UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

DESARROLLO E IMPLEMENTACIÓN DE UN SISTEMA IOT PARA LA MONITORIZACIÓN DE MAGNITUDES ELÉCTRICAS MEDIANTE COMUNICACIÓN MODBUS SOBRE RS485.

TRABAJO FINAL DEL

Grado en Ingeniería Electrónica Industrial y Automática

REALIZADO POR

Rogelio Adrian Rodríguez Rudametkin

TUTORIZADO POR

Salvador Orts Grau

CURSO ACADÉMICO: 2020/2021

1 Resumen

El presente TFG se plantea el desarrollo e implementación de una aplicación IoT para la monitorización de los parámetros eléctricos de una instalación eléctrica. El alcance del trabajo será el siguiente:

- Estudio de mercado y selección de un medidor de potencia monofásico económico y con posibilidad de comunicación vía Modbus RS485.
- Estudio de mercado y selección de un microcontrolador económico y de pequeño tamaño que ofrezca las máximas posibilidades de conexión. Comunicación serie, WiFi y Bluetooth.
- Desarrollo del software para la aplicación, que comprenderá:
    - Implementación de la comunicación ModBus con el medidor para la lectura de datos y configuración del mismo.
    - Implementación de una GUI para la monitorización remota de los datos leídos del medidor.
- Implementación experimental y testeo del sistema desarrollado.

2 Índice del Trabajo

MEMORIA

Índice de la Memoria

1 Introducción

1.1 Objeto y Objetivos

En el presente trabajo se tiene un medidor de potencia monofásico conectado a una red eléctrica. Este vatímetro es capaz de medir todos los parámetros eléctricos relevantes de la red y transmitirlos mediante comunicación RS-485 siguiendo el protocolo Modbus. El problema que se plantea consiste en encontrar una forma de "modernizar" este sensor, de manera que no se necesite ningún equipo permanente que tenga que estar conectado directa y físicamente a él para poder monitorizar sus parámetros.

Por ello, el objetivo de este trabajo es transformar este vatímetro a un dispositivo IoT. Es decir, conseguir que los datos que acumule el vatímetro sean enviados a través de internet, y recibidos por un programa en un ordenador independiente y externo.

Para conseguir la transformación de este sensor a un dispositivo IoT, el presente trabajo plantea añadir una unidad de procesamiento adicional que actúe como intermediario entre la colección de datos del vatímetro y el envío de estos mismos sobre internet; para luego ser visualizados en otro dispositivo externo al sistema. La comunicación entre el vatímetro y este dispositivo de procesamiento debe efectuarse mediante RS-485 y el protocolo Modbus.

El presente trabajo no pretende desarrollar un sistema integrado que haga esta función, sino demostrar la posibilidad de convertir, principalmente sensores y actuadores ya existentes en instalaciones industriales, a dispositivos IoT de una manera sencilla, eficiente y económica.

También se resolverá, de manera simplificada y como demostración del trabajo, una aplicación para ordenador que se conecte al dispositivo IoT y por donde se puedan visualizar los datos.

1.2 Estado del Arte

El problema que se pretende resolver no es para nada nuevo. Ya hace varias décadas que se hablaba de las comunicaciones máquina-a-máquina (M2M). Pero no fue hasta hace poco, con el auge de internet, que se dio un paso adelante y se actualizó este concepto a IoT; que recientemente ha cogido tracción y relevancia en muchos ámbitos, tanto comerciales, como industriales.

El IoT viene del inglés "Internet of Things" (el internet de las cosas). Esto no es un protocolo ni un estándar, es simplemente la idea de que cualquier objeto, que puede enviar, recibir o procesar datos, sea capaz de conectarse a internet y ser identificados en una red o sistema. En la *figura 1* se ve representado una abstracción de su funcionamiento.

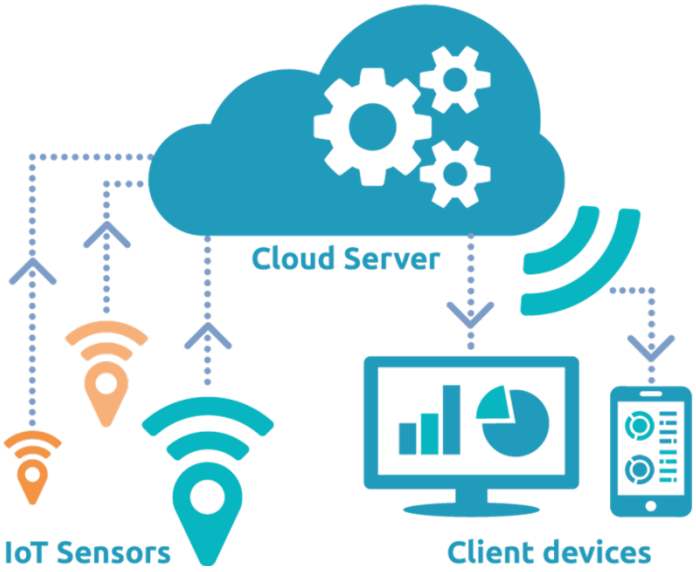


Figura 1: Diagrama abstracto de IoT.

La demanda de este tipo de sistemas interconectados, donde la interacción humana se minimiza, es cada vez mayor. En el mercado industrial ya se pueden encontrar multitud de sensores y actuadores que incluyen la tecnología para ser conectados a internet y funcionar de esta forma.

Lo fundamental de este concepto IoT, es simplificar la integración entre la realidad física y la virtual. De esta manera, la automatización se puede llevar a otro nivel; donde distintos sistemas pueden comunicarse entre ellos, ya que, en principio, un sistema puede tener acceso a los datos de otro distinto.

Para este trabajo, el enfoque está en conseguir que el vatímetro esté conectado a internet y, por tanto, formar parte de esta red de dispositivos IoT.

2 Alternativas y Justificación del Material Elegido

Como se ha explicado en el apartado anterior, el objetivo de este trabajo es "modernizar" un sistema ya existente. Por lo que el sensor elegido debe ser un medidor de potencia monofásico que posea comunicación mediante RS-485 con el protocolo Modbus. Además, se estudiarán alternativas para la unidad de procesamiento, el adaptador a RS-485 que será necesario para cualquiera de las unidades de procesamiento, y finalmente el lenguaje de programación que se utilizará para llevar a cabo este trabajo.

2.1 Vatímetro RS-485 con Modbus

El vatímetro que se eligió para este proyecto es el SDM230 Modbus de Eastron, el que se muestra en la *figura 2*. Este, además de la comunicación Modbus que se requiere, lee todos los parámetros de la red eléctrica que se requieren, por lo que no será necesario calcularlos. Y tiene un rango elevado de voltaje y corriente, por lo que se puede conectar a gran parte de las instalaciones industriales. Además, tiene un precio bastante asequible, rodeando los 100€.

Figura 2: Vatímetro SDM230 Modbus de Eastron.

En el otro lado del espectro de esta clase de vatímetros, esta el Acuvim II de Accuenergy, mostrado en la *figura 3*. Este ofrece mucha mas versatilidad y flexibilidad que el SDM230; tiene varios tipos de comunicación con mayores velocidades, mas rango de voltaje y corriente, y tiene una pantalla con los parámetros mas visibles. Obviamente este dispositivo es bastante mas grande y robusto, pero también es mucho mas caro, costando alrededor de los 350 €.

