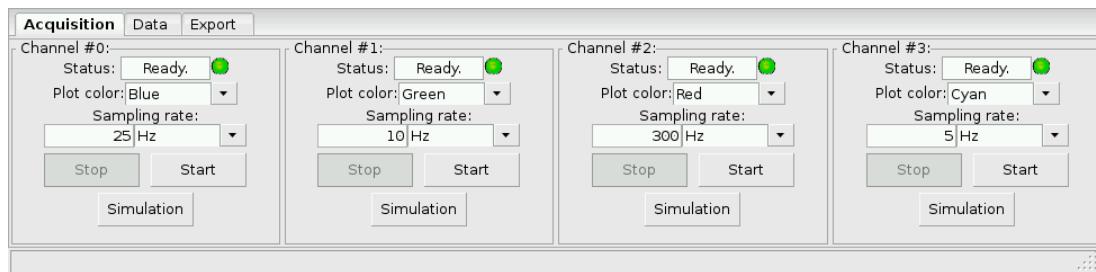


Figura 4.2: Pestaña para controlar las adquisiciones de datos.

Visualización de datos.

La segunda pestaña corresponde a la visualización de datos donde los datos son mostrados directamente como su valor x (tiempo) e y (voltaje en este prototipo). El diseño del panel formado por cinco columnas es el siguiente:

- Una columna con cinco botones cuadrados, cada uno con un diseño personalizado especificando su función. El primero actualizará los datos de todas las tablas, el

segundo los datos de la tabla correspondiente al primer canal, el tercero los datos de la tabla correspondiente al segundo canal, el cuarto los del tercer canal y el quinto los del cuarto canal.

- Cuatro columnas con cuatro tablas de dos columnas cada tabla para los datos de cada canal, rellenadas bajo petición del usuario por los botones de la columna anterior.

Las tablas han sido modificadas en una subclase interna ya que el control que provee WxPython no permite ajustar el tamaño de las columnas de datos. De esta manera el control una vez creado ajusta el tamaño de las columnas al espacio disponible.

Se ha decidido liberar de carga al procesador y que la actualización de los datos de las tablas se realice mediante petición expresa del usuario con los botones de la primera columna. Se ha creado un diseño para cada uno de los botones con Adobe Fireworks[15] para que ocupen el mínimo espacio posible al carecer de texto pero siga siendo intuitiva su función.

La modularización de este panel es una tarea pendiente.

En la Figura 4.3 se puede ver el resultado final.



Figura 4.3: Pestaña de visualización de datos.

Exportación de datos.

Por último, la tercera pestaña correspondiente a la exportación de datos consiste en dos filas principalmente:

- Una primera fila formada por cuatro columnas, cada una de ellas representando cada canal. Dentro de cada columna se han añadido campos de texto para personalizar el título de la exportación, la etiqueta del eje X, la etiqueta del eje Y y sendos botones para exportar los datos en formato CSV y PDF.
- La segunda fila se ha utilizado para exportar datos de varios canales a la vez a un único fichero. Se han incluido cuatro *checkboxes* para seleccionar los canales que se exportarán al fichero, una lista desplegable para truncar los valores de tiempo a segundos, milisegundos, microsegundos o nanosegundos y sendos botones para exportar a CSV a PDF.

El exportado de datos a PDF no está implementado en esta versión, y al igual que el anterior panel, éste tampoco ha sido modularizado.

El resultado final: Figura 4.4.

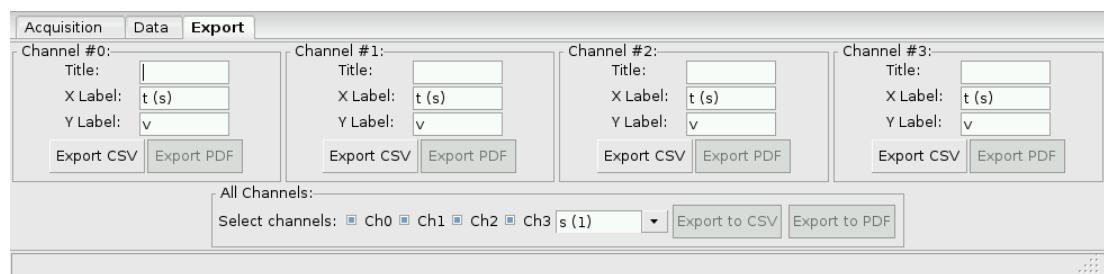


Figura 4.4: Pestaña de exportación de datos.

4.2. Resultado final.

El conjunto de los anteriores elementos forman la ventana principal del programa dando una apariencia sobria y un control intuitivo sobre las funciones.



Figura 4.5: El interfaz gráfico de nuestro programa.

“El hardware es lo que hace a una máquina rápida; el software es lo que hace que una máquina rápida se vuelva lenta”

Craig S. Bruce, desarrollador.

Capítulo 5

Diseño e implementación del modelo.

La lógica del programa se ha desarrollado de la manera más óptima y flexible, posible facilitando la ampliación de las funcionalidades del mismo así como la reutilización de código en el futuro. Para ello es importante la modularización de cada parte, en este capítulo se explica como se ha realizado esa tarea en el programa y su funcionamiento interno.

5.1. Modulos que forman el programa.

Un módulo en Python es un conjunto de funciones, clases u otros módulos que pueden ser importados en otros módulos o clases dando flexibilidad al código programado. El programa está formado por tres módulos principales llamados *classes*, *GUI*, y *utils*.

- *classes* contiene los objetos que utiliza el modelo para funcionar.
- *GUI* contiene todos los objetos del interfaz gráfico.
- *utils* contiene funciones o clases con propósitos específicos ubicadas aquí para liberar las clases principales de código o porque pueden ser de interés en diversas partes del programa.

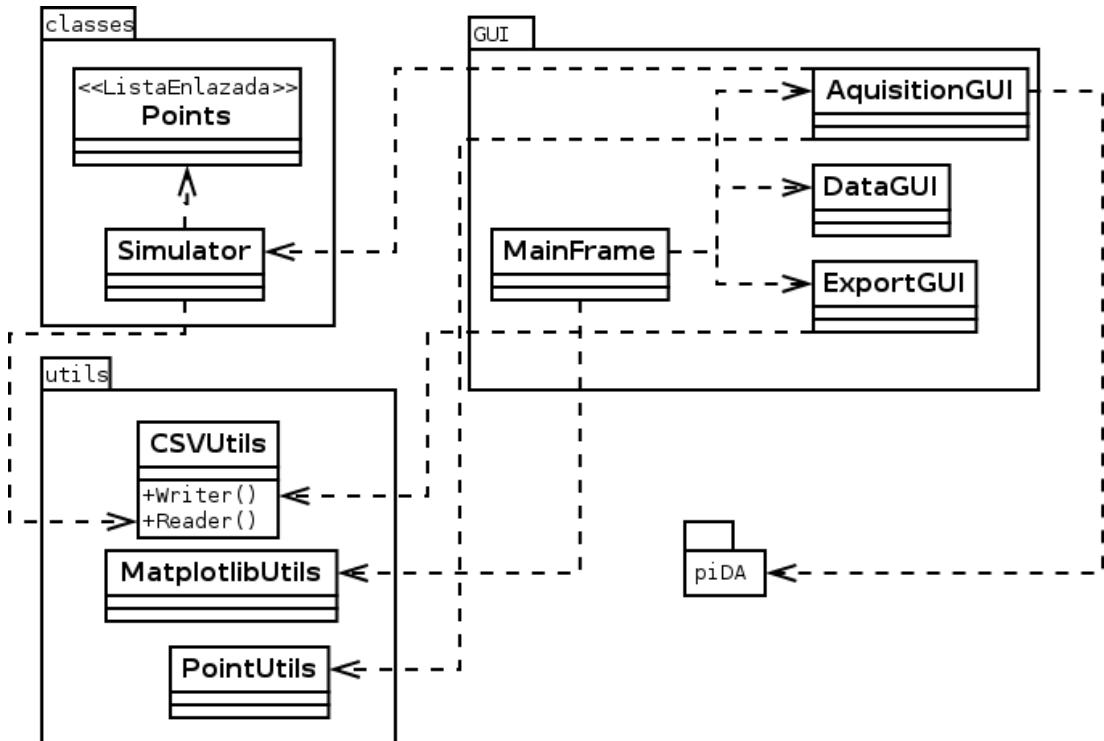


Figura 5.1: Diagrama de clases y sus dependencias..

