industriales
etsii

Escuela Técnica Superior de Ingeniería Industrial

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería Industrial

Monitor de parámetros clínicos en código abierto: interfaz de usuario /Adquisición de datos

TRABAJO FIN DE GRADO

GRADO EN ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Autor: **José Manuel Angosto Fernández**

Director: **Joaquín Roca González**

Cartagena, ……

Universidad Politécnica de Cartagena

Agradecimientos

A Joaquín Roca González, director de este TFG, por haberme dado la oportunidad de llevar a cabo un trabajo relacionado con la salud y el deporte

A mi madre Isabel y a mi padre José por haberme apoyado desde el primer día en el que comencé esta aventura de la ingeniería.

ÍNDICE

ÍNDICE DE TABLAS

1. ESTUDIO DEL ESTADO DEL ARTE Y OBJETIVOS DEL PROYECTO

1.1. MEDIDA DE PARÁMETROS CLÍNICOS. FRECUENCIA CARDIACA.

La medida de parámetros clínicos es una labor esencial en el tratamiento de un paciente, ya sea una persona hospitalizada, un deportista que quiera hacer un estudio de su estado físico, etc. Para llevar a cabo dicha tarea, existen equipos de medida de VO2 máximo, pulsaciones, presión sanguínea y otros parámetros de interés. A partir de estos datos, se almacena información del paciente. Posteriormente se podrá realizar un diagnóstico de una enfermedad, establecer unos parámetros de entrenamiento, además de poder ser utilizados, en tiempo real, durante una operación.

De todos los parámetros, uno de los que mayor relevancia tiene es la frecuencia cardiaca (HR o bpm, por sus siglas en inglés). Es el objeto de estudio en este proyecto. Tiene un amplio campo de aplicación, desde un quirófano hasta el entrenamiento deportivo. Es fácilmente medible. Por ejemplo, los pulsómetros utilizan un dispositivo unido a una banda elástica, la cual se coloca en el pecho, mide las pulsaciones y se comunica por Bluetooth con un reloj o con un teléfono móvil.

1.2. FRECUENCIA CARDIACA, RASPBERRY PI Y PYTHON.

Otros proyectos anteriores han abordado la medida y estudio de la frecuencia cardiaca. Se han desarrollado programas para el tratamiento de señales de ese tipo mediante el uso, ya sea de Raspberry Pi como plataforma principal, o del lenguaje de programación Python. Existen otros que utilizan Arduino, en vez de una Raspberry. Respecto al lenguaje de programación, también se han utilizado otros como C++ o Java. Por ello, se puede encontrar cierta información acerca de proyectos de bajo coste, cuyo fin es la medida de parámetros clínicos y su representación gráfica.

En cuanto a la medida de las pulsaciones, se han utilizado dispositivos de medida de frecuencia diseñados específicamente para trabajar con Arduino o Raspberry Pi, tales como e-Health Platform, y también bandas de medida de frecuencia cardíaca junto con receptores de Bluetooth, como es el caso del dispositivo Polar T34 y el receptor T31, que es un dispositivo Polar WearLink.

A nivel software se han desarrollado aplicaciones para la representación de datos en varios lenguajes (Matlab, C++, Java, Python). En Python se cuenta con varias librerías para la representación gráfica: Maplotlib (representación idéntica a Matlab), WxPython, PyQtGrpah, Chaco, etc. Uno de los problemas que se plantea a la hora de elegir un

lenguaje u otro, al igual que al elegir una librería de representación gráfica, es la capacidad de trabajo, de esta, en tiempo real.

En el caso de combinar el uso de Raspberry Pi y Python para desarrollar este tipo de proyectos, no se encuentran tantos proyectos, por lo que, en este aspecto, el trabajo a desarrollar es algo novedoso. La mayoría de los anteriores se desarrollaban con un ordenador o con Arduino, que tiene un lenguaje de programación propio.

El modelo de Raspberry Pi utilizado es el 2B. En comparación con la Raspberry Pi 3, no presenta ninguna desventaja considerable a la hora de desarrollar el presente proyecto. La versión del lenguaje de programación es Python2.7. La elección de esta versión y no la de una posterior, es porque las librerías que se van a utilizar están completamente desarrolladas y son estables en Python2.7.

Uno de los aspectos importantes de este proyecto es el uso de Bluetooth de baja energía, Bluetooth 4.0 o BLE (Bluetooth Low Energy). En vez de utilizar el puerto serie para comunicar la Raspberry con el dispositivo de medida, se utiliza un adaptador de BLE. No es compatible con el anterior protocolo Bluetooth, pero aporta mayor seguridad. También permite la transferencia de pequeñas cantidades de información a velocidad relativamente baja, pero consumiendo poca energía. El sensor de medida es el Polar H7, uno de los más utilizados en el mundo del deporte.

2. VISIÓN GENERAL DEL PROYECTO

2.1. HARDWARE

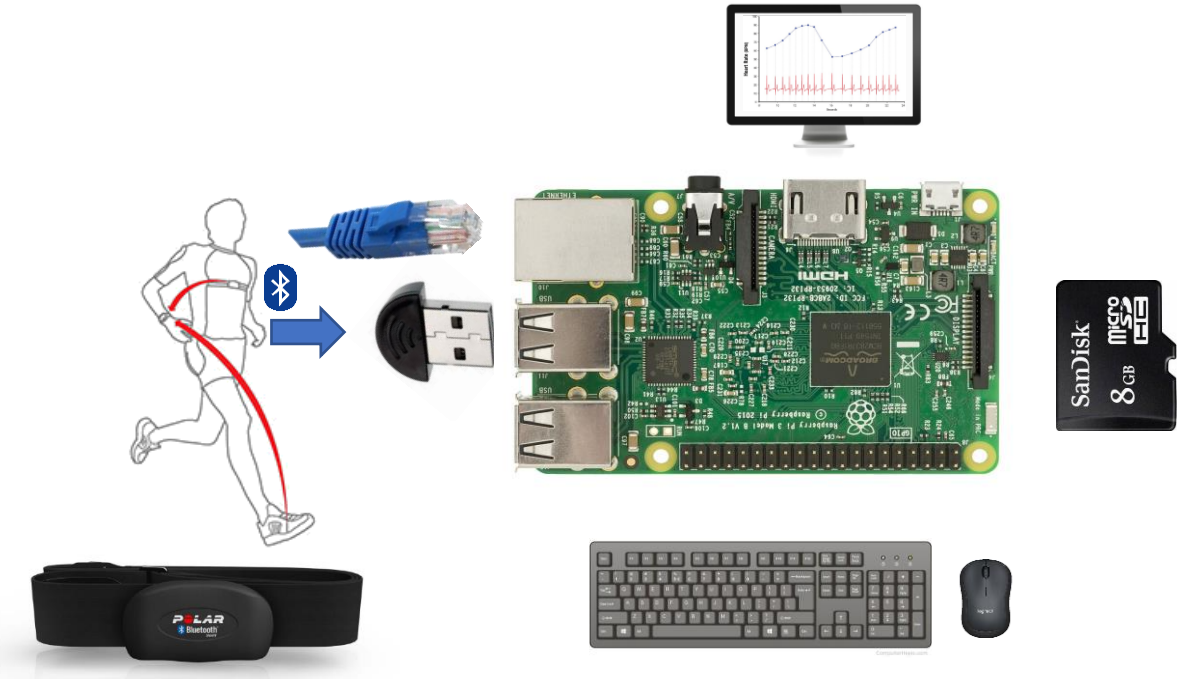

Figura 1. Vista general del Hardware del sistema.

En la imagen se pueden observar todos los componentes que van a formar parte del hardware del proyecto. En el centro aparece la Raspberry Pi y alrededor, los periféricos que se van a utilizar. Estos son: el sensor Polar H7, un sensor de Bluetooth 4.0, un cable de Ethernet, un monitor, una tarjera de memoria microSD de 8GB, un ratón y un teclado.

2.2. SOFTWARE

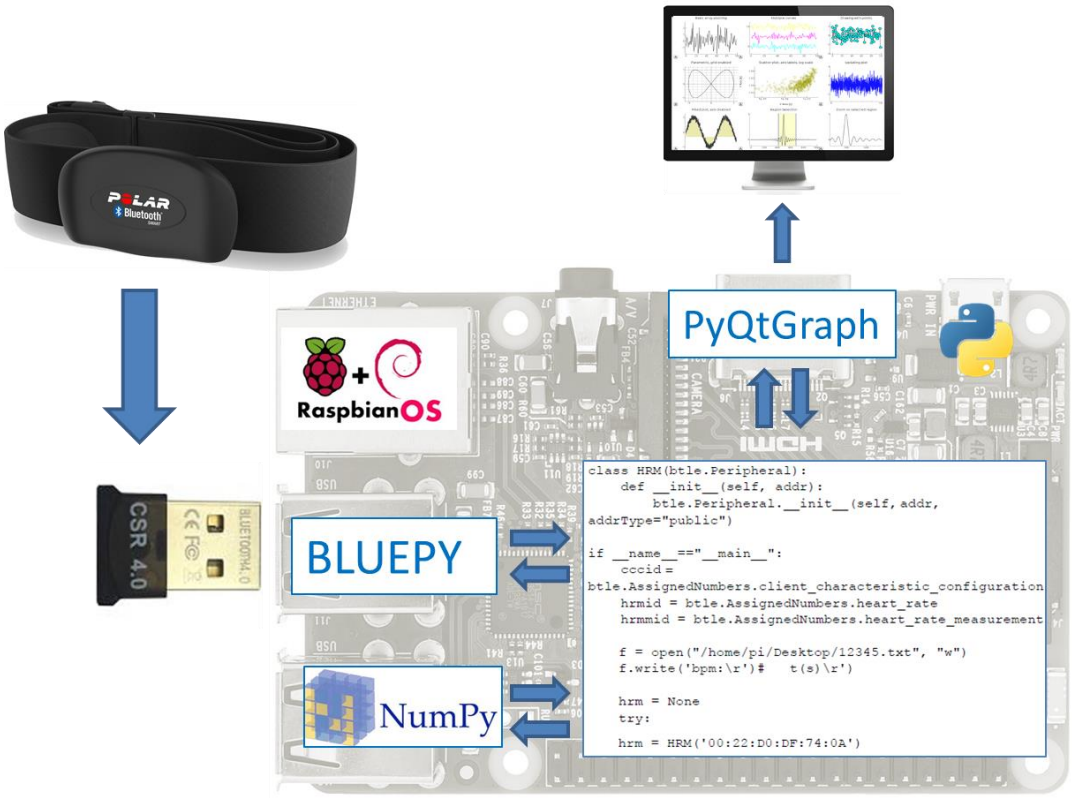

Figura 2. Vista general del Software del sistema

En la parte de software, dentro del lenguaje de programación Python, se va utilizar una serie de librería de comunicación serie, comunicación Bluetooth, procesamiento de datos y de programación multitarea. Algunas de ellas solo se utilizarán en las pruebas previas al programa final. En este, las necesarias son BluePy, PyQtGraph, NumPy, sys, time y datetime.

3. RASPBERRY PI

El proyecto se ha desarrollado utilizando una Raspberry Pi 2 modelo B. Esta placa es un computador de placa reducida (85 x 56 mm) desarrollado por la fundación Raspberry Pi de Reino Unido y que tiene un precio aproximado de 35€. Entre sus características principales se pueden encontrar las siguientes:

- Conexión a red mediante Ethernet 10/100 (RJ-45)
- Juego de instrucciones RISC de 32 bits
- CPU de 900MHz quad-core ARM Cortex A7
- Cuatro puertos USB 2.0
- Almacenamiento de información en MicroSD
- Consumo energético de 800 mA (4 W)

Figura 3. Raspberry Pi 2 modelo B.

A diferencia de otras placas, cuenta con un sistema operativo propio, lo que le permite funcionar como un pequeño ordenador. Los más conocidos son Raspbian, RISC OS, Arch Linux ARM y Pidora

SoC

El SoC (System On a Chip) que utiliza es el BCM2836, aunque en versiones anteriores era el BCM2835. La SDRM (memoria dinámica síncrona de acceso aleatorio) está soldada al SoC.

Figura 4. BCM2836.

CPU

Es un microprocesador de la familia ARM Cortex-A7 de cuatro núcleos que forma parte del SoC. Es un microprocesador RISC para el cual el fabricante (Acorn Computers) asegura que trabaja a 900 MHz. El juego de instrucciones del que hace uso es el ARMv7.

GPU

La unidad de procesamiento gráfico es la Broadcam VideoCore IV, compatible con OpenMAX. Esta GPU es capaz de decodificar el MPEG-2, utilizado por la TNT y el DVD. Además, permite administrar una cámara HD en directo y codificar vídeo, pudiendo mostrar imágenes Full HD en 1080p30 (1920 x 1080 a 30 frames por segundo).

Memoria

Se divide entre la GPU y la CPU. La cantidad que se asigna a cada procesador depende de la distribución de Linux que se utilice. Las memorias utilizadas, suministradas por Samsung y Hynix, tienen una capacidad de dos y de cuatro gigabits. Son de tipo DDR2, frecuencia de transferencia doble, alimentadas con valores de voltaje entre 1,2 V y 1,8 V.

Puertos Ethernet y USB

El puerto USB no está conectado directamente a la toma USB, sino que lo está a un circuito integrado LAN9514. Este tiene cuatro tomas USB. El puerto Ethernet está conectado a un controlador Ethernet 10/100. Este puerto controla la configuración half o full-duplex y la auto-MDIX (Medium Dependan Interface Crossover).

Salidas de vídeo

La salida de vídeo usada en este proyecto es la de HDMI. Mediante esta conexión se permite que las señales de audio y vídeo viajen a alta velocidad y se disminuyan las interferencias.

GPIO

La GPIO (Entrada/salida de propósito general) es una interfaz que permite conectar la Raspberry Pi con el mundo exterior. Consta de 40 conectores divididos en dos filas de 20 cada una y con una separación de una décima de pulgada, 2,54 mm.

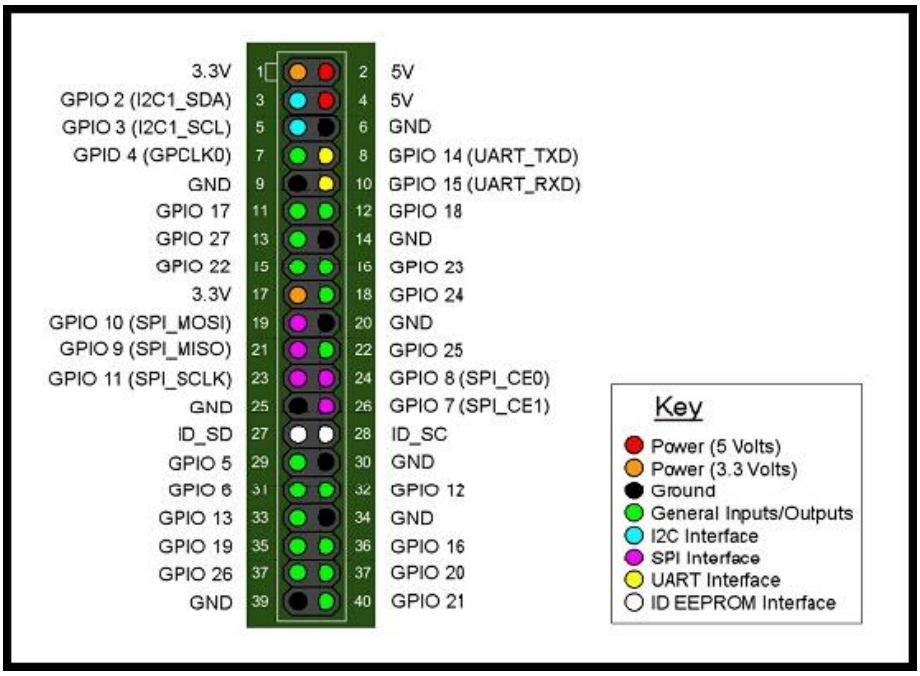

Figura 5. Disposición de los pines de la GPIO.

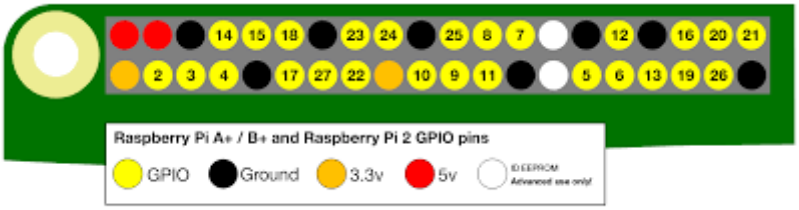

Figura 6. Distribución de los pines en la placa.

Las 40 clavijas se pueden reconfigurar. La 14 y la 15 se configuran al inicio para proporcionar las señales de UART. Podrían transformarse en clavijas GPIO estándares para proporcionar hasta 26 entradas/salidas. En cambio, la 27 y la 28 se reservan para la lectura de la EEPROM, presentes en las tarjetas de extensión.

Alimentación

Se realiza mediante una toma micro-USB. Lo que enciende o apaga la placa es la inserción o retirada de esta toma, ya que no existe ningún botón de encendido/apagado. El consumo de intensidad de la tarjeta ronda los 0,5 A, a lo que hay que sumar el consumo de periféricos en caso de que se añadan y las caídas de tensión, por lo que se recomienda una alimentación de 5 V y 1 A.

Conector de la tarjeta SD

Situado en la parte inferior de la tarjeta, presenta algunas variaciones respecto a versiones anteriores, que lo hacen más robusto y seguro. Cuenta con un sistema de bloqueo/desbloqueo que se acciona empujando la tarjeta. Es necesario que la placa este desenchufada para poder introducir y extraer la tarjeta, ya que en ella se encuentra el sistema operativo.

Raspberry Pi y Arduino

Existen otros tipos de placas, como Arduino, que podrían haberse utilizado para este proyecto. Sin embargo, la Raspberry Pi presentaba, en conjunto, unas características más adecuadas que las demás a la hora de afrontar este proyecto. En la siguiente tabla se presenta una comparativa entre las dos placas mencionadas:

	Raspberry Pi 2 modelo B	**Arduino Uno**
Sistema Operativo	Linux	Ninguno
Entornos de desarrollo	Linux (Debian, Raspbian, Arch Linux, Slackware), SUSE, Fedora	Arduino
Puertos USB 2.0	4	1
Conectividad de red	10/100 Ethernet RJ-45	Necesita un Shield para la conexión a red
Multitarea	Sí	No
Memoria Flash	Tarjeta MicroSD (2 a 16G)	32 KB
Voltaje de entrada	5 V	7 – 12 V
Frecuencia de reloj	900 MHz	16 MHz
Entradas/Salidas	40 Pines GPIO, USB, HDMI, RCA, audio de 3,5mm, bus HAT ID	14 entradas/salidas digitales, 6 entradas analógicas, 6 PWM, 1 UART

Tabla 1. Comparativa Raspberry pi 2 modelo B y Arduino Uno.

Otros aspectos que diferencian ambas placas es que Raspberry Pi es un microprocesador con memoria RAM de 512 Mb, mientras que Arduino tiene un microcontrolador ATMEGA 328. Además, para poder hacer uso de ciertas funciones, Arduino necesita de 'ampliaciones', las llamadas Shields. Este es el caso de la conexión a red. Haciendo una analogía: la primera puede hacer las funciones de un ordenador, mientras que la segunda podría equivaler a un autómata programable, pero ambos para operaciones no muy complicadas. Para proyectos de programación con gran cantidad de datos, Raspberry presenta mayor número de ventajas; para proyectos de electrónica, es decir, para el diseño de la parte hardware, utilizar Arduino es una mejor decisión. El proyecto actual se desarrolla en un entorno de software, más que de hardware, por tanto, es más conveniente utilizar la primera.

Dado que la Raspberry Pi puede funcionar como un pequeño ordenador, se programará el código directamente utilizando la misma placa. Para ello se conectarán todos los periféricos (teclado, monitor y ratón) a ella. El sistema operativo con el que se trabajará será Raspbian, que es el más común de los utilizados sobre la placa. El entorno de programación de Python, IDLE, está preinstalado en el sistema operativo.

3.1. RASPBIAN

Anteriormente se han mencionado sistemas operativos (SO) que soporta la Raspberry Pi. Como se ha dicho, el que se va a utilizar es Raspbian, concretamente la versión publicada el 2 de marzo de 2017.

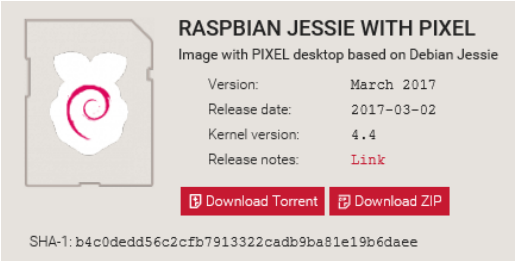

Figura 7. Versión de Raspbian instalada.

Este SO es una distribución de Linux basada en Debian Wheezy, desarrollada como solución al problema de que no existía un SO que se pudiera usar en la CPU ARMv6, la que incluía en un principio la placa Ahora es ARM Cortex A-7.

El escritorio usado es LXDE y el navegador de internet, Chromium. Cuenta con varios entornos de programación: BlueJ Java IDE, Grany Programmer's Editor, Greenfoot Java IDE, Mathematica, Node-RED, IDLE para Python2 y 3, Scratch, Sense HAT Emulator, Sonic Pi y Wolfram. También cuenta con LibreOffice (Editor de textos, hojas de cálculo, bases de datos).

Para la instalación, lo que se ha hecho ha sido descargar la versión nombrada anteriormente desde la página web de Raspberry. El enlace es el siguiente:

https://www.raspberrypi.org/downloads/raspbian/

Una vez hecho lo anterior se ha copiado el archivo de imagen que contenía la descarga en una tarjeta microSD de 8GB. El sistema operativo utilizado para instalar Raspbian ha sido Windows, por lo tanto, el programa para pasar el archivo de imagen ha sido Win32DiskImage. Al acabar, se ha introducido la tarjeta en el conector de la Raspberry y, a partir de este momento, ya se ha podido empezar a manejar la placa.

Al enchufar la Raspberry Pi a la corriente, por el monitor aparecen franjas de colores y el logo de la marca. Al ser la primera vez que se usa, aparece a continuación la herramienta 'raspi-config', la cual permite configurar la placa. Al inicio aparece solo la primera vez. Posteriormente, si se quiere volver a ejecutar, se debe escribir el comando `raspi-config` si se hace como adiministrador o `sudo raspi-config` si se hace como usuario normal.

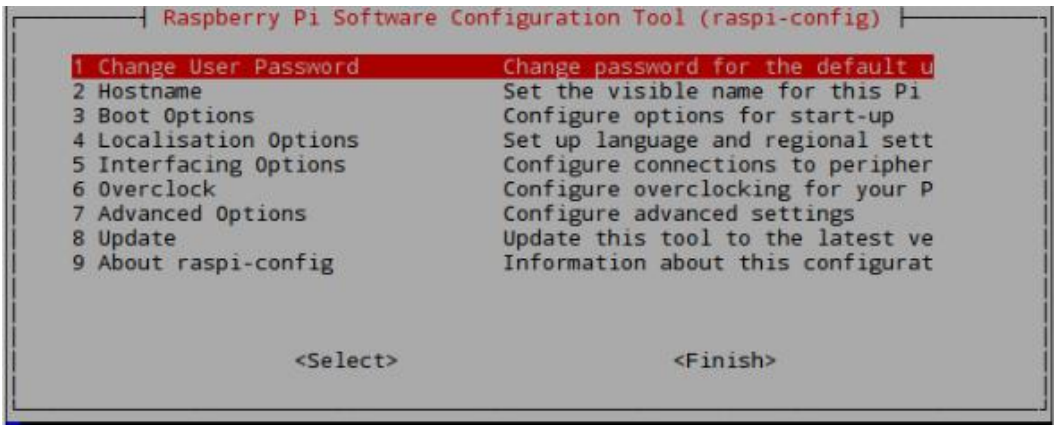

Figura 8. Raspi-config.

Para este proyecto se ha configurado la hora, el teclado, la contraseña para el acceso como administrador y el inicio en modo gráfico. Para lo primero, se han seguido los siguientes pasos:

1º.- Se selecciona la cuarta opción, 'Localisation Options'.

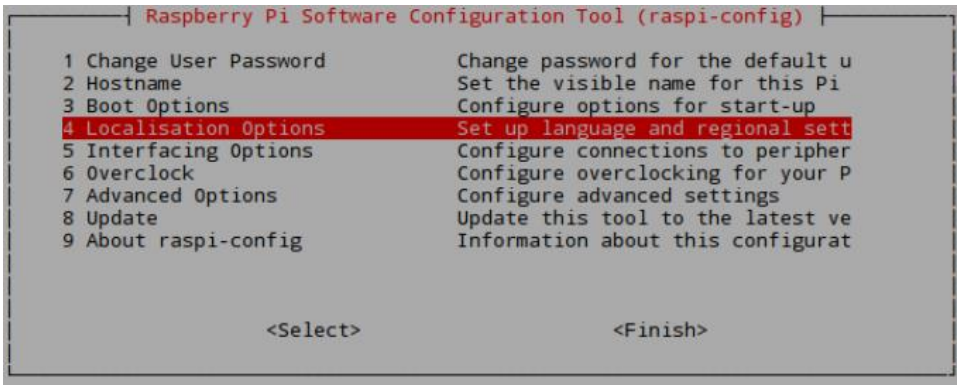

Figura 9. Raspi-config.

2º.- Ahora es la segunda, 'Change Timezone'.

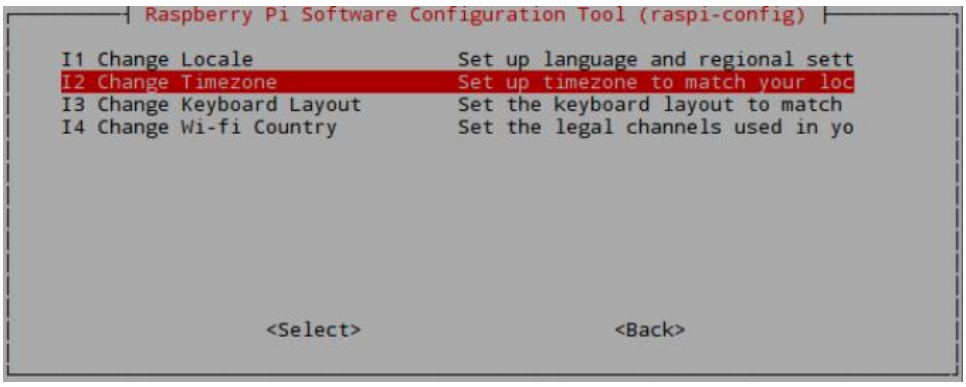

Figura 10. Raspi-config.

3º.- A continuación, aparece una serie de zonas geográficas. Se debe seleccionar la zona en la que se encuentre el usuario, en este caso, Europa.

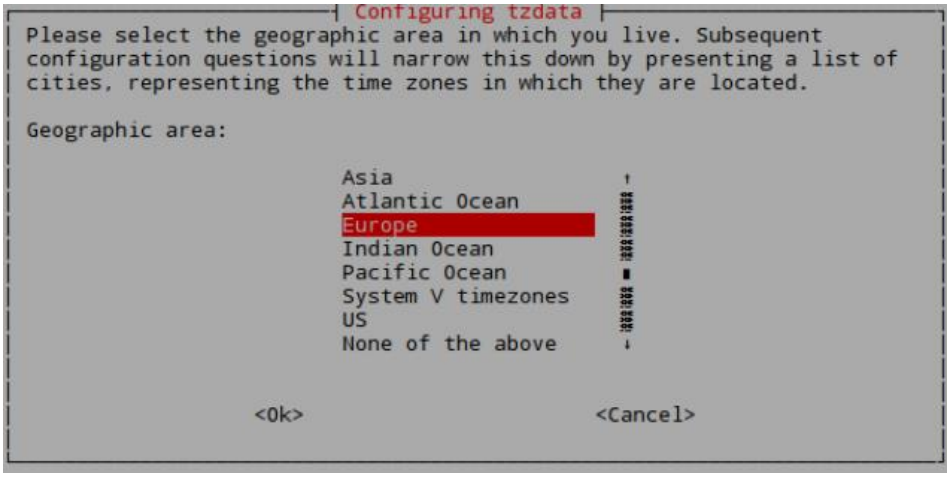

Figura 11. Raspi-config.

4º.- Por último, se selecciona la ciudad más cercana. En España coincide con la capital, Madrid.

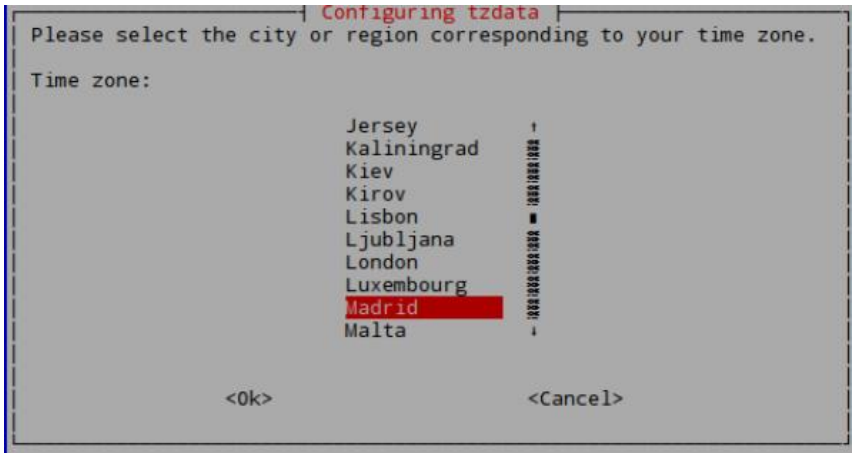

Figura 12. Raspi-config.

Después de la configuración horaria, se modifica el sistema para que se inicie en modo gráfico. Para ello se debe seleccionar la tercera opción del menú de inicio, 'Boot Options'. Al hacerlo aparecen otras tres opciones y se elige la primera, 'Desktop / CLI'.

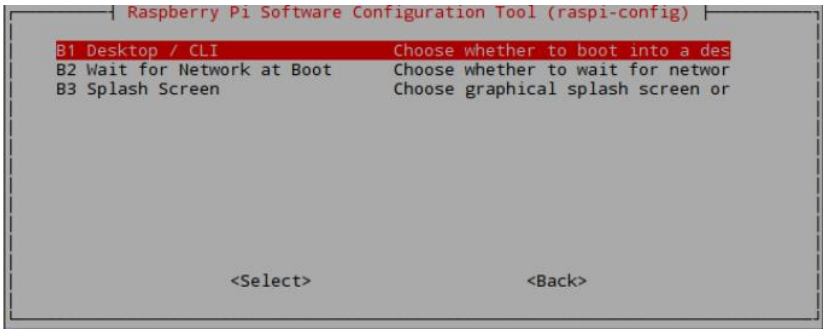

Figura 13. Raspi-config.

Por último, se muestran otras cuatro opciones.

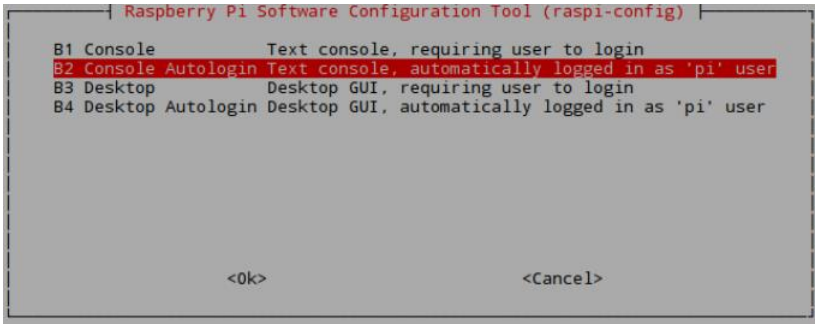

Figura 14. Raspi-config.

'Console' inicia el sistema en modo texto. Si se selecciona 'Console Autologin' el sistema comienza a funcionar en modo gráfico. En caso de optar por una de las otras dos opciones, 'Desktop' o 'Desktop Autologin', el sistema arranca en modo gráfico e inicia Scratch. Se ha optado por 'Console Autologin'.

El ajuste del teclado y el de la contraseña se han hecho una vez configurado el modo gráfico. El idioma del teclado se ha cambiado porque inicialmente está configurado en inglés. Para ello hay que acceder al menú del escritorio pulsando en el icono de Raspberry Pi situado en la zona superior izquierda de la pantalla. Una vez ahí se selecciona la pestaña 'Preferences' y se desplegarán otras seis pestañas. De estas, la que hay que elegir es 'Mouse and Keyboard Settings'. A continuación, aparece una ventana con las pestañas de 'Mouse' y 'Keyboard'. En la segunda se pulsa en 'Keyboard Layout y aparece una lista de países donde se debe buscar 'Spain' y seleccionar la configuración de teclado que se desea, ya que aparecen diversas alternativas de teclados en español.

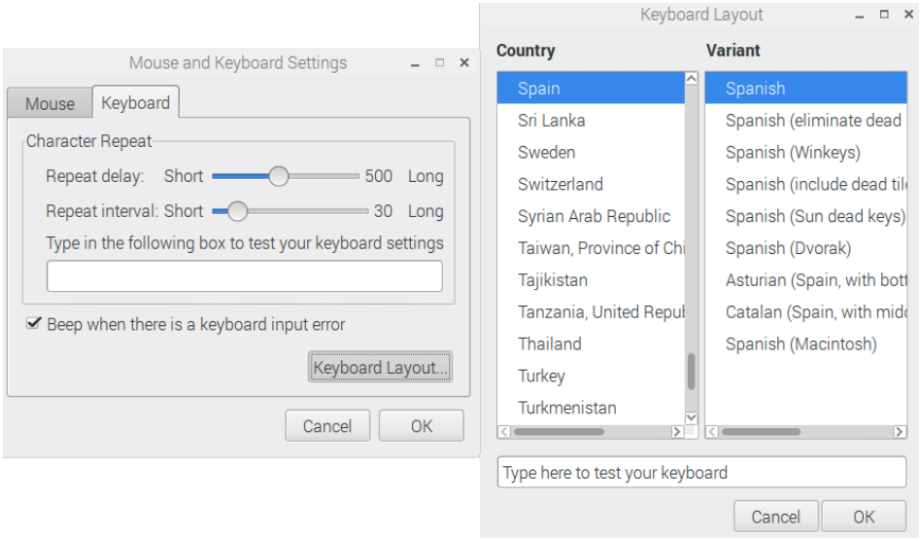

Figura 15. Configuración de ratón y teclado.

Como se puede observar, el texto que aparece y las opciones que se han nombrado anteriormente están en inglés. El idioma que aparece por pantalla se puede modificar, pero en este caso se ha dejado en inglés.