Figura 3: Vatímetro Acuvim II de Accuenergy.

Como este trabajo se enfoca más en lo económico y pequeño, este vatímetro, aun siendo mejor y más rápido, fue descartado. Para el uso que se le va a dar, la funcionalidad que proporciona el SDM230 es mucho más que suficiente.

2.2 Unidad de Procesamiento Adicional

Para esta parte se necesita de un dispositivo capaz de comunicarse con el vatímetro mediante RS-485 y, además, tenga un módulo Wi-Fi para poder transmitir los datos inalámbricamente. Los requerimientos de procesado no son muy exigentes, ya que es simplemente leer datos y enviarlos. Y en cuanto al consumo de energía, es preferible que consuma el mínimo posible.

Investigando esta parte del mercado, hay dos vertientes que cumplen con los requisitos y se puede utilizar para este trabajo: micros con sistema operativo, o sin sistema operativo.

La principal diferencia entre estos dos es la manera de programarlos. Para los dispositivos sin sistema operativo, toda la programación se debe hacer con un ordenador externo y se tiene que cargar físicamente en el dispositivo para poder ejecutarse. Mientras que los que sí tienen sistema operativo son, en esencia, un ordenador; por lo que se pueden crear y ejecutar programas en el mismo dispositivo.

Es más común encontrar dispositivos sin sistema operativo ya que no requieren de la compleja integración que se necesita para que funcione un ordenador. Uno de los dispositivos más usados para esta aplicación es el "Arduino Uno Wi-Fi", *figura 4*.

Es una de las mejores opciones porque incluye una plataforma muy extensa de desarrollo, para todo tipo de aplicaciones. En el caso del modelo "Uno Wi-Fi", incluye todas las librerías necesarias para poder crear la aplicación de este trabajo. Además, está específicamente diseñado para aplicaciones IoT.

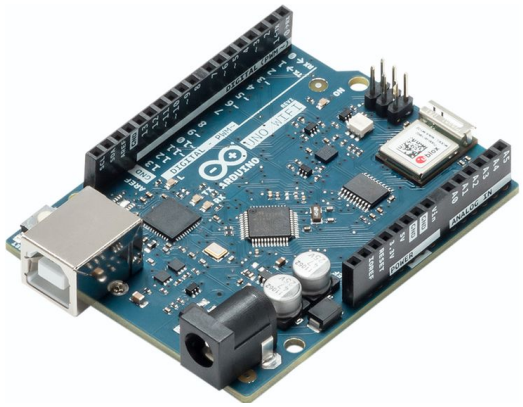


Figura 4: Imagen de un Arduino Uno Wi-Fi.

Por otro lado, están los que tienen sistema operativo. La marca más predominante en esta sección es "Raspberry Pi". Estos sistemas, aun siendo más complejos internamente, suelen ser comparables a sus alternativas en cuanto a consumo energético.

La placa que se utilizará para este trabajo será la "Raspberry Pi Zero W", mostrado en la *figura 5*. Principalmente por sus capacidades Wi-Fi, pero también por su enfoque en bajo consumo y huella física relativamente pequeña.

Figura 5: Imagen de un Raspberry Pi Zero W.

Se elige este dispositivo frente a sus alternativas por una razón: flexibilidad. Programar sobre un sistema operativo hace que el mantenimiento, mejoras o actualizaciones, sean mucho más sencillas frente a tener que desmontar toda la instalación para reprogramar la placa.

También reducirá a gran medida el desarrollo del programa de este trabajo, ya que, al ejecutarse el programa en lo que es esencialmente un ordenador, se pueden detectar y corregir errores mucho más fácil y rápidamente.

Además, la Raspberry Pi es sustancialmente más económica que la placa de Arduino.

Cabe mencionar que la placa de Arduino, al ser una plataforma muy utilizada, es capaz de ejecutar un sistema operativo. Pero de manera muy ineficiente y con funcionalidad reducida. Normalmente, este tipo de microcontroladores no tienen los requisitos necesarios para poder aguantar un sistema operativo.

2.3 Adaptador a RS-485

Para poder conectar cualquiera de los dispositivos mencionados anteriormente con el vatímetro SDM230, es necesario un adaptador de comunicación serie (UART o USB) a RS-485, que es la que utiliza el vatímetro.

La opción más económica y optimizada es mediante un conversor USB a RS-485. Este tipo de conversiones son muy comunes en ámbitos industriales y por tanto se utilizará para este trabajo.

Es una conversión relativamente simple y se pueden encontrar varios modelos. Para este trabajo se buscó uno genérico, el mostrado en la *figura 6*, que tuviese una buena relación calidad-precio.

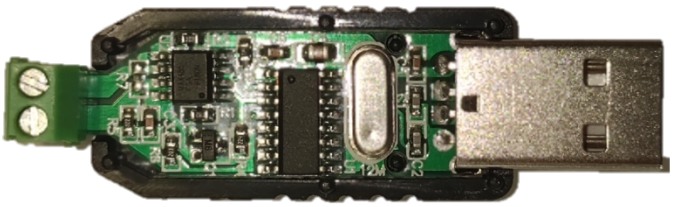


Figura 6: Adaptador USB a RS-485 utilizado para este trabajo.

La alternativa a esto sería diseñar un circuito que convierta las señales TTL de los puertos digitales del Arduino (o los GPIO de la Raspberry), a RS-485. Esto sería más costoso, en cuanto a tiempo y dinero, y no aportaría ninguna ventaja frente a una solución integrada.

Además, la ventaja de utilizar USB frente a otras alternativas digitales es la facilidad de manipular palabras (normalmente de 8 bits con paridad) en vez de tener que analizar uno por uno los bits que van llegando por los pines de las placas.

2.4 Lenguaje de Programación

Una de las ventajas, que no se mencionó, de utilizar un dispositivo con sistema operativo es que se puede utilizar cualquier lenguaje que sea compatible con dicho sistema operativo.

Para este trabajo se utilizará "C". Es un lenguaje que ofrece mucho control sobre la ejecución del programa y tiene una capa de abstracción más cercana al sistema operativo del dispositivo. Esto permitirá crear soluciones óptimas y con mucho más control que con cualquier otra alternativa.

"C" es también uno de los lenguajes más extendidos y utilizados, por lo que encontrar librerías compatibles será mucho más sencillo que con cualquier otro programa.

La única desventaja frente a otros lenguajes más abstractos es que la complejidad de desarrollar un programa es más elevada. Pero con el conocimiento adecuado para programar en C, esto no será tan gran inconveniente.

Para el desarrollar la interfaz del usuario se utilizará Java por su simplicidad a la ahora de crear interfaces gráficas. Y también por su compatibilidad multiplataforma; esto permitirá crear una sola aplicación que se podrá ejecutar en casi todos los sistemas operativos.

3 Material Utilizado y Software

3.1 SDM230-MODBUS

El SDM230 (*figura 7*) es un vatímetro monofásico, diseñado para ser montado en carriles DIN. Se le llama vatímetro, pero realmente mide muchos más parámetros de la red eléctrica; tal como el voltaje, la corriente, la frecuencia, etc. Además, es capaz de calcular la energía consumida y generada por la red que está midiendo.