Las clases mostradas en el diagrama de la Figura 5.1 se detallan a continuación:

5.1.1. Módulo *classes*.

- **Points.py:** Esta clase contiene una lista enlazada de puntos (x,y) con un iterador infinito: Cuando llega al final de la lista el siguiente elemento vuelve a ser primero de la lista pero con el valor x_i sumado al x_n y dejando y igual. Al alcanzar de nuevo el final de la lista, el valor sumado x_n se dobla para mantener la continuidad de la línea de puntos.
- **Simulator.py:** Esta clase contiene el simulador programado para realizar pruebas cuando la interfaz piDA de adquisición de datos aún no estaba operativa. Su funcionamiento está explicado en la Sección 5.3). El simulador implementa el diseño acordado en la Subsección 3.1.1 para que una vez el prototipo estuviese preparado para incluirse en el software la transición fuese directa.

El módulo incluye además las clases no desarrolladas:

- Acquisition.py: Diseño rudimentario del interfaz de adquisición de datos discontinuado.
- Graph.py: Clase que representa una gráfica con sus datos, propiedades y vistas, con objetivo de poder ser guardada como formato propio o dentro de un fichero de proyecto.
- Histogram.py: Idéntica a la clase Graph pero esta vez con un histograma para abordar la solución del laboratorio de radiactividad.
- Project.py: Clase que representa un proyecto, con sus datos y gráficas con objetivo de guardarlo en un fichero y poder recuperar el trabajo en otro momento.

5.1.2. Módulo *GUI*.

Dado a que el diseño del interfaz gráfico está explicado en su propio Capítulo 4, aquí solo se explican las funciones correspondientes a la parte del modelo.

- AcquisitionGUI.py: Implementa un panel con los controles para manejar un interfaz de adquisición o simulador creado dentro del mismo. Es una de las clases clave ya que incluye la lógica que envía los datos a la gráfica de la ventana principal, explicada en la Subsección 5.2.1.

Al instanciar esta clase se pasa un número de canal como argumento que intenta inicializarse en la interfaz piDA, si no es posible obtenerlo, el panel queda desactivado. Esto es útil por ejemplo si piDA soporta varios interfaces de adquisición con distinto número de canales cada uno. Por ejemplo, el primer prototipo (Sección 6.2) solo soportaba dos canales.

- DataGUI.py: Implementa un panel para visualizar los datos adquiridos. Sección 4.1.2.
- ExportGUI.py: Implementa un panel para exportar los datos adquiridos. Sección 4.1.2.

- MainFrame.py: Implementa la ventana principal. Sección 4.1.

Aquí están las funciones encargadas de controlar el refresco automático y manual de la gráfica y la petición de datos a los distintos módulos activos para la misma.

5.1.3. Módulo *utils*.

- CSVUtils.py: Contiene dos clases Reader, y Writer las cuales sirven para leer o escribir ficheros CSV.
 - La clase Reader recibe una ruta de archivo CSV en el formato del programa o del software del laboratorio termodinámica y física aplicada *DataStudio*. Reconoce el título, cabeceras de las columnas y cada par de datos x e y , los cuales los incluye en un lista de tipo Points (ver Subsección 5.1.1).
 - La clase Writer recibe una ruta para crear el fichero y provee de los métodos para escribir en él una o más filas con el formato del programa y codificación UTF-8.
- MatplotlibUtils.py: Contiene tres funciones para tratar con la API interna de Matplotlib de colores de líneas. Se ha añadido además la opción de no dibujar la línea.
 - La función IntToCharColor recibe un entero del 0 al 7 y devuelve un carácter de color para Matplotlib. Si se recibe un número fuera de ese rango la línea no se dibuja (carácter 'z'). Los códigos y el color que representa soportados por Matplotlib son, en orden:
 - b: blue
 - g: green
 - r: red
 - c: cyan
 - m: magenta
 - y: yellow
 - k: black
 - w: white

- IntToColor recibe un entero y devuelve una cadena de caracteres con su nombre en lenguaje natural, el orden y los nombres se pueden ver en la lista del apartado anterior. Cualquier valor fuera del rango retorna la cadena “None”.
 - GetColorList retorna una lista con los colores soportados por Matplotlib en lenguaje natural.
- PointUtils.py Contiene el método PointsToDoubleLists que recibe una lista de puntos en formato $(x_0, y_0), (x_1, y_1), (x_2, y_2) \dots (x_n, y_n)$ y retorna dos listas: Una con todas las x ($x_0, x_1, x_2 \dots x_n$) y otra con todas las y ($y_0, y_1, y_2 \dots y_n$). La librería Matplotlib requiere para dibujar que se le entreguen los puntos x e y por separado, por tanto se necesita hacer una conversión.

5.2. Lógica de control de la gráfica.

5.2.1. Actualización de los datos.

La actualización de los datos de la gráfica se realiza en un proceso de dos pasos:

Obtener los datos de cada canal. Se llama secuencialmente a cada canal activo o con datos para pintar y se le pide que envíe sus datos a la gráfica. Ésta llamada corre sobre un hilo diferente al que se está utilizando en ese canal para obtener datos, por tanto se han implementado semáforos de tipo mútex para la sincronía entre hilos concurrentes. La llamada espera a que no se estén utilizando los datos en el módulo para obtener una copia de los mismos, que la convierte mediante PointsToDoubleLists en datos que puede enviar a la gráfica. El color de la línea también se especifica en este instante, por lo que puede ser cambiado en cualquier momento de la adquisición. Los gráficos serán mostrados en el próximo redibujado de la gráfica.

En la Figura 5.2 se puede ver un diagrama de cómo se inicia el proceso desde “RefreshGraphOnce” en el la clase de la ventana principal.

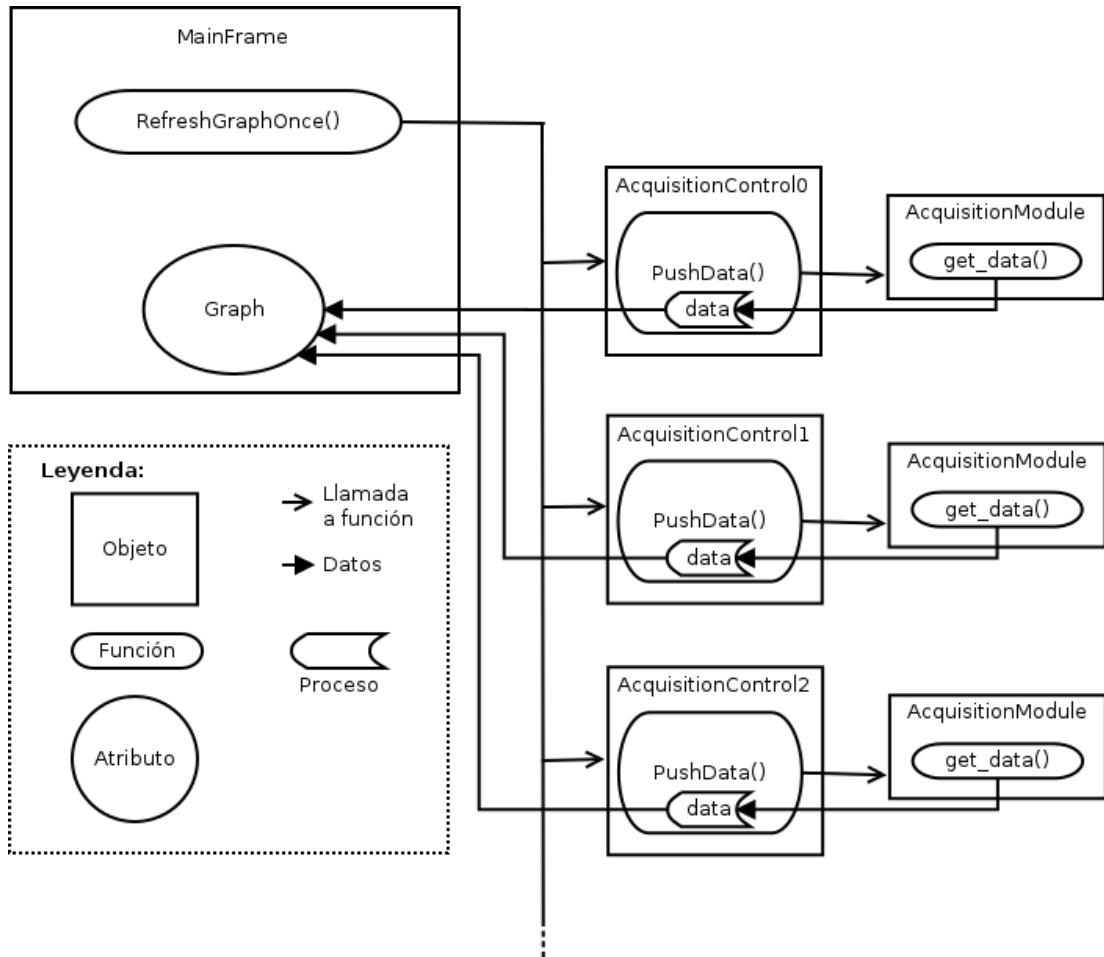


Figura 5.2: Diagrama de envío de datos a la gráfica.