Este SO tiene instalados varios lenguajes de programación (BlueJ Java IDE, Greenfoot Java IDE, Mathematica, Python 2 y 3 con el entorno de programación IDLE, Scratch, Sonic Pi y Wolfram), editores Office (texto, tablas de cálculo, bases de datos, etc.), navegador de internet (Chromium).

Figura 16. Menú de la Raspberry Pi.

4. DISPOSITIVO DE MEDIDA DE FRECUANCIA CARDIACA

La labor principal del proyecto es la medida de la frecuencia cardiaca, el tratamiento de los datos y la representación de los mismos. Para la primera fase, se necesita un dispositivo de medida que permita transmitir a la Raspberry las medidas realizadas. En este caso se opta por un sensor de comunicación Bluetooth. El modelo es el Polar H7. Su ámbito de uso es el deportivo, donde se utiliza con un reloj en el que se muestran las pulsaciones.

Figura 17. Sensor Polar H7.

Especificaciones técnicas

Voltaje	3V
Frecuencia	2.402 – 2.480 GHz
Consumo de corriente	400 µA
Potencia de transmisión	1.3 mW
Tipo de pila	CR 2025
Junta tórica de la pila	Junta tórica 20,0 x 1,0 Material: FPM
Duración de la pila	200 h
Rango de temperaturas de funcionamiento	De -10 a +50 ºC
Material del conector	Poliamida
Material de la correa	38% poliamida, 29% poliuretano, 20% elastano, 13% poliéster

Tabla 2. Especificaciones técnicas del sensor Polar H7.

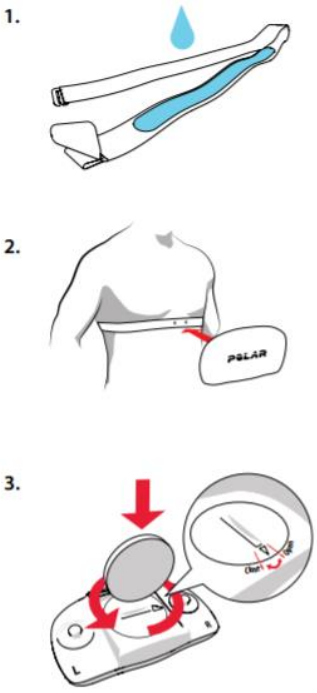

Figura 18. Colocación del sensor.

COLOCACIÓN DEL HR SENSOR

1. Humedece la zona de los electrodos del elástico (ilustración 1).
2. Fija el elástico alrededor de tu tórax y ajústalo para que quede ceñido pero cómodo.
3. Coloca el sensor (ilustración 2).

Después del entrenamiento, retira el transmisor y enjuaga la correa con agua corriente para mantenerla limpia.

PRIMEROS PASOS

Puedes utilizar tu Polar H7 con muchas de las principales apps de fitness como Polar Beat, además de con muchos productos Polar y máquinas de gimnasio. Polar H7 es compatible con muchos smartphones, como iPhone 4S y posteriores y dispositivos Android seleccionados. Comprueba todos los productos y dispositivos compatibles en polar.com/support.

Para comenzar con tu H7, primero tienes que vincularlo con la app o el producto Polar. Para obtener instrucciones más detalladas, consulta el Manual del fabricante de la app o la guía de usuario de tu producto Polar.

Recuerda que la señal de frecuencia cardíaca que tu H7 envía no atraviesa el cuerpo humano. Por tanto, no debes guardar el dispositivo receptor en una mochila, por ejemplo, sino en un lugar delante de ti.

CUIDADOS DEL HR SENSOR

Sensor: Retira el sensor del elástico después de cada uso y sécalo con un paño suave. Limpia el sensor con jabón suave y agua cada vez que sea necesario.

Elástico: Enjuágalo con agua corriente después de cada uso. Lávalo con regularidad. Consulta la etiqueta de tu elástico para obtener instrucciones detalladas del lavado.

Para obtener instrucciones detalladas de cuidado, consulta el Manual del usuario completo en polar.com/support.

CAMBIAR LA PILA ES SENCILLO

1. Con una moneda, abre la tapa de la pila girándola en el sentido contrario a las agujas del reloj, hasta OPEN (ilustración 3).
2. Inserta la pila (CR2025) dentro de la tapa con el lado positivo (+) orientado hacia la tapa. Asegúrate de que la junta de estanquidad sigue en la ranura para garantizar la resistencia al agua del dispositivo.
3. Presiona la parte trasera de la tapa contra el sensor.
4. Usa la moneda para girar en el sentido de las agujas del reloj, hasta CLOSE (Cerrar).

Figura 19. Instrucciones de uso.

5. PYTHON

Respecto al lenguaje de programación utilizado, inicialmente se plantearon tres alternativas: Python, Java y C/C++. De estos tres lenguajes de programación, el que se escogió finalmente fue Python, debido a que se adapta mejor a este proyecto, pues la Raspberry Pi y Raspbian están preparados para trabajar con él. Además, la información disponible y el hecho de que sea un lenguaje de código abierto, reforzaron la idea de elegir dicho programa. Para poder realizar una comparativa entre los tres lenguajes, se presentan ventajas y desventajas de cada uno de ellos:

C++

Ventajas:

- Lenguaje de programación orientada a objetos
- Muy didáctico, sirve como base para aprender otros lenguajes
- Muy potente y robusto
- Puede compilar programas desarrollados en C

Desventajas:

- Si se compara con otros lenguajes de programación, es más difícil de aprender.

Java

Ventajas:

- Es un lenguaje multiplataforma
- Lenguaje orientado a objetos
- Más rápido que otros lenguajes
- Gran cantidad de recursos
- Fácil de aprender en comparación con C++, que tiene una sintaxis similar pero un poco más complicada

Desventajas:

- Menos eficiente que otros lenguajes como C/C++
- Requiere de intérprete

Python

Ventajas:

- Simplificado y rápido
- Es sencillo de aprender
- Orden
- Es multiplataforma

Desventajas:

- El hecho de ser un lenguaje interpretado hace que sea más lento respecto a los compilados

Otras razones por las que se elige Python son: aunque C/C++ y Java puedan tener un mayor rendimiento y se hayan utilizado en más aplicaciones que Python, este último lenguaje presenta una mayor sencillez y claridad que los anteriores, añadiendo el hecho de que Python sea uno de los lenguajes más usados en programación con Raspberry Pi. Sobre Python, al igual que sobre C++ y Java, hay gran cantidad de información publicada en Internet, que puede servir de ayuda a la hora de realizar la programación.

Python es un lenguaje interpretado, multiparadigma y multiplataforma, además tiene un tipado dinámico. Al ser multiparadigma soporta la programación estructurada, la funcional y la orientada a objetos. Uno de los aspectos que lo hace idóneo para este proyecto es que tiene licencia de código abierto, lo cual es una ventaja a la hora de utilizar la información encontrada en internet. Es un lenguaje sencillo, que intenta favorecer el hecho de que sea legible y apto para principiantes en el mundo de la programación.

Actualmente, existen dos versiones, la 2.7 y la 3.x. En Debian, como se ha visto anteriormente, aparecen instaladas la 2.7 y la 3.4. El entorno de programación que se utiliza es IDLE. Se han utilizado ambas versiones, ya sea para realizar pruebas aisladas, ejemplos, desarrollo del programa final; aunque las librerías descargas, se han descargado en su mayoría para Python 2.7.

La Raspberry Pi tiene instalados Python 2 y 3 y utiliza como entorno de programación IDLE. En esta imagen se ve el menú desplegable de Raspbian donde se pueden ver los entornos de programación que vienen de serie con el dispositivo.

Figura 20. Menú de la Raspberry Pi.

El primer programa que se ha hecho ha sido el 'Hola Mundo', el cual solo ocupa una línea de comando. En este caso se ha incluido una nueva línea que imprima por pantalla el texto: 'Esto es una prueba'. El código implementado y el resultado del mismo aparecen en la siguiente imagen. Como se puede comprobar solamente hacen falta dos sentencias `print()` y entre los paréntesis el texto que se pretende que aparezca en pantalla.

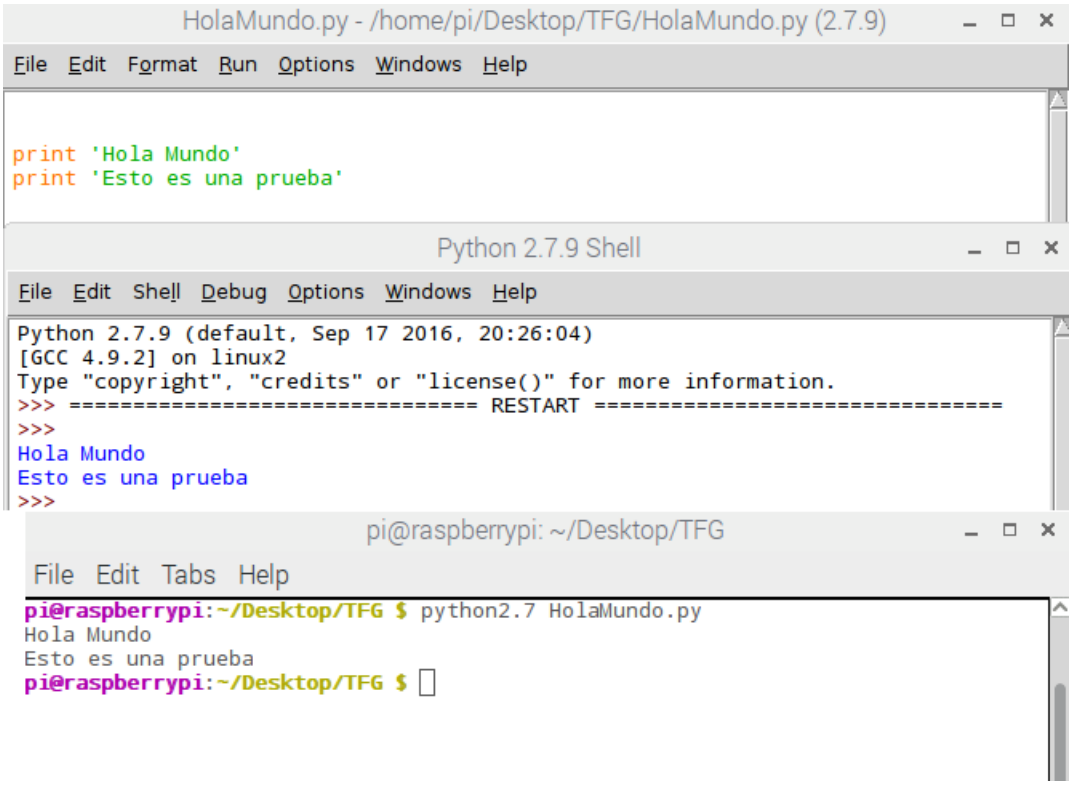

Figura 21. Programa Hola Mundo.

El siguiente programa desarrollado ha sido uno consistente en la comparación de dos listas. Después de pedir al usuario que introduzca dos listas, se comparan mediante una sentencia `if` y aparece un mensaje, por pantalla, indicando si son o no iguales. Luego se pregunta si se quiere seguir con el programa, que en caso afirmativo se pulsará la tecla 's'.

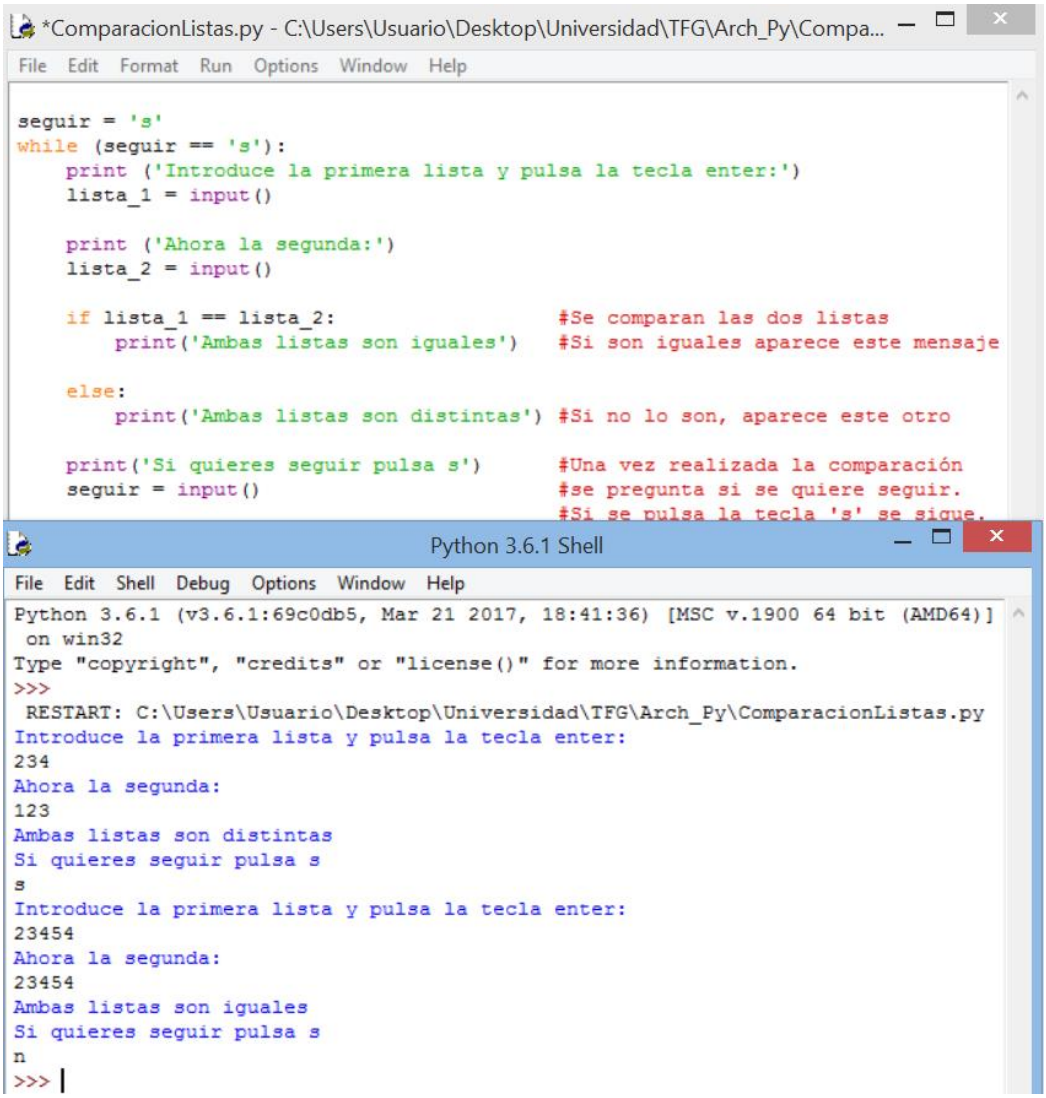

Figura 22. Programa de comparación de listas.

En primer lugar, se han introducido las listas '234' y '123'. Al comprobar que son distintas se ha mostrado el mensaje: 'Ambas listas son distintas'. Inmediatamente después aparece un mensaje que da la opción de seguir si se pulsa 's'. Se hace esto y se introducen dos listas iguales con los caracteres '23454', aparece un mensaje diciendo que 'Ambas listas son iguales' y aparece de nuevo el mensaje que permite continuar.

6. COMUNICACIÓN SERIE. PUERTO RS-232.

Para la comunicación entre la placa y el sensor de medida se usa Bluetooth. Pero primero se prueban los programas sin establecer comunicación con el dispositivo, con el fin de depurar parte de los errores. Entonces se establece comunicación serie con la misma placa, a través de un simulador de puerto serie virtual.

La Raspberry tiene un puerto serie que sigue el protocolo RS-232. Este, data de 1960, aunque posteriormente se han ido indtroduciendo modificaciones. Define la conexión serie entre el emisor y el receptor, que en el caso de este proyecto, ambos van a ser la misma placa. Consiste en que los niveles binarios de la señal se indican mediante niveles de tensión, positiva y negativa, respecto del punto de potencial común. Permite las transmisiones, tanto síncrona como asíncronamente. Está orientado a conexiones punto a punto. Sin embargo, presenta algunas desventajas como son las limitaciones de velocidad y distancia. Además, al utilizar la transmisión por voltaje, puede introducir fallos debido a las perturbaciones eléctricas.

Para establecer la comunicación serie entre dos puertos de la misma placa, se necesita un simulador de puerto serie y un programa que envíe y reciba los mensajes. El simulador de puerto serie que se utiliza es 'socat'. Para envío y recepción de mensajes se utiliza un programa desarrollado en Python. Pero antes de utilizar Python, se utiliza otro programa para probar solamente la comunicación, para comprobar que se envía y se recibe algún mensaje. Para esto hay varias alternativas. Las dos que se han utilizado son: minicom y GTKTerm. Lo que se hace para comprobar que se establece la comunicación correctamente es enviar mensajes entre los programas y verificar que se reciben. Una vez se ha comprobado que la comunicación entre minicom y GTKTerm funciona correctamente, en el siguiente paso se usa uno de estos programas junto con el que se ha desarrollado en Python. Finalmente se prueba solamente este último programa. Todo lo anterior se hace después de enlazar los puertos con socat.

6.1. SOCAT

Socat es una herramienta que permite enlazar varios puertos serie y protocolos, ya sea en la Raspberry o en un ordenador. Amplía las funciones que presenta Netcat, haciéndose su uso más complejo. No obstante, en este proyecto, el uso de socat va a ser muy limitado. En la comunicación serie que se va a establecer, solo se necesita ejecutar una línea de comando. En ella se deben indicar, al menos, dos direcciones. La primera indica el origen de la comunicación y los paquetes que se van a enviar. La segunda dirección indica el destino de la comunicación.

Para poder enviar y recibir mensajes de texto en la terminal mediante el comando socat, se deben escribir los siguientes pasos:

1º) (Opcional) Se teclea el comando `dmesg | grep tty` para ver los puertos serie habilitados.

2º) Se introduce `socat -d -d pty,raw,echo=0 pty,raw,echo=0`. Cuando se pulsa la 'Enter', debe aparecer por pantalla algo similar a lo siguiente:

```
2017/03/14 17:22:17 socat[6730] N PTY is /dev/pts/1
2017/03/14 17:22:17 socat[6730] N PTY is /dev/pts/2
2017/03/14 17:22:17 socat[6730] N starting data transfer
loop with FDs [5,5] and [7,7]
```

En las dos primeras líneas, `/dev/pts/1` y `/dev/pts/2` indican los puertos que se están enlazados y que por tanto se van a utilizar.

3º) Se abre otra ventana de la Terminal sin cerrar la que actualmente se encuentra abierta. Introduzco la línea que aparece a continuación y pulso 'Enter':

```
cat </dev/pts/1
```

Esta ventana es la que va a recibir el texto.

4º) Se vuelve a abrir otra ventana de la Terminal, la que va a enviar los mensajes. Se introduce el siguiente comando:

```
echo "Prueba de envío y recepción de mensajes” > /dev/pts/2
```

El texto que se encuentra entre comillas es el que aparecerá en la ventana de la terminal que se ha abierto anteriormente.

Si se quieren utilizar más pares de puertos, se abre otra ventana de la terminal y se vuelve a ejecutar `socat -d -d pty,raw,echo=0 pty,raw,echo=0`.

6.2. GTKTERM

Por otra parte, GTKTerm es un emulador gráfico de puerto serie desarrollado a partir de GTK+. Dicho emulador permite establecer comunicación con los puertos serie. Se caracteriza, entre otras cosas, por lo siguiente:

- Permite modificar las líneas de control manualmente (DTR, CTS)
- Envío de archivos
- Posibilidad de configuración de las características de la comunicación (velocidad, paridad, números de bits que contiene cada mensaje, bits de parada, etc.)

6.2.1. Instalación

GTKTerm se encuentra en los repositorios de Raspbian. Entonces, para su instalación, se deben ejecutar en la terminal, únicamente los siguientes dos comandos, con el primero se actualiza el sistema y con el segundo se instala:

```
sudo apt-get update
```

```
sudo apt-get install gtkterm
```

6.2.2. Comunicación entre dos ventanas de GTKTerm

Si se pretende enviar mensajes desde GTKTerm y recibirlos en el mismo programa, el proceso que se debe seguir es similar al mostrado anteriormente:

A) Igual que el paso 2º) de SOCAT, por lo que aparecerá lo mismo que en el caso anterior:

```
2017/03/14 17:22:17 socat[6730] N PTY is /dev/pts/1
2017/03/14 17:22:17 socat[6730] N PTY is /dev/pts/2
2017/03/14 17:22:17 socat[6730] N starting data transfer
loop with FDs [5,5] and [7,7]
```

B) Se abren dos ventanas de GTKTerm y se accede a la pestaña de Configuration.

C) En una de las ventanas se introduce el nombre de uno de los puertos serie:

```
Port: /dev/pts/1
```

Las casillas restantes permanecen sin variación.

D) En la otra ventana se hace lo mismo, pero con el puerto restante:

```
Port: /dev/pts/2
```

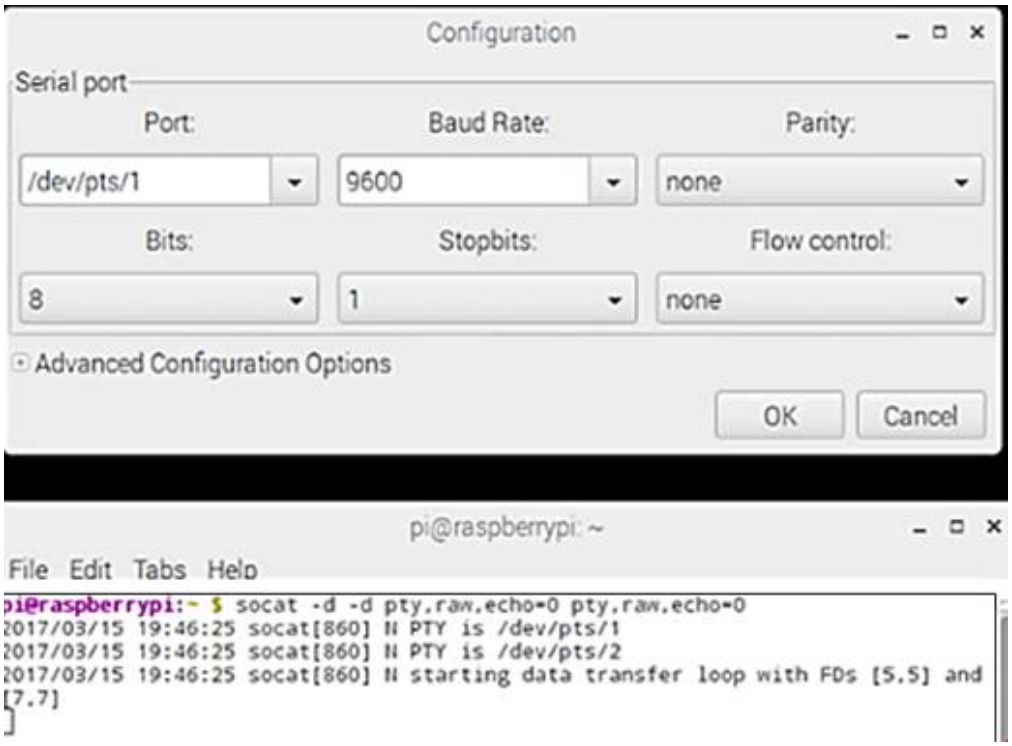

Figura 23. Configuración del primer puerto en GTKTerm y puertos enlazados.

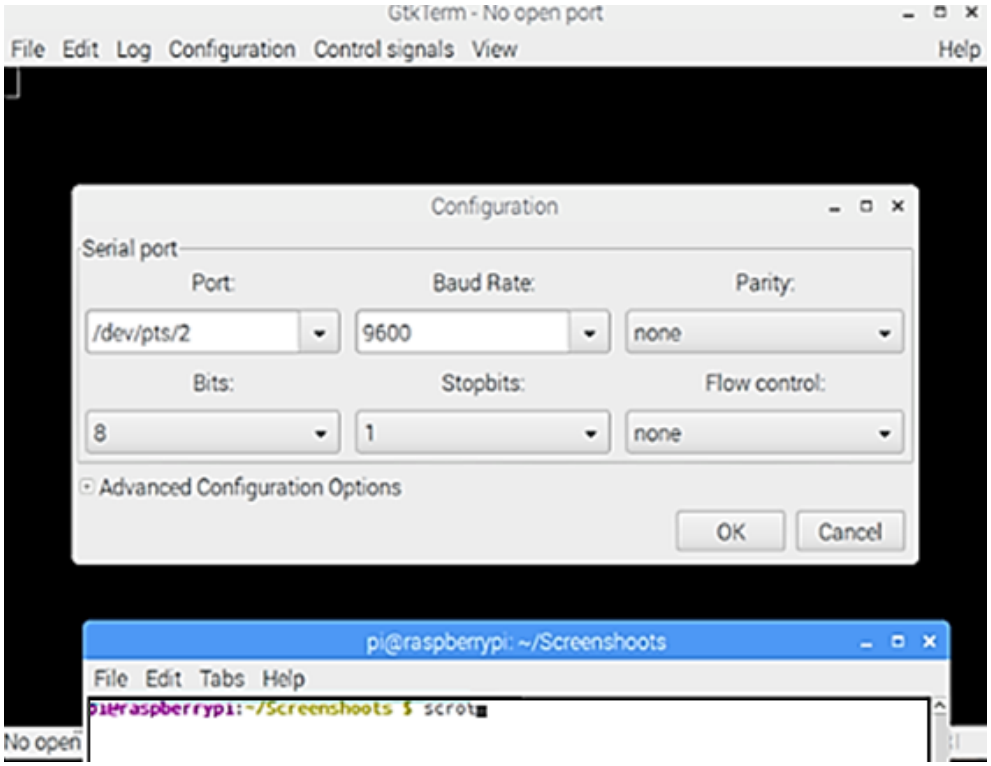

Figura 24. Configuración del segundo puerto en GTKTerm.

En las dos imágenes anteriores se puede observar que en cada ventana de 'Configuration' se ha introducido un puerto distinto. Ambos coinciden con los que aparecen en la terminal de Debian, en la primera imagen. Para realizar las capturas de pantalla se ha utilizado el comando `socat`.

Ahora se muestran las ventanas de envío de mensajes y de recepción:

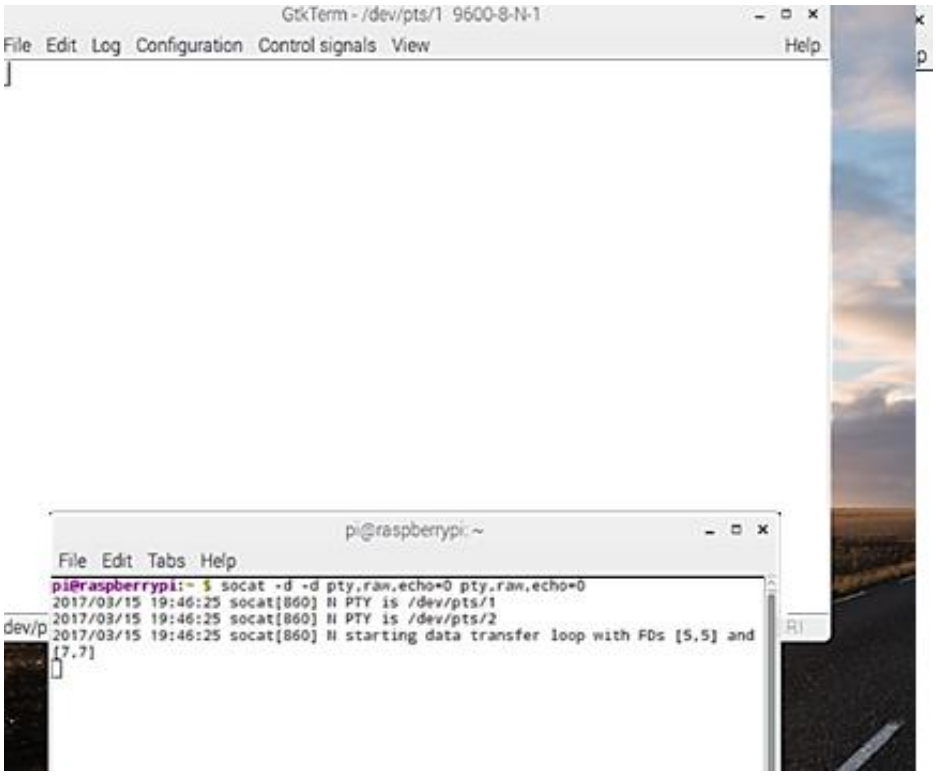

Figura 25. Ventana de envío de datos.

Figura 26. Ventana de recepción de datos.

En la primera imagen aparece la ventana de GTKTerm desde la cual se escribe y se envían los caracteres. En la segunda aparece el mensaje escrito en la anterior, conforme se va escribiendo. Si se escribiera en la segunda, el texto introducido aparecería en la primera.

6.3. MINICOM

Es un programa de menús de comunicación basado en emulación de terminal y en texto que permite enviar mensajes a través de puerto serie. En muchos casos sirve como alternativa a GTKTerm. Se utiliza en sistemas Unix. Incluye un directorio de marcación ANSI y emulación VT100.

6.3.1. Instalación

Al igual que GTKTerm, minicom también se encuentra en los repositorios de Debian, por lo tanto, los comandos que se necesitan para su instalación son los siguientes:

`sudo apt-get update` (Solamente se ejecuta si es necesario volver a actualizar el sistema)

```
sudo apt-get install minicom
```

Para configurar minicom, se ejecuta el siguiente comando:

```
sudo minicom -s
```

6.3.2. Comunicación entre minicom y GTKTerm

Otra alternativa, para establecer comunicación serie, es abrir una ventana de GTKTerm y otra de minicom y establecer comunicación entre ambas. Para recibir mensajes en minicom desde GTKTerm, se debe escribir en la terminal de Linux el siguiente comando:

```
sudo minicom -D /dev/pts/2
```

Y configurar GTKTerm como puerto de envío de mensajes, de la forma en la que se indicó anteriormente. En la ventana Configuration, en la casilla de Port, habría que escribir `/dev/pts/1`.

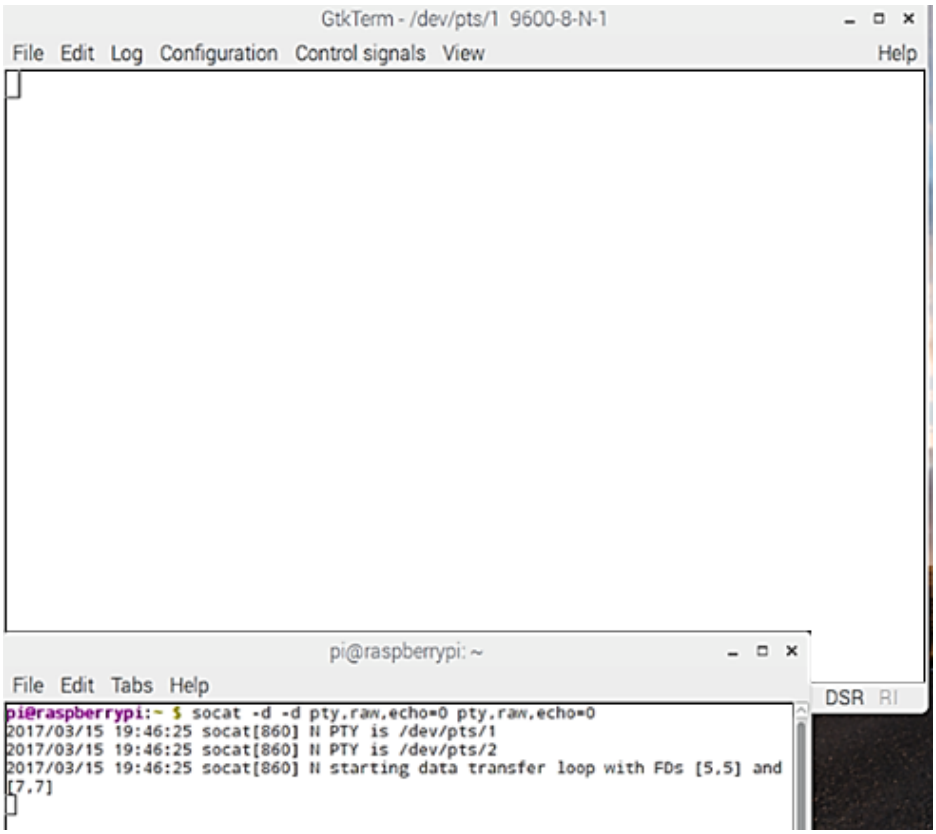

Figura 27. Ventana de envío. GTKTerm.

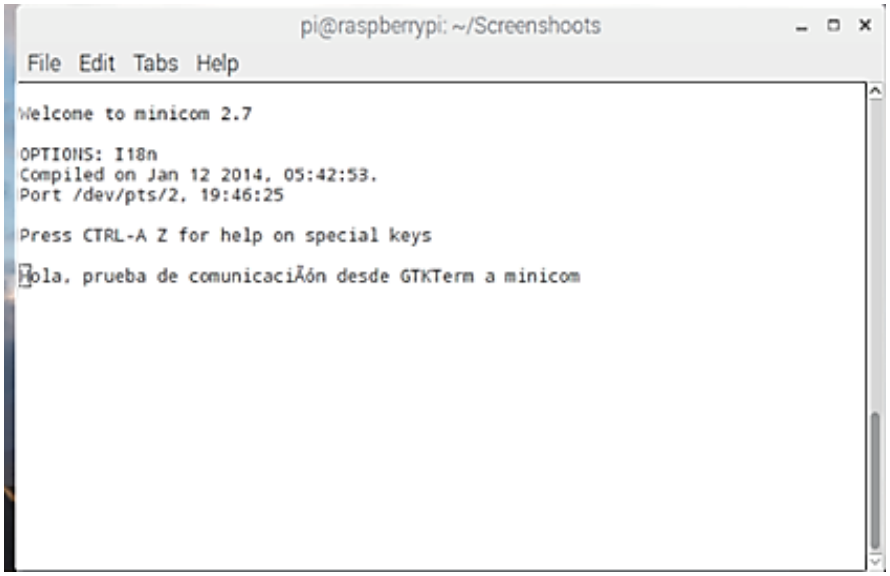

Figura 28. Ventana de recepción. Minicom.

En este ejemplo se ha enviado un mensaje desde GTKTerm y se ha recibido en minicom. Hay un fallo a la hora de recibir el mensaje. En la palabra comunicación aparece la letra 'A' con la virgulilla de la 'ñ' encima. Aparece al poner acentos en las letras.

Se puede hacer a la inversa, escribir texto en minicom y que aparezca en GTKTerm. Para esto se debe configurar ambos programas a la inversa de como se ha hecho.