Figura 7: Vatímetro SDM230-Modbus de Eastron.

Lo importante de este vatímetro es que tiene integrado un puerto de comunicación serie a través del protocolo MODBUS. Y, como se comentó anteriormente, sigue el estándar RS-485.

Este protocolo, aunque extenso, es fácil de implementar y controlar. Se explicará en más detalle en el apartado posterior de la solución adoptada.

3.2 Raspberry Pi Zero W

La marca Raspberry Pi (*figura 8*), en general, fabrica y diseña ordenadores de placa única, con un formato pequeño y asequible. En este caso concreto, es elegido es de la línea "Zero", que consiste en empaquetar toda la funcionalidad de la Raspberry Pi en un formato aun más pequeño y con el menor consumo posible.

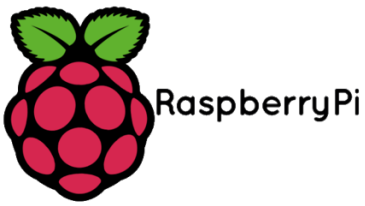


Figura 8: Logotipo de Raspberry Pi.

Además, esta placa, la mostrada en la *figura 9*, tiene un sistema de comunicación inalámbrica; principalmente Wi-Fi y Bluetooth. Que, como se ha comentado extensamente, es muy relevante para este trabajo.

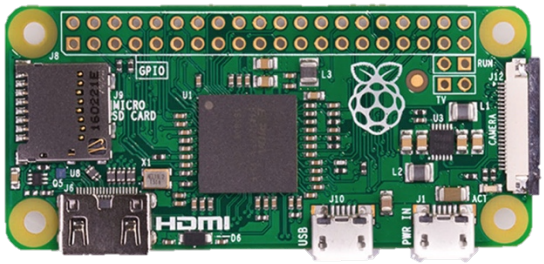


Figura 9: Vista frontal de la Raspberry Pi Zero W.

El kit que se compró, mostrado en la *figura 10*, viene con una fuente de alimentación USB, una tarjeta *microSD* de 16 gigabytes y varios cables adaptadores para poder empezar a utilizarla.

Figura 10: Imagen de lo incluido en el kit de la Raspberry Pi.

3.3 Convertidor USB a RS-485

El vatímetro, que se verá después, se comunica a través del estándar RS-485 y las Raspberry Pi solo tienen entradas USB. Estos dos estándares son físicamente similares, pero tanto la filosofía de transmisión como las características eléctricas son fundamentalmente distintas. Por tanto, es necesaria una conversión.

Por suerte, este tipo de conversión es muy común, ya que RS-485 era uno de los estándares predominantes antes de la llegada de USB. Se pueden encontrar en prácticamente cualquier tienda electrónica, y son muy asequibles. El que se utilizará para este trabajo se muestra en la *figura 11*.

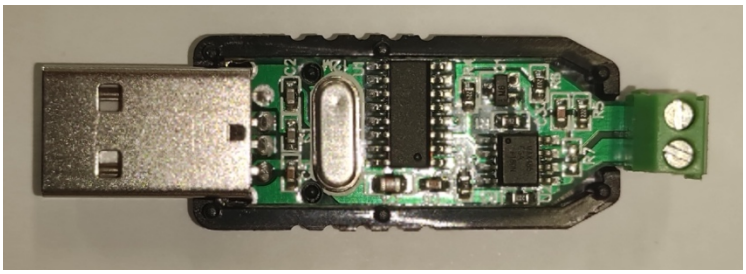


Figura 11: Imagen del conversor USB a RS-485 utilizado para este trabajo.

3.4 Visual Studio Code

Visual Studio Code es un programa ligero y fácil de usar. Es compatible con casi todos los lenguajes de programación y tiene muchos complementos que extienden la funcionalidad del programa. Su entorno se muestra en la *figura 12*.

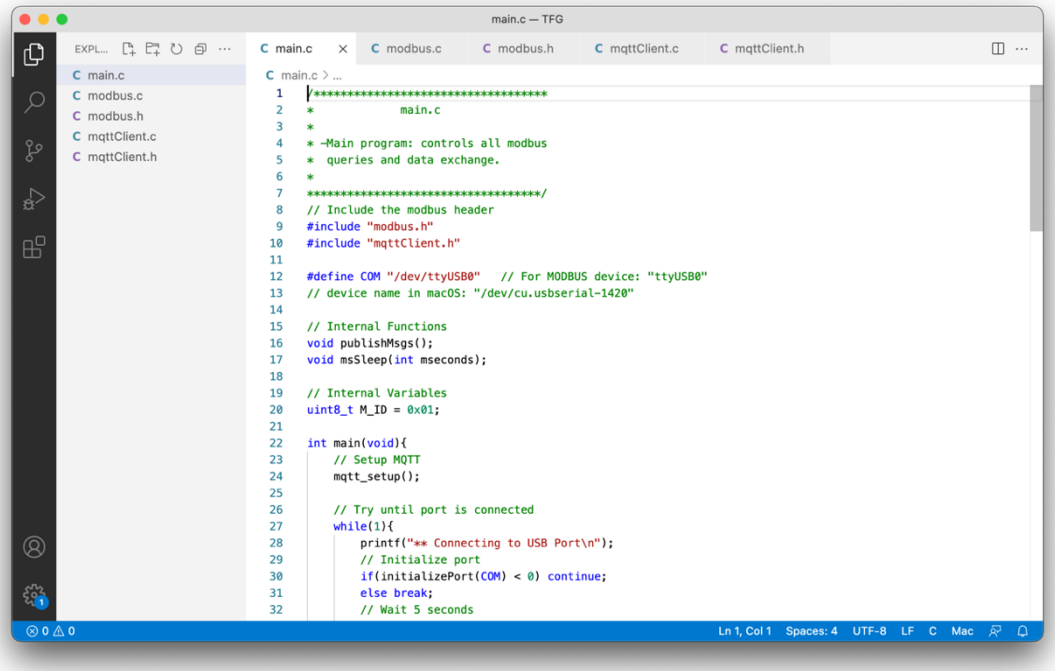


Figura 12: Imagen del entorno de Visual Studio Code.

Se utilizará para editar códigos escritos en C. Al ser un lenguaje bastante común, hay multitud de entornos que son compatibles. Uno de los principales motivos por el que se eligió este programa frente a otros, es el hecho de ser "open source". Esto permite que la funcionalidad del programa sea eficiente y efectiva.

Otra ventaja es que incluye acceso directo a una terminal, por lo que hace el proceso de exportar programas a la Raspberry Pi aún más sencillo. También incluye soporte para gran parte de las librerías de Linux, cosa que es útil para saber si el programa está bien escrito sin tener que compilarlo.

3.5 Eclipse IDE

Al igual que Visual Studio Code, Eclipse es un entorno de desarrollo "open source". Destaca por su uso en creación de aplicaciones Java, aunque también es capaz de interpretar otros lenguajes. Su entorno se muestra en la *figura 13*.