Redibujado de la gráfica. La primera implementación de esta tarea fue un redibujado total del lienzo de la gráfica, esto incluía los ejes, títulos e información que no necesitaba modificarse. Este modo resultaba muy intensivo de CPU y un verdadero problema si se deseaba realizar una actualización de datos más o menos frecuente, por tanto se buscó una optimización para el mismo.

La optimización encontrada forzaba el cambio de gran parte de la lógica de funcionamiento del programa, por lo que se realizaron tests de rendimiento comparando uno y otro sistema. Los resultados mostraron una mejora del 333 % con respecto al método inicial y reduciendo la carga en la CPU significativamente, haciendo obligatoria la implementación de la misma.

El funcionamiento de esta optimización se basa en la actualización de únicamente del cuadro dónde se dibujan las líneas previa a una restauración del mismo a un estado anterior guardado. De este modo, al iniciar una adquisición de datos se toma el estado de la gráfica y en cada refresco se restaura este estado, y acto seguido se dibujan las líneas de nuevo, con los nuevos valores ya preparados. En la Figura 5.3 viene una representación de este proceso.

Esta tarea se realiza tras obtener los datos de cada uno de los canales como se ha explicado en el apartado anterior, pero la actualización de un elemento del interfaz gráfico requiere que éste se ejecute sobre el hilo principal, por lo que se llama a esta función desde allí, pudiendo crear pequeños bloqueos si la cantidad de datos a dibujar es muy grande.

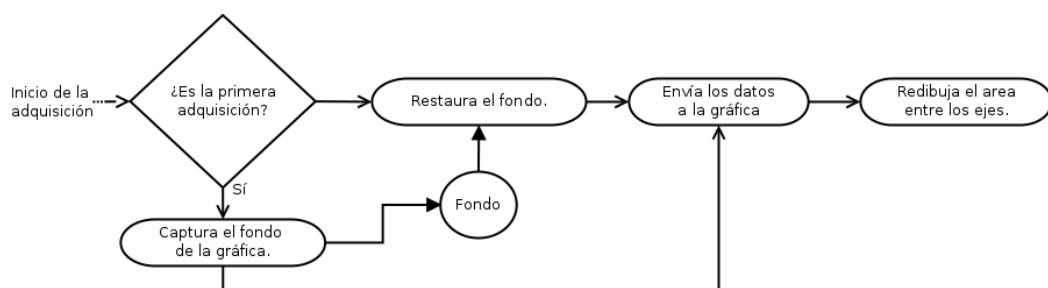


Figura 5.3: Diagrama de refresco selectivo de la gráfica.

5.2.2. Actualización automática.

Una función básica para poder ver los datos a la vez que se toman a través del interfaz de adquisición es que la gráfica se vaya actualizando automáticamente. El diseño del sistema es muy sencillo: Cuando es activado, comienza un hilo donde un bucle llama en el intervalo que ha especificado el usuario a la función de actualizar la gráfica. Cuando se desactiva esta función, el bucle finaliza, y con él la ejecución del hilo.

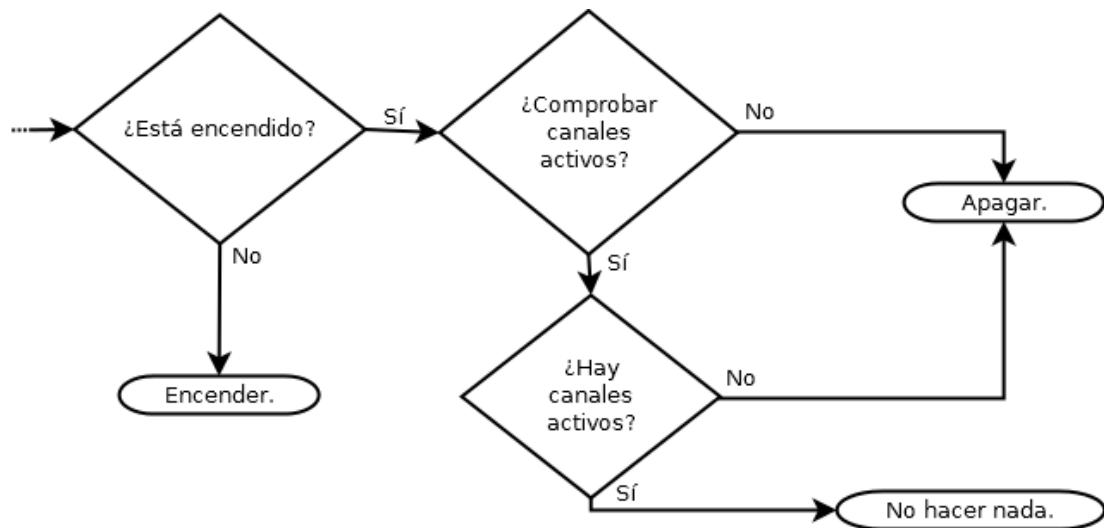


Figura 5.4: Función de encendido o apagado del refresco automático.

Ha supuesto un pequeño desafío programar cuándo esta función ha de activarse o desactivarse, ya que su estado encendido o apagado depende de un control de usuario general, y diversos canales ejecutándose en diferentes hilos de ejecución. Se ha creado una sola función *ToggleGraphRefreshing* encargada de activarlo o desactivarlo. La función usada sin ningún argumento hace su función natural: si está apagado lo enciende, si está encendido lo apaga. En cambio si se especifica que compruebe los canales pasando como argumento `check_channels=True` comprueba que no haya ninguno activo antes de apagar. En la Figura 5.4 se puede ver la representación de este proceso.

5.3. El simulador.

Durante el transcurso del proyecto desarrolló un simulador de interfaz de adquisición de datos para poder probar los elementos y no retrasar el desarrollo de esta parte.

La lógica de funcionamiento del mismo es simple: como su propio nombre indica, ha de simular el funcionamiento de un interfaz de adquisición. Por tanto se ha adoptado el diseño descrito en la Subsección 3.1.1 para que además llegado el momento de integrar el prototipo desarrollado en el proyecto paralelo no haya que modificar la lógica del

programa.

5.3.1. Lógica de funcionamiento.

El simulador dispone de una lista interna de puntos que es el “origen de datos”. Durante la adquisición, cada elemento de esta lista se copia a la frecuencia especificada a otra lista accesible mediante el interfaz.

Para ello, al iniciarse se crea un hilo de ejecución donde un bucle copia los valores de una lista a otra. Al final de cada bucle se comprueba si se ha ordenado detener la adquisición o si ha de continuar. Si es la segunda opción, se espera la inversa de la frecuencia de muestreo antes de volver a copiar el siguiente valor de la lista iterada, como se puede ver en la Figura 5.5.

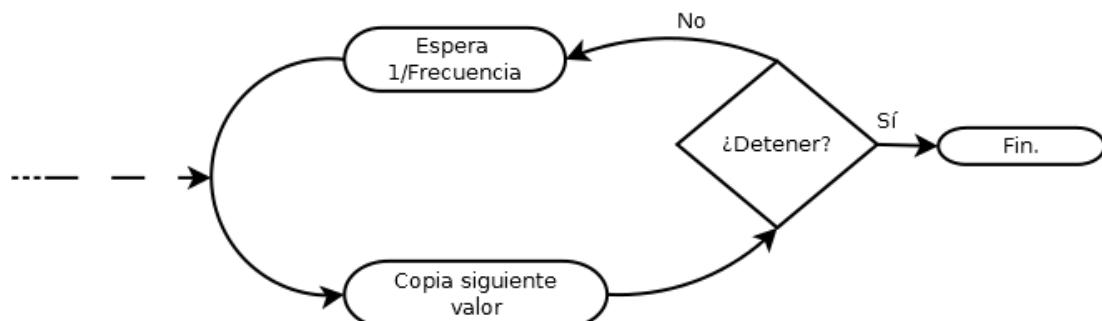


Figura 5.5: Función de encendido o apagado del refresco automático.