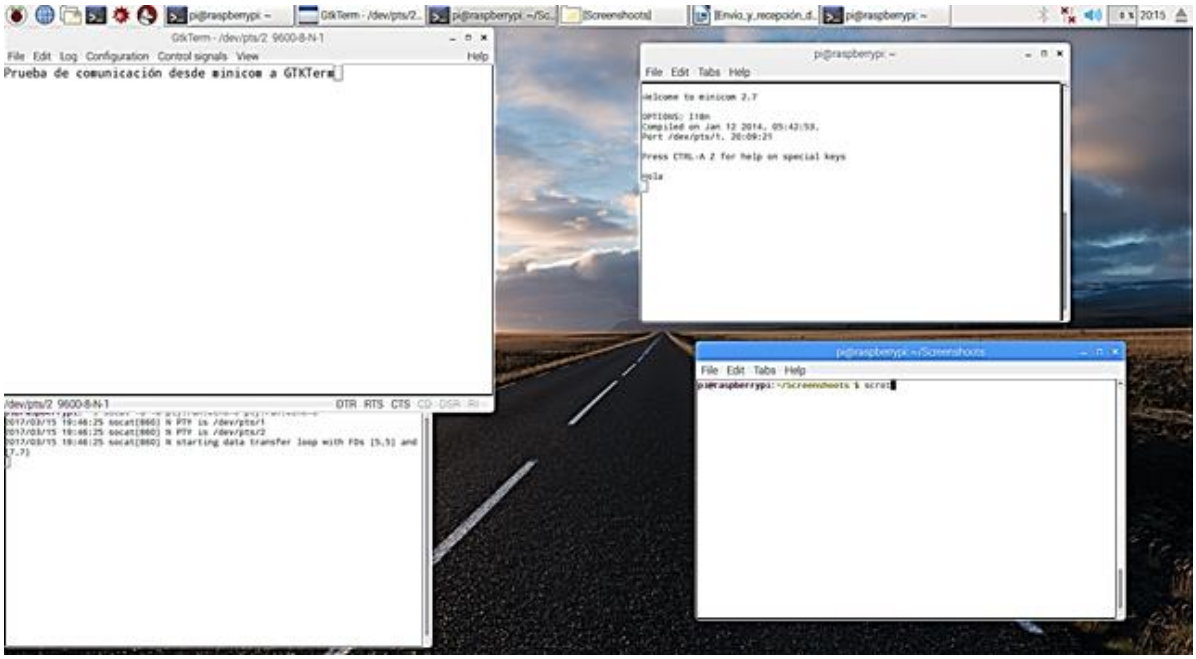

Figura 29. Comunicación entre GTKTerm y Minicom.

6.3.3. Envío de archivos de texto de minicom a GTKTerm

También se pueden enviar archivos desde minicom a GTKTerm. Para ello se deben seguir los siguientes pasos:

1º) Se realizan los pasos que se indicaron para enlazar dos puertos serie distintos.

2º) Se pulsa 'Ctrl-A' y posteriormente la tecla 'S'. Aparece un menú con cinco opciones, como el que se muestra en la imagen. Las cinco opciones: zmodem, ymodem, xmodem, kermit y ascii. Se selecciona la última opción.

Figura 30. Primer menú de envío de datos con minicom.

3º) Aparecerá un nuevo menú. Se selecciona esta vez la penúltima opción, que es `.profile`.

Figura 31. Segundo menú de envío de datos con minicom.

4º) Una vez hecho lo anterior, se presenta un espacio donde introducir la dirección del archivo a enviar. En este caso es `/home/pi/Desktop/Python/Py3/tabla`.

Figura 32. Introducción de la dirección del archivo a enviar.

5º) Por último, se muestra en la ventana de GTKTerm el texto que contiene el archivo enviado. En este caso son cien números repartidos en tres columnas. Este archivo se usará más adelante en Python.

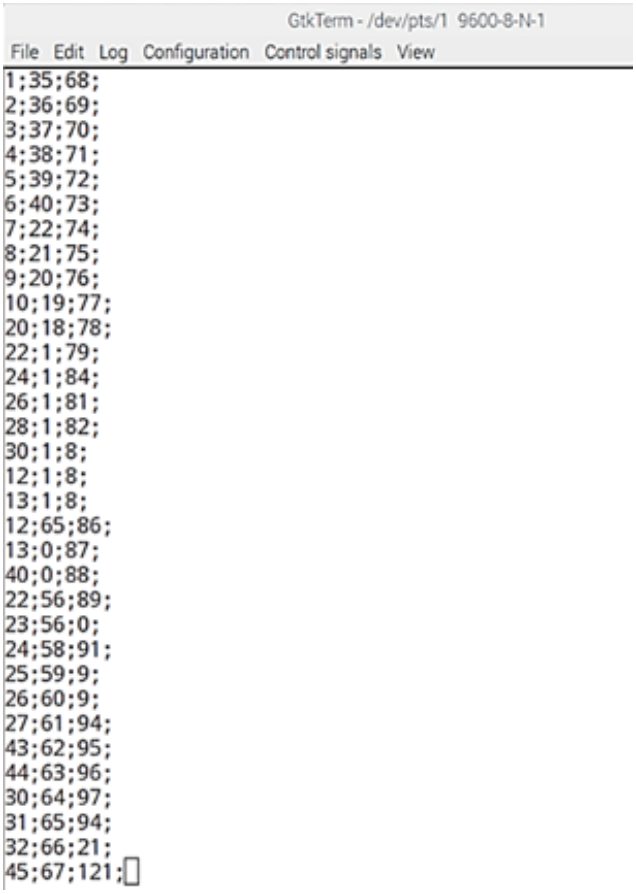

Figura 33. Lista de números enviada

6.4. PYTHON Y PYSERIAL

Para poder utilizar los puertos serie en Python, es necesario descargar una librería que permita establecer este tipo de comunicación. En este proyecto se ha utilizado 'Pyserial'. Permite hacer uso de un puerto serie basado en el protocolo RS-232. Se puede hacer uso de más de un puerto serie a un mismo tiempo, para lo cual, la librería permite el uso de números del 0 al 255 para configurar dichos puertos

6.4.1. Descarga e instalación de la librería 'pyserial'

Para la descarga de la librería se ha consultado la web oficial de Python. El enlace de descarga es el siguiente: https://pypi.python.org/pypi/pyserial.

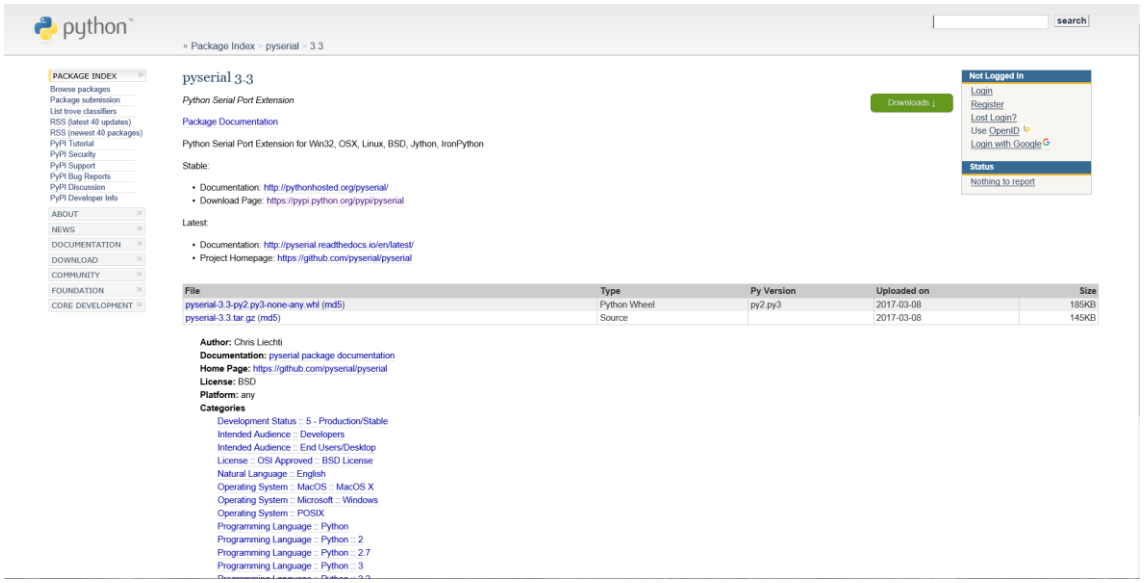

Figura 34. Página web de descarga.

Se descarga el segundo paquete que aparece, que tiene como nombre pyserial-3.3.tar.gz(md5).

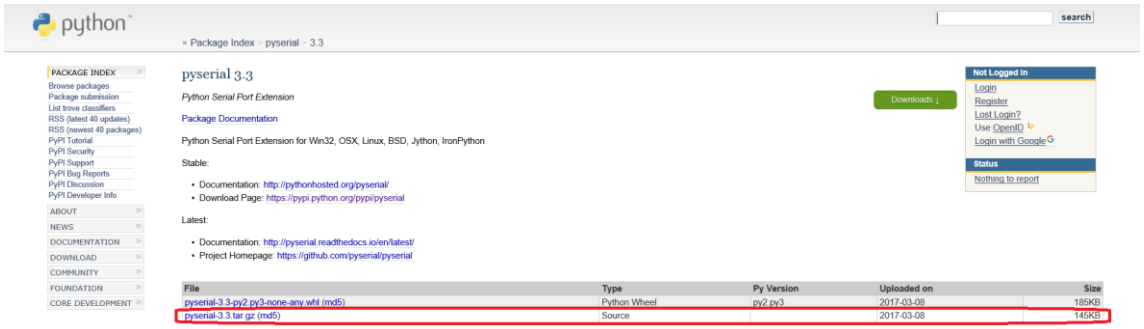

Figura 35. Enlace de descarga.

Una vez descargado, se ha descomprimido y se ha instalado según los pasos que aparecen en los archivos de texto README.ret y pyserial.rst, los cuales se encuentran dentro del paquete pyserial-3.3. Primero, desde la terminal, se debe acceder como superusuario y, posteriormente, se puede utilizar una de las siguientes líneas de comando:

```
python -m pip install pyserial
```

```
python setup.py install
```

En ese caso se ha utilizado la segunda, la cual se ha podido ejecutar una vez se ha accedido al directorio donde se encuentra el archivo setup.py. Esto se ha hecho mediante la siguiente línea de comando:

```
Cd /home/pi/Downloads/pyserial-3.3
```

6.4.2. Clases en PySerial

Algunas de las clases aportadas por la librería PySerial son las que se muestran a continuación:

`import serial`: Se importa la librería PySerial al programa. Todas las funciones de la librería deben empezar por `serial.`, como por ejemplo `serial.Serial()`

`ser = serial.Serial( port='/dev/pts/1', baudrate=9600, parity=serial.PARITY_ODD, stopbits=serial.STOPBITS_TWO, bytesize=serial.EIGHTBITS):`
Mediante la función `serial.Serial`, la cual se asigna a una variable (en este caso 'ser'), se declara un puerto serie ('/dev/pts/1').
`Baudrate` indica la velocidad a la que se transmite el mensaje. Los valores permitidos son: 110, 300, 1200, 2400, 9600, 19200, 38400, 57600, 115200.
Con `parity` y `bytesize` se indica la paridad y el tamaño de cada byte, respectivamente.
Es siempre obligatoria para el uso de un puerto serie.

`ser.open()`: Abre el puerto serie anteriormente declarado por `ser.Serial().`

`print(ser.name)`: Se imprime por pantalla el nombre del puerto que verdaderamente se está usando.

`ser.write()`: Imprime lo que se encuentra entre paréntesis. Ejemplo:

`ser.write('hello world')` imprime 'hello world'.

`ser.write(input)` imprime los caracteres que se han recibido como entrada.

`ser.close()`: Cierra los puertos abiertos con `ser.open()`.

`ser.read()`: Lee el número de bytes indicados entre paréntesis. En caso de que no se indique nada, se toma por defecto un byte.

`ser.readline()`: Lee una línea hasta que encuentre un salto de línea '\n'.

`ser.inWaiting()`: Devuelve el número de bytes del buffer recibido.

`outWaiting()`: Devuelve el número de bytes del buffer de salida.

`flushInput()`: Descarga el búfer de entrad.

`flushOutput()`: Borra el búfer actual, abortando la salida actual y descargando el búfer actual.

Un ejemplo de uso de esta librería es el que aparece a continuación. En él se genera una cantidad de números aleatorios y se transmite por el puerto serie.

```
import serial
import time
import random

color1 = 0
color2 = 0
color3 = 0

ser = serial.Serial(port = "/dev/pts/1", baudrate=9600)
ser.close()
ser.open()
random.seed()
while(1):
        if ser.isOpen():
                        temp = ''
                        color1 = random.randint(0,255)
                        color2 = random.randint(0,255)
                        color3 = random.randint(0,255)

                        if(color2 <0):
                                color2 = 0
                        if(color3 < 0):
                                color3 = 0
                        temp +=str(color1)
                        temp +=','
                        temp +=str(color2)
                        temp+=','
                        temp +=str(color3)
                        temp += "\r\n"
                        print(temp)
                        ser.write(temp)
                        time.sleep(1)
```

Figura 36. Generación de números aleatorios.

Como se puede observar, además de la librería `serial`, se han añadido `time` y `random`. La primera es útil por la función `sleep`. Esta permite dormir el programa tantos segundos como se pasen como parámetros; en este caso un segundo. De esta forma se concede el tiempo necesario para transmitir los datos por el puerto serie. La segunda librería permite generar los números aleatorios. Para ello se utiliza la función `random.randint(0,255)`, que genera números aleatorios entre los valores pasados por parámetro, aquí 0 y 255. El resultado final es la generación infinita de números representados en tres columnas y separados por coma. En las siguientes imágenes se muestra su funcionamiento, primero en Python 2.7 y luego en GTKTerm.

Figura 37. Código y resultado del programa en Python.

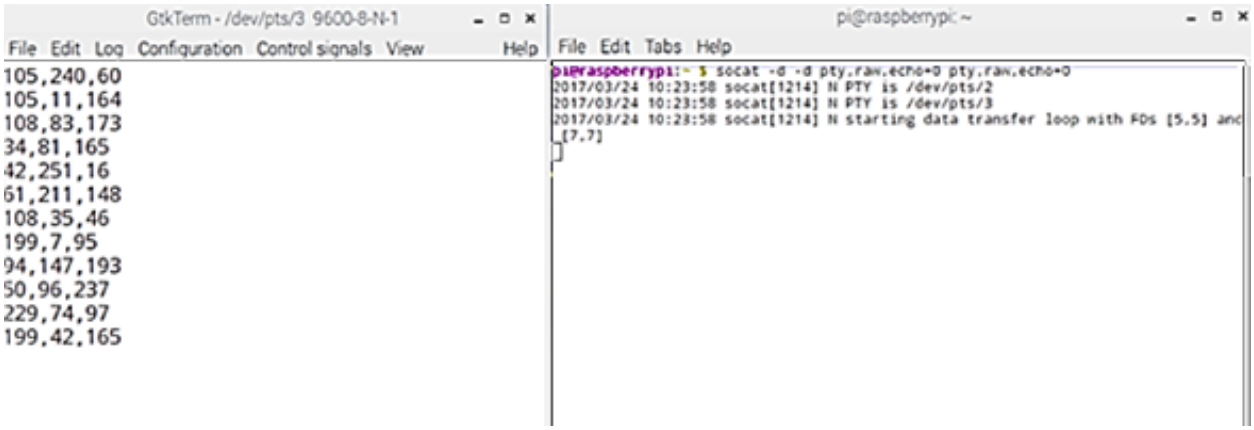

Figura 38. Resultado del programa en GTKTerm y puertos enlazados.

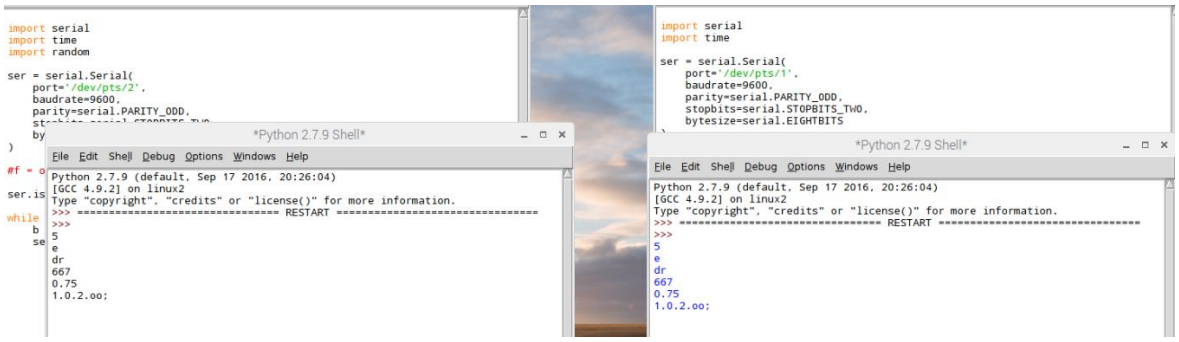

Figura 39. Resultado de programa que usa pyserial.

Este ejemplo consiste en introducir, en la ventana del programa emisor, cualquier carácter y pulsar la tecla 'enter' para que se imprima en la ventana del programa receptor. En la ventana de la izquierda, en negro, se muestran los caracteres introducidos por teclado. A la derecha, en azul, se muestran en la ventana del receptor. Se puede transmitir, de una sola vez, más de un carácter., incluyendo números, letras (también la ñ) y signos de puntuación.

6.4.3. Comunicación entre GTKTerm y Python

Para establecer comunicación entre ambos programas se ha escrito el código que aparece a continuación. Además, se han enlazado dos puertos como se ha indicado en casos anteriores.

```
#Este programa, hecho tomando como ejemplo uno encontrado en internet, sirve para comunicarse con GTKTerm o
#minicom. Si también se abre el programa PruGTK_Py_3.py, que es exactamente igual pero cambiando el puerto
#/dev/pts/1 por el /dev/pts/2, se puede establecer comunicación entre ambos

import time
import serial

# configuración del puerto serie /dev/pts/1
ser = serial.Serial(
    port='/dev/pts/1',
    baudrate=9600,
    parity=serial.PARITY_ODD,
    stopbits=serial.STOPBITS_TWO,
    bytesize=serial.EIGHTBITS
)

ser.isOpen()

print 'Introduce el mensaje abajo.\r En caso de querer salir introduce exit.'

input=1
while 1 :
    input = raw_input(">> ") #Por pantalla aparece >> y se puede empezar a escribir
    if input == 'exit':      #Este if me permite salir en caso de que introduzca input
        ser.close()
        exit()
    else:
        ser.write(input + '\r')  #Se envía el mensaje
        out = ''
        time.sleep(1)            #Se duerme el proceso 1 segundo para que de tiempo a que se transmita el mensaje
        while ser.inWaiting() > 0:
            out += ser.read(1)

        if out != '':            #Se imprime por pantalla el mensaje que se envía
            print ">>" + out
```

Figura 40. Código de prueba de comunicación serie entre GTKTerm y Python.

Si se hace funcionar este código, aparecerá en la pantalla del Shell de Python: `Serial port is open` en caso de que funcione correctamente y `Error` si hay algún fallo. En el primer caso, una vez se haya indicado por pantalla que el puerto está abierto, se podrá teclear por GTKTerm un conjunto de caracteres. Cada uno de ellos aparecerá en una línea diferente. Si no se introduce nada, se irán saltando líneas de una en una. Si se ejecuta el programa en Python 2.7, las líneas irán saltando y desde que se imprime la primera línea no se imprimirá nada hasta que se le indique. Si se ejecuta en Python 3.4, en cada línea se imprimirá b' ' y las letras que introduzca desde GTKTerm aparecerán entre las comillas, una en cada línea, de la siguiente forma:

Serial port is open:

```
b'P'
b'y'
b't'
b'h'
b'o'
b'n'
b'3'
```

En Python 2.7 las letras se mostrarán como en la imagen:

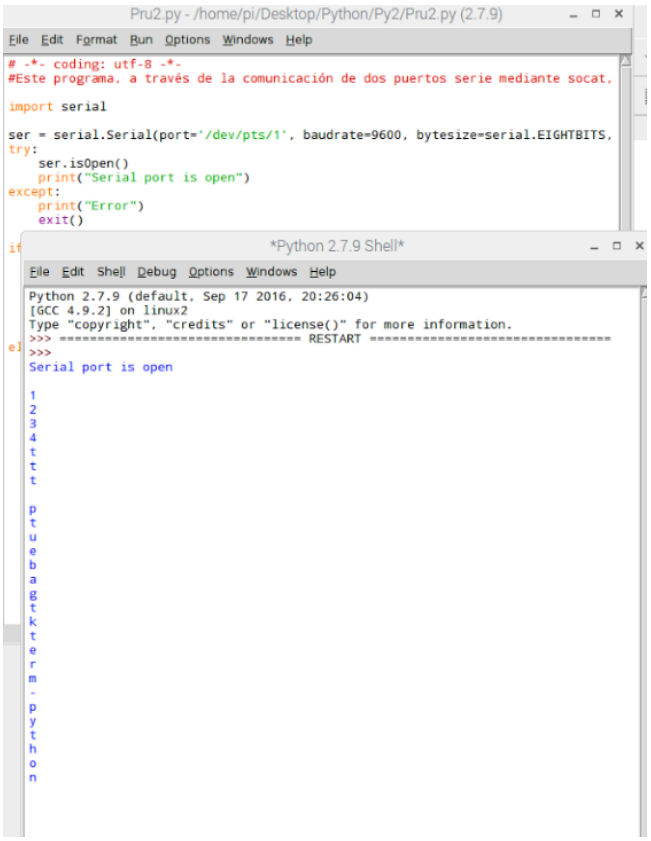

Figura 41. Resultados del programa en Python 2.

7. CREACIÓN DE APLICACIONES GRÁFICAS EN PYTHON. MATPLOTLIB, PYQTGRAPH Y QT4 DESIGNER.

7.1. MATPLOTLIB

Inicialmente, para la representación gráfica de datos numéricos, se ha optado por la biblioteca Matplotlib, junto con NumPy, ya que es una de las librerías gráficas más usadas. Está basada en el programa matemático 'Matlab'. Permite la generación de diversos tipos de gráficas en dos dimensiones y de imágenes. Por ejemplo, permite mostrar por pantalla histogramas, gráficos de errores, diagramas de dispersión, entre otros.

Mediante una serie de comandos, se puede controlar cualquier propiedad de los gráficos a mostrar (grosor de línea, color, escala, etc.). Posee una gran cantidad de elementos incorporados desde su instalación, lo que la hace más flexible que otras librerías y, además, facilita su uso.

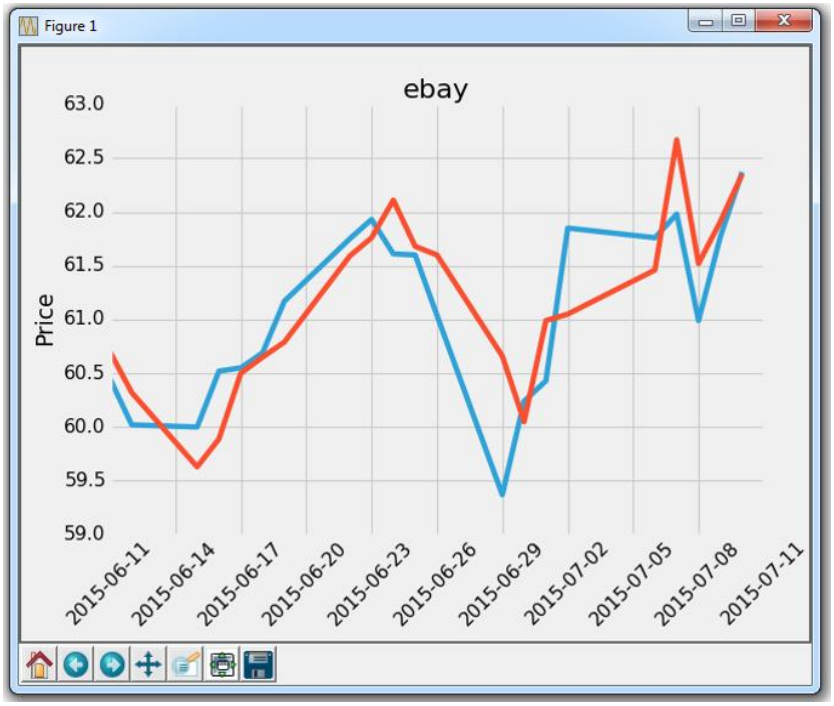

Figura 42. Ejemplo de gráfica utilizando Matplotlib.

7.1.1. Instalación

La instalación de Matplotlib se ha llevado a cabo a través de la terminal de Linux. La versión de Debian que se instaló en la Raspberry tiene preinstalada la librería, por lo que se puede proceder directamente a la instalación. El comando que se debe introducir, para ello, en la terminal, es:

```
sudo apt-get install python-matplotlib
```

Este es el comando de instalación en Debian, pero también en Ubuntu. En caso de que la distribución de Linux fuera Fedora o Redhat, habría que escribir:

```
sudo yum install python-matplotlib
```

Si se quieren instalar las dependencias, lo que hay que hacer es introducir el siguiente comando, dado que se trabaja con una distribución Debian:

```
sudo apt-get build-dep python-matplotlib
```

Matplotlib se puede utilizar en las últimas versiones de Python, desde la 2.7. Tras la instalación, se puede realizar un test para comprobar si realmente funciona. Se puede realizar de dos maneras. Una de ellas hace uso, nuevamente, de la terminal, donde hay que escribir `py.test`. Otra forma es haciendo uso, directamente, de Python, donde se crea un programa de solo dos líneas:

```
Import matplotlib
Matplotlib.test()
```

7.1.2. Ejemplos con Matplotlib

Ejemplo 1:

```
# Import the necessary packages and modules
import matplotlib.pyplot as plt
import numpy as np

# Prepare the data
x = np.linspace(0, 10, 100)

# Plot the data
plt.plot(x, x, label='linear')

# Add a legend
plt.legend()

# Show the plot
plt.show()
```

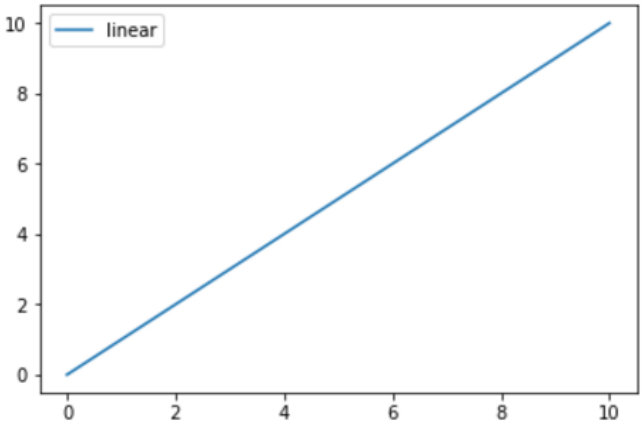

Figura 43. Ejemplo 1 de Matplotlib.

Este ejemplo encontrado en internet, se ha utilizado directamente. En él se importan el módulo 'pyplot', que se encuentra dentro de la librería 'matplotlib' y que permite crear gráficas, y la librería 'numpy', la cual se explicará más adelante. Se han importado mediante el comando `import … as …`. Esto permite asociar un nombre más corto a cada biblioteca, de forma que no haya que escribir el nombre real entero cada vez que se utilice una instrucción de la misma. Se puede observar en las siguientes cuatro líneas de comando que conforman el programa:

```
x = np.linspace(0, 10, 100)
plt.plot(x, x, label='linear')
plt.legend()
plt.show()
```

La primera instrucción pertenece a `numpy` y las tres siguientes a `pyplot`. Las funciones se describen a continuación:

- `numpy.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None)`: Devuelve una cantidad de números distanciados la misma cantidad de espacio, en un intervalo especificado. Tal intervalo viene dado por los valores de `start` y `stop`, que en este caso son 0 y 10, respectivamente. El tercer número que se indica en el ejemplo corresponde al valor de `num`, que indica la cantidad de números a devolver en ese intervalo. El valor por defecto es 50. El único parámetro obligatorio es `start`. Los demás son opcionales.

- `pyplot.plot()`: Permite generar una gráfica con los parámetros escritos entre paréntesis. En este ejemplo se pasan como parámetros (x, x, label = 'linear'). Los dos primeros indican los ejes 'x' e 'y', respectivamente. Por lo tanto, según lo escrito arriba, los valores asignados a 'x' en el comando `x=np.linspace(1, 10, 100)` aparecerán representados, tanto en el eje 'x' como en el eje 'y' de la gráfica. `label = 'linear'` permite que aparezca el recuadro con la palabra linear y con la línea azul.

- `pyplot.legend()`: Añade la leyenda donde aparece el 'label' definido en el comando anterior.

- `pyplot.show()`: Permite mostrar por pantalla la gráfica creada anteriormente.

Ejemplo 2:

Prueba_Grafica_txt.py - /home/pi/D...n/Py3/Prueba_Grafica_txt.py (2.7.9)

File Edit Format Run Options Windows Help

```
import numpy as np
import matplotlib.pyplot as plt

def una_columna(x):
    fichero = open(x)
    columna_1 = []
    columna_2 = []
    columna_3 = []
    a = []
    b = []
    c = []

    for l in fichero:
        campo = l.split(';')
        a = campo[0]
        b = campo[1]
        c = campo[-2]
        columna_1.append(a)
        columna_2.append(b)
        columna_3.append(c)

    plt.subplot(3,1,1)
    plt.plot(columna_1,'r')

    plt.subplot(3,1,2)
    plt.plot(columna_2,'g')

    plt.subplot(3,1,3)
    plt.plot(columna_3)

    plt.show()

una_columna('/home/pi/Desktop/Python/Py3/tabla')
```

Figura 44. Ejemplo 2, código.

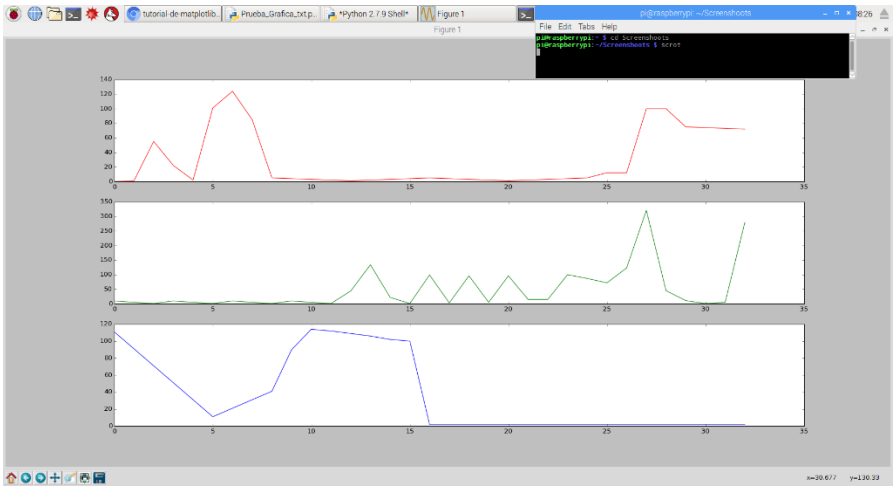

Figura 44. Ejemplo 2, resultado.

En este caso se abre un archivo de texto constituido por tres columnas de números separadas por punto y coma en cada fila. Se definen tres listas, una para cada columna. Además, se incluye un nuevo comando:

`pyplot.subplot()`: Permite dibujar varias gráficas en una misma ventana. En este caso se pasan tres números como parámetros. El primero es igual en los tres casos, un 3. Es el número de gráficas a dibujar. El segundo también es igual, un '1', lo que indica el número de columna en el que se van a colocar. El tercer número establece la fila en la que se va a colocar cada gráfica.

En `plt.plot()` se han introducido dos parámetros diferentes a los que se han escrito en el ejemplo anterior. 'columna_1', 'columna_2' y 'columna_3' establecen que en cada caso se va a representar una de esas columnas. En el caso de 'columna_2'.

Ejemplo 3:

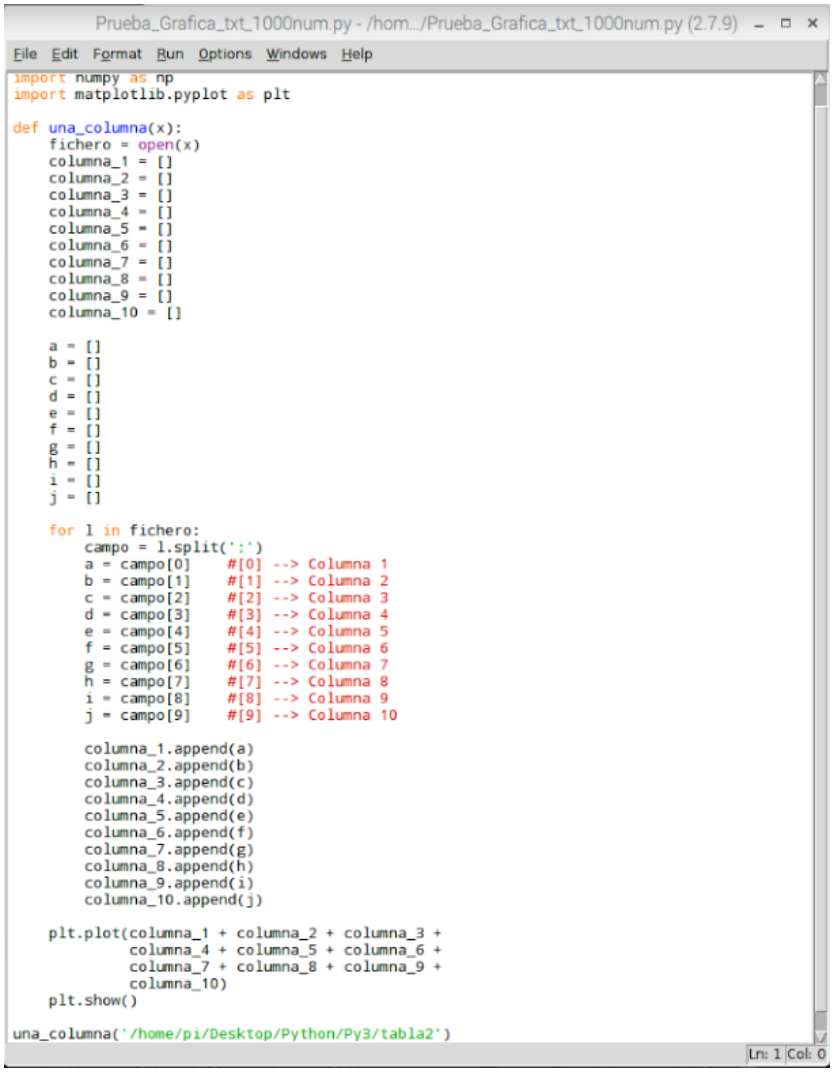

Figura 46. Ejemplo 3, código.