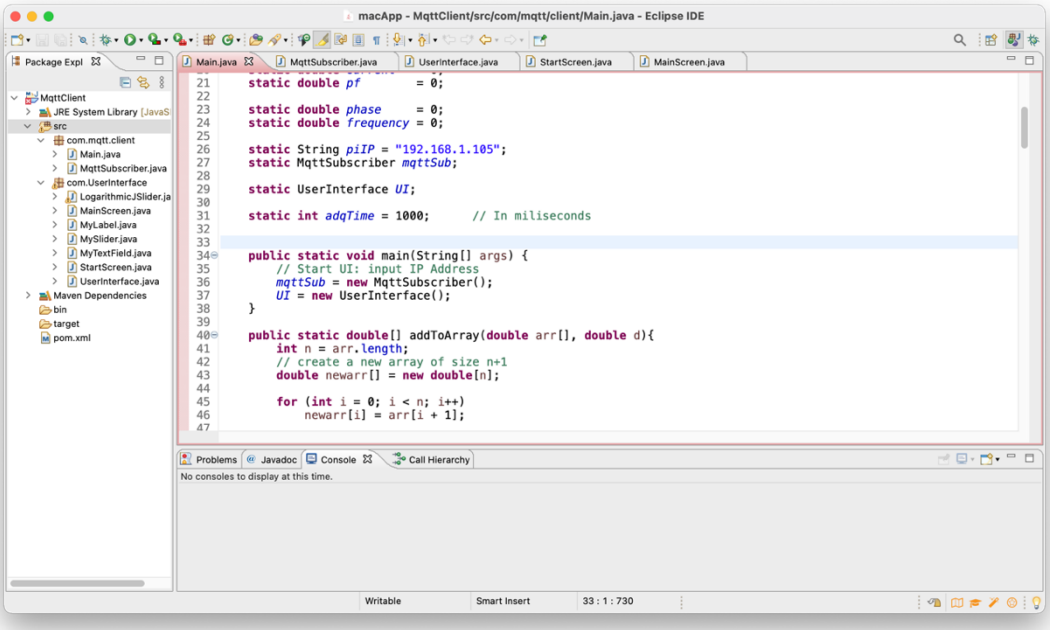


Figura 13: Imagen del entorno de Eclipse IDE.

Se eligió porque proporciona una de las maneras más rápidas para crear una interfaz gráfica, pero con el suficiente control para poder implementar fácilmente código personalizado.

4 Solución Adoptada

La solución se divide en dos componentes y la aplicación gráfica. El primer componente consiste en la comunicación entre el vatímetro SDM230 y la Raspberry Pi. Donde se hará uso de la comunicación serie mediante el protocolo MODBUS. El segundo es la comunicación que tiene la Raspberry Pi con internet mediante el protocolo MQTT. Y, por último, la aplicación Java del dispositivo externo, que consistirá en una interfaz gráfica de usuario para visualizar los parámetros eléctricos.

Por tanto, en la primera parte se explicará el primer componente y en la segunda el segundo. Luego, en la tercera parte, se hablará del programa que se ejecutará en la Raspberry Pi, donde se combinan los dos componentes en uno. En la cuarta parte se comentará el diseño de la aplicación gráfica. Y por último se hablará de la fase experimental, donde se comprueba el funcionamiento de todo el sistema a la vez.

4.1 Comunicación Modbus

A diferencia del resto de apartados, este consta de una pequeña complejidad añadida. Además de tener la parte digital de programación, también tiene una parte física de hardware, ya que la comunicación serie entre ambos dispositivos se hace mediante un par de cables, por lo que se hablará del montaje.

Pero antes de entrar directamente a la programación o el montaje, se tienen que configurar tanto la Raspberry Pi, como el vatímetro SDM230.

4.1.1 Configuración Inicial: Raspberry Pi

Empezando por la Raspberry Pi, lo primero que se tiene que hacer es instalar el sistema operativo en la tarjeta SD que viene incluida. Para ello, es necesario tener un ordenador con "Raspberry Pi Imager" instalado, mostrado en la *figura 14*. Después es solo cuestión de seguir los pasos del programa e instalar "Raspbian OS" en la tarjeta SD.

Figura 14: Imagen del programa Raspberry Pi Imager.

Una vez terminado el proceso, insertamos la tarjeta SD en la Raspberry Pi y la conectamos a una fuente de alimentación. Es necesario tener una pantalla, teclado y ratón conectada a la Raspberry Pi para poder configurarla; al menos inicialmente.

Al conectar todo, aparecerá en la pantalla el escritorio del sistema operativo de la Raspberry Pi, como se muestra en la *figura 15*.

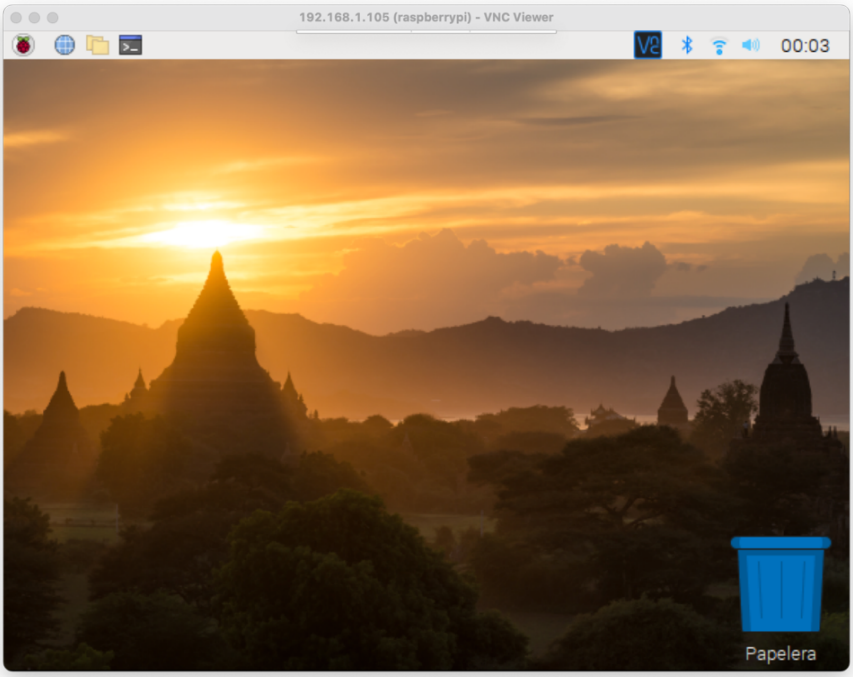


Figura 15: Imagen del escritorio de la Raspberry Pi.

Lo primero será configurar la Raspberry Pi para poder utilizarla remotamente. Para ello se tiene que conectar a internet y luego activar el protocolo SSH.

Para conectarla a internet simplemente se tiene que pulsar sobre el icono de Wi-Fi en la parte superior derecha de la pantalla (como se indica en la *figura 16*), seleccionar la red e introducir la contraseña en el recuadro mostrado en la *figura 17*.