El origen de los datos.

El simulador obtiene sus datos de un fichero CSV que se carga con clase Writer descrita en apartados anteriores, obteniendo así una lista iterable infinita Points.

Se ha tomado la decisión de tomar datos reales procedentes de un fichero en lugar de valores generados aleatoriamente o por una función matemática para que éstos sean lo más parecidos a los que se van a manejar con el prototipo. Además, de este modo se ha desarrollado un módulo de importación de ficheros CSV útil en el futuro.

5.4. Los distintos hilos de ejecución.

El software está programado concurrentemente, con los problemas de sincronía que esto puede ocasionar si no son tratados adecuadamente.

El programa ejecuta un hilo principal donde el interfaz gráfico es ejecutado, y pueden crearse hasta cinco hilos más, esto es uno por cada canal adquiriendo datos y otro controlando la actualización y obtención de datos de cada uno de los canales.

El punto más susceptible para que un error de sincronía suceda es en el momento que el hilo correspondiente a la obtención de datos de cada canal accede a la lista donde éstos están guardados a la vez que se está almacenando un valor. Esto provocaría por ejemplo que en la lista esté escrito el valor x pero no el y , por tanto enviando a la gráfica una lista de puntos x_n con un tamaño diferente al de y_n , provocando un fallo en la gráfica.

Se ha solucionado este problema de sincronización mediante el uso de secciones críticas controladas por semáforos binarios (*mútex*) en cada interfaz de adquisición. De este modo nunca se accederá a la lista de datos de un interfaz de adquisición si este se encuentra escribiendo sobre él, si no que, como muestra el diagrama de la Figura 5.6, esperará a que el semáforo de dicho interfaz esté libre para tomarlo, obtener los datos y dejarlo libre de nuevo.

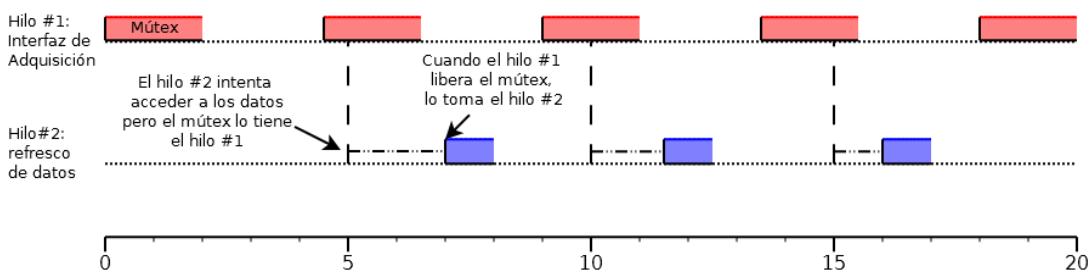


Figura 5.6: Diagrama de sincronización entre hilos.

5.5. Otras funciones integradas: Instalador.

Junto con el resto de ficheros se incluye un “setup.sh” que configura el sistema para la ejecución del programa.

Este fichero de comandos instala las librerías de las que se depende *WxPython* y *Matplotlib* del repositorio oficial de Raspbian. Además, obtiene del repositorio de Git la versión más actual del interfaz de adquisición de datos piDA y lo instala en el sistema.

Por tanto, la replicación de este programa en las distintas estaciones es tan sencillo como ejecutar los siguientes comandos:

```
$ git clone https://github.com/dmcelectrico/PFC.git  
$ ./PFC/setup.sh
```


“Hoy en día la mayoría del software existe no para resolver un problema, sino para actuar de interfaz con otro software”

Ian O. Angell, profesor.

Capítulo 6

Los prototipos y el interfaz piDA.

En este capítulo se explican los dos prototipos desarrollados en el proyecto paralelo y su integración en nuestro programa junto con el interfaz piDA.

6.1. El interfaz piDA.

Para operar con los prototipos se ha desarrollado junto a los mismos la librería bautizada como piDA¹ que implementa el diseño detallado en la Subsección 3.1.1.

En lugar de empotrar la librería en el programa y hacerlos una sola entidad, se ha decidido que tanto la librería como el programa sigan siendo dos elementos independientes. De este modo ambos proyectos pueden seguir mejorando por separado y las mejoras implementadas en uno u otro no necesitan una nueva integración en el código, sino que son totalmente actualizables por separado gracias a github.

Además, se incluye un instalador con la librería que configura la Raspberry Pi, instala las dependencias de piDA y deja el sistema preparado para funcionar con los prototipos.

¹Se puede encontrar en el siguiente repositorio git: <http://github.com/noeldiazro/piDA>

6.2. El primer prototipo.

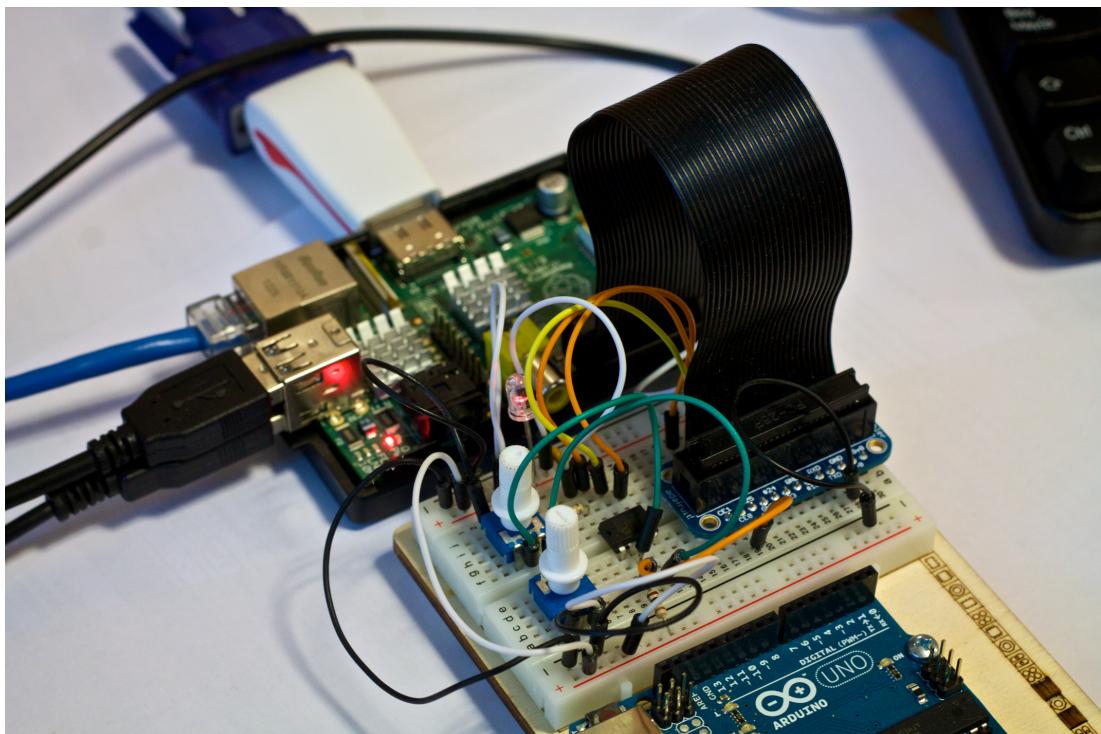


Figura 6.1: El primer prototipo conectado a la Raspberry Pi.

El primer prototipo (Figura 6.1) cuenta con un chip convertidor analógico/digital MCP3202 que provee dos entradas analógicas y precisión de 12 bits. El chip se comunica con la Raspberry Pi a través de un interfaz de serie compatible con el protocolo SPI, la conexión con la misma es mediante los pines GPIO. Podemos controlar la tensión que lee el chip mediante dos potenciómetros conectados a sendos canales. Todo el conjunto está montado sobre una placa de prototipado o *protoboard*. Los datos que se obtienen del mismo son voltios, concretamente entre 0 y 3,3 ya que la misma placa se alimenta de la propia Raspberry, no requiriendo así alimentación externa. Durante el desarrollo de los siguientes prototipos esto cambiará para poder conectar los sensores del interfaz original PASCO (éstos operan entre ± 10 voltios y algunos requieren alimentación extra de 5 y/o 12 voltios).