Ahora, la tabla a la que se accede tiene diez columnas de cien números cada una. En vez de representarlas por separado, se representan en la misma gráfica. Además, se representan una a continuación de la otra. Para ello, se pasa como parámetro del comando `plt.plot` la suma de todas las columnas.

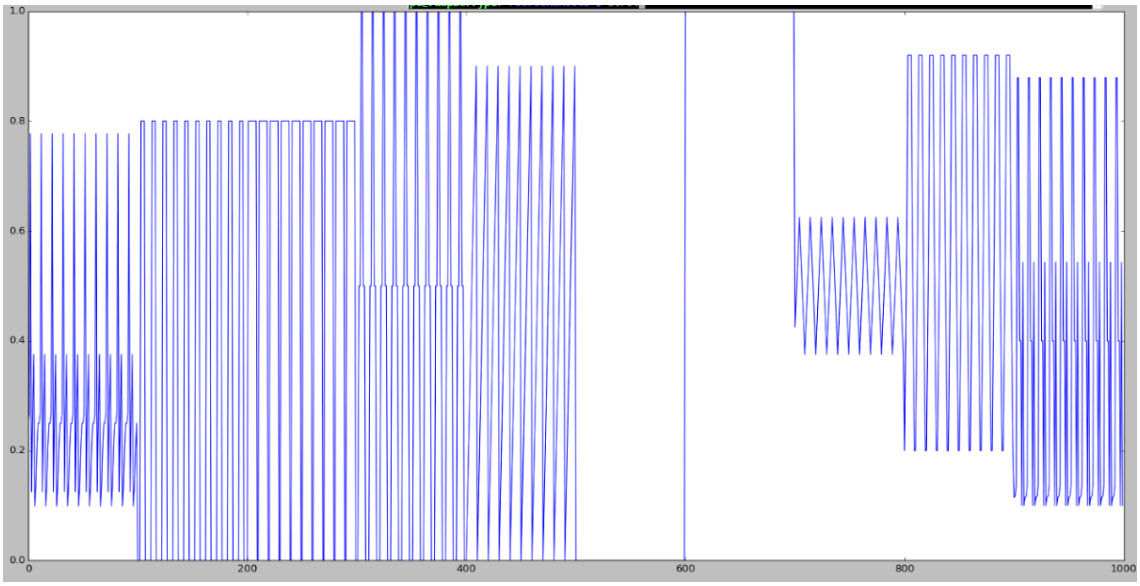

Figura 47. Ejemplo 3, gráfica.

7.2. NUMPY

Otra librería que se ha utilizado para la representación gráfica con Python ha sido NumPy. Se utiliza conjuntamente con Matplotlib. Es el principal paquete utilizado para la computación de datos científicos. Pero también permite realizar otras funciones, como almacenar datos en forma de matrices multidimensionales. Algunos de los principales contenidos de los que dispone, son:

- Un potente array N-dimensional
- Herramientas para la integración de código C/C++ y Fortran
- Capacidad para desarrollar álgebra lineal, transformadas de Fourier y para generar números aleatorios

7.2.1. Instalación

La instalación de NumPy en la Raspberry se puede realizar fácilmente. Como en casos anteriores, se hace uso de la terminal. Antes de la instalación se puede comprobar si hay alguna actualización reciente para el sistema operativo. De esta manera, si hay una nueva versión del mismo es probable que contenga los nuevos repositorios, desde donde descargar las últimas versiones de los programas que se desean instalar. Para ellos se utiliza el comando

```
sudo apt-get update
```

Hecho lo anterior, para instalar la librería NumPy, se debe introducir el comando

```
sudo apt-get install Python-numpy
```

7.2.2. Ejemplos

Ejemplo 1: Creación de un array

```
>>> import numpy as np
>>> a = np.array([11, 3, 7, 1])
>>> a
array([11, 3, 7, 1])
>>> b = np.arange(7)
>>> b
array([0, 1, 2, 3, 4, 5, 6])
```

Para el uso de la librería solo es necesario importarla mediante el comando `import`. Aquí se ha utilizado `import as` para acortar el nombre y que posteriormente sea más fácil llamar a las funciones contenidas en ellas.

Ejemplo 2: Álgebra lineal

```
>>> from numpy.random import rand
>>> from numpy.linalg import solve, inv
>>> a = np.array([[1, 2, 3], [3, 4, 6.7], [5, 9.0, 5]])
>>> a.transpose()
array([[ 1. ,  3. ,  5. ],
       [ 2., 4., 9.],
       [ 3., 6.7, 5.]])
>>> inv(a)
array([[-2.27683616,  0.96045198,  0.07909605],
       [ 1.04519774, -0.56497175, 0.1299435],
       [ 0.39548023, 0.05649718, -0.11299435]])
>>> b =  np.array([3, 2, 1])
>>> solve(a, b)  # solve the equation ax = b
array([-4.83050847,  2.13559322,  1.18644068])
>>> c = rand(3, 3) * 20  # create a 3x3 random matrix of
values within [0,1] scaled by 20
>>> c
array([[  3.98732789,   2.47702609,   4.71167924],
       [ 9.24410671, 5.5240412, 10.6468792],
       [ 10.38136661,   8.44968437, 15.17639591]])
>>> np.dot(a, c)  # matrix multiplication
array([[  53.61964114,   38.8741616 , 71.53462537],
       [ 1180.4935668,   86.14012835, 158.40440712],
       [ 155.04043289, 104.3499231, 195.26228855]])
>>> a @ c # Starting with Python 3.5 and NumPy 1.10
```

```
array([[  53.61964114,   38.8741616 ,   71.53462537],
       [ 118.4935668,   86.14012835, 158.40440712],
       [ 155.04043289, 104.3499231, 195.26228855]])
```

7.3. ALTERNATIVAS A MATPLOTLIB

En los primeros programas de representación gráfica, se han utilizado las librerías matplotlib y numpy. Sin embargo, la primera de ellas tiene algunas desventajas respecto a otras librerías, por ejemplo, a la hora de trabajar con programas de lectura de datos en tiempo real. La precisión en la representación de los datos disminuye y no permite el correcto desarrollo de la aplicación donde se utiliza, sobre todo cuando se trabaja con intervalos de tiempo muy pequeños. Estas desventajas representan una limitación a la hora de desarrollar este proyecto. Es por ello, que se decide optar por buscar una alternativa. Se han buscado varias opciones para realizar una comparación correcta y valorar cuál puede ser de mayor utilidad en este caso:

- TKinter
- PyQtgraph
- Chaco Screen Shots
- WxPython
- PyGTK
- Streamplot

La información acerca de las características de las librerías anteriores, así como algunos ejemplos, se presentan a continuación:

7.3.1. Tkinter

Tkinter es la librería, por defecto, para el desarrollo de interfaces gráficas en Python, aunque Tk se encuentra disponible en otros lenguajes. En sistemas operativos Windows es la que viene instalada por defecto; en distribuciones de Linux, como Raspbian, viene preinstalada. Es la más utilizada. Es orientada a objetos y rápido. Además, generalmente está incluido en Python. La documentación que se incluye por defecto es un poco pobre. Sin embargo, existe una gran cantidad de información que amplía el contenido inicial. También, otras alternativas a este módulo son wxPython, PySide, PyGTK, PyQtgraph y Chaco Screen Shots.

Con TKinter se puede crear una gran variedad de interfaces gráficas. Desde simples cuadros de diálogo con un botón, el ingreso de datos, captura de la pulsación de teclas, hasta juegos como el tres en raya.

```
23          else:
24              self.turno = 'X'
25
26      def __str__(self):
27          return "Le toca a las " + self.dame_tur
28
29  class Casilla:
30      def __init__(self, row, column, turno):
31          self.row = row
32          self.column = column
33          self.clicked = False
34
35          self.imagen = Tkinter.PhotoImage(file='
36
37          self.boton = Tkinter.Button(frame, imag
38                      width=95, height=82, comman
39          self.boton.grid(row=self.row, column=se
40
41      def clic(self):
42          if not self.clicked:
43              #cambiar imagen
44              if turno.dame_turno() == 'X':
45                  self.cambiar_imagen('equis.gif')
```

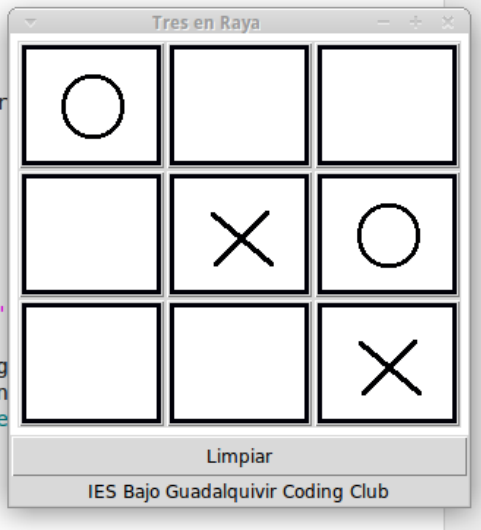

Figura 48. Extracto del código del juego 3 en raya

Las ventajas y desventajas de Tkinter son las siguientes:

- Ventajas:
  - En sistemas Windows ya viene instalada y en sistemas Linux, preinstalada con Python
  - Es sencilla de aprender, siendo recomendada para aprendices
  - Existe mucha documentación disponible

- Desventajas:
  - Es lenta, por lo que no es muy conveniente para aplicaciones de tiempo real. Aunque las últimas versiones han mejorado este punto, permitiendo que se pueda utilizar en un campo más amplio de programas
  - Limitado control del comportamiento de la interfaz
  - Su apariencia difiere de las aplicaciones nativas
  - Presenta pocos elementos gráficos

Un ejemplo de esta librería se presenta a continuación:

```
from Tkinter import *

app = Tk()
app.title("Aplicacion grafica en python")
etiqueta = Label(app, text="Hola mundo!!!")
boton = Button(app, text="OK!!")

etiqueta.pack()
boton.pack()
app.mainloop()
```

7.3.2. PyQtGraph

Esta librería está desarrollada con PyQt4 y NumPy. Consta de herramientas útiles para aplicaciones del ambiente científico y matemático:

- Gráficas en dos y tres dimensiones
- Amplio control sobre las imágenes que se muestran por pantalla
- Selección de datos y áreas de interés
- Librerías de módulos útiles para aplicaciones científicas y de ingeniería

Para su utilización se necesitan PyQt 4.7+, PySide o PyQt5; Python 2.6, 2.7 o 3.x; y NumPy; además, para los gráficos en tres dimensiones, también necesita pyopengl y qt-opengl. Los sistemas operativos donde funciona son Windows, Linux y Mac.

A continuación, se enuncian las ventajas e inconvenientes de PyQt:

- Ventajas:
    - Sí que es rápida, lo que la hace adecuada para una aplicación en tiempo real, como la de este proyecto
    - La apariencia es nativa
    - Arquitectura opcional para Modelo/Vista para las tablas, listas, árboles
    - La variedad de posibilidades que presenta a la hora de crear elementos gráficos, es elevada
    - En cuanto al control de la interfaz, es más potente y versátil que otros
    - Se puede separar el diseño de la interfaz
    - Arquitectura opcional para Modelo/Vista para las tablas, listas árboles

- Desventajas:
    - No viene preinstalada con Python, por lo que se debe instalar por separado. En Linux no resulta difícil su instalación
    - Es un poco más complejo de aprender
    - En alguna ocasión se debe hacer conversión de tipos de datos, ya que está implementado en C++
    - Existe menos documentación

Ejemplo de código escrito utilizando PyQt (Hola Mundo!):

```
import sys
from PyQt4 import QtGui

def main():
  app = QtGui.QApplication(sys.argv)

  etiqueta_hola = QtGui.QLabel("Hola Mundo!")
  etiqueta_hola.show()

  return app.exec_()
```

```
if __name__ == '__main__':
  sys.exit(main())
```

7.3.3. Chaco

Chaco conforma un conjunto de herramientas que permite el desarrollo de aplicaciones gráficas de distinta complejidad, desde scripts sencillos hasta aplicaciones gráficas con grandes interrelaciones de datos y multitud de herramientas interactivas. Permite la representación en dos dimensiones de imágenes estáticas y de elementos interactivos. Chaco incluye renderizadores para muchos tipos de trama, implementaciones integradas de interacciones comunes con esas parcelas y un marco para extender y personalizar tramas e interacciones. También puede hacer gráficos de manera no interactiva a las imágenes, ya sea en formato raster o vectorial, y tiene un sub-paquete para hacer trazado de línea de comandos o scripting simple.

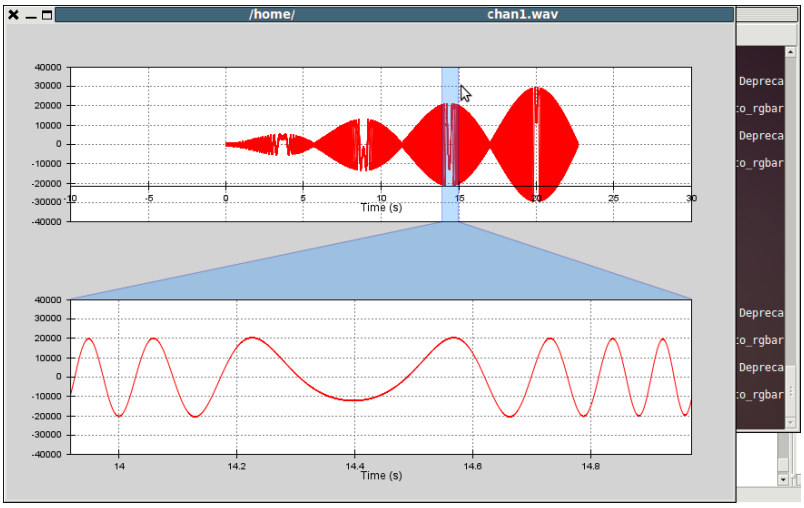

Figura 49. Ejemplo de Chaco Screen Shot

7.3.4. WxPython

Es una librería multiplataforma, para el desarrollo de aplicaciones gráficas, el cual pertenece a la biblioteca gráfica wxWidgets para Python. Cuenta con licencia de código abierto y está programado en dos lenguajes, C y Python.

- Ventajas:
  - Está más desarrollada en Python
  - No está orientada a ningún entorno
  - Es flexible y presenta una gran cantidad de herramientas gráficas
  - Es válido en los tres sistemas operativos más usados: Windows, Linux y Mac OS
  - Se puede encontrar mucha información, tanto teórica como práctica

- Desventajas:
    - No se encuentra preinstalada en ningún sistema operativo
    - Las representaciones gráficas de otras plataformas, no siempre llegan a ejecutarse correctamente
    - En proyectos que superen cierta extensión y complejidad se vuelve inestable
    - Es más compleja de aprender que otras
    - Se desarrollan versiones frecuentemente, lo que termina siendo un problema en cuanto a la compatibilidad

Ejemplo de wxPython:

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

import wx

class TestFrame(wx.Frame):
   def __init__(self, parent, ID, title):
       wx.Frame.__init__(self,  parent,  wx.ID_ANY,
title, pos=(0, 0), size=(320, 240))
       panel = wx.Panel(self, wx.ID_ANY)
       text   =   wx.StaticText(panel,   wx.ID_ANY,
"Hello, World!", wx.Point(10, 5), wx.Size(-1, -1))

class TestApp(wx.App):
   def OnInit(self):
       frame  =  TestFrame(None,  wx.ID_ANY,  "Hello,
world!")
       self.SetTopWindow(frame)
       frame.Show(True)
       return True

if __name__ == '__main__':
   app = TestApp()
    app.MainLoop()
```

7.3.5. PyGTK

Esta es la librería de desarrollo de aplicaciones gráficas creado mediante la biblioteca GTK. Es también multiplataforma, pero programado solamente en Python. Entre otros usos, se utiliza para desarrollar GNOME, entorno de escritorio para sistemas GNU/Linux y Unix.

- Ventajas:
  - Es flexible y potente
  - Estable
  - Al igual que algunas de las anteriores librerías, tiene un gran conjunto de elementos gráficos

- Desventajas:
  - Es compleja de aprender
  - Tampoco está preinstalada en Python
  - En Windows resulta algo lenta y de apariencia extraña. En este sistema operativo, también es la que más dependencias tiene, las cuales se deben instalar aparte.
  - Se puede encontrar escasa información

Ejemplo de PyGTK (Hola Mundo):

```
import gtk

def crear_ventana():
    ventana = gtk.Window()
    ventana.set_default_size(200, 200)
    ventana.connect('destroy', gtk.main_quit)

    etiqueta = gtk.Label('Hola mundo')
    ventana.add(etiqueta)

    etiqueta.show()
    ventana.show()

crear_ventana()
gtk.main()
```

7.3.6. Streamplot

Esta es la librería de la que menos información se tiene, por lo tanto, es la que menos posibilidades tiene de ser elegida. Un ejemplo desarrollado con Streamplot es el siguiente:

```
from streamplot import PlotManager

plt_mgr = PlotManager(title="My first plt with streamplot")

x1, y1 = 1, 2        #El primer numero indica el punto medio del eje x
x2, y2 = 2, 4        #y el segundo el del eje y
plt_mgr.add(name="1", x=x1, y=y1)
plt_mgr.add(name="2", x=x2, y=y2)
plt_mgr.update()
plt_mgr.close()
```

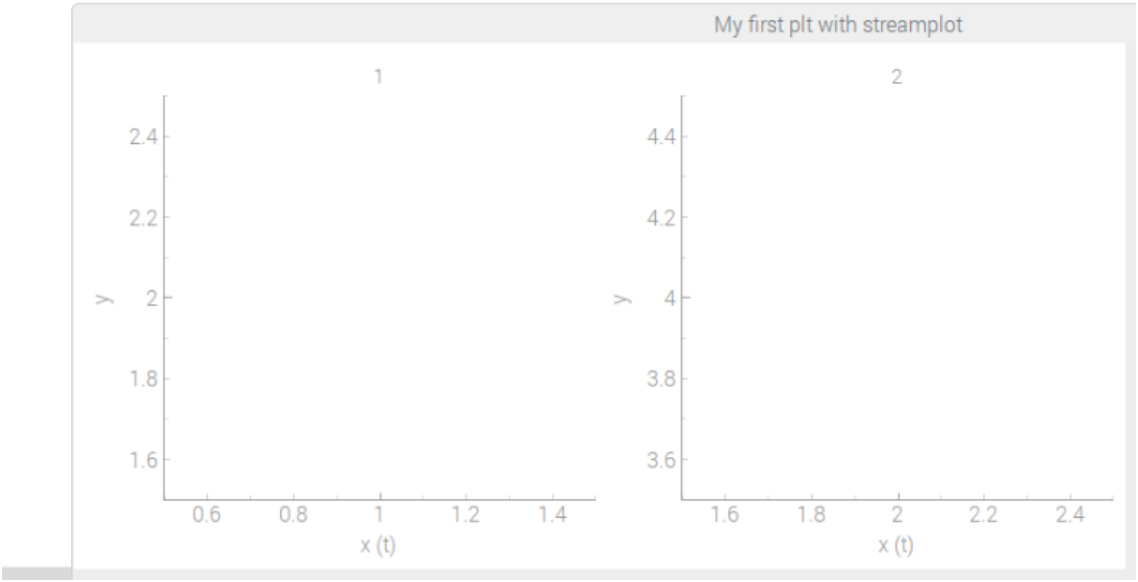

Figura 50. Ejemplo de StreamPlot.

Lo único que se hace en este programa es mostrar dos gráficas sin contenido. Sin embargo, para hacer esto la Raspberry Pi utiliza aproximadamente un setenta y cinco por ciento de su capacidad.

7.3.7. TKinter y PyQtGraph

De las seis opciones presentadas, se pre-seleccionan dos: TKinter y PyQtGraph. La primera se valora por ser el módulo, por defecto para aplicaciones gráficas en Python, por su sencillez y por la información disponible. De la segunda destacan la rapidez y versatilidad, la apariencia nativa y la variedad de herramientas con las que cuenta para realizar gráficos.

Se han elaborado algunos ejemplos para comprobar, en términos generales, cuál de los módulos es el que, definitivamente, se va a seleccionar. A continuación, se muestran algunos de esos programas de ejemplo.

TKinter

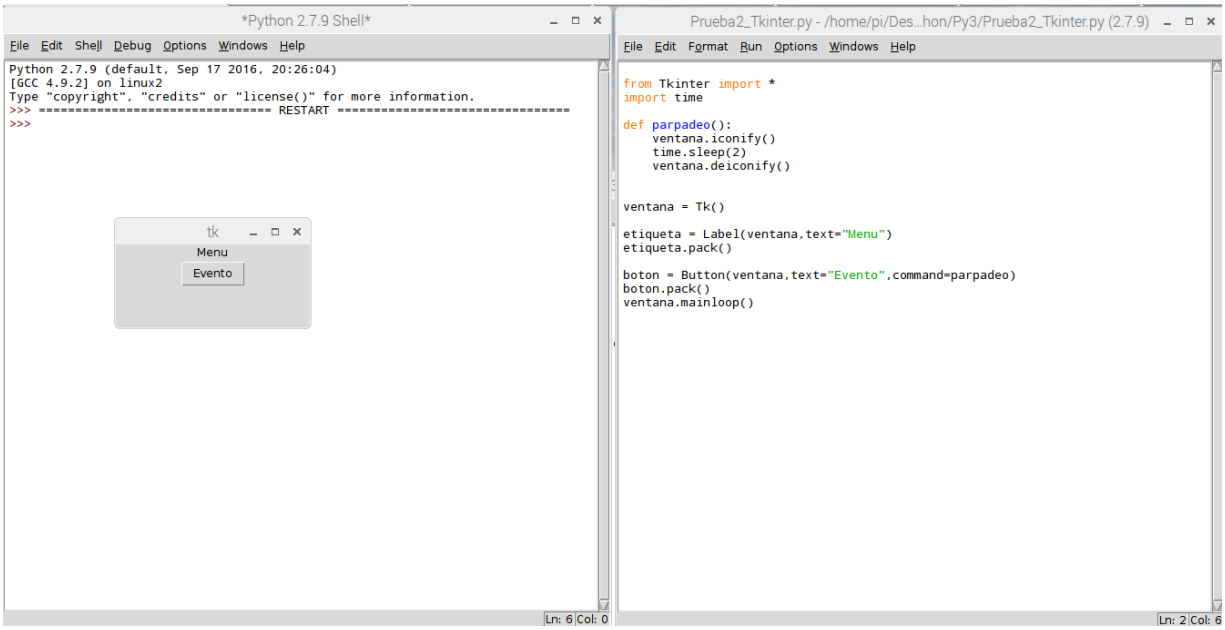

Figura 51. Ejemplo de Tkinter. Creación de ventana con botón.

En este ejemplo, lo que se hace es crear una ventana con el título 'Menú' y con un botón en el que aparece escrito 'Evento'. Si se pulsa dicho botón, la ventana se minimiza durante dos segundos y vuelve a aparecer. Para que suceda esto, se define la función parpadeo, que mediante el comando `ventana.iconify()` minimiza la ventana una vez el botón es pulsado, luego `contime.sleep(2)` duerme el programa dos segundos y con `ventana.deiconify()` vuelve a mostrar la ventana.

PyQt

Ejemplo 1:

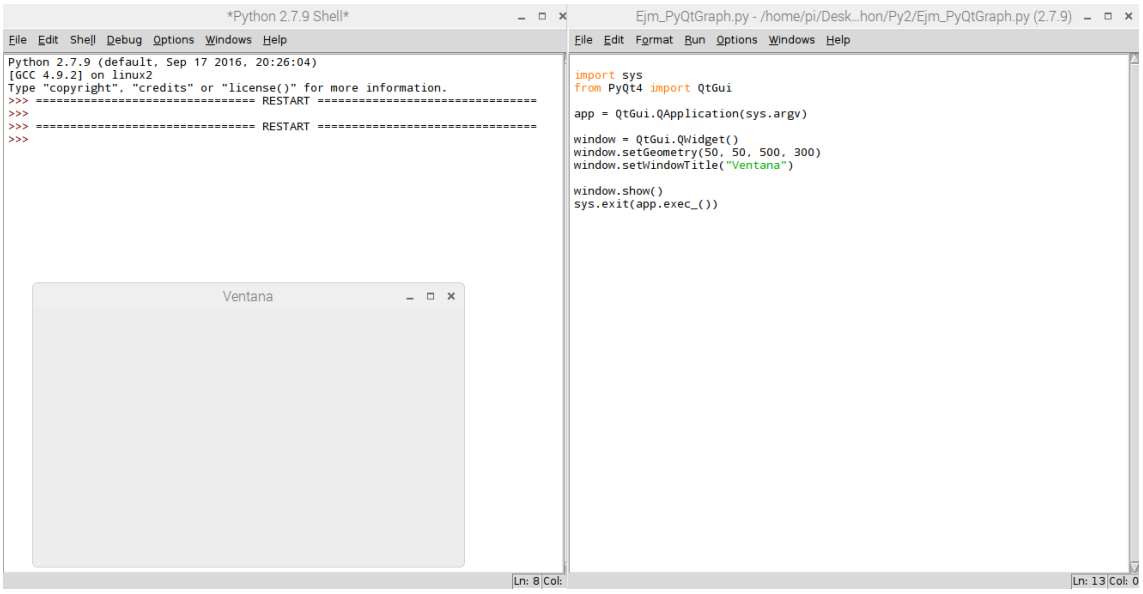

Figura 52. Ejemplo de PyQt. Creación de ventana.

Este primer ejemplo de la biblioteca PyQt solo muestra una ventana con un título, el cual se determina mediante el comando `window.serWindowTitle('Ventana')`. También se especifican las dimensiones de la ventana mediante `window.setGeometry(50,50,500,300)`.

Ejemplo 2:

En este caso se abre un fichero con veintitrés números, los cuales se muestran y se representan gráficamente.

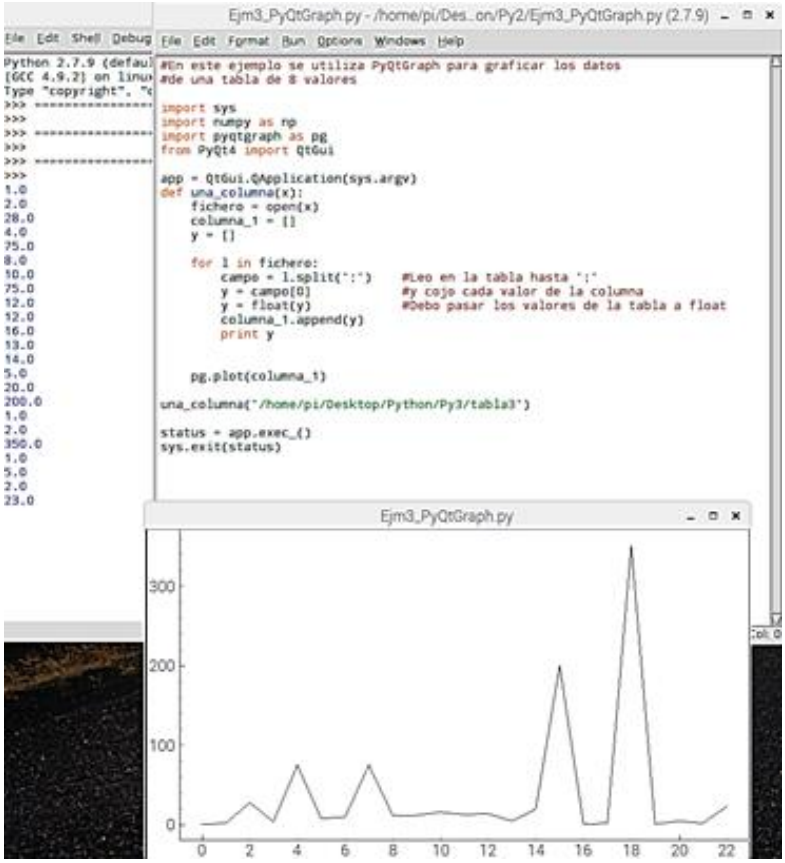

Figura 53. Ejemplo 2 de PyQt. Representación de tabla con 23 valores.

Ejemplo 3:

Al igual que en el ejemplo anterior, se abre un archivo de texto que contiene veintitrés números, y se representan. Primero se define una función llamada 'una_columna' que recibe como parámetro la dirección donde se encuentra la tabla. Mediante un bucle 'for' recorre los veintitrés números mientras los va mostrando por pantalla. Dentro del mismo bucle se crea una lista llamada 'columna_1'. Una vez finalizado el 'for' se representa la lista mediante *pg.plot()*.

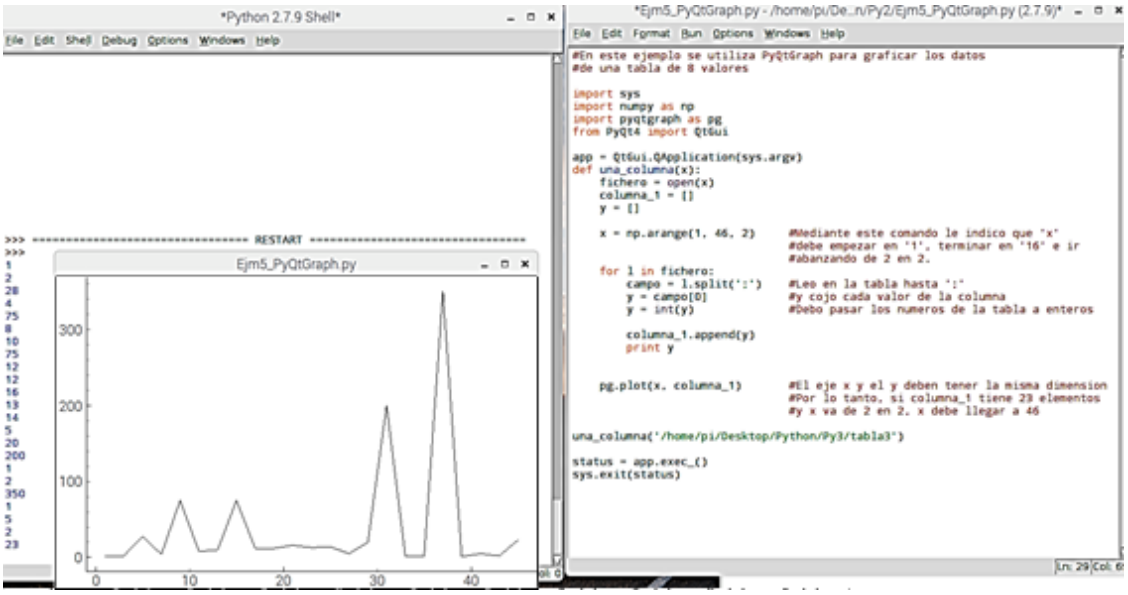

Figura 54. Ejemplo 3 de PyQt. Representación de tabla de 23 valores.

Ejemplo 4:

```
#Este programa es igual al anterior, con un cambio: ahora
el eje x, en vez de avanzar de uno en uno, lo hace de dos
en dos, lo cual se consigue mediante el comando x =
np.arange(1, 46, 2). Lo que indico aquí es que el eje x
empiece en '1', termine en '46' y avance de dos en dos. Las
dimensiones de los ejes x e y deben ser siempre las mismas.