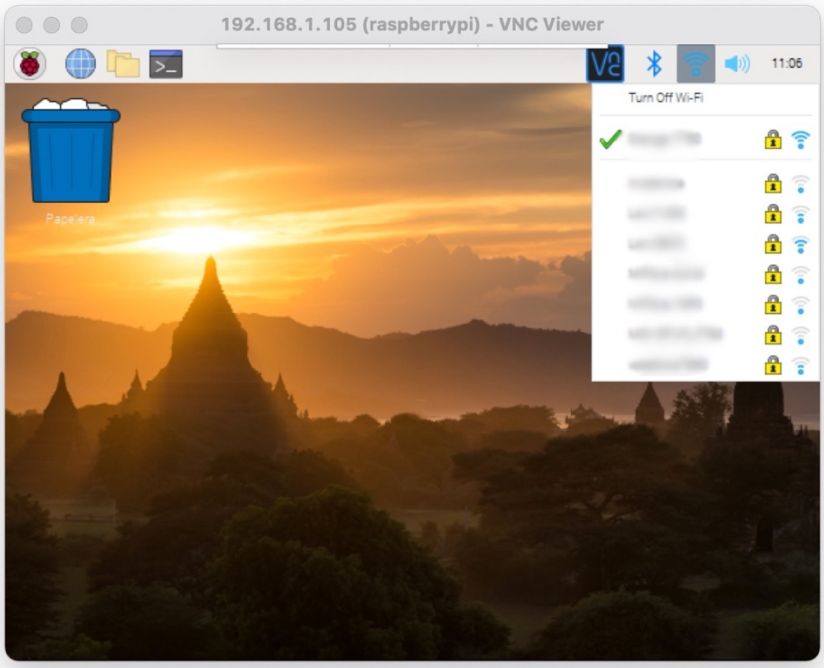


Figura 16: Demostración de la posición de la configuración Wi-Fi.

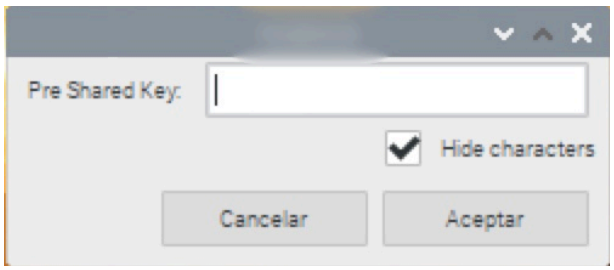


Figura 17: Ventana de configuración Wi-Fi.

Otra manera de hacerlo, que resultará más útil cuando accedemos a ella remotamente, es modificando el archivo `wpa_suplicant.conf`. Para ello se tiene que abrir la aplicación de la terminal (mostrada en la *figura 18*) y escribir el siguiente comando:

```
$ sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
```

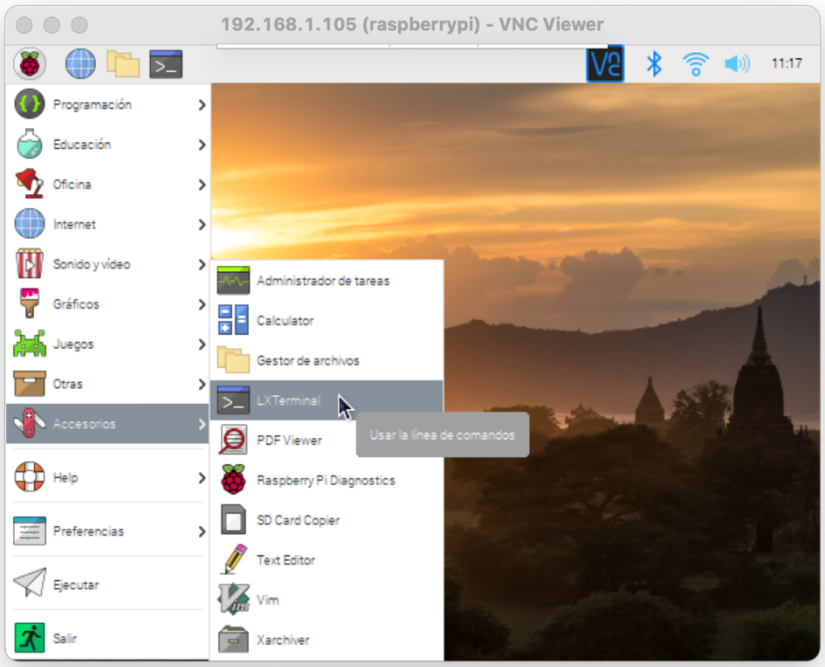


Figura 18: Demostración del lugar donde se encuentra la terminal.

Aparecerá un archivo (mostrado en la *figura 19*) para editar donde introduciremos los datos de la red a la que se quiere conectar, como se muestra en la *figura 18*. Una vez hecho eso, se pulsa "Ctrl+O" para guardar la configuración.

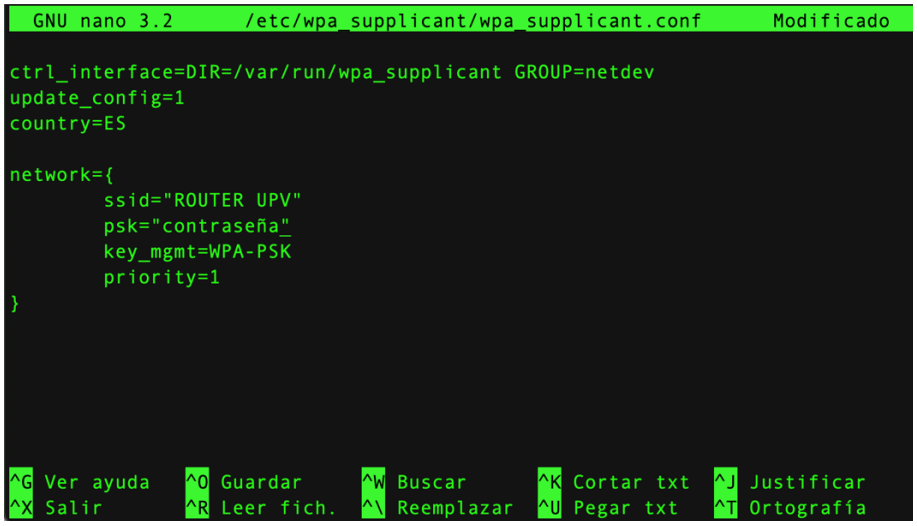


Figura 19: Archivo "wpa_supplicant.conf" en la terminal.

Para activar el protocolo de SSH, se tiene que volver a la aplicación del terminal. Escribiendo el comando de abajo, se abrirá una ventana de configuración. De ahí, se tendrá que navegar al menú "3. Interface Options" y luego a "P2. SSH" con las flechas del teclado y seleccionar sí al final como se muestran en las imágenes posteriores (*figura 20*).

```
$ sudo raspi-config
```