6.2.1. Implementación en el programa.

Se ha decidido realizar la implementación de la librería directamente y prescindir de la portabilidad del programa a otros sistemas operativos dejando esta característica como una futura mejora. Al ser una librería de Python mas, con solo importarla en nuestro programa ya tenemos acceso a sus módulos, especialmente al módulo Acquisition, que es nuestro nexo de conexión.

Al disponer solo de dos canales disponibles según el prototipo pero tener el interfaz gráfico preparado para cuatro canales y no tener ninguna manera de obtener a través de la librería el número de canales ofrecidos, se ha programado de manera que la interfaz gráfica intenta inicializar cada uno de los cuatro canales y si alguno falla simplemente queda marcado como “no disponible” y se desactivan los controles asociados al mismo en el interfaz gráfico.

6.2.2. Utilización del prototipo.

La librería se comportó según lo acordado en el diseño, por tanto tras la integración de ésta el funcionamiento fue el esperado y los cambios en el código original fueron mínimos. En la Figura 6.2 se puede ver un ejemplo de su funcionamiento.



Figura 6.2: Funcionamiento del primer prototipo de dos canales.

6.3. Segundo prototipo.

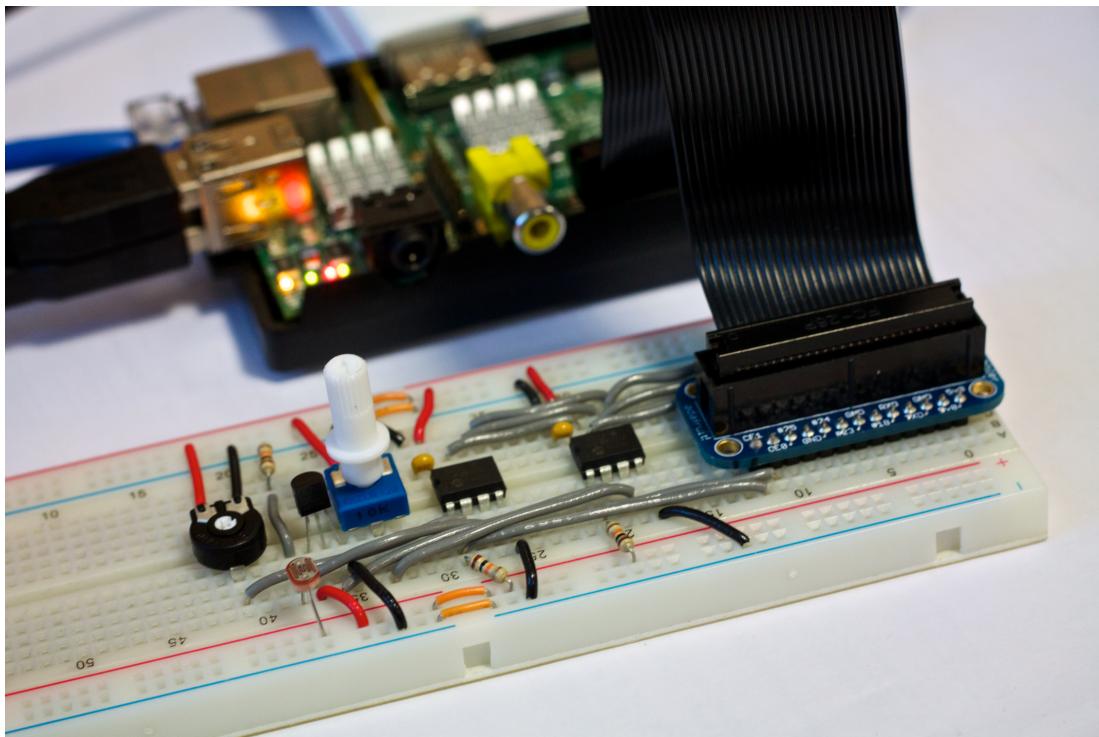


Figura 6.3: El segundo prototipo conectado a la Raspberry Pi.

El segundo prototipo (Figura 6.3) cuenta con las mismas características que el primero pero esta vez con dos chips MCP3202 dando soporte para cuatro canales. Dos de ellos son sensores: uno de luz y otro de temperatura. La distribución de canales por tanto es la siguiente:

- Canal 0: Potenciómetro (azul).
- Canal 1: Sensor de temperatura
- Canal 2: Sensor de luz.
- Canal 3: Potenciómetro (negro).

Los sensores no se han calibrado por lo que los datos obtenidos de ellos siguen siendo voltaje entre 0 y 3,3 voltios.

6.3.1. Implementación y utilización en el programa.

El código estaba preparado para ser utilizado hasta con cuatro canales, por tanto no hizo falta acondicionar nada del mismo para la conexión del nuevo prototipo.

El funcionamiento una vez más fue el esperado, recibiendo datos de los cuatro canales y toda la mecánica funcionando sin ningún problema, incluida la visualización y exportación de los datos. En la Figura 6.4 se puede ver una captura del programa funcionando a distintas frecuencias e interactuando con los distintos canales.



Figura 6.4: Funcionamiento del segundo prototipo con cuatro canales.

“Las máquinas son cada vez más eficientes y mejores, por lo que queda claro que la imperfección es la grandeza del hombre.”

Ernst Fischer, filósofo.

Capítulo 7

Evaluación y pruebas.

En este capítulo se describen las pruebas que se han ejecutado sobre el software con el fin de ponerlo a prueba, conocer la exactitud temporal con la que los datos son medidos y sus limitaciones.

7.1. Descripción de las pruebas.

Para la realización de este capítulo se van a obtener dos elementos: la frecuencia real a la que el sistema está tomando medidas y la carga del sistema en cada una de ellas. Para obtener el primer dato hemos calculado el intervalo medio entre medidas dividiendo el tiempo real que ha tomado la adquisición entre el número de datos y luego se ha realizado su inversa, como indica la Ecuación 7.1.

$$\bar{F}_{Real} = \frac{1}{T_{total}/n_{medidas}} \quad (7.1)$$

El procedimiento de las pruebas ha sido el siguiente:

- Se han elaborado una serie de tests basados en la frecuencia y canales usados en la adquisición de datos para obtener una medida de cuál es la frecuencia máxima a la que se puede operar con nuestra solución actual y se ha anotado la carga del sistema para evaluar cual es el motivo de la limitación en cada caso.
- Se han adquirido datos a distintas frecuencias durante un tiempo mínimo de cinco

minutos en cada caso, variando desde 1Hz hasta 2000Hz con un canal, hasta 1000Hz con dos canales simultáneamente y hasta 750Hz con cuatro canales.

- Se han exportado los datos mediante el módulo interno del programa, trasladado a una hoja de cálculo y obtenido la frecuencia media real de la adquisición mediante 7.1.
- Se ha comparado la frecuencia real obtenida en el apartado anterior con la frecuencia teórica en orden de calcular el error absoluto medio y error relativo porcentual mediante las fórmulas 7.2 y 7.3 respectivamente. donde \bar{F}_r representa la frecuencia media real obtenida en 7.1 y F la frecuencia teórica.

$$\bar{\epsilon}_a = |\bar{F}_r - F| \quad (7.2)$$

$$\bar{\epsilon}_r = \left| \frac{\bar{F}_r - F}{F} \right| \cdot 100 \quad (7.3)$$

Los datos se representarán en tablas con los siguientes datos:

- F (Hz): Frecuencia en herzios especificada en la interfaz del programa.
- T_{total}: Tiempo total de la adquisición de datos, medido en segundos.
- N° muestras: Número de muestras realizadas en la adquisición.
- ΔT_{medio}: Intervalo medio entre medidas, especificado en segundos.
- \bar{F}_{Real} : Frecuencia real a la que se ha realizado la adquisición, calculado por la Ecuación 7.1.
- $\bar{\epsilon}_a$: Error absoluto de la frecuencia, calculado por la Ecuación 7.2 y medido en Hz.
- $\bar{\epsilon}_r$: Error relativo de la frecuencia, calculado por la Ecuación 7.3, en porcentaje (%).

7.1.1. Carga del sistema.

La carga del sistema se ha medido mediante el comando *uptime*¹ y anotando el segundo valor de carga al representar la misma durante los últimos cinco minutos.

Por realizar una explicación rápida de como se mide la carga del sistema, si el valor obtenido es de 0.60, el sistema ha estado de media un 40 % del tiempo libre de carga. En cambio si es por ejemplo de 1.60, el sistema ha tenido de media un 60 % de sobrecarga, y quiere decir al ser este un sistema monoprocesador que 0.60 procesos de media han tenido que esperar para obtener tiempo de CPU[16]. Como nota, la carga del sistema en *idle* (sin utilizar) es de 0.11, por tanto un 11 % del tiempo de procesamiento lo utiliza el sistema operativo.