#En este ejemplo se muestran los datos de la tabla2, la de
1000 valores, y se representan en una grafica
Hay lineas de comando comentadas. Se debe quitar '#' en
funcion de lo que se quiera hacer:
Graficar y/o mostrar la lista de numeros.

import sys
import numpy as np
import pyqtgraph as pg
from PyQt4 import QtGui
import time

app = QtGui.QApplication(sys.argv)
def una_columna(x):
    fichero = open(x)
    columna_1 = []
    columna_2 = []
    columna_3 = []
    columna_4 = []
    columna_5 = []
    columna_6 = []
    columna_7 = []
    columna_8 = []
    columna_9 = []
```

```
    columna_10 = []

    a = []
    b = []
    c = []
    d = []
    e = []
    f = []
    g = []
    h = []
    i = []
    j = []

    for l in fichero:
        campo = l.split(';')
        a = campo[0]    #[0] --> Columna 1
        b = campo[1]    #[1] --> Columna 2
        c = campo[2]    #[2] --> Columna 3
        d = campo[3]    #[3] --> Columna 4
        e = campo[4]    #[4] --> Columna 5
        f = campo[5]    #[5] --> Columna 6
        g = campo[6]    #[6] --> Columna 7
        h = campo[7]    #[7] --> Columna 8
        i = campo[8]    #[8] --> Columna 9
        j = campo[9]    #[9] --> Columna 10

        a = float(a)
        b = float(b)
        c = float(c)
        d = float(d)
        e = float(e)
        f = float(f)
        g = float(g)
        h = float(h)
        i = float(i)
        j = float(j)

        columna_1.append(a)
        columna_2.append(b)
        columna_3.append(c)
        columna_4.append(d)
        columna_5.append(e)
        columna_6.append(f)
        columna_7.append(g)
        columna_8.append(h)
        columna_9.append(i)
        columna_10.append(j)

        #print a, b, c, d, e, f, g, h, i, j Mediante esta
fila se imprimen por
       # pantalla los numeros conforme aparecen en el
```

```
archivo de texto
        # Si en vez de ',' pongo '+' aparece la suma de cada
fila

    for i in columna_1:    #Mediante estos 10 bucles for se
muestran
        print(i)             #todos los numeros en una sola
columna
    #for i in columna_2:
    #    print(i)
    #for i in columna_3:
    #    print(i)
    #for i in columna_4:
    #    print(i)
    #for i in columna_5:
    #    print(i)
    #for i in columna_6:
    #    print(i)
    #for i in columna_7:
    #    print(i)
    #for i in columna_8:
    #    print(i)
    #for i in columna_9:
    #    print(i)
    #for i in columna_10:
    #    print(i)

    pg.plot(columna_1)# + columna_2 + columna_3 +
    #         columna_4 + columna_5 + columna_6 +
    #         columna_7 + columna_8 + columna_9 +
    #         columna_10)

una_columna('/home/pi/Desktop/Python/Py3/tabla2')

status = app.exec_()
sys.exit(status)
```

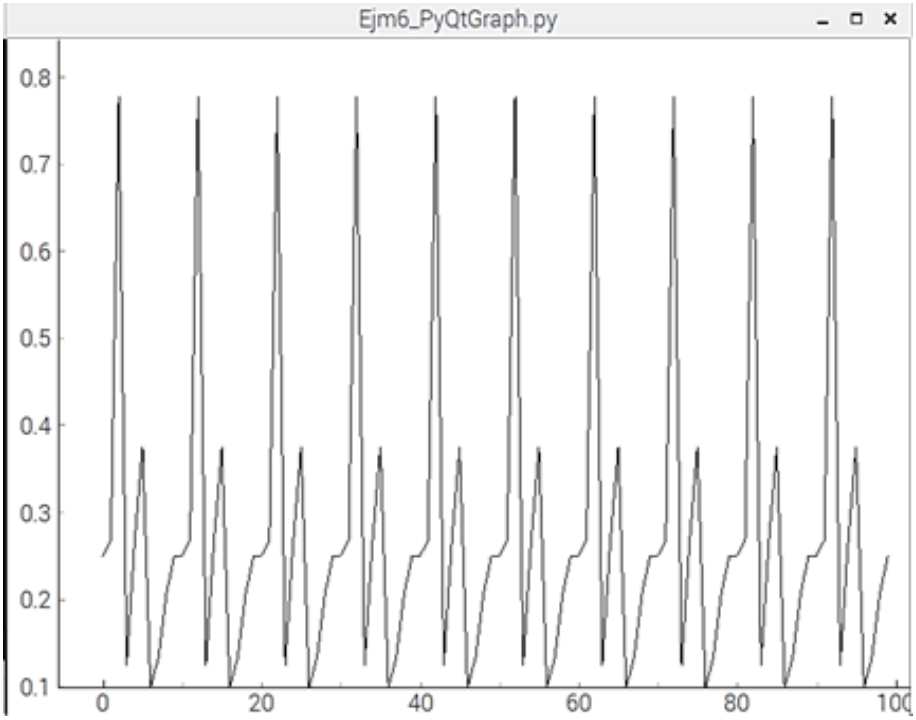

Figura 55. Gráfica del ejemplo 4 de PyQt. Representación de lista de 1000 números.

En este último ejemplo, lo que se hace es representar, mediante PyQt, la primera columna de una lista de mil números. Esos mil números se encuentran repartidos en diez columnas de cien, cada una. En cada columna, hay diez repeticiones de los mismos diez números.

Conclusión

Finalmente, dadas las ventajas e inconvenientes de estos dos módulos y después de haber hecho varias pruebas, la decisión es la de utilizar PyQtGraph como librería gráfica. Las razones concretas por las que se selecciona son: a pesar de que la información disponible es menor, es más rápido que Tkinter. Como se dijo anteriormente, en las últimas versiones de este último módulo han aumentado su rapidez, pero siguen sin llegar al nivel de PyQtGraph. Además, la apariencia de PyQt es nativa, al contrario que en TKinter. Y la cantidad de herramientas y funciones a disposición del usuario es mucho mayor.

7.4. QT4 DESIGNER

Otra herramienta que se ha utilizado para ayudar a la elaboración de un menú, es Qt4 Designer. Es un programa que permite desarrollar aplicaciones gráficas basadas en PyQt, de una forma sencilla y sin tener que elaborar directamente el código. Presenta una gran variedad de opciones que ayudan al diseño de las aplicaciones. Estas pueden consistir, únicamente, en un botón que permite realizar una acción; o incluso, en menús más avanzados, en los cuales se puede encontrar una barra de herramientas, varios botones, etc. Además, una vez acabada una aplicación, puede generar el código en Python.

Para su instalación, se ha ejecutado en la terminal este comando:

```
sudo apt-get install python-qt4 qt4-designer
```

La apariencia inicial del programa es la que aparece en la siguiente imagen. Al abrir el programa aparecen dos ventanas: la de trabajo (la grande), y la de selección de forma (la pequeña). Primero se debe seleccionar lo que se quiere desarrollar. Para desarrollar la ventana de menú en este proyecto, se ha seleccionado la cuarta opción, Main Window. Una vez hecho esto, se pasa a la ventana de trabajo, donde se irán colocando todos los elementos que se quieren ver en la aplicación. En el lado izquierdo de la imagen aparecen todos los elementos que se pueden añadir. La función que se quiere que desarrollen se puede seleccionar desde el programa o, una vez creada la ventana, en el código generado.

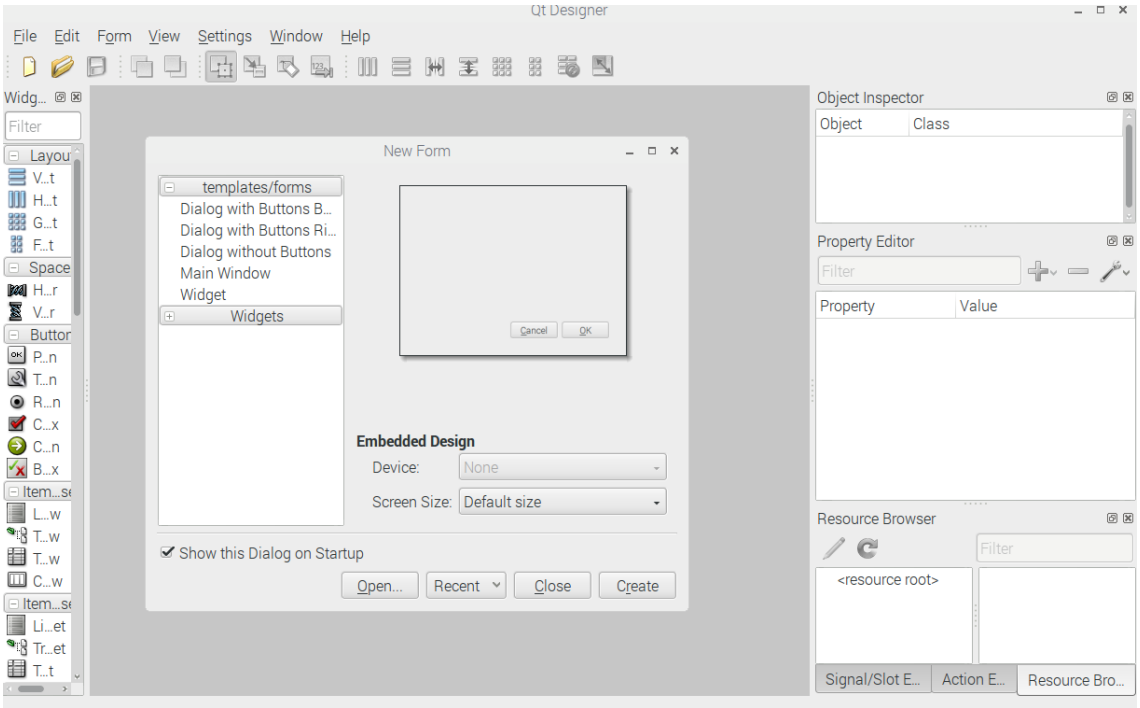

Imagen 56. Vista inicial de Qt4 Designer.

8. COMUNICACIÓN BLUETOOTH

Para la comunicación por Bluetooth se necesita un adaptador USB y una librería específica en Python. Como el medidor de frecuencia cardíaca utiliza BLE (Bluetooth Low Energy) o Bluetooth 4.0, el adaptador debe soportar lo mismo.

8.1. BLUETOOTH 4.0

La versión 4.0 de Bluetooth fue adoptada en junio de 2.010 y actualmente es utilizada por una gran variedad de dispositivos electrónicos. Algunas de las características que presenta son:

- Menor consumo de energía, en comparación con las versiones anteriores. Lo que aumenta la duración de la batería Este factor es especialmente útil en dispositivos pequeños.
- Velocidad de transmisión de 32 Mb/s.
- Distancia de lectura de hasta 100 metros.
- Opera a una frecuencia de 2,4 GHz
- En cuanto a seguridad, emplea el sistema de cifrado AES y presenta esquemas de seguridad configurables.

8.2. UNOTEC ADAPTADOR BLUETOOTH 4.0 USB

Se han hecho pruebas con un adaptador 2.0 y otro 4.0. El primero de ellos, a pesar de detectar dispositivos como el teléfono móvil, no puede detectar el sensor Polar. Sin embargo, con el segundo sí que se establece conexión con éxito. El modelo utilizado para BLE es el 'Unotec Adaptador Bluetooth 4.0 USB'. Es compatible con Raspbian, Windows 98, 98SE, ME, 2000, XP, Vista y Windows 7 y 8. La velocidad de transferencia de información es de 3MPB/s, y el rango máximo, de 20 metros.

Figura 57. Unotec Adaptador Bluetooth 4.0 USB.

8.3. ESTABLECIMIENTO DE LA COMUNICACIÓN

Para poder establecer comunicación, a partir de la terminal, entre la Raspberry Pi y otro dispositivo mediante Bluetooth se deben introducir una serie de comandos. Todos se ejecutan como 'super usuario'.

`sudo hciconfig hci0 up`. Este comando sirve para activar el Bluetooth de la Raspberry.

`sudo hciconfig hci0 down`. Cambiando `up` por `down` se deja de apaga la comunicación Bluetooth.

`sudo rfkill unblock all`. A veces, al intentar activar el Bluetooth aparece el siguiente eroor: `Can't init device hci0: Operation not posible due to RF-kill(132)`. Se debe a que hci0 ya aparece como `up`. Para solucionarlo es necesario introducir este comando.

`sudo service Bluetooth stop`. Este comando permite finalizar definitivamente la comunicación.

`sudo hcitool lescan/scan`. Para buscar dispositivos con los que establecer comunicación existen dos comandos. Aparte de los dispositivos, también muestra las direcciones de los mismos. Se introduce uno u otro en función de si se utiliza el protocolo de Bluetooth clásico o si se utiliza Bluetooth 4.0. En el primer caso se debe escribir `sudo hcitool scan`, mientras que en el segundo, `sudo hcitool lescan`.

`sudo hciconfig dev` y `sudo hcitool dev`. Son opcionales. Muestran otros dispositivos.

En las siguientes imágenes se muestra el uso de estos comandos. En la primera se utilizan `sudo hciconfig hci0 down` y `sudo hciconfig hci0 up` para apagar la conexión Bluetooth y para volverla a conectar. Luego se observa el resultado de introducir `sudo hciconfig dev`, `sudo hcitool dev` y `sudo hciconfig lescan`. En la segunda se ve un ejemplo de `sudo hcitool lescan` en el que se detecta el dispositivo de medición de pulsaciones. En la tercera aparece el error comentado anteriormente al usar `sudo hciconfig hci0 up`, por lo que se debe introducir `sudo rfkill unblock all`; también se utiliza `sudo hcitool`, que muestra un teléfono móvil y su dirección.

```
pi@raspberrypi:~ $ sudo hciconfig hci0 down
pi@raspberrypi:~ $ sudo hciconfig hci0 up
pi@raspberrypi:~ $ sudo hciconfig dev
hci0:   Type: BR/EDR  Bus: USB
        BD Address: 00:1A:7D:DA:71:13  ACL MTU: 310:10  SCO MTU: 64:8
        UP RUNNING PSCAN
        RX bytes:1794 acl:0 sco:0 events:102 errors:0
        TX bytes:1772 acl:0 sco:0 commands:102 errors:0

pi@raspberrypi:~ $ sudo hcitool dev
Devices:
        hci0    00:1A:7D:DA:71:13
pi@raspberrypi:~ $ sudo hciconfig lescan
hci0:   Type: BR/EDR  Bus: USB
        BD Address: 00:1A:7D:DA:71:13  ACL MTU: 310:10  SCO MTU: 64:8
        UP RUNNING PSCAN
        RX bytes:1794 acl:0 sco:0 events:102 errors:0
        TX bytes:1772 acl:0 sco:0 commands:102 errors:0
```

Figura 58. Configuración de Bluetooth.

```
pi@raspberrypi:~ $ sudo hcitool lescan
LE Scan ...
00:22:D0:DF:74:0A (unknown)
00:22:D0:DF:74:0A Polar H7 DF740A1C
00:22:D0:DF:74:0A (unknown)
00:22:D0:DF:74:0A Polar H7 DF740A1C
00:22:D0:DF:74:0A (unknown)
00:22:D0:DF:74:0A Polar H7 DF740A1C
00:22:D0:DF:74:0A (unknown)
00:22:D0:DF:74:0A Polar H7 DF740A1C
00:22:D0:DF:74:0A (unknown)
00:22:D0:DF:74:0A Polar H7 DF740A1C
00:22:D0:DF:74:0A (unknown)
00:22:D0:DF:74:0A Polar H7 DF740A1C
00:22:D0:DF:74:0A (unknown)
00:22:D0:DF:74:0A Polar H7 DF740A1C
00:22:D0:DF:74:0A (unknown)
00:22:D0:DF:74:0A (unknown)
00:22:D0:DF:74:0A Polar H7 DF740A1C
```

Figura 59. Ejecución de hcitool lescan.

```
pi@raspberrypi:~ $ sudo hciconfig hci0 up
Can't init device hci0: Operation not possible due to RF-kill (132)
pi@raspberrypi:~ $ sudo rfkill unblock all
pi@raspberrypi:~ $ sudo hciconfig hci0 up
pi@raspberrypi:~ $ sudo hcitool scan
Scanning ...
        58:48:22:12:27:6B       Xperia M4 Aqua
```

Figura 60. Configuración de Bluetooth.

8.4. LIBRERÍAS PARA COMUNICACIÓN BLUETOOTH: BLUEZ Y BLUEPY

Se pueden encontrar varias librerías de Python para la comunicación Bluetooth. Dos de las más utilizadas son Bluez y Bluepy. Esta última cuenta con un módulo, llamado btle.py, de especial importancia para este proyecto.

8.4.1. Bluez

Se utiliza en uno de los ejemplos encontrados donde se explica cómo conectar la Raspberry y el sensor Polar (enlace del ejemplo: http://forum.arduino.cc/index.php?topic=377587.0. Entrada: 02/08/2017).

Las características más importantes de esta librería son: implementación modular completa, seguridad multiprocesamiento simétrica, procesamiento de datos multihilo, compatibilidad con varios dispositivos Bluetooth, interfaz de socket estándar para todas las capas y soporte de seguridad a nivel de dispositivos y servidores. Además, la misma librería está constituida por varios módulos: el núcleo del subsistema del kernel, capas de kernel de audio L2CAP y SCO, RFCOM, BNEP, CMTP y HIDP y otras implementaciones del kernel, etc.

Los Sistemas Operativos en los que tiene soporte son Linux (Debian, Ubuntu, Fedora, OpenSuSE, Mandrake y Gentoo) y Chrome OS.

Para su descarga e instalación se han seguidos los siguientes pasos:

Primero, al igual que con otras librerías o programas en Raspbian, se actualiza el sistema operativo, lo cual se consigue mediante las dos siguientes líneas de comando.

```
sudo apt-get update
sudo apt-get upgrade
```

Lo siguiente es descargar bluez. El primer grupo de comandos descarga elementos necesarios para la instalación de la librería. Luego se crea un directorio llamado bluez, al que se accede. `wget` descarga la librería desde la dirección dada.

```
sudo apt-get install libdbus-1-dev libdbus-glib-1-dev libglib2.0-dev libical-dev libreadline-dev libudev-dev libusb-dev make
mkdir -p bluez
cd bluez
wget https://www.kernel.org/pub/linux/bluetooth/bluez-5.37.tar.xz
```

Una vez descargada, para su instalación se debe introducir en la terminal, lo siguiente:

```
cd bluez-5.37
```

```
./configure -disable-systemd
make
sudo make install
```

Para visualizar su uso, se presentan algunos ejemplos:

Ejemplo 1

En este primer ejemplo se imprimen por pantalla los dispositivos detectados, su dirección y su nombre.

```
# simple inquiry example
import bluetooth

nearby_devices=
bluetooth.discover_devices(lookup_names=True)
print("found %d devices" % len(nearby_devices))

 for addr, name in nearby_devices:
print("  %s - %s" % (addr, name))
```

Ejemplo 2

```
# bluetooth low energy scan
    from bluetooth.ble import DiscoveryService


    service = DiscoveryService()
    devices = service.discover(2)


    for address, name in devices.items():
        print("name: {}, address: {}".format(name,
        address))
```

8.4.2. BluePy

Otra de las interfaces de conexión Bluetooth y Python es Bluepy. Está desarrollada principalmente para funcionar en Linux. Funciona tanto en Python 2.7 como en Python 3.4.

Para la descarga e instalación de la misma se hace uso de la terminal. Hay dos formas de hacerlo. Una de ellas, la cual se puede utilizar en casi todos los sistemas Linux, es mediante dos comandos:

```
sudo apt-get install python-pip libglib2.0-dev
sudo pip install bluepy
```

Sin embargo, si se quiere instalar desde la fuente, es preciso proceder de la siguiente manera:

```
sudo apt-get install git build-essential libglib2.0-dev
git clone https://github.com/IanHarvey/bluepy.git
cd bluepy
python setup.py build
sudo python setup.py install
```

Como se ha indicado con anterioridad, esta librería contiene un módulo, btle.py, que permite conectar dispositivos mediante Bluetooth. Por tanto, es la que se utiliza en todas las pruebas. A continuación, se presenta un ejemplo en el que se reciben diez pulsaciones y el instante de tiempo, en segundos, en el que se reciben. El código incluye dos modificaciones sobre el original. Una de ellas es el guardado de las pulsaciones en un archivo de texto. La otra aparece comentada más adelante. El ejemplo se ha encontrado en la página www.github.com.

Primero se importan el módulo btle y la librería time.

```
from bluepy import btle
import time
```

Después se crea una clase que permite definir la dirección y el tipo de dirección del dispositivo. Aquí se ha realizado un cambio respecto al código original, ya que daba error. En la tercera línea, donde aparece escrito `addrType="public"`, en vez de public aparecía otro término entre comillas.

```
class HRM(btle.Peripheral):
    def __init__(self, addr):
        btle.Peripheral.__init__(self,addr,
addrType="public")

if __name__=="__main__":
    cccid                                                =
btle.AssignedNumbers.client_characteristic_configuration
    hrmid = btle.AssignedNumbers.heart_rate
    hrmmid = btle.AssignedNumbers.heart_rate_measurement
```

Mediante la instrucción `open("Nombre del archivo", "w")` se crea el archivo en la dirección indicada, en modo escritura (“w”), en el que se van a guardar los valores de las pulsaciones. Luego se escribe en la primera línea del archivo `bpm:`.

```
    f = open("/home/pi/Desktop/12345.txt", "w")
    f.write('bpm:\r')#   t(s)\r')

    hrm = None
    try:
```

Se indica la dirección del dispositivo: 00:22:D0:DF:74:0A.

```
service, = [s for s in hrm.getServices() if s.uuid==hrmid]
        ccc,                                                    =
service.getCharacteristics(forUUID=str(hrmmid))

        if 0: #No funciona
            ccc.write('\1\0')

        else:
            desc = hrm.getDescriptors(service.hndStart,
                                      service.hndEnd)
            d, = [d for d in desc if d.uuid==cccid]
            hrm.writeCharacteristic(d.handle, '\1\0')
```

Ahora se define una función para mostrar los datos. Primero se determina el tiempo inicial, a partir del cual se va a calcular el instante en el que se recibirá cada pulsación. La función recibe dos parámetros. `Data` es el que contiene las pulsaciones.

```
        t0=time.time()
        def print_hr(cHandle, data):
            bpm = ord(data[1])
            print bpm,"%.2f"%(time.time()-t0) # "%.2f"%(
time.time()-t0) indica que el tiempo se indica con dos
decimales (0.2f).
            z = bpm
            y = time.time()-t0
            z = str(z)
            y = str(y)
```

Mediante `f.write()`, en este caso, escribo cada variable, seguida de un punto y coma, en una línea distinta.

```
          f.write(z + ';' + '\r')#' ' + y + ';' + '\r')
        hrm.delegate.handleNotification = print_hr

        for x in range(10):         #Para variar el numero
de muestras, cambiar el numero
            hrm.waitForNotifications(3.)

    finally:
        if hrm:
            f.close()                #Cierro el fichero
            hrm.disconnect()
```

```
usuario@usuario-G56JK:~$ cd Escritorio/TFG/Python/Prog_avances
usuario@usuario-G56JK:~/Escritorio/TFG/Python/Prog_avances$ python Blue1.py
55 0.13
54 1.08
54 2.09
54 3.10
54 4.12
53 5.13
53 6.07
53 7.09
53 8.10
53 9.11
```

Figura 61. Ejecución del programa Blue1.py.

La columna de la izquierda contiene las medidas de las pulsaciones por minuto (bpm) y, la de la derecha, el instante de tiempo en el que se toman, medido desde el inicio del programa.

Después de probar ambas librerías, se ha elegido la segunda, bluepy. La elección se debe a que con esta se ha conseguido conectar la banda de medición más fácilmente, y los ejemplos encontrados resultan más fáciles de utilizar y adaptar al código que ya se tiene.

9. PRUEBAS REALIZADAS CON EL MEDIDOR DE FRECUENCIA CARDIACA Y LA RASPERRY PI.

Las primeras pruebas han tenido el fin de comunicar el sensor de bluetooth con la Raspberry Pi. Para ello se ha hecho uso del programa que aparece como ejemplo en el apartado 6.1.2. Bluepy que, además, se muestra a continuación. Se han realizado algunas modificaciones. Sobre todo, orientadas a la cantidad máxima de muestras tomadas.

```
from bluepy import btle
import time


class HRM(btle.Peripheral):
    def __init__(self, addr):
        btle.Peripheral.__init__(self,addr,
addrType="public")

if __name__=="__main__":
    cccid                                                    =
btle.AssignedNumbers.client_characteristic_configuration
    hrmid = btle.AssignedNumbers.heart_rate
    hrmmid = btle.AssignedNumbers.heart_rate_measurement


    f = open("/home/pi/Desktop/12345.txt", "w")
    f.write('bpm:\r')#   t(s)\r')

    hrm = None
    try:
        hrm = HRM('00:22:D0:DF:74:0A')

        service,  =  [s  for  s  in  hrm.getServices()  if
s.uuid==hrmid]
        ccc,                                                 =
service.getCharacteristics(forUUID=str(hrmmid))

        if 0: #No funciona
            ccc.write('\1\0')

        else:
            desc = hrm.getDescriptors(service.hndStart,
                                      service.hndEnd)
            d, = [d for d in desc if d.uuid==cccid]

            hrm.writeCharacteristic(d.handle, '\1\0')
```

```
        t0=time.time()
        def print_hr(cHandle, data):
            bpm = ord(data[1])
            print bpm,"%.2f"%(time.time()-t0) # "%.2f"%(
time.time()-t0) indica que el tiempo se indica con dos
decimales (0.2f).
            z = bpm
            y = time.time()-t0
            z = str(z)
            y = str(y)
          f.write(z + ';' + '\r')#' ' + y + ';' + '\r')
        hrm.delegate.handleNotification = print_hr

        for x in range(10):
            hrm.waitForNotifications(3.)

    finally:
        if hrm:
            f.close()
            hrm.disconnect()
```

En esta parte de código se limita el número de muestras.

```
for x in range(10):
            hrm.waitForNotifications(3.)
```

Lo único que se debe hacer es aumentar o disminuir el número contenido dentro del paréntesis de `range(10):`. En caso de querer tomar un número indefinido de medidas, se puede cambiar el bucle y escribir lo siguiente:

```
x = 0
while x == 0:
            hrm.waitForNotifications(3.)
```

En caso de optar por la opción de un número indeterminado de muestras, hay tres formas de poder salir del bucle. Una es añadiendo una condición que haga que la variable 'x' cambie de valor, por ejemplo, si se recibe un valor de pulsaciones mayor a cien. Otra es pulsando `'Ctrl + C'`. La tercera es mediante la función 'keep alive' que tienen los dispositivos BLE y que viene incluida en el módulo btle.py. En este proyecto interesa que el número de medidas sea indefinido, para poder tomar muestras durante el tiempo que se desee.

Una vez se ha conseguido conectar el Polar H7 con la Raspberry y se ha transmitido información, lo que se pretende es recibir los datos y representarlos simultáneamente. Se utilizan dos códigos que se comunican entre sí por puerto serie:

mientras uno recibe los datos por bluetooth, los envía por puerto serie y los guarda en un archivo .txt, otro recibe los datos por el puerto serie y los representa gráficamente.

El código empleado para representar las medidas realizadas es el siguiente.

Importación de librerías

```
import time
import serial
import pyqtgraph as pg
import sys
from PyQt4 import QtGui, QtCore
import numpy as np
import pyqtgraph.console
from pyqtgraph.dockarea import *
```

Definición de la clase 'RingBuffer', utilizada en pruebas anteriores, que permite almacenar los datos recibidos.

```
class RingBuffer:

    def __init__(self, size):
        self.data = [0 for i in xrange(size)]

    def append(self, x):
        self.data.pop(0)
        self.data.append(x)

    def get(self):
        return self.data

    def __getitem__(self, index):
        return self.data[index]

    def __len__(self):
        l = len(self.data)
        return l

    def __setitem__(self, index, value):
        self.data[index] = value
```

Definición del puerto serie

```
ser = serial.Serial(
    port='/dev/pts/1',
    baudrate=9600,
    parity=serial.PARITY_ODD,
    stopbits=serial.STOPBITS_TWO,
    bytesize=serial.EIGHTBITS
)

ser.isOpen()

app = QtGui.QApplication(sys.argv)
```

Declaración de variables

```
c_1 = RingBuffer(180)
media = RingBuffer(180)
ptr = 0
rx_len = 0
suma = 0
m = 0
b1 = 0
bpm = 0


print ('Recepcion de datos:')
```

Definición de la función recepcion(), que cumple la función de obtener los datos del puerto serie.

```
def recepcion():
    global rx_len, c_1, media, suma, bpm

    j = 0
    while 1:
        out=''
        while ser.inWaiting() > 0:
            out += ser.read(1)
        if '\r' in out:
            a = out.split(';')
            b1 = float(a[0])

            c_1.append(b1)
            rx_len += 1
            suma += b1
            m = suma/rx_len
            media.append(m)
```

```
            c_1.append(c1)
            rx_len += 1
            suma += c1
            m = suma/rx_len
            media.append(m)

            c_1.append(d1)
            rx_len += 1
            suma += d1
            m = suma/rx_len
            media.append(m)

            c_1.append(e1)
            rx_len += 1
            suma += e1
            m = suma/rx_len
            media.append(m)

            c_1.append(f1)
            rx_len += 1
            suma += f1
            m = suma/rx_len
            media.append(m)

            j+=5
            if j == 20:
                return c_1
```

Definición de la función `update()` para actualizar los datos recibidos mediante

```
def update():
    global rx_len, ptr, bpmMax, bpmMin, bpmMMax, bpmMMin,
text1, text2
    x = recepcion()
    print x.get()
    ptr+=2
    x[:1] = x[-1:]
    media[:-1] = media[1:]

    curve1.setData(x.get())
    curve1.setPos(ptr, 0)
    curve2.setData(media.get())
    curve2.setPos(ptr, 0)
```

Creación de la ventana de representación de datos. Se ha desarrollado una forma diferente para crear la ventana que representa los datos. La sección de código mostrada a continuación es la que permite crear la ventana. En pruebas posteriores se utilizará esta forma. Las últimas tres líneas activan la rejilla y definen el color y el grosor de línea de las gráficas.

```
win = pg.GraphicsLayout(border=(100,100,100))
view = pg.GraphicsView()
view.setCentralItem(win)
view.show()
view.setWindowTitle('Prueba de bpm')
#El eje x se encuentra por defecto en el lado derecho de
la pantalla.
#El eje y, en el izquierdo.
ancho = 800
alto = 600
view.resize(ancho, alto)

#DEFINICION DE LAS GRAFICAS
p1 = win.addPlot(col=1)
win.nextRow()
p2 = win.addPlot(col=1)

curve1 = p1.plot(pen = pg.mkPen(1, width=2)) #El primer
numero entre parentesis es el color, el segundo es el
ancho
curve2 = p2.plot(pen = pg.mkPen(5, width=2))
p1.setRange(yRange=[50,100])
p1.showGrid(x=True, y=True) #Activacion de la rejilla en
el eje 'x' y en el 'y'.
p2.setRange(yRange=[50,100])
p2.showGrid(x=True, y=True)

timer = pg.QtCore.QTimer()
timer.timeout.connect(update)
timer.start(5)

status = app.exec_()
ser.close()
sys.exit(status)

curve1 = p1.plot(pen = pg.mkPen(1, width=2))
curve2 = p2.plot(pen = pg.mkPen(5, width=2))
p1.showGrid(x=True, y=True)
```

Recepción Bluetooth

```
from bluepy import btle
import time
import serial
import pyqtgraph as pg
import sys
from PyQt4 import QtGui, QtCore
import numpy as np

class RingBuffer:
    def __init__(self, size):
        self.data = [0 for i in xrange(size)]

    def append(self, x):
        self.data.pop(0)
        self.data.append(x)

    def get(self):
        return self.data

    def __getitem__(self, index):
        return self.data[index]

    def __len__(self):
        l = len(self.data)
        return l

    def __setitem__(self, index, value):
        self.data[index] = value

class HRM(btle.Peripheral):
    def __init__(self, addr):
        btle.Peripheral.__init__(self,addr,
addrType="public")

ser = serial.Serial(
    port='/dev/pts/2',
    baudrate=9600,
    parity=serial.PARITY_ODD,
    stopbits=serial.STOPBITS_TWO,
    bytesize=serial.EIGHTBITS
)

ser.isOpen()

app = QtGui.QApplication(sys.argv)
```

```
if __name__=="__main__":
    cccid                                                    =
btle.AssignedNumbers.client_characteristic_configuration
    hrmid = btle.AssignedNumbers.heart_rate
    hrmmid = btle.AssignedNumbers.heart_rate_measurement

    f = open("/home/pi/Desktop/12345.txt", "w") #"w" Si
quiero sobreescribir el .txt con nuevos datos
                                                  #"a"   Si
no quiero sobreescribir
    hrm = None
    bpm = 0

    try:
        hrm = HRM('00:22:D0:DF:74:0A')

        service, = [s for s in hrm.getServices() if
s.uuid==hrmid]
        ccc,                                                 =
service.getCharacteristics(forUUID=str(hrmmid))

        if 0: #No funciona (Segun ponia en el programa
original)
            ccc.write('\1\0')

        else:
            desc = hrm.getDescriptors(service.hndStart,
                                      service.hndEnd)
            d, = [d for d in desc if d.uuid==cccid]
            hrm.writeCharacteristic(d.handle, '\1\0')

        t0=time.clock()
        cont = 0
        z = ''
        def print_hr(cHandle, data):
            global bpm, ptr, z
            bpm = ord(data[1])
            z = bpm
            y = time.clock()-t0
            z = str(z)
            y = str(y)
            bpm = float(bpm)

            if bpm != 0:
                print bpm,"%.2f"%(time.clock()-t0)
                f.write(z + '\n')
```