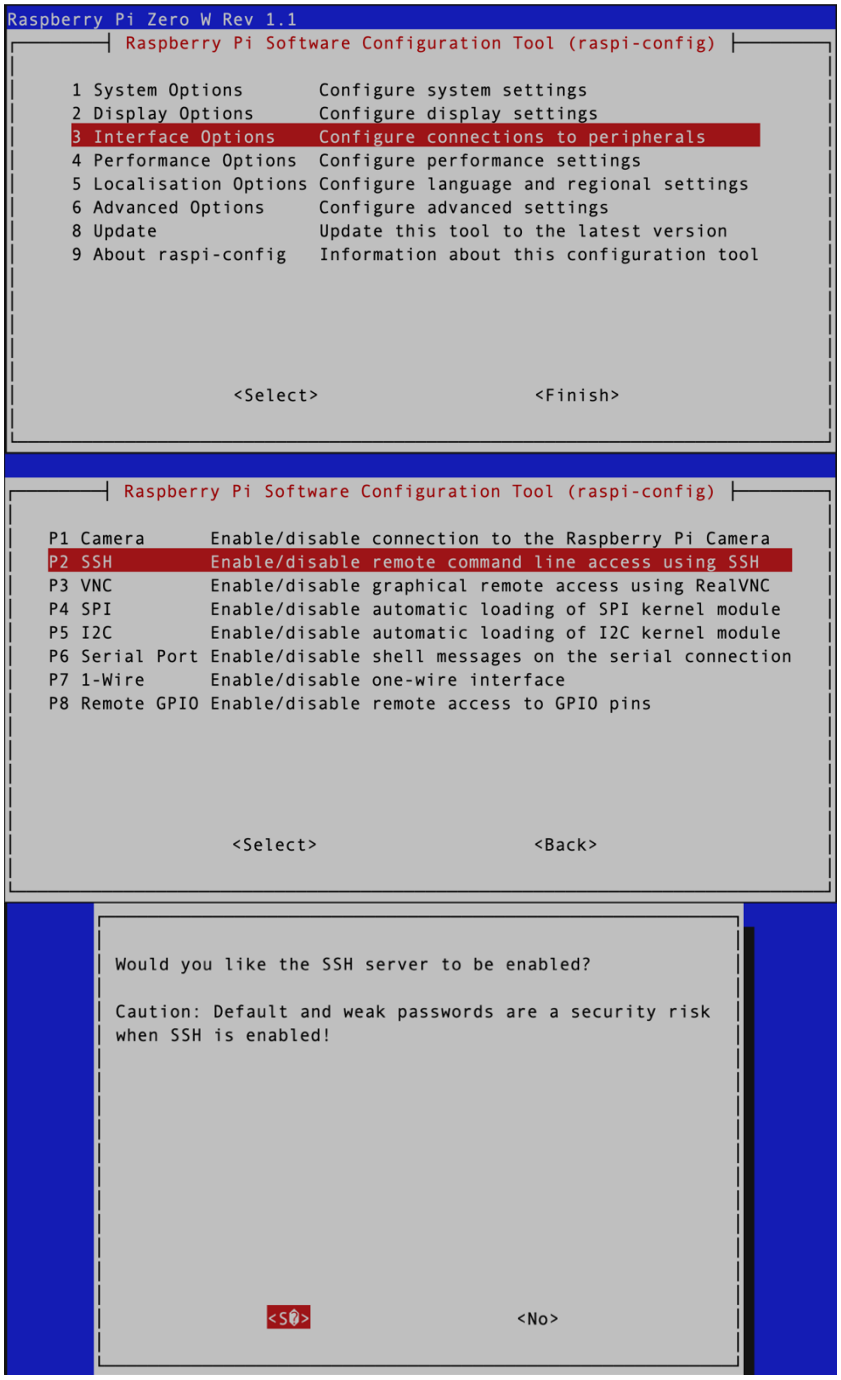


Figura 20: De arriba a abajo, pasos a seguir para habilitar la interfaz SSH.

Con esto hecho, se pueden desconectar la pantalla, el teclado y el ratón. A partir de ahora, todo se hará remotamente desde otro ordenador mediante el protocolo SSH. Para ello, se abre la terminal desde el otro ordenador (como se muestra en la *figura 21*), y se escribe el siguiente comando:

```
$ sudo ssh pi@raspberrypi.local
```

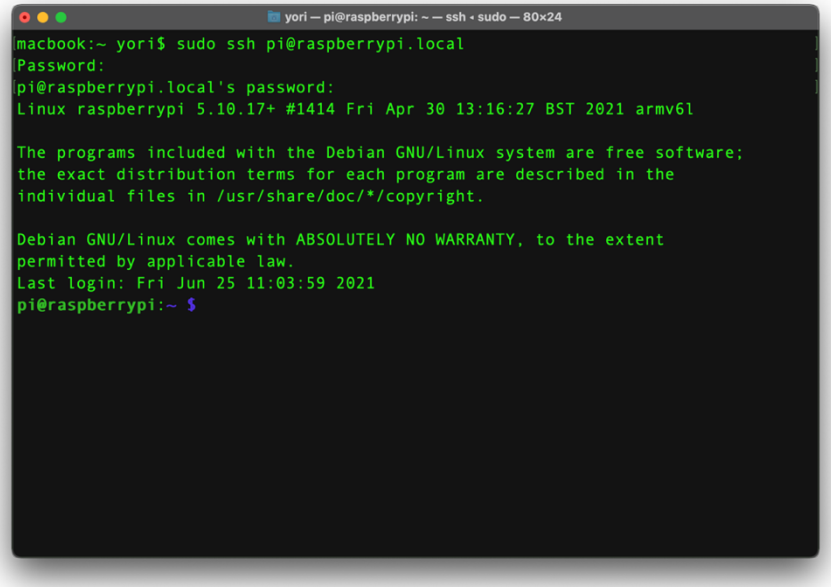


Figura 21: Conexión a la Raspberry Pi desde otro ordenador mediante SSH.

Pedirá la contraseña del usuario "Pi" de Raspberry. Por defecto, la contraseña es "raspberry", así que se introduce eso.

Una vez se conecte a la Raspberry Pi, todos los comandos que se escriban en la terminal del ordenador externo se ejecutarán en la Raspberry Pi.

Para empezar con el trabajo, se creará un directorio donde guardar todos los archivos C y el programa que se ejecutará. Luego, se tendrá que entrar en el directorio, para que todos los comandos que se escriban estén sean relativos a este. Por ejemplo, si se quiere compilar el archivo "X", basta con escribir el comando de compilar y el nombre del archivo; y mientras el archivo "X" se encuentre en el directorio donde se está trabajando, el comando funcionará. Para conseguir esto, se escriben los siguientes comandos:

```
$ sudo mkdir modbus/
$ sudo cd modbus/
```

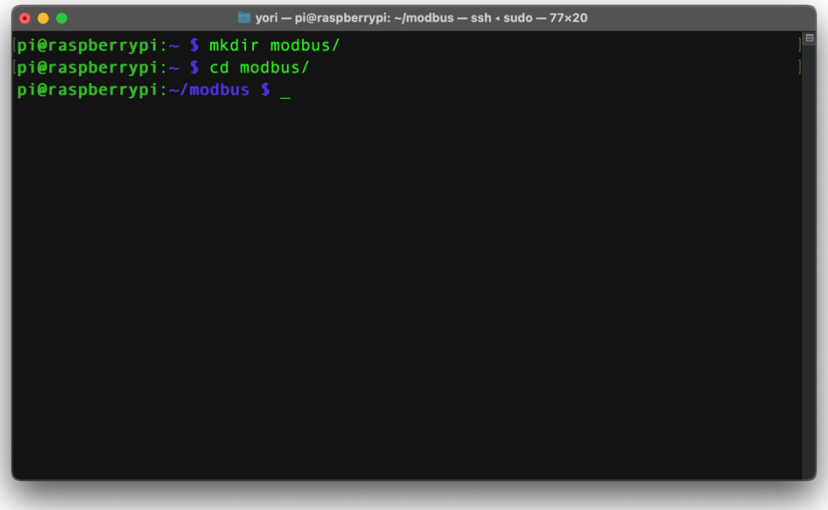


Figura 22: Creación del directorio y situando el área de trabajo sobre el.