No se debe confundir la sobrecarga con que el sistema no es capaz de realizar la tarea especificada o que las restricciones temporales no se van a poder cumplir. Como se puede comprobar más adelante en las pruebas, el uso intensivo de la CPU por el programa no ha provocado medidas imprecisas.

7.1.2. ¿Cuándo una medida no es válida y el sistema ha tocado su límite?

Se ha decidido que una frecuencia deja de ser precisa a partir del 0,25 % de error relativo (ϵ_r). A partir de la adquisición tachada como no válida se realiza una o dos medidas más y se comparan las cargas en el sistema en ambos casos para investigar la causa del error.

7.1.3. ¿De dónde puede proceder el límite?

Los orígenes que se toman de la limitación pueden ser bien de la propia capacidad de cómputo de la Raspberry Pi, o del sistema de entrada/salida o el mismo hardware que no es capaz de entregar los datos al ritmo que se exigen. Para discernir entre ambas posibilidades hemos obtenido los siguientes niveles de carga que tiene nuestro programa sin utilizar el interfaz de adquisición mediante el uso del simulador funcionando a la máxima velocidad que nos permite el sistema:

¹<http://en.wikipedia.org/wiki/Uptime#Linux> (en inglés)

Número de canales activos	Carga del sistema
1	2.02
2	2.40
4	4.92

Tabla 7.1: Cargas máximas en el sistema sin utilizar el interfaz de adquisición.

Por otra parte, se ha comprobado empíricamente que la carga total por la que el sistema se queda totalmente congelado e inusable es a partir de aproximadamente 5.00.

Con estos valores, si la carga del sistema durante la adquisición de datos en los casos donde el error no es tolerable es similar o superior a la determinada en la Tabla 7.1, el cuello de botella lo estará creando la Raspberry; mientras que si no es así el cuello de botella estará en el sistema de entrada/salida o el hardware de adquisición de datos, y el sistema aún tendrá recursos para ir más rápido.

Destacar que la carga del sistema es un valor muy abstracto y variable, la única manera de llegar a una conclusión es mediante la realización de todas las pruebas y la puesta en común de las mismas, que se realiza al final del capítulo.

7.2. Pruebas con un solo canal.

Las primeras pruebas se realizan con utilizando un solo canal adquiriendo datos a distintas frecuencias. Se recogen los resultados en la Tabla 7.2.

Como se puede observar, las adquisiciones tienen un error relativo tolerable hasta los 1000Hz (por debajo del 1 %) habiendo siendo el pico más alto de un 0,16 % dado en los 500Hz y seguramente por algún fallo puntual, puesto que en la siguiente medida de 1000Hz el error es bastante inferior (0,015 %).

No obstante, a partir del kilohertzio hay errores del 24 % en 1500Hz y del 43 % en 2000Hz, dejando frecuencias medias de 1135,8Hz y 1127,1Hz reales respectivamente, errores ya por tanto no tolerables.

F (Hz)	T _{total} (s)	Nº muestras	ΔT_{medio} (s)	\bar{F}_{Real} (Hz)	$\bar{\epsilon}_a$	$\bar{\epsilon}_r$ (%)
1	358,0001	358	1,000000	0,9999	3,7454E-7	3,7454E-5
2	324,0001	648	0,500000	1,9999	9,0651E-7	4,5325E-5
5	310,2001	1551	0,200000	4,9999	2,2781E-6	4,5562E-5
10	302,5014	3025	0,100000	9,9999	4,8082E-5	0,000480
50	340,2401	17012	0,020000	49,9999	1,8537E-5	3,7075E-5
100	304,9405	30494	0,010000	99,9998	0,000168	0,000168
500	308,0007	153750	0,002000	499,1871	0,812882	0,162576
1000	312,9217	312872	0,001000	999,8411	0,158891	0,015889
1250	348,1531	393899	0,000883	1131,392	118,6071	9,488575
1500	320,4741	364017	0,000880	1135,869	364,1300	24,27533
2000	333,6778	376117	0,000887	1127,186	872,8139	43,64069

Tabla 7.2: Resultados de los tests utilizando un solo canal.

7.2.1. Cargas del sistema en cada caso.

En la Tabla 7.3 se comparan los errores con la carga del sistema.

F (Hz)	$\bar{\epsilon}_a$	$\bar{\epsilon}_r$ (%)	Carga del sistema
1	3,7454E-7	3,7454E-5	0.38
2	9,0651E-7	4,5325E-5	0.37
5	2,2781E-6	4,5562E-5	0.32
10	4,8082E-5	0,0004808	0.35
50	1,8537E-5	3,7075E-5	0.37
100	0,0001680	0,0001680	0.52
500	0,8128820	0,1625764	0.71
1000	0,1588911	0,0158891	1.45
1250	118,60719	9,4885755	1.55
1500	364,13002	24,275335	1.68
2000	872,81390	43,640695	1.69

Tabla 7.3: Comparación del error con la carga del sistema al utilizar un solo canal.

Entre 1 y 100Hz la carga es prácticamente inexistente, el sistema funciona ligero y con más del 50 % de ciclos libres.

A partir de 100Hz la carga empieza a ser más notable y a crecer conforme las frecuencias suben, y a los 1000Hz el sistema lleva ya un 45 % de sobrecarga. De todos modos, la sobrecarga no es alarmante ya que el error relativo sigue siendo bajo, indicando que las restricciones temporales se cubren razonablemente.

Analizando los tres últimos valores testados donde los errores han sido muy altos, 1250Hz, 1500Hz y 2000Hz, las cargas son muy similares y por debajo de los 2.02 que consideramos en 7.1 que podría correr el programa sin problema en este tipo de prueba. Por tanto se considera que el cuello de botella en este test no se encuentra en el programa por una carga excesiva en la CPU si no en la entrada/salida o el hardware de adquisición.

7.3. Pruebas con dos canales simultáneamente.

El siguiente test es con dos canales adquiriendo datos cuyos resultados se pueden ver en la Tabla 7.4.

F (Hz)	T _{total} (s)	Nº muestras	ΔT_{medio} (s)	\bar{F}_{Real} (Hz)	$\bar{\epsilon}_a$	$\bar{\epsilon}_r$ (%)
1	329,0002	328	1,00000	0,9999	8,8734E-7	8,8734E-5
	329,0002	328	1,00000	0,9999	7,2321E-7	7,2321E-5
2	317,0001	634	0,50000	1,9999	8,2384E-7	4,1192E-5
	316,5001	633	0,50000	1,9999	7,6650E-7	3,8325E-5
...						
500	325,1106	162551	0,00200	499,9867	0,0132585	0,0026517
	324,6987	162349	0,00200	499,9987	0,0012213	0,0002442
750	319,3583	199363	0,00160	624,2611	125,7388	16,76517
	318,8745	201618	0,00158	632,2799	117,7200	15,69601
1000	349,3195	219508	0,00159	628,3873	371,6126	37,16126
	348,9656	216182	0,00161	619,4936	380,5063	38,05063

Tabla 7.4: Resultados de los tests utilizando dos canales.

En este caso el error relativo se dispara a partir de los 500Hz, teniendo en los tests

a 750Hz una frecuencia real de algo más de 624Hz y 632Hz en cada canal, sugiriendo que el límite para este tipo de adquisición se encuentra alrededor de esos valores.. En los casos precedentes el error sigue siendo contenido y satisfactorio dada la naturaleza de propósito general de este sistema. Se han ignorado los valores entre 2Hz y 500Hz al ser todos muy parecidos y no aportar nada al lector.

7.3.1. Cargas del sistema en cada caso.

En la Tabla 7.5 se comparan los resultados con las cargas del sistema.

F (Hz)	$\bar{\epsilon}_a$	$\bar{\epsilon}_r$ (%)	Carga del sistema
1	8,8734E-7 7,2321E-7	8,8734E-5 7,2321E-5	0.24
2	8,2384E-7 7,6650E-7	4,1192E-5 3,8325E-5	0.32
5	2,7614E-6 2,7165E-6	5,5228E-5 5,4330E-5	0.32
10	4,4472E-6 2,1952E-5	4,4472E-5 0,0002195	0.33
50	2,3018E-5 0,0001063	4,6037E-5 0,00021274	0.43
100	4,9838E-5 0,0007617	4,98389E-5 0,00076173	0.56
500	0,0132585 0,0012213	0,00265171 0,00024427	1.36
750	125,7388 117,7200	16,76517 15,69601	1.80
1000	371,6126 380,5063	37,16126 38,05063	2.08

Tabla 7.5: Comparación del error con la carga del sistema al utilizar dos canales.