```
        hrm.delegate.handleNotification = print_hr

        x = 0
        b = 0
        input = ''
        while x == 0:
            hrm.waitForNotifications(.5)

            input += z + ';\n'

            b+=1
            print b
            if b == 5:
                print input + '\r'
                ser.write (input + '\r')
                b = 0
                input = ''
            time.sleep(0.005)

            if bpm == 0:
                x = 1
    finally:
        if hrm:
            f.close()
            hrm.disconnect()

ser.close()

status = app.exec_()
sys.exit(status)
```

El resultado del programa se muestra en las siguientes fotos.

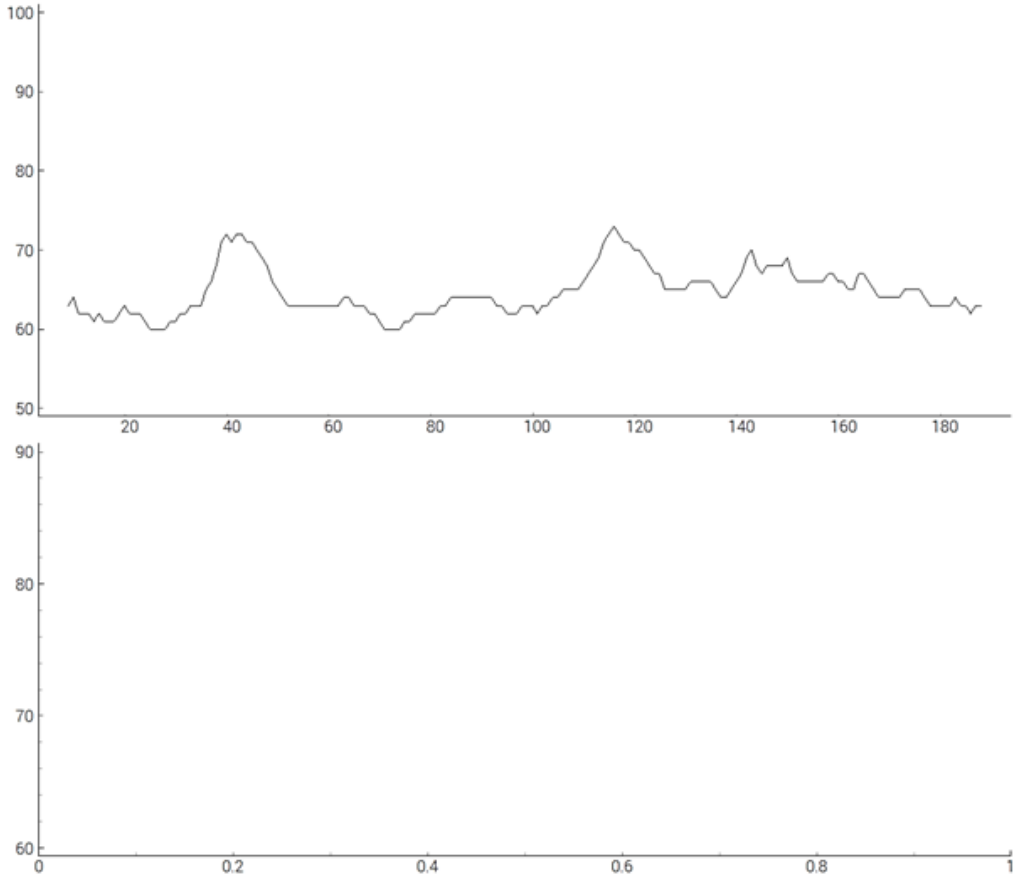

Figura 62.

En este primer caso, solo se muestra la gráfica de las pulsaciones instantáneas. Sin la rejilla, con el mínimo grosor posible y sin modificar el color.

Posteriormente modifica el programa para que aparezca la segunda gráfica, que representa la media de las pulsaciones. No se realiza ningún cambio más.

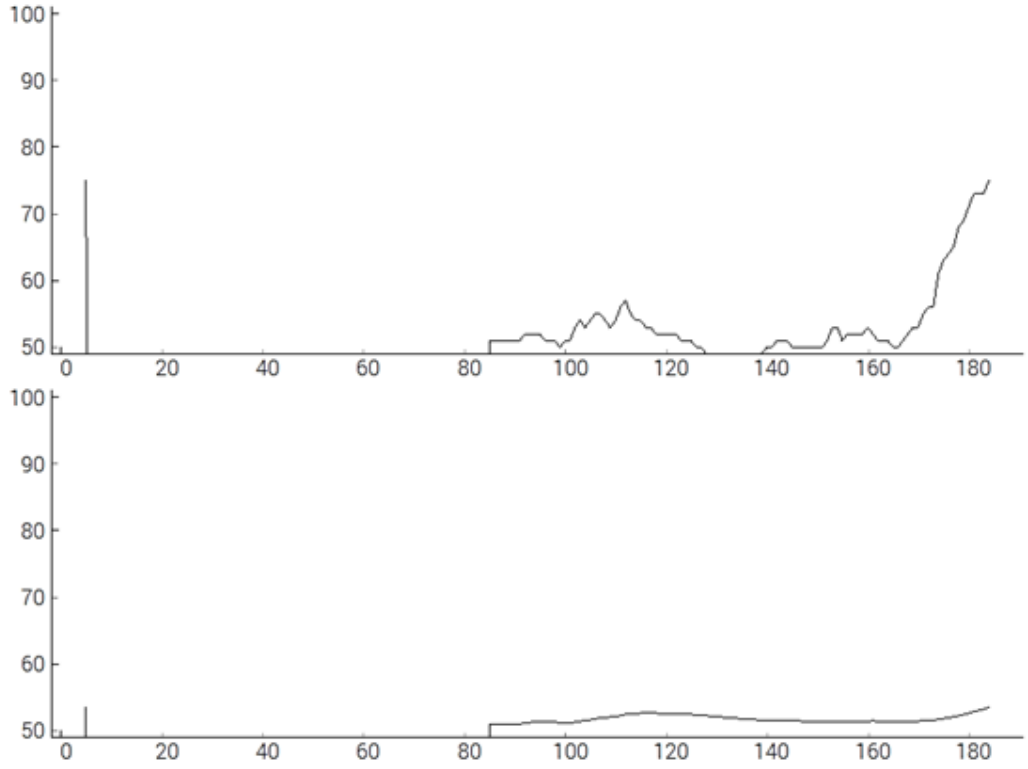

Figura 63.

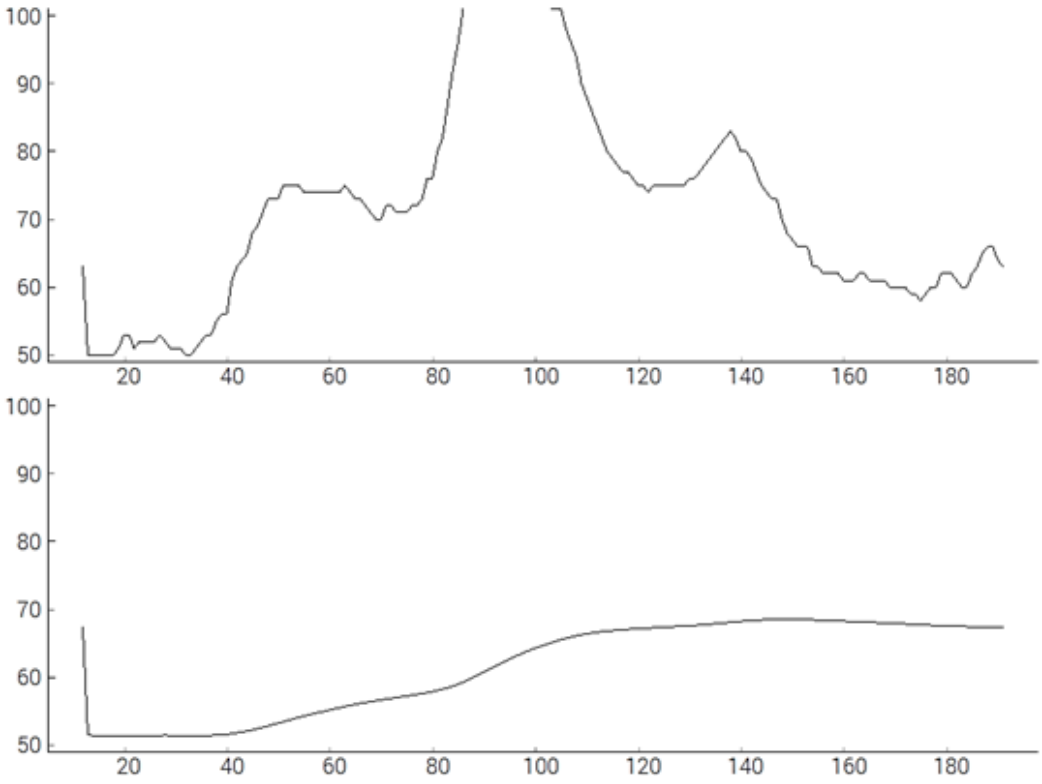

Figura 64.

Las dos imágenes anteriores pertenecen al mismo programa. Se muestran en la gráfica superior las pulsaciones instantáneas, en la inferior la media. Se pueden hacer dos observaciones: la primera es que al principio de la gráfica se representa un valor que no se corresponde con el real. En la primera imagen debería ser cero, porque no se ha

llegado a las suficientes muestras como para completar la gráfica; en la segunda, debería ser más bajo que el valor que se muestra. Aparece como consecuencia de una línea de comando en la que se indica: `x[:1] = x[-1:]`. Para solucionarlo se debe cambiar el signo de ambos números, quedando `x[:-1] = x[1:]`. Respecto a la comunicación serie, las medidas se envían en bloques de cinco. Una vez se reciben en el segundo programa, se espera a que hayan veinte muestras, es decir, que se hayan producido cuatro envíos. Esto hace que la representación gráfica sea lenta. Pasa demasiado tiempo entre la representación de un grupo de veinte datos y el siguiente. Por lo tanto, lo que se decide hacer es enviar cada medida directamente y mostrarla. En la siguiente gráfica se muestra la gráfica con el envío realizado de esta última forma mencionada. También se activa la rejilla, se aumenta el ancho de línea y se cambia el color.

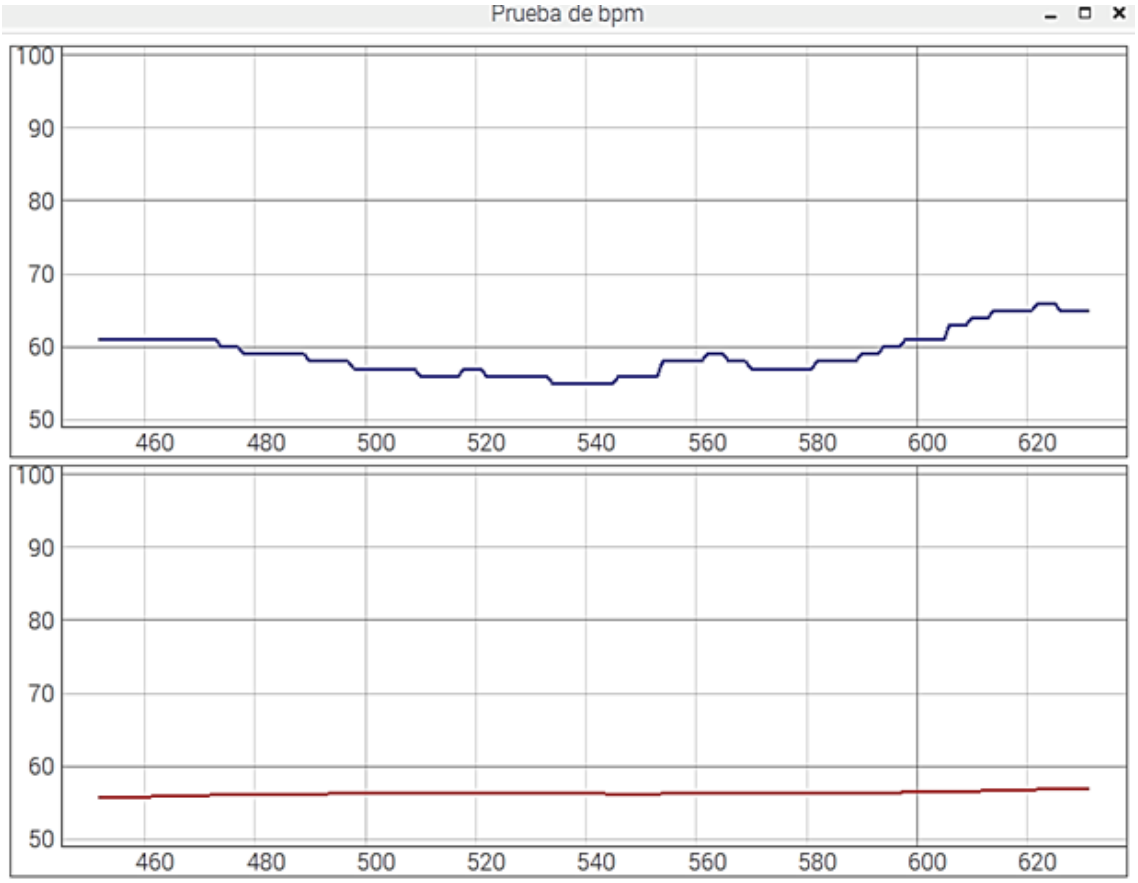

Figura 65.

Una vez se ha conseguido mostrar gráficamente las medidas, el siguiente paso que se ha llevado a cabo es mostrar los datos numéricamente. A esto se ha procedido de varias formas, con el fin de valorar cuál es la mejor presentación.

El código del programa receptor de datos por Bluetooth es idéntico al anterior. En cambio, el que genera la representación sí que se le añaden comandos. A continuación, este último se muestra con las modificaciones:

```
import time
import serial
import pyqtgraph as pg
import sys
from PyQt4 import QtGui, QtCore
import numpy as np
import pyqtgraph.console
from pyqtgraph.dockarea import *

class RingBuffer:
    def __init__(self, size):
        self.data = [0 for i in xrange(size)]

    def append(self, x):
        self.data.pop(0)
        self.data.append(x)

    def get(self):
        return self.data

    def __getitem__(self, index):
        return self.data[index]

    def __len__(self):
        l = len(self.data)
        return l

    def __setitem__(self, index, value):
        self.data[index] = value

ser = serial.Serial(
    port='/dev/pts/1',
    baudrate=9600,
    parity=serial.PARITY_ODD,
    stopbits=serial.STOPBITS_TWO,
    bytesize=serial.EIGHTBITS
)

ser.isOpen()

app = QtGui.QApplication(sys.argv)

#FORMA 1 DE CREAR UNA VENTANA
#win = pg.GraphicsWindow()
#win.setWindowTitle('Reprsentacion de bpm y su media')

c_1 = RingBuffer(180)
```

```
media = RingBuffer(180)
ptr = 0
rx_len = 0
suma = 0
m = 0
b1 = 0
bpm = 0
bpmMax = 0
bpmMin = 1000
bpmMMax = 0
bpmMMin = 1000
bpmMed = 0

print ('Recepcion de datos:')

def recepcion():
    global rx_len, c_1, media, suma, bpm

    j = 0
    while 1:
        out=''
        while ser.inWaiting() > 0:
            out += ser.read(1)
        if '\r' in out:
            a = out.split(';')
            b1 = float(a[0])

            c_1.append(b1)
            rx_len += 1
            suma += b1
            m = suma/rx_len
            media.append(m)

            j+=1
            if j == 1:
                return c_1



def update():
    global rx_len, ptr, bpmMax, bpmMin, bpmMMax, bpmMMin,
text1, text2
    x = recepcion()
    print x.get()
    ptr+=2
    x[:-1] = x[1:]
    media[:-1] = media[1:]
```

```
    bpm = x[-1]
    bpm1 = x[-2]
    bpmMed = media[-1]
    bpmMed1 = media[-2]
```

Cálculo de máximos y mínimos de las medidas

```
#Se calcula el maximo
    if bpm > bpm1:
        if bpm > bpmMax:
            bpmMax = bpm
    elif bpm < bpm1:
        if bpm1 > bpmMax:
            bpm1Max = bpm1
    elif bpm == bpm1:
        if bpm > bpmMax:
            bpmMax = bpm

    if bpmMed > bpmMed1:
        if bpmMed > bpmMMax:
            bpmMMax = bpmMed
    elif bpmMed < bpmMed1:
        if bpm1 > bpmMMax:
            bpm1Max = bpm1
    elif bpmMed == bpmMed1:
        if bpmMed > bpmMMax:
            bpmMMax = bpmMed

#Se calcula el minimo
    if bpm < bpm1:
        if bpm < bpmMin:
            bpmMin = bpm
    elif bpm > bpm1:
        if bpm1 < bpmMin:
            bpmMin = bpm1
    elif bpm == bpm1:
        if bpm < bpmMin:
            bpmMin = bpm

    if bpmMed < bpmMed1:
        if bpmMed < bpmMMin:
            bpmMMin = bpmMed
    elif bpmMed > bpmMed1:
        if bpmMed1 < bpmMMin:
            bpmMin = bpmMed1
    elif bpmMed == bpmMed1:
        if bpmMed < bpmMMin:
```

```
            bpmMMin = bpmMed

    curve1.setData(x.get())
    curve1.setPos(ptr, 0)
    curve2.setData(media.get())
    curve2.setPos(ptr, 0)
```

Definición de la representación numérica

```
    text1.setText('%0.1f max bpm\r %0.1f bpm\r %0.1f min
bpm' % (bpmMax, bpm, bpmMin))
    text2.setText('%0.1f Med max bpm\r %0.1f Med bpm\r
%0.1f Med min bpm' % (bpmMMax, bpmMed, bpmMMin))
    vb1.addItem(text1)
    vb2.addItem(text2)

win = pg.GraphicsLayout(border=(100,100,100))
view = pg.GraphicsView()

view.setCentralItem(win)
view.show()
view.setWindowTitle('Prueba de bpm')
#El eje x se encuentra por defecto en el lado derecho de
la pantalla.
#El eje y, en el izquierdo.
ancho = 800
alto = 600
view.resize(ancho, alto)
```

Definición de la representación numérica

Las siguientes líneas de código corresponden a la primera prueba de representación. Se definen dos textos a mostrar, `text1` y `text2`. Se determina el ángulo respecto a la horizontal con el que aparecerán, el color de fondo de los cuadrados y el borde de los mismos. También se indica la posición en la ventana. Se han desarrollado dos formas para mostrar los textos. Ambas presentan fallos: la primera no actualiza bien los números; con la segunda, el texto desaparece pasado un tiempo. En la siguiente sección de código aparecen ambas formas habilitadas. En las pruebas solo se ha usado una de las dos a un mismo tiempo.

```
#FORMA 1 DE MOSTRAR LOS DATOS NUMERICAMENTE. NO ACTUALIZA
BIEN.
text1 = pg.TextItem(angle=0, border='w', fill=(0, 0, 255,
100))
text1.setPos(ancho-750, alto-585)
view.addItem(text1)
```

```
text2 = pg.TextItem(angle=0, border='w', fill=(0, 0, 255,
100))
text2.setPos(ancho-750, alto-285)
view.addItem(text2)

l = win.addLayout(col=2, border=(50,0,0))
l.addLabel('%0.1f max bpm\r %0.1f bpm\r %0.1f min bpm' %
(bpmMax, bpm, bpmMin))
l.setContentsMargins(10, 10, 10, 10)
l.addLabel('123')

#FORMA 2 DE MOSTRAR LOS DATOS NUMERICAMENTE. DESAPARECEN
PASADO POCO TIEMPO.
vb1 = win.addViewBox(col = 2, lockAspect=True)
text1 = pg.TextItem('%0.1f max bpm\r %0.1f bpm\r %0.1f min
bpm' % (bpmMax, bpm, bpmMin))
vb1.setContentsMargins(-100, 10, 10, 10)
view.addItem(text1)
vb1.setXRange(750, 20)
vb1.setYRange(0, 0)
nm = vb1.screenGeometry()
print 'Geometria de la pantalla = ', nm
vb1.autoRange()
```

Definición de gráficas

```
#DEFINICION DE LAS GRAFICAS
p1 = win.addPlot(col=1)
#p1.setContentsMargins(10, 10, 10, 10)
win.nextRow()
p2 = win.addPlot(col=1)

vb2 = win.addViewBox(col = 2, lockAspect=True)
text2 = pg.TextItem('%0.1f max bpm\r %0.1f bpm\r %0.1f min
bpm' % (bpmMMax, bpmMed, bpmMMin))
view.addItem(text2)
vb2.setXRange(750, 20)
vb2.setYRange(0, 0)
vb2.autoRange()
curve1 = p1.plot(pen = pg.mkPen(1, width=2)) #El primer
numero entre parentesis es el color, el segund es el ancho
curve2 = p2.plot(pen = pg.mkPen(5, width=2))
p1.setRange(yRange=[50,100])
p1.showGrid(x=True, y=True) #Activacion de la rejilla en
el eje 'x' y en el 'y'.
p2.setRange(yRange=[50,100])
```

```
p2.showGrid(x=True, y=True)

timer = pg.QtCore.QTimer()
timer.timeout.connect(update)
timer.start(5)

status = app.exec_()
ser.close()
sys.exit(status)
```

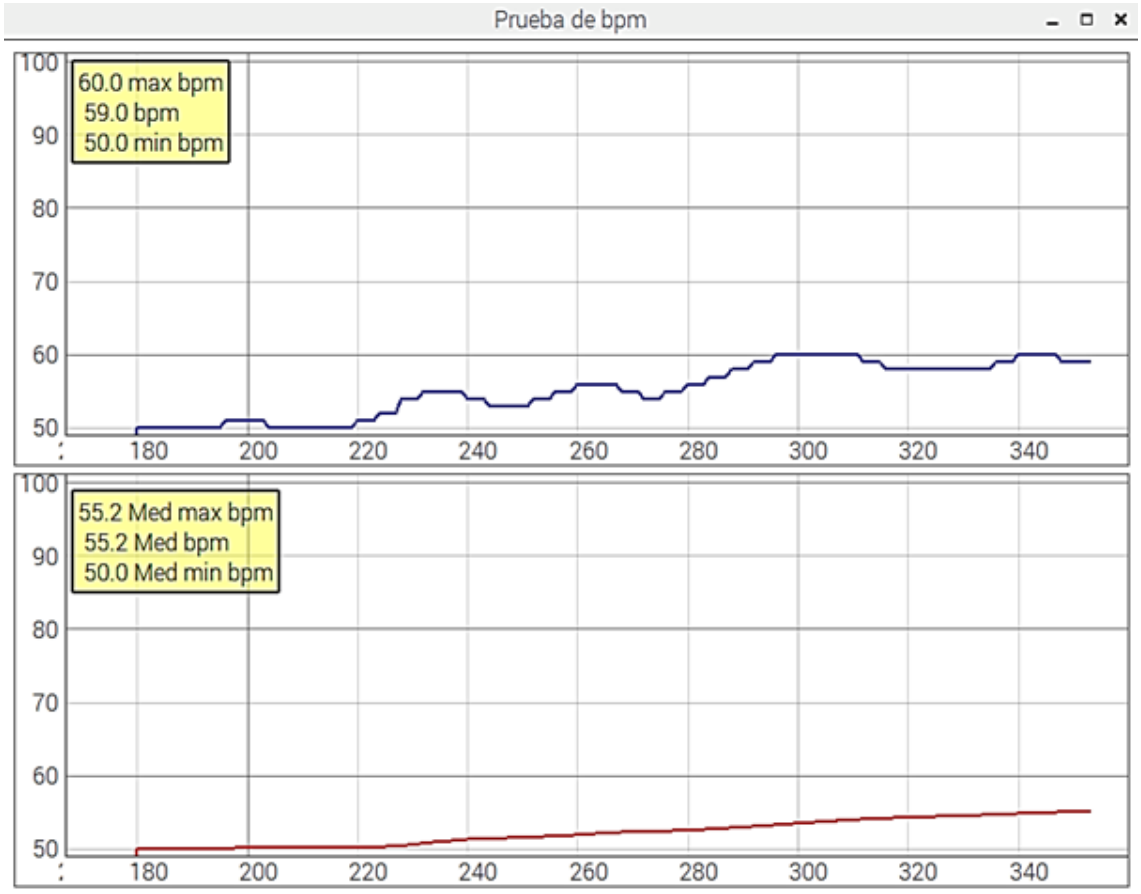

Figura 66.

Aparecen dos cuadrados, cada uno para cada gráfica. En el superior, los números que se muestran son:

60.0 max bpm: Las pulsaciones máximas que se han medido.

59.0 bpm: Las pulsaciones instantáneas.

50.0 min bpm: Las pulsaciones mínimas medidas.

Bpm viene del inglés, ‘Beats per second’.

Esta forma de mostrar los números tiene un problema. La posición de los cuadrados es fija. Es decir, cuando el tamaño de la ventana cambia, los cuadrados no se adaptan a la nueva geometría. En la siguiente imagen se puede observar lo que ocurre.

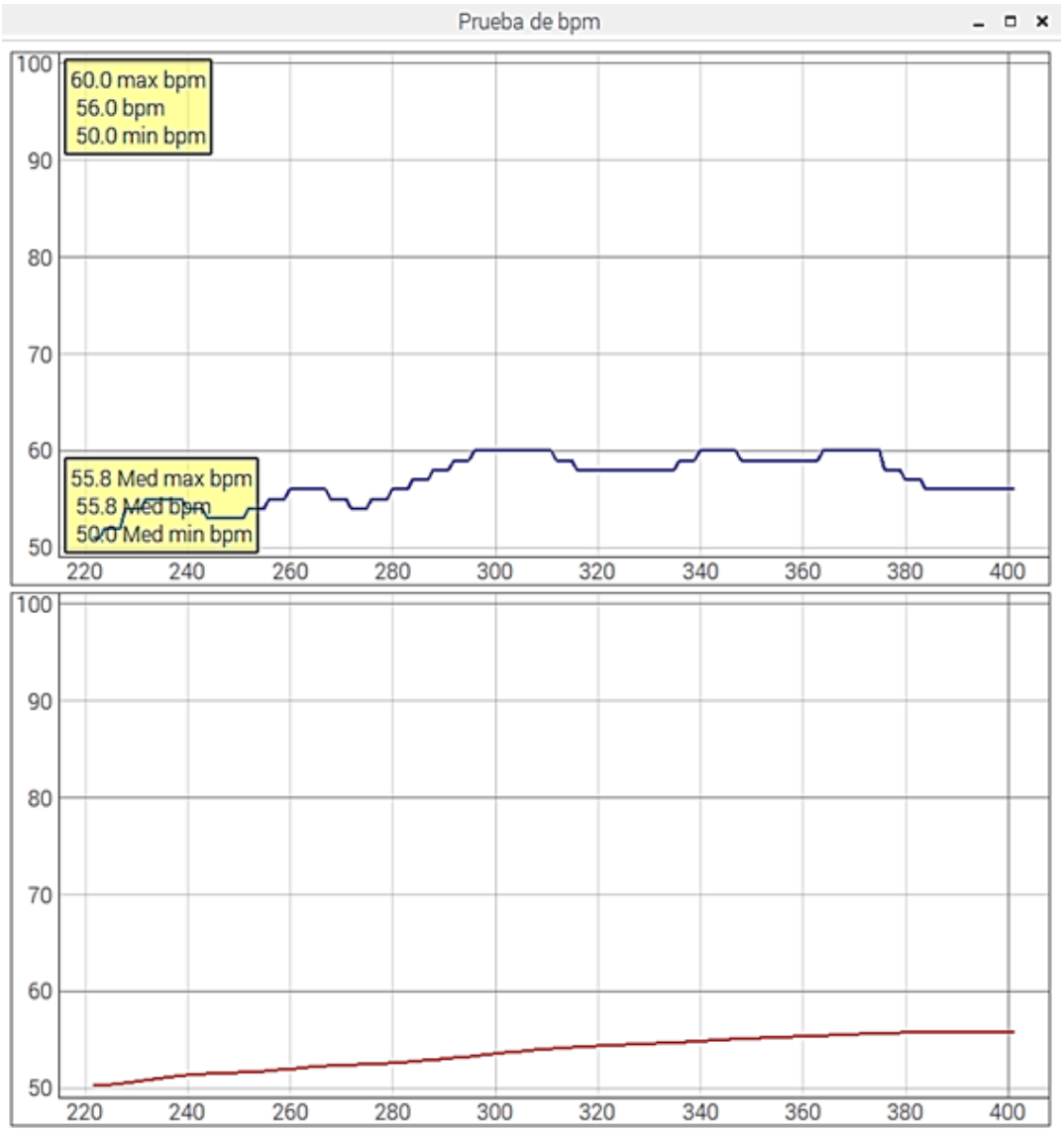

Figura 67.

Posteriormente se ha desarrollado una nueva forma de mostrar los datos numéricamente. El cuadrado en el que se contienen los datos no depende del tamaño de la ventana, pero sí que es demasiado grande. En la primera imagen las dimensiones de la ventana son 800x600. En las siguientes, se amplían.

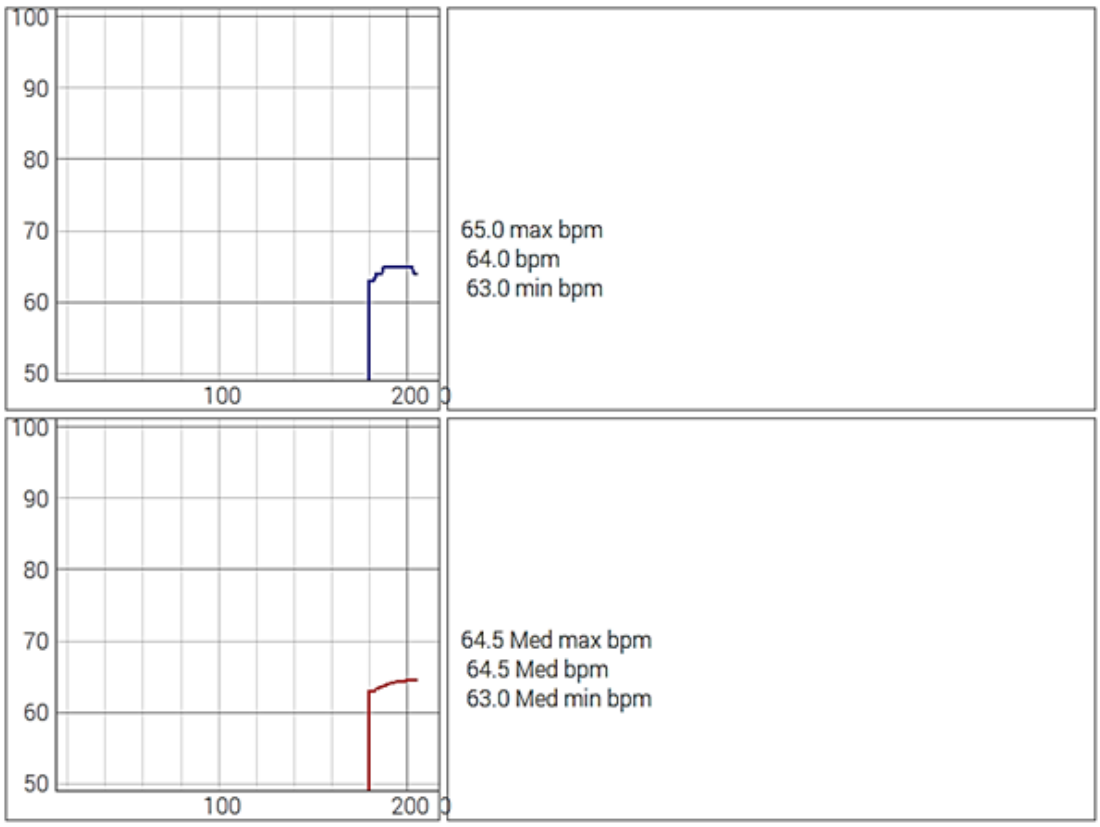

Figura 68.

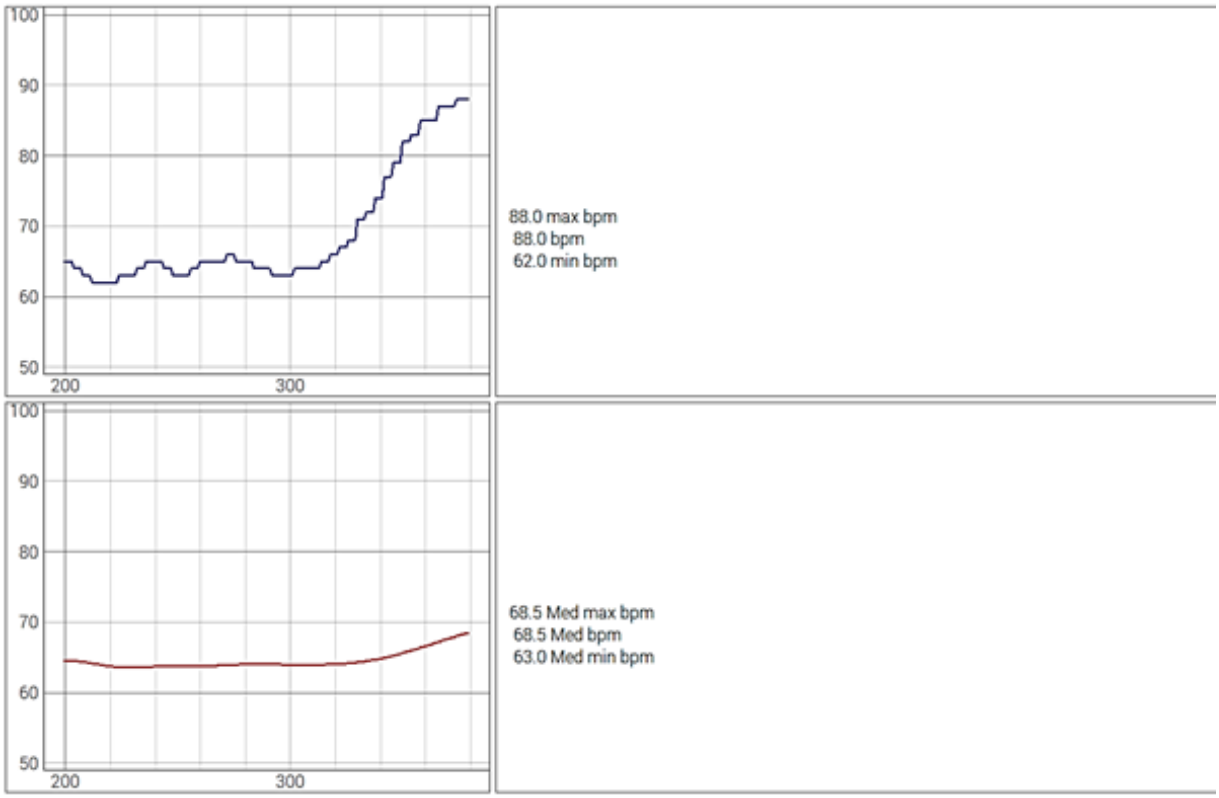

Figura 69.

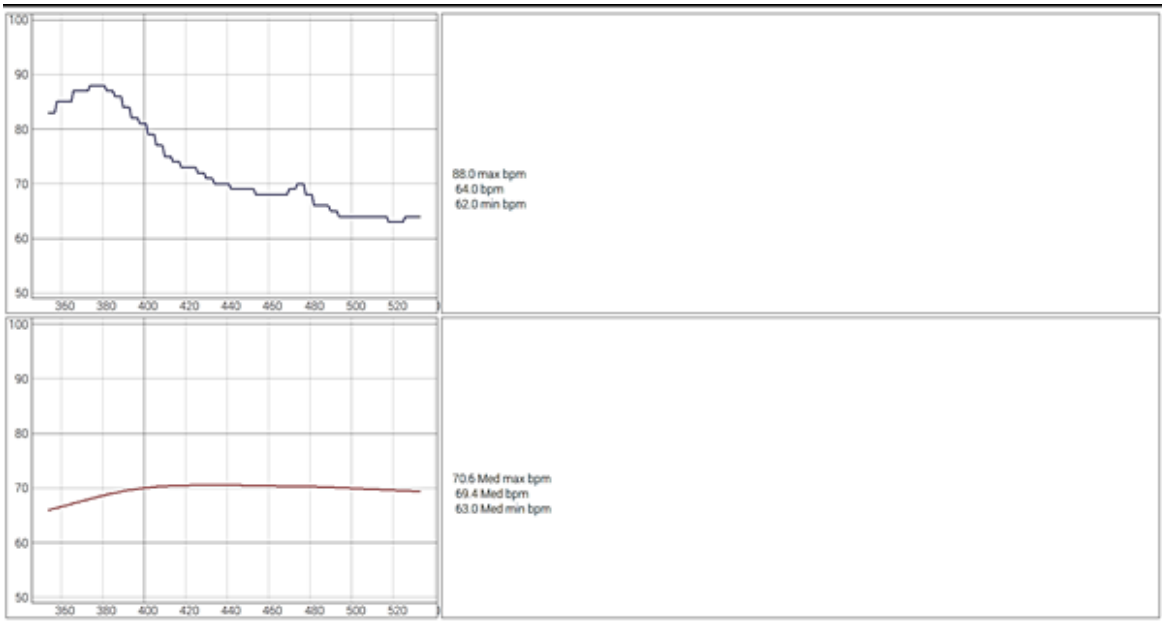

Figura 70.

Para disminuir el tamaño de los cuadrados de datos numéricos, se ha utilizado un comando, el cual no afecta directamente a dichos cuadrados sino a los que contienen las gráficas.