Como se observa en la *figura 22*, se pude ver en todo momento cual es el directorio donde se están ejecutando los comandos entre el nombre del usuario y el símbolo `$`; en este caso es el directorio `~/modbus`.

Continuando con el tema de comandos, además de los que ya se han visto, es importante explicar unos que serán de utilidad para este trabajo.

Cómo se trabajará remotamente, será conveniente saber copiar archivos desde el ordenador a la Raspberry Pi. Para ello se utiliza este comando, similar al de SSH:

```
$ sudo scp *.c *.h pi@raspberrypi.local:~/modbus/
```

Los asteriscos al lado del tipo de archivo indican que se copien todos los archivos de ese tipo. Como se comentaba anteriormente, es importante apuntar a la terminal al directorio en el que se encuentran estos archivos. Al final del comando, se indica el directorio al que queremos que se copien; en este caso queremos copiar todos los archivos ".c" y ".h" del directorio en el ordenador, al directorio `~/modbus/` de la Raspberry Pi que ya se había creado.

Al ejecutarse este comando, aparecerán los archivos que se copiaron y si resultó exitosa la transacción o no. Tal y como se muestra en la *figura 23*.

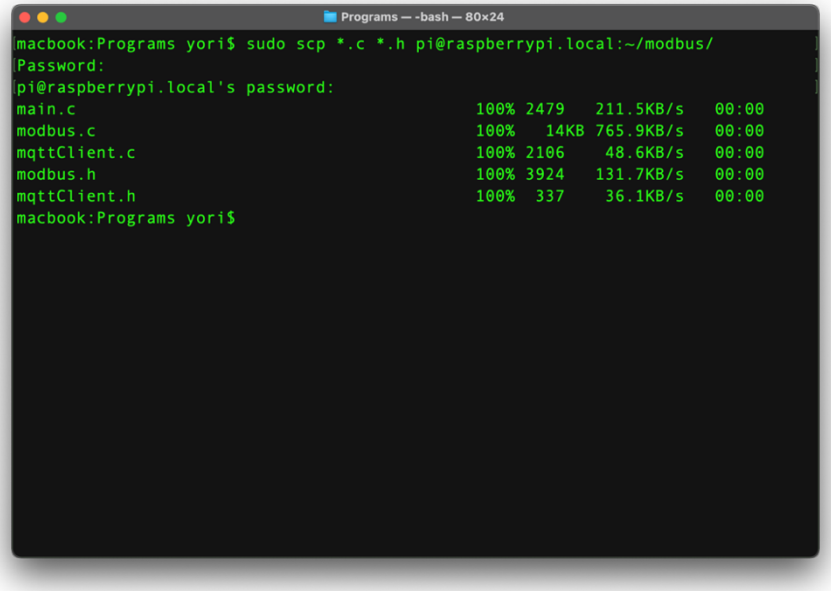


Figura 23: Ejemplo de una transferencia de archivos exitosa.

Una vez copiados estos archivos, y desde la Raspberry Pi, se utiliza el siguiente comando para compilar el código a una aplicación ejecutable:

```
$ gcc modbus.c -o modbus.exec
```

Igual que en el comando "scp", también se puede añadir asterisco para indicar que se compilen todos los archivos ".c". Para ejecutar el programa creado, simplemente se tiene que añadir "./" seguido del nombre. Por ejemplo, para ejecutar "modbus.exec", se escribirá:

```
$ ./modbus.exec
```

El comando "nano" sirve para editar cualquier archivo desde la terminal, como se mostró anteriormente para modificar la configuración Wi-Fi. De esta forma, si se tiene que hacer un cambio pequeño, no es necesario utilizar el comando "scp" desde el ordenador externo y esperar a que se envíen los archivos. Para ello se escribirá:

```
$ nano modbus.c
```

Para cerrar o cancelar cualquier aplicación o comando activo en el terminal, se tiene que utilizar la combinación de teclas "Ctrl+C". Esto resulta útil para parar el ejecutable sin tener que reiniciar la Raspberry Pi o la conexión SSH.

4.1.2 Configuración Inicial: SDM230-MODBUS

La configuración del vatímetro es más sencilla. Los únicos parámetros que interesan cambiar son la velocidad de transmisión a 9600 Baudios, y el número de identificación del dispositivo esclavo.

Una vez encendido, se deja pulsado el botón "Enter" (indicado en la *figura 24*) durante 3 segundos hasta que aparezca una pantalla de introducir contraseña, como se muestra en la *figura 25*. La contraseña es 1-0-0-0 por defecto, y se puede cambiar en este mismo menú. En este caso se dejará intacta, ya que no es relevante para esta aplicación.

Figura 24: Localización de los botones Scroll y Enter en el vatímetro SDM230.

Figura 25: Pantalla del vatímetro para introducir la contraseña.

Una vez introducida la contraseña y aceptada, se pulsa el botón Scroll (indicado en la *figura 24*) hasta llegar a la pantalla de "Address ID" (*figura 26*). Se cambia a "001", ya que es el único dispositivo de nuestra red, y dejamos pulsado Enter otros 3 segundos para guardar el cambio. Si se diera el caso de ampliar la red MODBUS, cada dispositivo debería tener una identificación diferente.

Figura 26: Pantalla del vatímetro donde introducir la dirección del esclavo.

Ahora, se pulsa Scroll de nuevo hasta que aparezca la pantalla de "Baud Rate" (*figura 27*). Por defecto está a 2400 Baudios, así que se cambiará a 9600 Baudios que es la velocidad más alta que permite el vatímetro.

Figura 27: Pantalla del vatímetro donde introducir la velocidad en baudios.

La configuración se queda guardada permanentemente, incluso después de haberse apagado. Por lo que ya no se volverá a tocar el vatímetro.

Todas estas instrucciones se encuentran más detalladamente en las hojas de datos del SDM230, que se encontrarán adjuntos a este trabajo.

4.1.3 Motaje

El vatímetro SDM230, consta de una línea de comunicación serie por el estándar RS485, utilizando el protocolo MODBUS.

RS-485 es un estándar económico, relativamente rápido y muy extendido en la industria de sensores. Este estándar, a pesar de emplear un simple bus con un par de cables, requiere de unas especificaciones eléctricas concretas tanto para los transmisores como los receptores. Como se mencionó en el estudio de las alternativas, la Raspberry Pi no posee dicha transmisión serie, por lo que se tiene que emplear un conversor USB a RS-485.

Para los cables de transmisión, se utilizará un par trenzado; una de las opciones más comunes para este estándar, ya que evita interferencias. Además, habrá que terminar ambos finales del bus con resistencias de 120 ohmios, como se indica en la *figura 28*.