El sistema comienza a sobrecargarse en los 500Hz pero al igual que en las pruebas

anteriores, este 36 % extra de procesamiento no compromete la frecuencia a la que se toman los datos ya que los errores siguen siendo muy bajos.

No obstante, los dos valores donde el error se dispara, 750Hz y 1000Hz, la sobrecarga alcanza en este último caso el 108 % frente a los 140 % estimados en 7.1 y puede indicar que en este caso la capacidad de la propia Raspberry Pi es la limitante. Pero volviendo a los datos sobre la frecuencia media real de estos dos casos en la Tabla 7.4 se puede observar que la suma de la misma en cada caso son 1256,541Hz y 1247,880Hz, cuando el caso límite de las pruebas con un solo canal arrojaron una frecuencia límite de 1135,8Hz. Los valores son muy próximos entre sí por lo que realmente no queda claro si puede ser una cosa u otra.

7.4. Pruebas con cuatro canales simultáneamente.

Las últimas pruebas corresponden a las realizadas con cuatro canales adquiriendo datos simultáneamente y cuyos resultados se pueden ver en la Tabla 7.6.

Se han vuelto a ignorar los resultados entre 2Hz y 250Hz al no ser útiles para el lector. En este caso el funcionamiento es aceptable hasta el test de los 500Hz, donde el error supera el 35 % en todos los canales y la frecuencia media real es de casi 317Hz en el canal más lento.

Se puede comprobar como parece haber un límite global en la frecuencia en la que el interfaz puede funcionar ya que la suma de todas las frecuencias medias reales obtenidas en 500Hz es 1284,06Hz y a 750Hz es 1286,831Hz, tal y como se sugirió en la prueba anterior obteniendo un límite máximo de 1256,541Hz y en la primera prueba con 1135,8Hz.

F (Hz)	T _{total} (s)	Nº muestras	ΔT_{medio} (s)	\bar{F}_{Real} (Hz)	$\bar{\epsilon}_a$	$\bar{\epsilon}_r$ (%)
1	322,0001	323	1,00000	0,9999	5,6303E-7	5,6303E-5
	322,0001	323	1,00000	0,9999	4,5744E-7	4,5744E-5
	322,0001	323	1,00000	0,9999	4,8540E-7	4,8540E-5
	322,0001	323	1,00000	0,9999	4,6055E-7	4,6055E-5
2	322,5001	646	0,50000	1,9999	1,0587E-6	5,2939E-5
	322,5001	646	0,50000	1,9999	1,2282E-6	6,1413E-5
	322,5001	646	0,50000	1,9999	1,0735E-6	0,0000536
	322,5001	646	0,50000	1,9999	9,4368E-7	4,7184E-5
. . .						
250	327,0160	81738	0,00400	249,9479	0,0520258	0,0208103
	327,0132	81739	0,00400	249,9532	0,0467881	0,0187152
	327,0402	81761	0,00400	249,9997	0,0002219	8,8778E-5
	327,0321	81759	0,00400	249,9998	0,0001128	4,5144E-5
500	347,3804	112507	0,00308	323,8696	176,13030	35,22606
	347,3536	111828	0,00310	321,9398	178,06011	35,61202
	347,3543	111592	0,00311	321,2598	178,74013	35,74802
	347,8928	110281	0,00315	316,9941	183,00588	36,60117
750	335,8788	107725	0,00311	320,7228	429,27716	57,23695
	335,9379	107917	0,00311	321,2379	428,76208	57,16827
	335,9450	108357	0,00310	322,5408	427,45914	56,99455
	335,9791	108291	0,00310	322,3116	427,68835	57,02511

Tabla 7.6: Resultados de los tests utilizando los cuatro canales.

7.4.1. Cargas del sistema en cada caso.

En la Tabla 7.7 se puede ver la comparación de los errores con la carga del sistema (C_S) en cada caso.

Con cuatro canales no parece ser diferente al resto de pruebas con menos canales y aunque en los casos donde el error es alto la carga en el sistema también es importante, no llega al nivel máximo especificado en la Tabla 7.1 (esta vez 2.36 frente a 4.92), y en

ambos casos es similar entre sí (2.33 y 2.36).

Una vez más, no es la capacidad de cálculo la que nos está limitando la frecuencia de los canales. Esta vez se ve más claramente ya que si se hace el recuento de hilos que se están ejecutando (un hilo por cada canal de adquisición, otro hilo de actualización de la gráfica y el hilo principal) no puede ser que el sistema esté sobrecargado cuando solo 1.36 procesos tienen que esperar tiempo de CPU y hay al menos seis ejecutándose en nuestro programa.

F (Hz)	$\bar{\epsilon}_a$	$\bar{\epsilon}_r$ (%)	C _S	F (Hz)	$\bar{\epsilon}_a$	$\bar{\epsilon}_r$ (%)	C _S
1	5,6303E-7	5,6303E-5	0.26	100	0,0022902	0,0022902	1.64
	4,5744E-7	4,5744E-5			6,6360249	6,6360E-5	
	4,8540E-7	4,8540E-5			0,0072389	0,0072389	
	4,6055E-7	4,6055E-5			0,0002015	0,0002015	
2	1,0587E-6	5,2939E-5	0.35	250	0,0520258	0,0208103	2.12
	1,2282E-6	6,1413E-5			0,0467881	0,0187152	
	1,0735E-6	0,0000536			0,0002219	8,8778E-5	
	9,4368E-7	4,7184E-5			0,0001128	4,5144E-5	
5	1,7358E-6	3,4716E-5	0.38	500	176,13030	35,22606	2.33
	9,3425E-6	0,0001868			178,06011	35,61202	
	4,5575E-6	9,1151E-5			178,74013	35,74802	
	3,0716E-6	6,1433E-5			183,00588	36,60117	
10	4,8712E-6	4,8712E-5	0.56	750	429,27716	57,23695	2.36
	4,2986E-6	4,2986E-5			428,76208	57,16827	
	1,1686E-5	0,0001168			427,45914	56,99455	
	2,2300E-5	0,0002230			427,68835	57,02511	
50	9,3884E-5	0,0001877	0.63				
	0,0000194	3,8860E-5					
	1,9863E-5	3,9726E-5					
	0,0018233	0,0036467					

Tabla 7.7: Comparación del error con la carga del sistema al utilizar los cuatro canales.

7.5. Puesta en común de los resultados y conclusión.

7.5.1. El límite.

En las pruebas se ha detectado un patrón de limitación por el que el interfaz tiene una frecuencia máxima total a la que puede funcionar en todo caso independientemente del número de canales.

Canales	$\sum F_{max}$
1	1135.869
2	1256.541
4	1286.831

Tabla 7.8: F_{max} por canal.

Como se puede ver en la Tabla 7.8, el interfaz alcanza un umbral cercano a los 1300Hz del que no es capaz de pasar.

Se puede asegurar por tanto aplicando una margen de error del 10 % en el peor de los valores máximos obtenidos que esta solución funciona en unos umbrales de precisión seguros siempre y cuando la suma de las frecuencias de todos los canales no supere los 1000Hz aproximadamente.

7.5.2. El porqué del límite.

Descartada la limitación de la CPU (conclusión del test de cuatro canales en 7.4.1) y comprobado que el sistema admite más carga de la que se está obteniendo en las pruebas con el interfaz (ver Tabla 7.1) se achacan la falta de rendimiento al sistema de entrada/salida que parece no proveer de los datos a la velocidad necesaria a partir de dichas frecuencias y por tanto hace al software esperar ocioso.

Por tanto, se debe proceder al cambio de interfaz E/S o la optimización del mismo si se desean realizar medidas a frecuencias por encima de las obtenidas en estas pruebas.

“Ningún conocimiento humano puede ir más allá de su experiencia.”

John Locke, filósofo.

Capítulo 8

Conclusiones y trabajos futuros.

8.1. Conclusiones personales.

Desde el principio he querido realizar un proyecto relacionado de alguna manera con una Raspberry Pi o un SBC con el objetivo de demostrar que muchas de las soluciones informáticas que hay en el mundo real en distintos ámbitos, especialmente en sistemas embebidos, son excesivas y pueden realizarse con presupuestos muy inferiores. Esto en parte es posible gracias al *Open Source* tanto referido al software como al hardware y la Raspberry Pi es totalmente libre en los dos ámbitos.