```
win.layout.setColumnMinimumWidth(int column, int minSize)
```

El primer parámetro que se le pasa a la función es el número de columna que se quiere modificar. En este caso es la primera. El segundo es el ancho mínimo de la columna. Existe la posibilidad de modificar la altura de las filas. El mayor inconveniente que se presenta ahora es la dependencia de la función del tamaño de la ventana. Según las dimensiones que tenga la ventana donde se muestra la aplicación, los cuadrados aparecerán de mayor o menor tamaño. Esto se puede ver en las siguientes gráficas, en las que se han utilizado dos tamaños (`int minSize)` distintos, 640 (dos primeras imágenes) y 1640 (dos últimas imágenes).

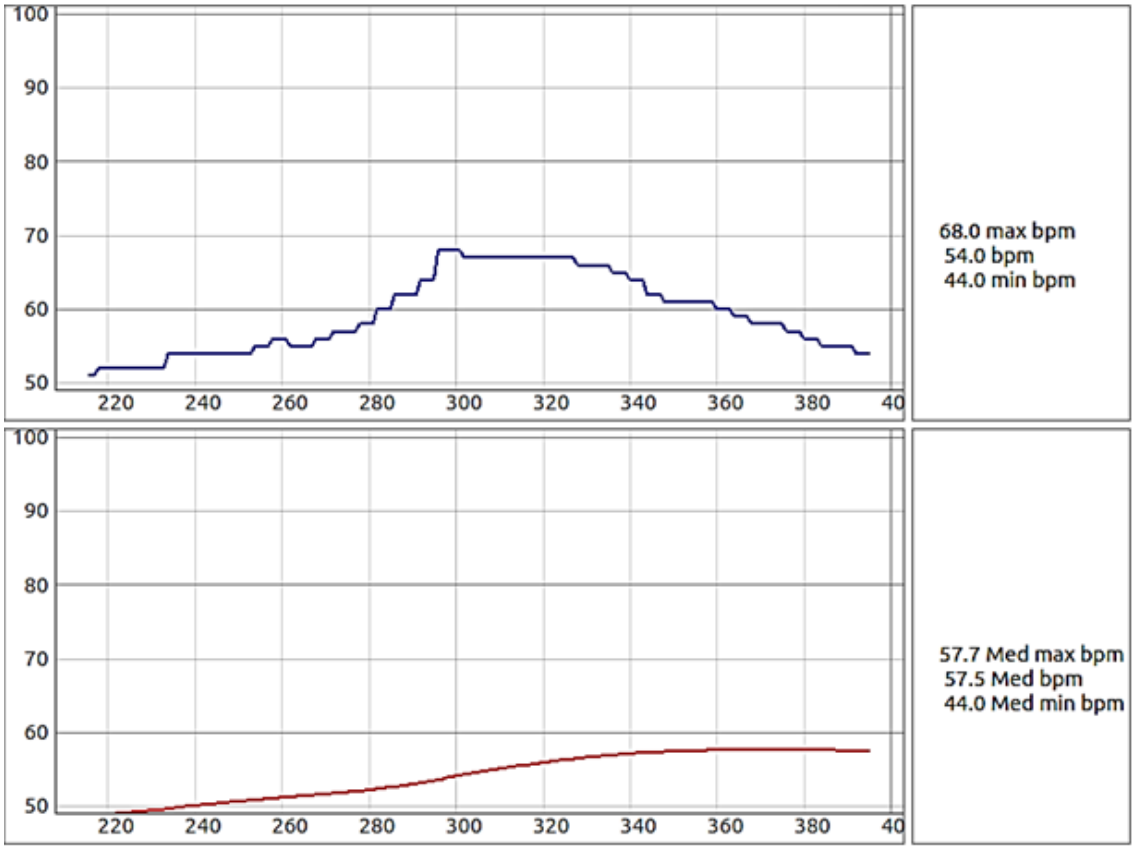

Figura 71.

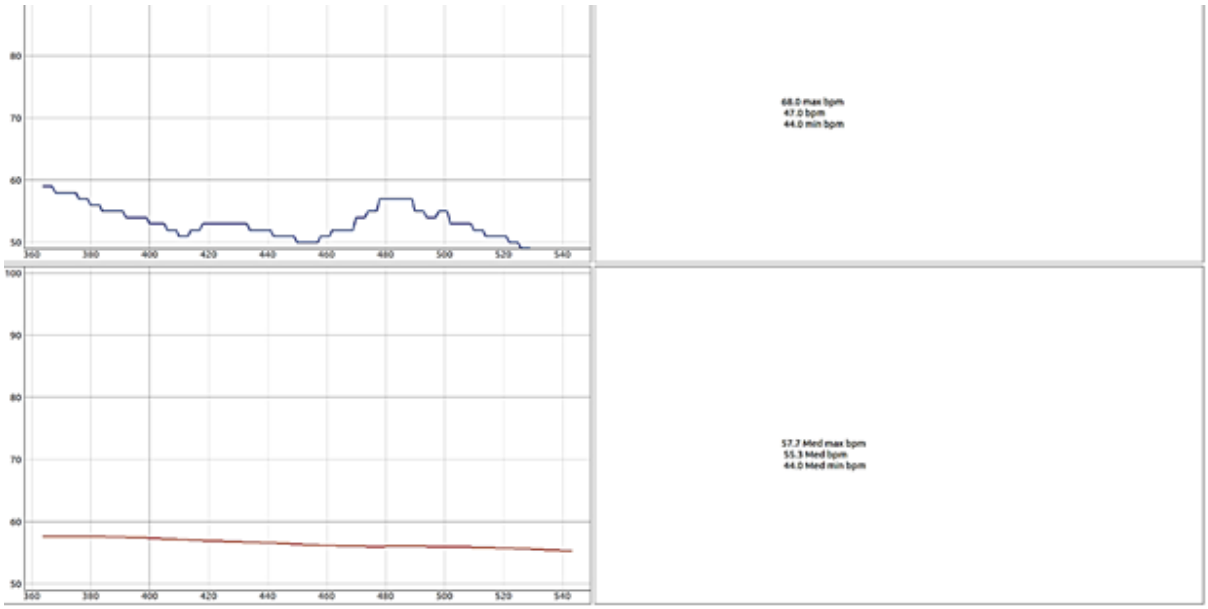

Figura 72.

En la imagen superior, el tamaño de la ventana es de 800x600. En la de abajo ocupa casi toda la pantalla. Los datos numéricos se ajustan mejor en la primera imagen.

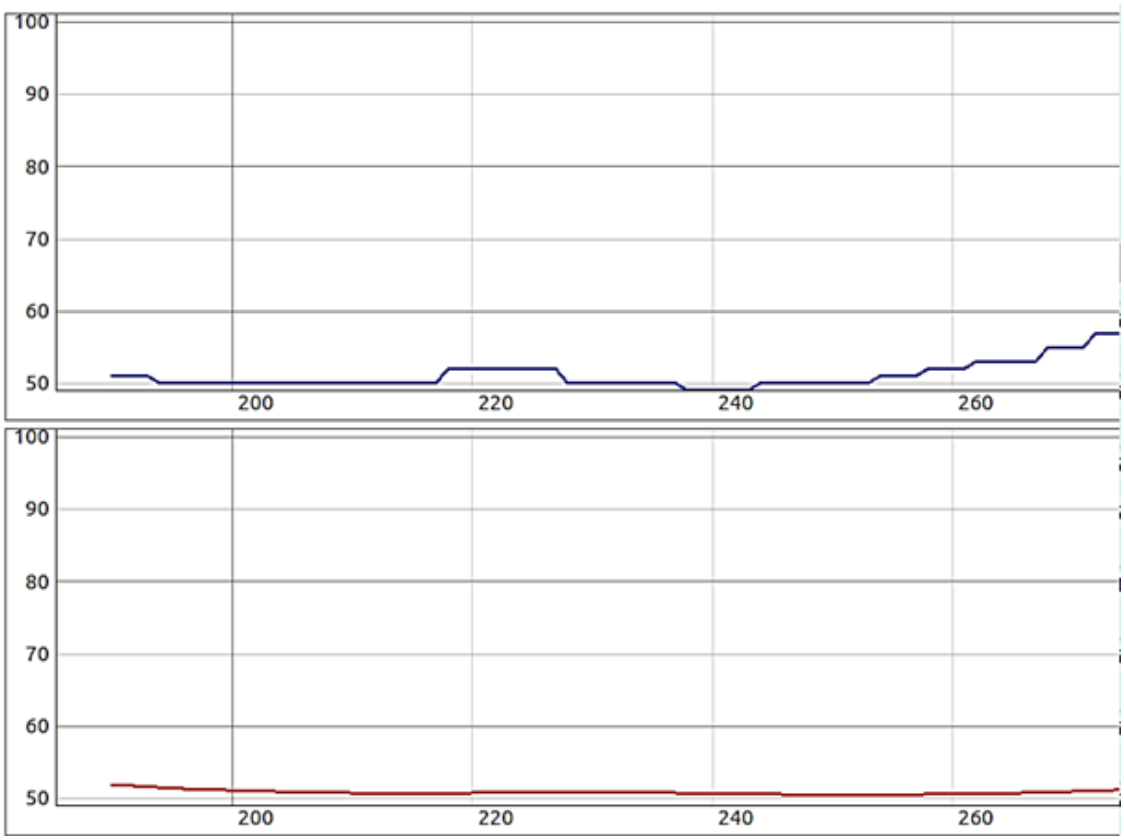

Figura 73.

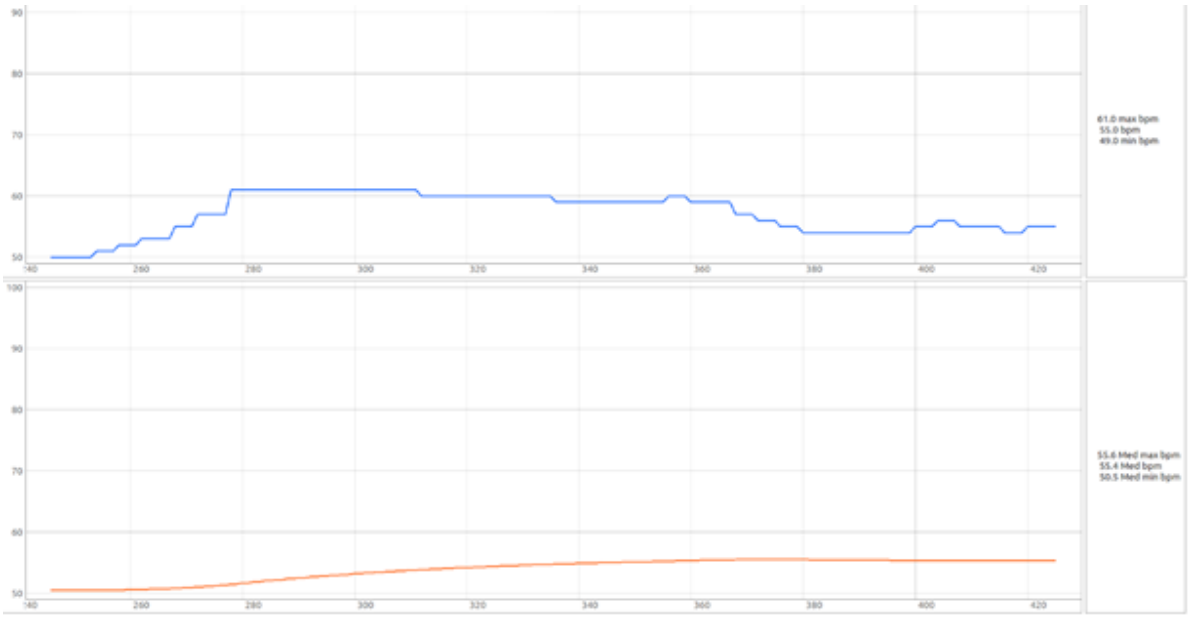

Figura 74.

Ahora ocurre lo contrario que en el caso anterior. Los datos numéricos aparecen bien ajustados cuando el tamaño de la ventana se hace más grande.

10.FICHERO DE DATOS Y FICHERO DE CONFIGURACIÓN

10.1. FICHERO DE DATOS

Los datos que se reciben, se guardan en un fichero de texto para su posterior utilización. El nombre del fichero es la fecha y hora en la que se ha comenzado la medición. Para obtener esta información se necesitan dos librerías que ya se incluyen en Python. Son `datetime` y `calendar`. De la primera se utilizan los módulos `datetime`, `date` y `timedelta`.

Para determinar el nombre y crear el fichero se utilizan las siguientes líneas de comando. Se escriben antes de los comandos de recepción bluetooth.

```
ahora = datetime.now()

nombre = ''

nombre = ahora

f = open("/home/usuario/Escritorio/%s.txt" % (nombre), "a")
```

Primero se declara una variable (`ahora`) para determinar el momento en el que comienza la medición y, así, poder establecer el nombre. Luego se establece la variable `nombre` que va a contener los datos de `ahora`.

Con la función `open()` se crea un archivo en la dirección indicada y con el nombre indicado. En este caso, el nombre que se le asigna es el contenido de la variable `nombre`. La asignación se hace añadiendo al final `%s.txt" % (nombre)`. Dentro de la función, también entre comillas, aparece la letra 'a'. Dependiendo de en qué modo se quiera abrir el fichero, se debe escribir una letra u otra. Las opciones que existen son las que siguen:

'r': Opción por defecto. Sirve para abrir un fichero en modo lectura.

'w': Crea un fichero en modo escritura. Si ya existe, lo sobreescribe.

'a': Sirve para añadir datos al final de un fichero ya existente. Si no existe crea uno nuevo.

't': Apertura en modo texto. Opción por defecto.

'b': Apertura en modo binario.

'+': Apertura en modo escritura y lectura.

Al final de la parte de recepción bluetooth se utiliza la función `f.close()`, donde `f` corresponde al nombre del fichero creado, para cerrar el archivo.

10.2. FICHERO DE CONFIGURACIÓN

El código se desarrolla de forma que, a partir de los datos de un fichero se determinen algunos de los parámetros de funcionamiento del programa. Esos parámetros son:

- La dirección Bluetooth del dispositivo de medida. (`Direccion`)

- El tamaño de los buffers de pulsaciones y de la media de las mismas. (`Muestras_bpm` y `Muestras_media`)

- Los límites máximo y mínimo de las gráficas, es decir, las pulsaciones máximas y mínimas que se mostrarán. (`HRMax` y `Hrmin`).

- Los colores de las gráficas. (`Color_g1` y `Color_g2`).

- Las dimensiones de los ejes 'x' e 'y'. (`EjeX` y `EjeY`).

El fichero donde se guardan los datos es un fichero de texto donde aparecen los datos en el mismo orden que arriba. Todos se encuentran en una sola fila y separados entre ellos por un punto y coma, de la siguiente manera:

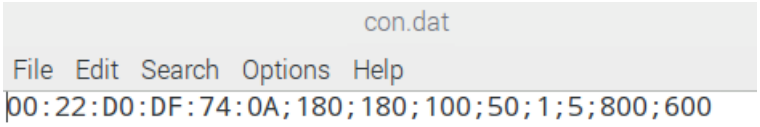

Figura 75.

Para poder leer el archivo, se inserta la siguiente sección de código:

```
c = open("/home/usuario/Escritorio/con.dat", "r")
for i in c:
     a = i.split(';')
     Direccion = a[0]
     Muestras_bpm = int(a[1])
     Muestras_media = int(a[2])
     HRMax = int(a[3])
     HRmin = int(a[4])
     Color_g1 = int(a[5])
     Color_g2 = int(a[6])
     EjeX = int(a[7])
     EjeY = int(a[8])
```

Se utiliza, además, otro archivo de configuración para que se muestre al ejecutar la función `con_dat` del menú de inicio. Esta función se explicará más adelante. El contenido del archivo es igual al anterior. Pero se cambia la disposición de los datos y se añade texto para identificar cada número con el dato al que representa.

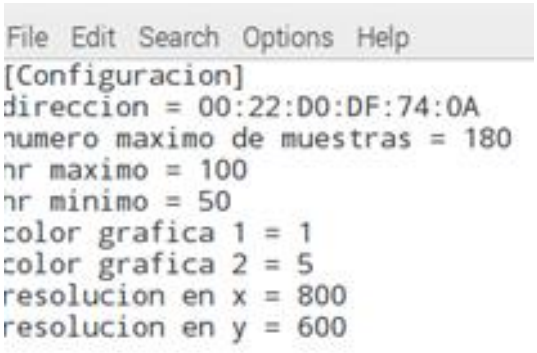

Figura 76. Fichero config.dat.

11. HILOS

11.1. HILOS CON LA LIBRERÍA THREADING

Después de realizar las pruebas de los programas por separado, comunicados por puerto serie, se deben unir ambos en uno solo. Para ello hay que adaptar partes de ambos programas. Sin embargo, si se hace esto directamente no se ejecuta de forma correcta. Aparecen distintos fallos, dependiendo de la parte de programa que se modifique. Los dos más comunes son: la ventana donde se deben mostrar las gráficas no muestra nada; o se muestran las gráficas, pero llegada una determinada cantidad de muestras, la representación se ralentiza. Esto se debe a que se llama a una misma función repetidas veces sin dar tiempo a que se ejecute nada más. La función en cuestión es:

```
hrm.waitForNotifications(3.)
```

Se encarga de esperar a que se reciban datos a través de bluetooth. Se la llama continuamente y no se permite mostrar la gráfica o si se representan los datos, provoca saturación y, por tanto, una disminución de la velocidad de dibujo.

Para solucionar este problema se propone la utilización de hilos. De esta manera, se puede llamar a varias funciones y que operen de forma sincronizada. Python cuenta con la librería threading para programar con hilos.

Primero se ha diseñado un programa con dos hilos: el hilo del programa principal, encargado de la actualización de buffers, y el que contiene la recepción por bluetooth. La definición de variables y la creación de la ventana donde va a aparecer la gráfica se hace fuera de los hilos. El inicio del hilo receptor de bluetooth depende del valor de una variable. Esta se define fuera de los hilos, con valor 'OFF'. En el hilo principal se le asigna el valor 'ON' y permite que el segundo hilo se inicie. La definición de los hilos se hace como se muestra en estas dos líneas de comando:

```
h1 = threading.Thread(name ='h1', target=principal,
args=())
h2 = threading.Thread(name ='h2', target=BLE, args=())
```

El primer parámetro entre paréntesis es el nombre del hilo. El segundo es el nombre de la función que ejecuta ese hilo. El último indica los parámetros que se le pasan. En este caso no se le pasa ninguno.

Después de la definición hay que llamarlos. Esto se hace mediante los comandos:

```
h1.start()
h2.start()
```

Una vez se ha comprobado que el programa funciona con dos hilos, se desarrolla uno nuevo con tres hilos. En este caso se añade uno que actualice las gráficas y los datos numéricos. El código queda de la siguiente manera:

```
from bluepy import btle
import time
import pyqtgraph as pg
import sys
from PyQt4 import QtGui, QtCore
import numpy as np
from datetime import datetime, date, timedelta
import calendar
import ConfigParser
import threading
import menu

app = QtGui.QApplication(sys.argv)

class RingBuffer:
        def __init__(self, size):
                self.data = [0 for i in xrange(size)]

        def append(self, x):
                self.data.pop(0)
                self.data.append(x)

        def get(self):
                return self.data

        def __getitem__(self, index):
                return self.data[index]

        def __len__(self):
                l = len(self.data)
                return l

        def __setitem__(self, index, value):
                self.data[index] = value

class HRM(btle.Peripheral):
        def __init__(self, addr):
                btle.Peripheral.__init__(self, addr, addrType="public")

c_1 = RingBuffer(180)
media = RingBuffer(180)
ptr = 0
rx_len = 0
suma = 0
m = 0
b1 = 0
bpm = 0
bpmMax = 0
bpmMin = 1000
bpmMMax = 0
bpmMMin = 1000
bpmMed = 0
envio = ''
b = 0

BT_ENABLE = 'OFF'
GRF_ENABLE = 'OFF'
DATA_READY = 'OFF'
```

Imagen 77. Código con hilos, librería threading.

```
#CREAR UNA VENTANA
win = pg.GraphicsLayout(border=(100,100,100))
view = pg.GraphicsView()

view.setCentralItem(win)
view.show()
view.setWindowTitle('Prueba de bpm')
#El eje x se encuentra por defecto en el lado derecho de la pantalla.
#El eje y, en el izquierdo.
ancho = 800
alto = 600
view.resize(ancho, alto)

#DEFINICION DE LAS GRAFICAS
p1 = win.addPlot(col=1)
#win.nextRow()
p2 = win.addPlot(row = 2, col=1)

curve1 = p1.plot(pen = pg.mkPen(1, width=2)) #El primer numero entre parentesis es el color, el segundo es el ancho
p1.setRange(yRange=[50,100])
p1.showGrid(x=True, y=True)

curve2 = p2.plot(pen = pg.mkPen(5, width=2))
#Activacion de la rejilla en ambos ejes
p2.setRange(yRange=[50,100])
p2.showGrid(x=True, y=True)

win.layout.setColumnMinimumWidth(1, 430)

def principal():
        global BT_ENABLE, GRF_ENABLE, rx_len, c_1, media, suma, bpm, ptr, bpmMax, bpmMin, bpmMMax, bpmMMin

        BT_ENABLE = 'ON'
        GRF_ENABLE = 'ON'

        time.sleep(2.5)

        while BT_ENABLE == 'ON':

                c_1.append(bpm)
                rx_len += 1
                suma += bpm
                m = suma/rx_len
                media.append(m)

                print c_1.get()
                ptr+=2
                c_1[:-1] = c_1[1:]
                media[:-1] = media[1:]
                bpm1 = c_1[-1]
                bpm2 = c_1[-2]
                bpmMed = media[-1]
                bpmMed1 = media[-2]

                #Se calcula el maximo
                if bpm > bpm1:
                        if bpm > bpmMax:
```

Imagen 78. Código con hilos, librería threading.

```
                        if bpm > bpmMax:
                                bpmMax = bpm
                elif bpm < bpm1:
                        if bpm1 > bpmMax:
                                bpm1Max = bpm1
                elif bpm == bpm1:
                        if bpm > bpmMax:
                                bpmMax = bpm

                if bpmMed > bpmMed1:
                        if bpmMed > bpmMMax:
                                bpmMMax = bpmMed
                elif bpmMed < bpmMed1:
                        if bpm1 > bpmMMax:
                                bpm1Max = bpm1
                elif bpmMed == bpmMed1:
                        if bpmMed > bpmMMax:
                                bpmMMax = bpmMed

                #Se calcula el minimo
                if bpm < bpm1:
                        if bpm < bpmMin:
                                bpmMin = bpm
                elif bpm > bpm1:
                        if bpm1 < bpmMin:
                                bpmMin = bpm1
                elif bpm == bpm1:
                        if bpm < bpmMin:
                                bpmMin = bpm

                if bpmMed < bpmMed1:
                        if bpmMed < bpmMMin:
                                bpmMMin = bpmMed
                elif bpmMed > bpmMed1:
                        if bpmMed1 < bpmMMin:
                                bpmMin = bpmMed1
                elif bpmMed == bpmMed1:
                        if bpmMed < bpmMMin:
                                bpmMMin = bpmMed

                if BT_ENABLE == 'OFF':
                        sys.exit()

                time.sleep(1.2)         #Pongo un valor de 1.1 segundos para que la primera matriz que aparezca no s

def BLE():
        global BT_ENABLE, DATA_READY, GRF_ENABLE

        #time.sleep(1)
        if BT_ENABLE == 'ON':
                DATA_READY = 'ON'
                #time.sleep(1)
                if __name__=="__main__":
                        cccid = btle.AssignedNumbers.client_characteristic_configuration
                        hrmid = btle.AssignedNumbers.heart_rate
                        hrmmid = btle.AssignedNumbers.heart_rate_measurement

                        ahora = datetime.now()
```

Imagen 79. Código con hilos, librería threading.

```
nombre = ''
nombre = ahora

f = open("/home/usuario/Escritorio/%s.txt" % (nombre), "a") #"w" Si se quiere sobrees
                                                            #"a" Si no se quiere sobreesc
Config = ConfigParser.ConfigParser()
cfgfile = open("/home/usuario/Escritorio/config.dat", "w")
Config.add_section('Configuracion')
Config.set('Configuracion', 'Direccion', '00:22:D0:DF:74:0A')
Config.set('Configuracion', 'Numero maximo de muestras', '180')
Config.set('Configuracion', 'HR maximo', '100')
Config.set('Configuracion', 'HR minimo', '50')
Config.set('Configuracion', 'Color grafica 1', '1')
Config.set('Configuracion', 'Color grafica 2', '5')
Config.set('Configuracion', 'Resolucion en X', '800')
Config.set('Configuracion', 'Resolucion en Y', '600')
Config.write(cfgfile)
cfgfile.close()
hrm = None
bpm = 0
b = 0

try:
        hrm = HRM('00:22:D0:DF:74:0A')

        service, = [s for s in hrm.getServices() if s.uuid==hrmid]
        ccc, = service.getCharacteristics(forUUID=str(hrmmid))

        if 0: #No funciona
                ccc.write('\1\0')

        else:
                desc = hrm.getDescriptors(service.hndStart,
                service.hndEnd)
                d, = [d for d in desc if d.uuid==cccid]
                hrm.writeCharacteristic(d.handle, '\1\0')

                t0=time.time()
                z = ''
                def print_hr(cHandle, data):
                        global bpm, ptr, z, rx_len, m, suma, media, envio
                        bpm = ord(data[1])
                        if bpm == 0:
                                sys.exit()
                        z = bpm
                        y = time.clock()-t0
                        z = str(z)
                        y = str(y)
                        bpm = float(bpm)
                        envio = z + ';\r'

                        if bpm != 0:
                                print bpm,"%.2f"%(time.time()-t0)
                                f.write(z + '\n')

                hrm.delegate.handleNotification = print_hr

                while 1:
```

Imagen 80. Código con hilos, librería threading.

```
                        hrm.waitForNotifications(3.)   #Valor inicial, 3.
                        time.sleep(1)

finally:
        if hrm:
                f.close()
                BT_ENABLE = 'OFF'
                GRF_ENABLE = 'OFF'
                view.close()
                sys.exit()
                print 'disconnect'
                hrm.disconnect()
```

Imagen 81. Código con hilos, librería threading.

Imagen 82. Código con hilos, librería threading.

11.2. HILOS CON QTHREAD

Después de desarrollar el código con hilos a partir de la librearía threading, siguen apareciendo algunos fallos. Sobre todo aparece un 'warning' que indica: `QTimer can only be used with threads started with QThread`. Para eliminarlo es necesario utilizar QThread. Es un módulo incluido en la librería PyQt4.QtCore, el cual proporciona las funciones necesarias para utilizar hilos en códigos de aplicaciones gráficas desarrolladas con pyqtgraph o PyQt4. Entonces, se modifica el programa para que, en vez de trabajar con la librería threading, trabaje con QThread.

Primero se incluye la clase `MainThreading`. Esta recibe como parámetro la ventana de menú para iniciar el programa. En la misma clase se definen tres funciones: `__init__` inicializa la clase, `start_threads` inicializa los hilos y `salir`, que una vez ejecutada hace que el programa termine.

```
class                    MainThreading(QtGui.QMainWindow,
design.Ui_MainWindow):   #MainThreading(QtGui.QWidget)
     def __init__(self, parent = None):
          super(self.__class__, self).__init__()
          self.setupUi(self)

     self.iniciar.clicked.connect(self.start_threads)
          self.detener.clicked.connect(self.salir)

     def start_threads(self):
          self.principal = principal()
          self.BLE = BLE()
```

```
            self.grafica = grafica()
            self.principal.start()
            self.BLE.start()
            self.grafica.start()

      def salir(self):
            sys.exit()
```

Una vez se ha hecho lo anterior, se definen las tres clases de los hilos que, a su vez, contienen tres funciones: `__init__` al igual que anteriormente, aquí también inicializa las clases; `__del__` es un destructor que permite eliminar datos almacenados en un objeto una vez hayan sido eliminadas las referencias al mismo; `run` contiene las instrucciones propias de cada hilo. La sección de código que se muestra a continuación es un ejemplo de la estructura de un hilo:

```
class WorkThread(QtCore.QThread):
      def __init__(self):
            QtCore.QThread.__init__(self)

      def __del__(self):
            self.wait()

      def run(self):
            for i in range(6):
                  time.sleep(0.3) # artificial time delay
                  self.emit(
QtCore.SIGNAL('update(QString)'), "from work thread " +
str(i) )

      self.terminate()
```

Finalmente se incluye la sentencia `if __name__ == "__main__":`. En ella se definen las variables y la ventana donde se muestran las gráficas y se llama a la función `MainThreading()`.

En la siguiente imagen se presenta un esquema con el funcionamiento del programa con los hilos, utilizando QThread.

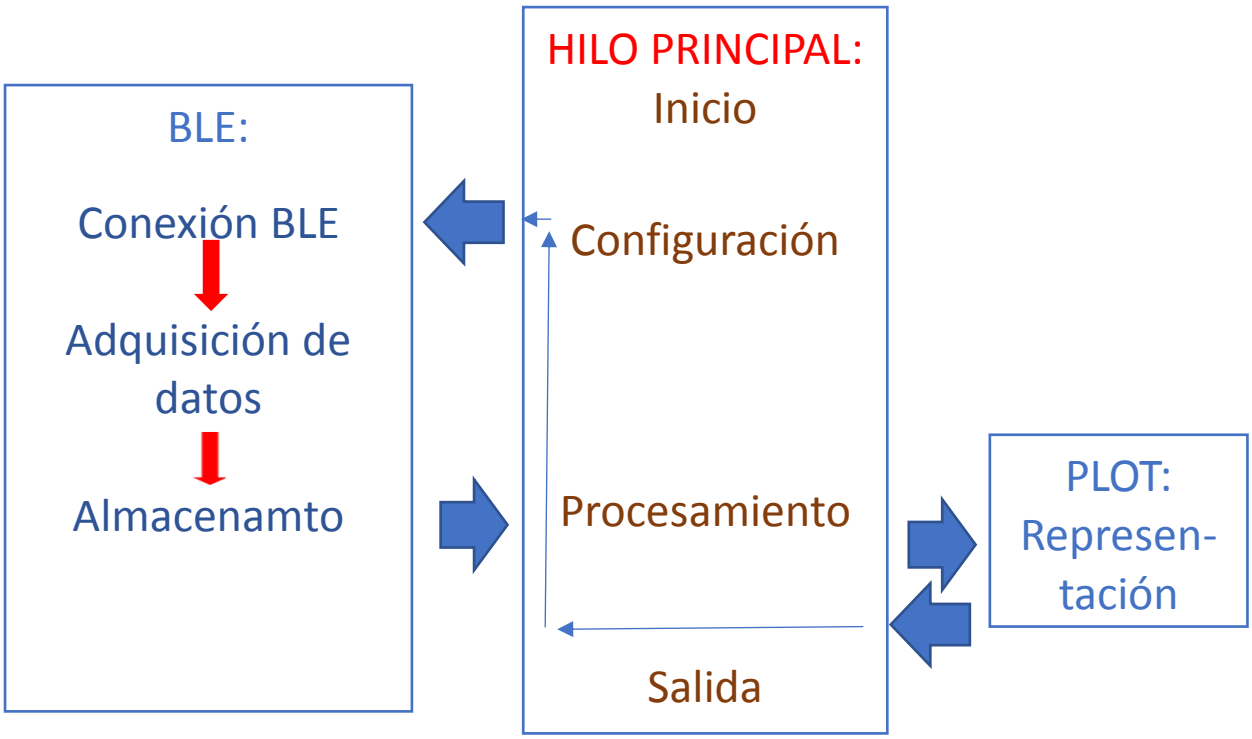

Figura 83. Vista del funcionamiento del programa con hilos

11.2.1. LIBRERÍA DESIGN

Se ha añadido una librería creada para que el programa principal contenga menor cantidad líneas de código y sea más legible. El nombre de la misma es `design`. En ella se define la ventana de menú que se utiliza para dar comienzo a la medición. Se ha desarrollado mediante el programa de diseño Qt4 Designer. La librería contiene el código generado por este programa. Con el fin de ampliar las opciones del menú, se han incluido algunas funcione en el código. En la imagen 77 se muestra la ventana de inicio de Qt4 Designer. En ella aparecen las posibles formas de la aplicación a desarrollar. La de mayor interés en este caso es la cuarta, Main Window. En la imagen 78 aparece el menú que se ha creado. Se pueden observar la ventana de trabajo y la vista previa. Se han utilizado dos botones (‘Push Button’), un layout horizontal (‘Horizontal Layout’) y una barra de herramientas. Basta con arrastrar los botones a la ventana principal y enmarcarlos en el ‘layout’. Luego, escribiendo en la casilla ‘Type Here’, se han añadido las ventanas de la barra de herramientas. Y seleccionando cada una de ellas, han incluido más opciones.

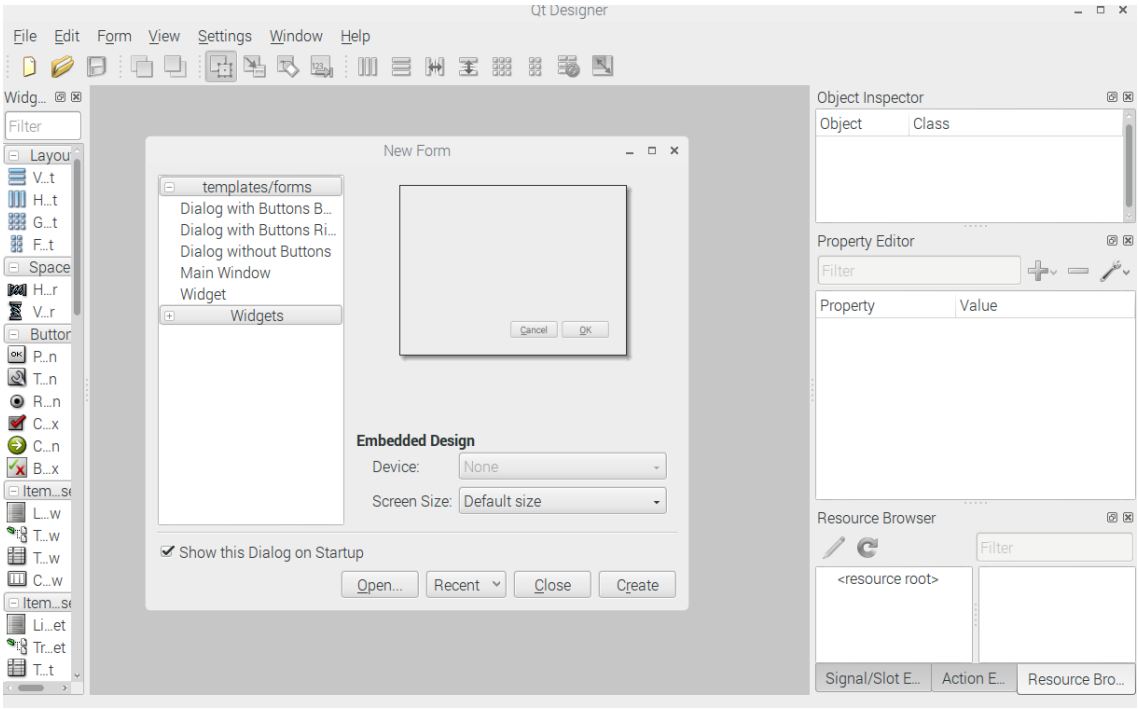

Imagen 84. Vista inicial de Qt4 Designer.

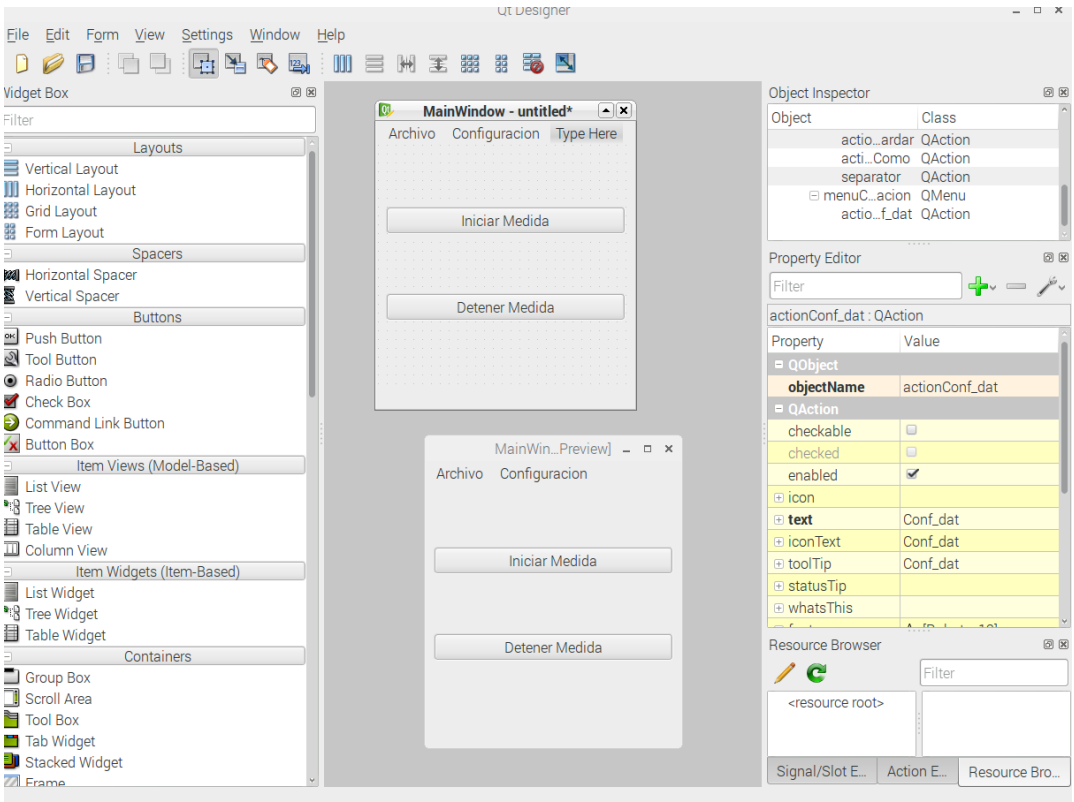

Imagen 85. Creación de la ventana de menú.

En la siguiente imagen se muestra el menú que se ha creado. Al ejecutar el programa, aparecen la ventana donde se muestran las gráficas y este menú. Consta de dos botones y una barra de herramientas. El botón 'Iniciar Medida' permite comenzar a tomar medidas con el pulsómetro y a dibujar las gráficas. Por el contrario, 'Detener Medida' termina el programa y sale. En la barra de herramientas aparecen dos casillas.

La primera, ‘Archivo’, despliega una barra más, con tres opciones: Guardar, Guardar como y Salir. La única que realiza una función es la última que, como indica su nombre, hace que se salga del programa. La segunda ventana, ‘Configuracion’, despliega una ventana con la opción ‘Conf_dat’. Si se selecciona esta última, se imprime por pantalla el contenido del archivo del mismo nombre. Esto se muestra en la imagen 80.

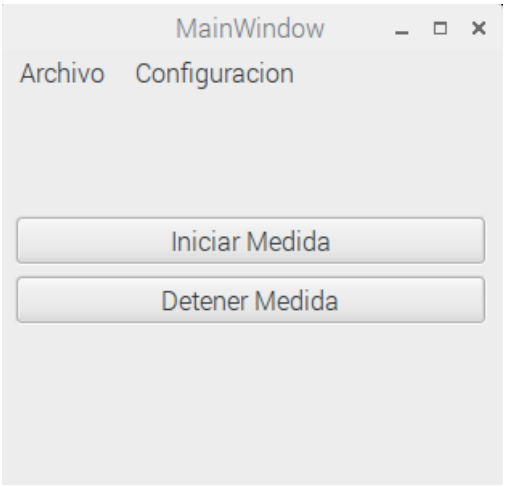

Imagen 86. Menú creado con Qt4 Designer

Imagen 87. Resultado de seleccionar Conf_dat (Configuración)

12. ANEXOS

12.1. LIBRERÍA DESIGN

```
import sys
from PyQt4 import QtCore, QtGui
import pyqtgraph as pg

try:
    _fromUtf8 = QtCore.QString.fromUtf8
except AttributeError:
    def _fromUtf8(s):
        return s

try:
    _encoding = QtGui.QApplication.UnicodeUTF8
    def _translate(context, text, disambig):
        return      QtGui.QApplication.translate(context,
text, disambig, _encoding)
except AttributeError:
    def _translate(context, text, disambig):
        return      QtGui.QApplication.translate(context,
text, disambig)

class Ui_MainWindow(object):
    def Salir(self):
      sys.exit()

    def setupUi(self, MainWindow):
      MainWindow.setObjectName(_fromUtf8("MainWindow"))
      MainWindow.resize(201, 156)
      self.centralwidget = QtGui.QWidget(MainWindow)

self.centralwidget.setObjectName(_fromUtf8("centralwidge
t"))
      self.verticalLayout_2                              =
QtGui.QVBoxLayout(self.centralwidget)

self.verticalLayout_2.setObjectName(_fromUtf8("verticalL
ayout_2"))
      self.verticalLayout = QtGui.QVBoxLayout()

self.verticalLayout.setObjectName(_fromUtf8("verticalLay
out"))
      self.iniciar                                       =
QtGui.QPushButton(self.centralwidget)

self.iniciar.setObjectName(_fromUtf8("Iniciar_Medida"))
      self.verticalLayout.addWidget(self.iniciar)
      self.detener                                       =
QtGui.QPushButton(self.centralwidget)
```

```
self.detener.setObjectName(_fromUtf8("Detener_Medida"))
      self.verticalLayout.addWidget(self.detener)
self.verticalLayout_2.addLayout(self.verticalLayout)
      MainWindow.setCentralWidget(self.centralwidget)
      self.menubar = QtGui.QMenuBar(MainWindow)
      self.menubar.setGeometry(QtCore.QRect(0,  0,  201,
25))
      self.menubar.setObjectName(_fromUtf8("menubar"))
      self.menuArchivo    =    QtGui.QMenu(self.menubar)
self.menuArchivo.setObjectName(_fromUtf8("menuArchivo"))
      self.menuConfiguracion = QtGui.QMenu(self.menubar)
self.menuConfiguracion.setObjectName(_fromUtf8("menuConf
iguracion"))
      MainWindow.setMenuBar(self.menubar)
      self.statusbar    =    QtGui.QStatusBar(MainWindow)
self.statusbar.setObjectName(_fromUtf8("statusbar"))
      MainWindow.setStatusBar(self.statusbar)
      self.actionGuardar = QtGui.QAction(MainWindow)

self.actionGuardar.setObjectName(_fromUtf8("actionGuarda
r"))
      self.actionGuardar_Como = QtGui.QAction(MainWindow)

self.actionGuardar_Como.setObjectName(_fromUtf8("actionG
uardar_Como"))
      self.actionSalir = QtGui.QAction(MainWindow)

self.actionSalir.setObjectName(_fromUtf8("actionSalir"))
      self.actionConf_dat = QtGui.QAction(MainWindow)

self.actionConf_dat.setObjectName(_fromUtf8("actionConf_
dat"))
      self.menuArchivo.addAction(self.actionGuardar)

self.menuArchivo.addAction(self.actionGuardar_Como)
      self.menuArchivo.addSeparator()
      self.menuArchivo.addAction(self.actionSalir)

self.menuConfiguracion.addAction(self.actionConf_dat)
self.menubar.addAction(self.menuArchivo.menuAction())
self.menubar.addAction(self.menuConfiguracion.menuAction
())
      self.retranslateUi(MainWindow)
      QtCore.QMetaObject.connectSlotsByName(MainWindow)
      QtCore.QObject.connect(self.actionConf_dat,
QtCore.SIGNAL(_fromUtf8("activated()")), self.PrintConf)
      QtCore.QObject.connect(self.actionSalir,
QtCore.SIGNAL(_fromUtf8("activated()")), self.Salir)

    def PrintConf(self):
```

```
      conf = open("/home/pi/Desktop/TFG/config.dat", "r")
      for i in conf:
            print i

    def inic(self):
      global view
      h1.start()
      h2.start()
      h3.start()
      view.show()
    def det(self):
      h1.close()
      h2.close()
      h3.close()

    def retranslateUi(self, MainWindow):
      MainWindow.setWindowTitle(_translate("MainWindow",
"MainWindow", None))
      self.iniciar.setText(_translate("MainWindow",
"Iniciar Medida", None))
      self.iniciar.clicked.connect(self.inic)
      self.detener.setText(_translate("MainWindow",
"Detener Medida", None))
      self.detener.clicked.connect(self.det)
      self.menuArchivo.setTitle(_translate("MainWindow",
"Archivo", None))

self.menuConfiguracion.setTitle(_translate("MainWindow",
"Configuracion", None))

self.actionGuardar.setText(_translate("MainWindow",
"Guardar", None))

self.actionGuardar_Como.setText(_translate("MainWindow",
"Guardar Como", None))
      self.actionSalir.setText(_translate("MainWindow",
"Salir", None))

self.actionConf_dat.setText(_translate("MainWindow",
"conf_dat", None))
```

12.2. PROGRAMA PRINCIPAL

```
from bluepy import btle
import time
import pyqtgraph as pg
import sys
from PyQt4 import QtGui, QtCore
import numpy as np
from datetime import datetime, date, timedelta
```

```
import calendar
from PyQt4.QtCore import QThread, SIGNAL
import ConfigParser
import design

class RingBuffer:
      def __init__(self, size):
            self.data = [0 for i in xrange(size)]

      def append(self, x):
            self.data.pop(0)
            self.data.append(x)

      def get(self):
            return self.data

      def __getitem__(self, index):
            return self.data[index]

      def __len__(self):
            l = len(self.data)
            return l

      def __setitem__(self, index, value):
            self.data[index] = value

class HRM(btle.Peripheral):
      def __init__(self, addr):
            btle.Peripheral.__init__(self,         addr,
addrType="public")

class                    MainThreading(QtGui.QMainWindow,
design.Ui_MainWindow):    #MainThreading(QtGui.QWidget)
      def __init__(self, parent = None):
            super(self.__class__, self).__init__()
            self.setupUi(self)

      self.iniciar.clicked.connect(self.start_threads)
            self.detener.clicked.connect(self.salir)

      def start_threads(self):
            self.principal = principal()
            self.BLE = BLE()
            self.grafica = grafica()
            self.principal.start()
            self.BLE.start()
            self.grafica.start()

      def salir(self):
            sys.exit()
```

```
class principal(QThread):
	def __init__(self):
		QThread.__init__(self)

	def __del__(self):
		self.wait()

	def run(self):
		global BT_ENABLE, GRF_ENABLE, rx_len, c_1,
media, suma, ptr, bpmMax, bpmMin, bpmMMax, bpmMMin, bpm
		BT_ENABLE = 'ON'
		GRF_ENABLE = 'ON'
		time.sleep(0.5)
		while BT_ENABLE == 'ON':
			if bpm != 0:
				c_1.append(bpm)
				rx_len += 1
				suma += bpm
				m = suma/rx_len
				media.append(m)

			ptr+=2
			c_1[:-1] = c_1[1:]
			media[:-1] = media[1:]
			bpm1 = c_1[-1]
			bpm2 = c_1[-2]
			bpmMed = media[-1]
			bpmMed1 = media[-2]

			#Calculo de maximos
			if bpm > bpm1:
				if bpm > bpmMax:
					bpmMax = bpm
			elif bpm < bpm1:
				if bpm1 > bpmMax:
					bpm1Max = bpm1
			elif bpm == bpm1:
				if bpm > bpmMax:
					bpmMax = bpm

			if bpmMed > bpmMed1:
				if bpmMed > bpmMMax:
					bpmMMax = bpmMed
			elif bpmMed < bpmMed1:
				if bpm1 > bpmMMax:
					bpm1Max = bpm1
			elif bpmMed == bpmMed1:
				if bpmMed > bpmMMax:
					bpmMMax = bpmMed
```

```
                    #Calculo de minimos
                    if bpm < bpm1:
                          if bpm < bpmMin:
                                bpmMin = bpm
                    elif bpm > bpm1:
                          if bpm1 < bpmMin:
                                bpmMin = bpm1
                    elif bpm == bpm1:
                          if bpm < bpmMin:
                                bpmMin = bpm

                    if bpmMed < bpmMed1:
                          if bpmMed < bpmMMin:
                                bpmMMin = bpmMed
                    elif bpmMed > bpmMed1:
                          if bpmMed1 < bpmMMin:
                                bpmMin = bpmMed1
                    elif bpmMed == bpmMed1:
                          if bpmMed < bpmMMin:
                                bpmMMin = bpmMed

                    if BT_ENABLE == 'OFF':
                          sys.exit()

                         time.sleep(0.5)

            self.terminate()

class BLE(QThread):
      def __init__(self):
            QThread.__init__(self)

      def __del__(self):
            self.wait()

      def run(self):
            global  BT_ENABLE,  DATA_READY,  GRF_ENABLE,
view, bpm
               time.sleep(0.6)
            if BT_ENABLE == 'ON':
                  DATA_READY = 'ON'
                  cccid                                    =
btle.AssignedNumbers.client_characteristic_configuration
                  hrmid                                    =
btle.AssignedNumbers.heart_rate
                  hrmmid                                   =
btle.AssignedNumbers.heart_rate_measurement

                  ahora = datetime.now()
                  nombre = ''
                  nombre = ahora
```

```
                    f = open("/home/pi/Desktop/TFG/%s.txt"
% (nombre), "a")
                    hrm = None

                    try:
                          hrm = HRM(Direccion)
                          service,   =   [s   for   s   in
hrm.getServices() if s.uuid==hrmid]
                          ccc,                          =
service.getCharacteristics(forUUID=str(hrmmid))


                          desc                          =
hrm.getDescriptors(service.hndStart,
                                service.hndEnd)
                                d, = [d for d in desc if
d.uuid==cccid]

hrm.writeCharacteristic(d.handle, '\1\0')

                                def     print_hr(cHandle,
data):
                                      global bpm, ptr,
z, rx_len, m, suma, media, envio
                                      bpm             =
ord(data[1])
                                      if bpm == 0:

sys.exit()
                                      bpm = float(bpm)
                                      z = bpm
                                      z = str(z)
                                      f.write(z + '\n')

hrm.delegate.handleNotification = print_hr

                                    while 1:
                                hrm.waitForNotifications(3.)

                    finally:
                          if hrm:
                                f.close()
                                BT_ENABLE = 'OFF'
                                GRF_ENABLE = 'OFF'
                                view.close()
                                sys.exit()
                                hrm.disconnect()
            elif BT_ENABLE == 'OFF':
                  time.sleep(0.5)
            self.terminate()
```

```
class grafica(QThread):
      def __init__(self):
            QThread.__init__(self)

      def __del__(self):
            self.wait()

      def run(self):
            global GRF_ENABLE, DATA_READY, rx_len, ptr,
bpmMax, bpmMin, bpmMMax, bpmMMin, bpm, bpmMed, media, c_1

            time.sleep(0.7)
            if GRF_ENABLE == 'ON' and DATA_READY == 'ON':

                  vb1 = win.addViewBox(col = 2)

                  text1     =     pg.TextItem(html='<div
style="text-align:      center"><span      style="color:
#FFF;"></span><br><span  style="color:  #FF0;  font-size:
32pt;">%0.1f  bpm</span><br><span  style="color:  #FF0;
font-size:    12pt;">%0.1f    Max    bpm</span><br><span
style="color:   #FF0;   font-size:   12pt;">%0.1f   Min
bpm</span></div>'  %  (bpm,  bpmMax,  bpmMin),  anchor=(-
0.1,0.3), border='w', fill=(0, 0, 255, 100))
                  vb1.setXRange(20, 40)
                  vb1.setYRange(0, 40)
                  vb1.autoRange()
                  vb1.addItem(text1)

                  vb2 = win.addViewBox(row = 2, col = 2)

                  text2     =     pg.TextItem(html='<div
style="text-align:      center"><span      style="color:
#FFF;"></span><br><span  style="color:  #FF0;  font-size:
32pt;">%0.1f  Media</span><br><span  style="color:  #FF0;
font-size:    12pt;">%0.1f    Med    Max</span><br><span
style="color:   #FF0;   font-size:   12pt;">%0.1f   Med
Min</span></div>' % (bpmMMax, bpmMed, bpmMMin), anchor=(-
0.1, 0.3), border='w', fill=(0, 0, 255, 100))
                  vb2.setXRange(20, 40)
                  vb2.setYRange(0, 40)
                  vb2.autoRange()
                  vb2.addItem(text2)

                  while 1:
                        time.sleep(0.5)
                        bpmMed = media[-1]
                        curve1.setData(c_1.get())
                        curve1.setPos(ptr, 0)
                        curve2.setData(media.get())
```

```
                        curve2.setPos(ptr, 0)

                        text1.setHtml('<div
style="text-align:      center"><span      style="color:
#FFF;"></span><br><span  style="color:  #FF0;  font-size:
32pt;">%0.1f   bpm</span><br><span   style="color:   #FF0;
font-size:    12pt;">%0.1f    Max    bpm</span><br><span
style="color:    #FF0;    font-size:    12pt;">%0.1f    Min
bpm</span></div>' % (bpm, bpmMax, bpmMin))

                        text2.setHtml('<div
style="text-align:      center"><span      style="color:
#FFF;"></span><br><span  style="color:  #FF0;  font-size:
32pt;">%0.1f  Media</span><br><span  style="color:  #FF0;
font-size:    12pt;">%0.1f    Med    Max</span><br><span
style="color:    #FF0;    font-size:    12pt;">%0.1f    Med
Min</span></div>' % (bpmMed, bpmMMax, bpmMMin))
                        vb1.addItem(text1)
                        vb2.addItem(text2)

                        if   GRF_ENABLE   ==   'OFF'   or
DATA_READY == 'OFF':
                              sys.exit()


            self.terminate()

if __name__ == "__main__":

      app = QtGui.QApplication(sys.argv)

      c = open("/home/pi/Desktop/TFG/con.dat", "r")
      for i in c:
            a = i.split(';')
            Direccion = a[0]
            Muestras_bpm = int(a[1])
            Muestras_media = int(a[2])
            HRMax = int(a[3])
            HRmin = int(a[4])
            Color_g1 = int(a[5])
            Color_g2 = int(a[6])
            EjeX = int(a[7])
            EjeY = int(a[8])

      ancho = int(EjeX)
      alto = int(EjeY)

      c_1 = RingBuffer(Muestras_bpm)
      media = RingBuffer(Muestras_media)
      ptr = 0
      rx_len = 0
```

```
        suma = 0
        m = 0
        b1 = 0
        bpm = 0.0
        bpmMax = 0.0
        bpmMin = 1000.0
        bpmMMax = 0.0
        bpmMMin = 1000.0
        bpmMed = 0
        envio = ''
        b = 0

        BT_ENABLE = 'OFF'
        GRF_ENABLE = 'OFF'
        DATA_READY = 'OFF'

        #CREACION DE LA VENTANA DE LAS GRAFICAS
        win = pg.GraphicsLayout(border=(100,100,100))
        view = pg.GraphicsView()

        view.setCentralItem(win)
        view.setWindowTitle('Prueba de bpm')
        view.resize(ancho, alto)
        view.show()

        #DEFINICION DE LAS GRAFICAS
        p1 = win.addPlot(col=1)
        p2 = win.addPlot(row = 2, col=1)

        curve1 = p1.plot(pen = pg.mkPen(Color_g1, width=2))
        p1.setRange(yRange=[HRmin,HRMax])
        p1.showGrid(x=True, y=True)

        curve2 = p2.plot(pen = pg.mkPen(Color_g2, width=2))
        p2.setRange(yRange=[HRmin,HRMax])
        p2.showGrid(x=True, y=True)

        win.layout.setColumnMaximumWidth(2, 250)

        MW = MainThreading()
        MW.show()
        app.exec_()
```

12.3. SOLUCIÓN FINAL

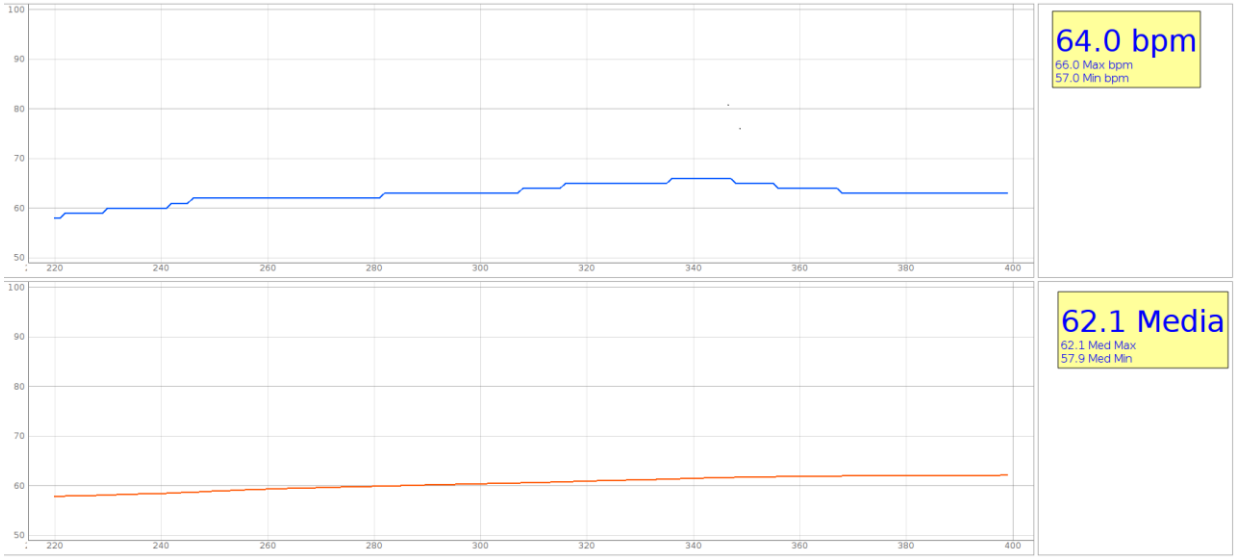

Figura 88. Resultado final 1.

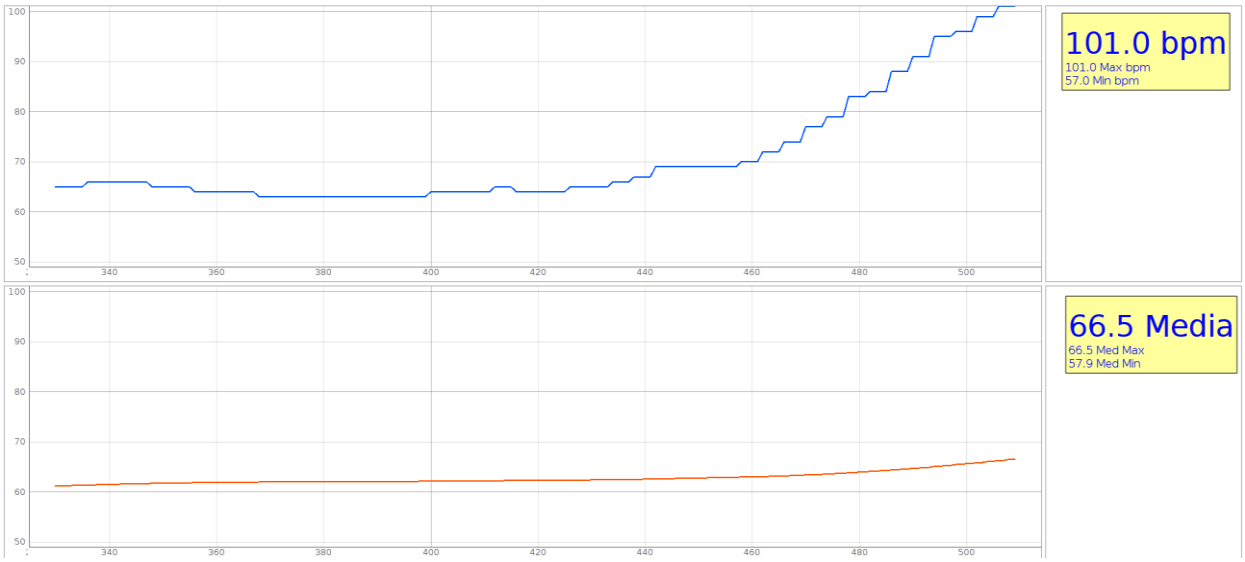

Figura 89. Resultado final 2.

13. BIBLIOGRAFÍA

- https://es.wikipedia.org/wiki/Raspbian

- Raspberry Pi 2, utilice todo el potencial de su nano-ordenador. Autor: François MOCQ. Editorial: eni ediciones. ISBN: 978-2-7460-9998-2.

- http://hacedores.com/arduino-o-raspberry-pi-cual-es-la-mejor-herramienta-para-ti/

- http://www.leantec.es/blog/22_Diferencias-entre-Arduino-y-Raspberry-Pi.html

- https://es.wikipedia.org/wiki/Python

- http://www.maestrosdelweb.com/ventajas-python/

- Sistemas SCADA, 2ª Edición. Autor: Aqulino Rodríguez Penin. Editorial: Marcombo, ediciones técnicas. Afaomega Grupo Editor. ISBN (MARCOMBO): 978-84-267-1450-3. ISBN (ALFAOMEGA GRUPO EDITOR): 978-970-15-1305-7

- http://imasdemase.com/programacion-2/descubriendo-socat/

- https://thehackerway.com/2011/08/08/herramientas-para-hacking-en-entornos-de-red-%E2%80%93-conectividad-con-socat-parte-iii/

- http://www.armadeus.org/wiki/index.php?title=GTKTerm

- https://apps.ubuntu.com/cat/applications/oneiric/gtkterm/

- https://es.wikipedia.org/wiki/Minicom

- http://pyserialuvg.wikispaces.com/M%C3%B3dulo+pySerial

- http://pyserial.sourceforge.net/

-http://www.instructables.com/id/Control-Bluetooth-LE-Devices-From-A-Raspberry-Pi/

- http://pyserial.readthedocs.io/en/latest/shortintro.html

- https://media.readthedocs.org/pdf/pyserial/latest/pyserial.pdf

- https://matplotlib.org/

-https://www.datacamp.com/community/tutorials/matplotlib-tutorial-python#gs.1SFb2Us

- http://www.numpy.org/

- https://en.wikipedia.org/wiki/NumPy

-https://askubuntu.com/questions/763877/how-to-install-and-run-qt-designer-for-python

- https://www.pccomponentes.com/unotec-adaptador-bluetooth-4-0-usb-para-pc

- http://www.bluez.org/about/

- https://github.com/IanHarvey/bluepy

- https://github.com/IanHarvey/bluepy/issues/53

- https://www.programiz.com/python-programming/methods/built-in/open

- http://www.global-tag.com/es/tecnologia-ble/

- https://es.wikipedia.org/wiki/Bluetooth

- https://es.wikipedia.org/wiki/Bluetooth_de_baja_energ%C3%ADa

- https://stackoverflow.com/questions/16879971/example-of-the-right-way-to-use-qthread-in-pyqt

- https://support.polar.com/es/ayuda/H7_HR_Sensor

14. FIGURAS

FIGURA	DESCRPCIÓN	FUENTE
1	Vista general del Hardware del sistema	Propia
2	Vista general del Software del Sistema	Propia
3	Raspberry Pi 2 modelo B	http://www.etechpk.net/wp-content/uploads/2016/02/RPI2A.png
4	SoC (System on a Chip) BCM 2836	http://www.zive.cz/GetThumbNail.aspx?id_file=119582484&width=5000&height=120&q=100
5	Disposición de los pines en la GPIO de la Raspberry Pi	http://elektronik.skyline-service.de/wp-content/uploads/2016/11/RaspiPins.png
6	Distribución de los pines de la GPIO en la Raspberry Pi	https://i.stack.imgur.com/nM5GM.png
7	Versión de Raspbian instalada en la Raspberry Pi	https://www.raspberrypi.org/downloads/raspbian/ (Consulta: febrero de 2017)
8	Vista del menú de Raspi-config	Propia
9	Vista del menú de Raspi-config	Propia
10	Vista del menú Raspi-config. Cambio de zona horaria.	Propia
11	Vista del menú Raspi-config. Cambio de zona horaria.	Propia
12	Vista del menú Raspi-config. Cambio de zona horaria.	Propia
13	Vista del menú Raspi-config. Selección de inicio en modo gráfico.	Propia
14	Vista del menú Raspi-config. Selección de inicio en modo gráfico.	Propia
15	Configuración del ratón y del teclado. Selección del idioma,	Propia
16	Menú principal de Raspbian.	Propia
17	Sensor Polar H7	https://d243u7pon29hni.cloudfront.net/images/products/banda_toracica_polar_H7_HR_Negro_ppal_l.jpg
18	Colocación del sensor Polar H7	https://support.polar.com/e_manuals/H7_Heart_Rate_Sensor/Polar_H7_Hear

		t_Rate_Sensor_Getting_Started_Guide_Espanol.pdf
19	Instrucciones de uso del sensor Polar H7	https://support.polar.com/e_manuals/H7_Heart_Rate_Sensor/Polar_H7_Heart_Rate_Sensor_Getting_Started_Guide_Espanol.pdf
20	Manú principal de Raspbian. Entornos de programación disponibles.	Propia
21	Ejemplo de código en Python: 'Hola mundo'.	Propia
22	Ejemplo de código en Python: Comparación de listas.	Propia
23	Configuración del primer puerto en GTKTerm y puertos serie enlazados.	Propia
24	Configuración del segundo puerto en GTKTerm y puertos serie enlazados.	Propia
25	Ventana de envío de datos de GTKTerm.	Propia
26	Ventana de recepción de datos de GTKTerm.	Propia
27	Ventana de envío de datos de GTKTerm.	Propia
28	Ventana de recepción de datos de minicom.	Propia
29	Ejemplo de comunicación entre GTKTerm y Minicom.	Propia
30	Primer menú de envío de datos de minicom.	Propia
31	Segundo menú de envío de datos de minicom.	Propia
32	Introducción de la dirección del archivo a enviar desde minicom.	Propia
33	Lista de números enviada.	Propia
34	Página web de descarga de pyserial.	https://pypi.python.org/pypi/pyserial/2.7 (Consulta: marzo de 2017)
35	Enlace de descarga de pyserial.	https://pypi.python.org/pypi/pyserial/2.7 (Consulta: marzo de 2017)
36	Ejemplo de código Python: generación de números aleatorios.	Propia

37	Código y resultado de programa en Pythob: Generación de números aleatorios.	Propia
38	Resultado de Programa en GTKTerm y puertos serie enlazados.	Propia
39	Ejemplo de código en Python: Uso de pyserial.	Propia
40	Código en Python: Prueba de comunicación entre GTKTerm y Python.	Propia
41	Resultado de la prueba de comunicación entre GTKTerm y Python.	Propia
42	Ejemplo de gráfica en Matplotlib.	https://pythonprogramming.net/static/images/matplotlib/matplotlib-fivethirtyeight-style-example.png (Consulta: abril 2017)
43	Ejemplo 1 de programa con Matplotlib.	https://www.datacamp.com/community/tutorials/matplotlib-tutorial-python#gs.1SFb2Us (Consulta: abril 2017)
44	Ejemplo 2 de programa con Matplotlib.	Propia
45	Resultado del ejemplo 2 de programa con Matplotlib.	Propia
46	Ejemplo 3 de código con matplotlib.	Propia
47	Resultado del ejemplo 3 de código con matplotlib.	Propia
48	Ejemplo de TKinter: Parte del código implementado para el juego de tres en raya.	https://sites.google.com/site/codingleb rija/ /rsrc/1434659815403/curso-2014-15/python/captura3raya.png
49	Resultado de un ejemplo de Chaco.	http://sdaaubckp.sourceforge.net/tute/wfview/python-chaco.png
50	Código y resultado de un ejemplo de StreamPlot.	Propia
51	Ejemplo de código con TKinter: Creación de una ventana con botón.	Propia
52	Ejemplo 1 de código con PyQt4: Creación de una ventana.	Propia

53	Ejemplo 2 de código con PyQt4: Representación gráfica de los 23 valores de una tabla.	Propia
54	Ejemplo 3 de PyQt4. Variación del código de la figura anterior (48).	Propia
55	Gráfica correspondiente al ejemplo 4 de PyQt4. Representación de una lista de 1.000 valores.	Propia
56	Vista inicial de Qt4 Designer	Propia
57	Dispositivo Unotec Adaptador Bluetoth 4.0 USB.	https://img.pccomponentes.com/articles/7/73794/adaptador-bluetooth-4-0-usb.jpg
58	Uso de los comandos `sudo hciconfig hci0 down`, `sudo hciconfig hci0 up`, `sudo hcitool dev` y `sudo hciconfig lescan.`	Propia
59	Uso del comando `sudo hcitool lescan`	Propia
60	Uso de los comandos `sudo hciconfig hci0 up`, `sudo rfkill unblock all`, `sudo hciconfig hci0 up` y `sudo hcitool scan.`	Propia
61	Resultado de transmisión de datos por Bluetooth.	Propia
62	Representación gráfica de la medida de frecuencia cardiaca.	Propia
63	Representación gráfica de la medida de frecuencia cardiaca y de la media.	Propia
64	Representación gráfica de la medida de frecuencia cardiaca y de la media.	Propia
65	Representación gráfica de la medida de frecuencia cardiaca y de la media.	Propia
66	Representación gráfica y numérica de la medida de frecuencia cardiaca y de la media.	Propia
67	Representación gráfica y numérica de la medida de frecuencia cardiaca y de la media.	Propia

68	Representación gráfica y numérica de la medida de frecuencia cardiaca y de la media.	Propia
69	Representación gráfica y numérica de la medida de frecuencia cardiaca y de la media.	Propia
70	Representación gráfica y numérica de la medida de frecuencia cardiaca y de la media.	Propia
71	Representación gráfica y numérica de la medida de frecuencia cardiaca y de la media. Tamaño de columna 640. Tamaño de ventana 800x600.	Propia
72	Representación gráfica y numérica de la medida de frecuencia cardiaca y de la media. Tamaño de columna 640. Ventana maximizada.	Propia
73	Representación gráfica y numérica de la medida de frecuencia cardiaca y de la media. Tamaño de columna 1640. Tamaño de ventana 800x600.	Propia
74	Representación gráfica y numérica de la medida de frecuencia cardiaca y de la media. Tamaño de columna 1640. Ventana maximizada.	Propia
75	Fichero de texto donde se guarda la configuración del programa	Propia
76	Fichero config.dat.	Propia
77 – 82	Código usando hilos con la librería threading.	Propia
83	Vista general del funcionamiento del programa con hilos	Propia
84	Vista inicial de Qt4 Designer	Propia
85	Creación de la ventana de menú.	Propia
86	Menú creado con Qt4 Designer	Propia
87	Resultado de seleccionar Conf_dat (Configuración)	Propia
88	Resultado final	Propia
89	Resultado final	Propia