Este proyecto ha demostrado que efectivamente es posible abaratar costes y realizar soluciones viables mediante el uso de estos ordenadores de bajo coste, además de suponerme un desafío en general. No solo porque es el proyecto de fin de carrera donde uno ha de poner en práctica todos los conocimientos adquiridos durante estos años, sino porque además he tenido que convivir con otro proyecto paralelo con el que se perseguía un objetivo común.

Ésto ha provocado un montón de acondicionamiento e incertidumbre durante todo el proceso. Pero como en todo trabajo en equipo, la buena comunicación y la ayuda mutua han cumplido su cometido y hemos podido cumplir no todos, pero gran parte de los objetivos cuando en las primeras reuniones no se tenía claro si realmente esta solución iba a ser viable.

Es otra de las razones que han hecho de este proyecto un desafío: No se sabía

exactamente si la plataforma iba a ser capaz de sustituir a un equipo mucho más caro, lo cual lo ha provocado que una buena parte del mismo haya sido un estudio de viabilidad e investigación.

Personalmente estoy contento con el resultado. Si bien es un trabajo que puede llegar mucho más lejos, se ha demostrado lo que quería probar en primer lugar, y en el proceso he podido aprender un método de trabajo nuevo, un lenguaje de programación importante como Python, y además en la realización de esta memoria también he conocido L^AT_EXy gran parte de su potencia.

8.2. Conclusiones profesionales.

El trabajo colaborativo ha sido sin duda el pilar más importante de este proyecto, puesto que de otra manera no habría sido posible llegar a donde se ha llegado. Ambos proyectandos provenimos de distintas carreras, yo por una parte de Ingeniería Informática y mi compañero de Ingeniería de Telecomunicaciones. Cada uno se ha centrado en su campo de estudio y ha sacado su parte adelante respetando la opuesta para que al final la unión de ambas haya sido un éxito.

8.3. Trabajos futuros.

8.3.1. Nuevas funcionalidades..

Además de las funcionalidades que no se han desarrollado por no ser un objetivo del proyecto pero se han propuesto y en ocasiones preparado a falta de escribir el modelo como la exportación PDF o la utilización del mismo en otros sistemas operativos, este programa puede albergar otras funcionalidades como por ejemplo la obtención de estadísticos con los datos obtenidos.

8.3.2. Optimización de la solución.

Aunque el rendimiento actual es aceptable y muy válido, siempre se puede optimizar por ejemplo utilizando una versión del sistema operativo para tiempo real o modificando

éste para que la integración en el mismo sea total, recortando programas y servicios que no resulten útiles para el propósito de este programa.

También es posible conseguir una mejora a través de un estudio de la optimización del diseño de la aplicación o portando la solución a lenguajes más eficientes como puede ser C.

8.3.3. Adaptación a otras SBCs.

La Raspberry Pi es realmente la SBC menos potente de todas, sería interesante hacer el programa compatible con su clon de doble núcleo la Banana Pi.

Se dispuso de una unidad durante el desarrollo y se comprobó que el funcionamiento el programa y todo el sistema en general era muchísimo más ligero. Solo se pudo probar el funcionamiento con el simulador de datos ya que la librería piDA no está adaptada a trabajar con una Banana Pi, pero una vez realizada esa adaptación sería interesante ver de qué es capaz esta placa con toda su potencia y tan solo 20€ más de sobreprecio con respecto a la Raspberry Pi. Es muy probable que resulte en una solución más viable.

8.3.4. Integración del producto final en los laboratorios de la Facultad.

La solución funcional podría integrarse a pequeña escala en los laboratorios de la Facultad con el fin de que los alumnos hagan uso de ella. Se puede evaluar así la utilidad de la misma y la calidad en el propio entorno para el que ha sido diseñada, obteniendo así comentarios y opiniones del usuario final muy útiles.

8.3.5. Explotación comercial.

Sí la integración en los laboratorios de la Facultad es un éxito, se puede dar el paso a crear un producto comercial. La explotación del mismo empezaría por una producción industrial de la interfaz de adquisición de datos y su venta vía web.

Si realmente es un producto funcional y supone un ahorro tan grande con respecto a las soluciones de la competencia puede resultar una fuente de ingresos que remunere el esfuerzo y la idea que ha habido tras el mismo.

Bibliografía

- [1] eLinux.org, *Raspberry Pi Verified Peripherals*. VGA Cables and converter boxes.
[Fecha de consulta: 15 Febrero 2014]
http://elinux.org/RPi_VerifiedPeripherals#HDMI-.3EVGA_Cables
- [2] Amazon.es, *Departamento de informática*. Tarjeta de memoria SDHC 4GB.
[Fecha de consulta: 22 Agosto 2014]
<http://www.amazon.es/Transcend-TS4GSDHC10-Tarjeta-memoria-SecureDigital/dp/B002WE6CTU/>
- [3] DHgate.com, *China Wholesale*. HDMI to VGA Output Projector Monitors Adapter. [Fecha de consulta: 18 Febrero 2014]
<http://www.dhgate.com/product/hdmi-to-vga-output-projector-monitors-adapter/170509392.html>
- [4] Amazon.es, *Departamento de informática*. Genius - Teclado Kb-110X Black Usb.
[Fecha de consulta: 22 Agosto 2014]
<http://www.amazon.es/Genius-Teclado-Kb-110X-Black-Usb/dp/B005E08ADU/>
- [5] Amazon.es, *Departamento de informática*. Genius XScroll - Ratón óptico con cable USB. [Fecha de consulta: 22 Agosto 2014]
<http://www.amazon.es/Genius-XScroll-Raton-Optico-cable/dp/B000EHPPZ4/>
- [6] Amazon.es, *Departamento de electrónica*. Samsung - Cargador de móvil. [Fecha de consulta: 22 Agosto 2014]

- <http://www.amazon.es/Samsung-ETA0U10EBECSTD-Cargador-movil-red/dp/B004S6NUNG/>
- [7] Python.org, *The Python Wiki*. Integrated Development Environments. [Fecha de consulta: 5 Marzo 2014]
<https://wiki.python.org/moin/IntegratedDevelopmentEnvironments>
- [8] Python.org, *The Python Wiki*. GUI Programming in Python. [Fecha de consulta: 5 Marzo 2014]
<https://wiki.python.org/moin/GuiProgramming>
- [9] Effbot.org, *Tkinterbook*. An Introduction To Tkinter. [Fecha de consulta: 5 Marzo 2014]
<http://effbot.org/tkinterbook/tkinter-index.htm>
- [10] WxWidgets, *A Cross-Platform GUI Library*. [Fecha de consulta: 1 Abril 2014]
<http://www.wxwidgets.org/>
- [11] WxPython.org, *API Documentation*. WX Package. [Fecha de consulta: 1 Abril 2014]
<http://www.wxpython.org/docs/api/wx-module.html>
- [12] WxPython.org, *Phoenix API Documentation*. Core Classes. [Fecha de consulta: 9 Abril 2014]
<http://wxpython.org/Phoenix/docs/html/1classindex.html>
- [13] Python.org, *The Python Wiki*. Graphical Representations of Data. [Fecha de consulta: 20 Marzo 2014]
<https://wiki.python.org/moin/NumericAndScientific/Plotting>
- [14] Matplotlib.org, *Python 2D plotting library*. [Fecha de consulta: 20 Marzo 2014]
<http://matplotlib.org/>
- [15] Adobe.com, *Adobe Creative Cloud*. Adobe Fireworks. [Fecha de consulta: 22 Julio 2014]
<https://creative.adobe.com/products/fireworks>

- [16] Wikipedia.org, *Load (computing)*. Unix-style load calculation. [Fecha de consulta: 15 Agosto 2014]
[http://en.wikipedia.org/wiki/Load_\(computing\)#Unix-style_load_calculation](http://en.wikipedia.org/wiki/Load_(computing)#Unix-style_load_calculation)
- [17] Facultad de Ciencias Exactas y Naturales de la Universidad de Antioquia, *Laboratorio de Física*, Cuantificación de errores. [Fecha de consulta: 15 Agosto 2014]
http://fisica.udea.edu.co/~lab-gicm/Labradorio_Fisica_1_2012/2012_Cuantificacion%20de%20errores.pdf