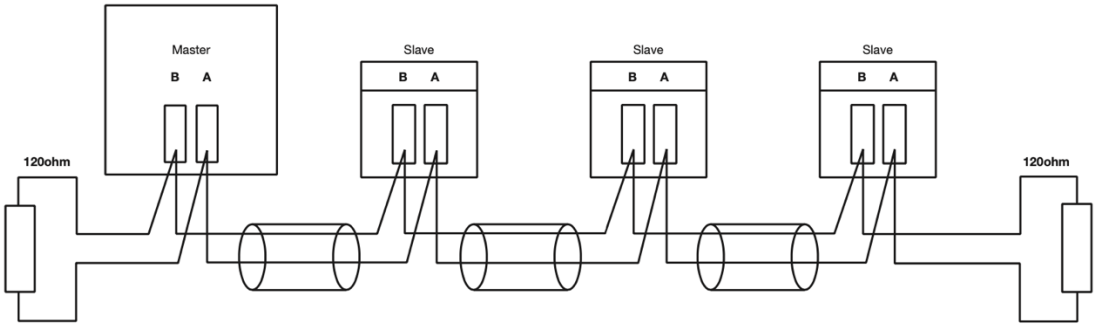


Figura 28: Diagrama para ejemplificar la conexión del bus RS-485.

Tanto el conversor USB-RS485 como los terminales del SDM230 constan de entradas atornillables para hacer fácil la instalación de dichos cables.

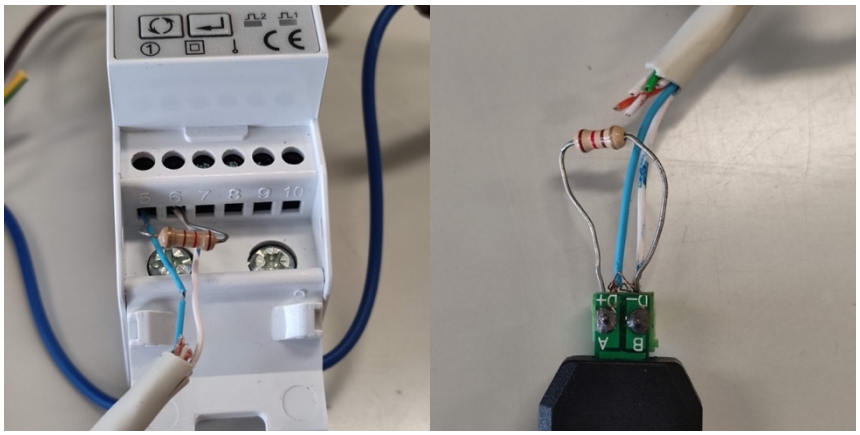


Figura 29: Conexionado de las resistencias y cable del vatímetro (izquierda) y el adaptador USB a RS-485 (derecha).

Las imágenes de la *figura 29* son del prototipo creado y con un solo esclavo conectado al maestro. Idealmente, se crearían nodos en una línea de bus común de donde se conectarán todos los dispositivos, como se muestra en la *figura 28*.

4.1.4 Programa

El objetivo principal de este componente es mandar una serie datos por la comunicación serie de la Raspberry Pi (que, a su vez, está conectado con el convertidor USB-RS485) y recibir los datos provenientes del vatímetro.

Para conseguir esto en código, hay que entender cómo se envían datos al puerto serie y que datos hay que enviar para que la comunicación sea efectiva. Para ello, primero se profundizará en el protocolo MODBUS y el sistema operativo de Raspberry Pi.

4.1.4.1 Protocolo MODBUS

El protocolo MODBUS fue diseñado para aplicaciones centralizadas de sensores; un ordenador central (el maestro) recibe datos de los sensores conectados a un bus (los esclavos). Este protocolo es muy extenso y cubre muchas funciones que el SDM230 no contempla. De manera simplificada, y para lo que se va a utilizar en este trabajo, el maestro envía un mensaje por el bus y espera hasta recibir una respuesta del esclavo. Si no la recibe, el mensaje se vuelve a enviar.

El mensaje del maestro puede variar, pero para este caso solo consiste de 8 palabras con 8 bits en cada una (es decir, 8 bytes). Este mensaje consta de varias partes (en la *figura 30* se muestra el orden del mensaje):

- **Identificación del esclavo:** un byte que representa el número del esclavo al que se dirige el mensaje. En el caso del vatímetro configurado es 0x01.
- **Código de función:** es el "comando" que se le envía al sensor. En el caso de este proyecto, solo nos interesa el comando 0x03, que nos permite leer los contenidos de los registros donde se almacenan los valores del vatímetro.
- **Dirección:** son 2 bytes que indican la dirección de los registros a los que se quieren acceder. Cada parámetro eléctrico que lee el sensor tendrá su propia dirección.
- **Número de registros:** son 2 bytes que indican cuántos registros se quieren acceder. Cada registro almacena 2 bytes, y como veremos en la respuesta del esclavo, requerimos de 4 bytes como respuesta. Por lo que este valor siempre será 0x0002 para este trabajo, que equivale a 2 registros.
- **Error CRC:** son 2 bytes para comprobar la integridad del mensaje. Mediante un cálculo, que incluye el mensaje entero (sin contar el CRC), se generan este par de bytes. Luego el esclavo repetirá el cálculo para comprobar que el mensaje recibido es correcto.

First Byte							Last Byte
Slave Address	Function Code	Start Address (Hi)	Start Address (Lo)	Number of Points (Hi)	Number of Points (Lo)	Error Check (Lo)	Error Check (Hi)

Figura 30: Orden de bytes en el mensaje del maestro.

Para la respuesta ocurre algo similar. El esclavo tiene varios tipos de respuesta, pero la más importante es la de datos (en la *figura 31* se muestra el orden del mensaje):

- **Identificación del esclavo:** repite su propia identificación.
- Código de función: el mismo comando que fue enviado por el maestro.
- **Número de bytes:** un byte que indica cuántos bytes tiene los datos que se están enviando. Como es el caso del SDM230, los datos que se enviarán serán un número en formato coma flotante del estándar IEEE 754. Este consiste de 4 bytes, por tanto, este byte siempre será 0x04 para este trabajo.
- **Datos:** es la respuesta de 4 bytes, en formato de coma flotante.
- **Error CRC:** 2 bytes para comprobar la integridad del mensaje.

First Byte								Last Byte
Slave Address	Function Code	Byte Count	First Register (Hi)	First Register (Lo)	Second Register (Hi)	Second Register (Lo)	Error Check (Lo)	Error Check (Hi)

Figura 31: Orden de bytes en la respuesta del esclavo.

La respuesta también puede ser una respuesta de Error. Donde, en vez de tener los datos y número de bytes, viene con un código del error. En la *figura 32* se puede ver el orden de estos bytes.

First Byte				Last Byte
Slave Address	Function Code	Error Code	Error Check (Lo)	Error Check (Hi)

Figura 32: Orden de bytes en la respuesta de error del esclavo.

4.1.4.2 Raspbian OS

El sistema operativo que utiliza la Raspberry Pi es Raspbian OS. Este, es un sistema operativo basado en Linux, y contiene todas las herramientas para ejecutar las tareas básicas de un ordenador; incluidas entre ellas, las comunicaciones al puerto serie y protocolos de internet. Por lo que no se tendrá que preocupar de recibir los bits individualmente o de manejar los tiempos de transmisión.

Una cosa a tener en cuenta cuando se habla de sistemas operativos basados en Linux es que "todo es un archivo". Por ejemplo, en el caso del puerto serie, lo que se haría es abrir el archivo del USB y escribir o leer sobre él. Esta es una de las ideas fundamentales del sistema operativo y es algo que dará forma a la manera de crear el programa para este trabajo.

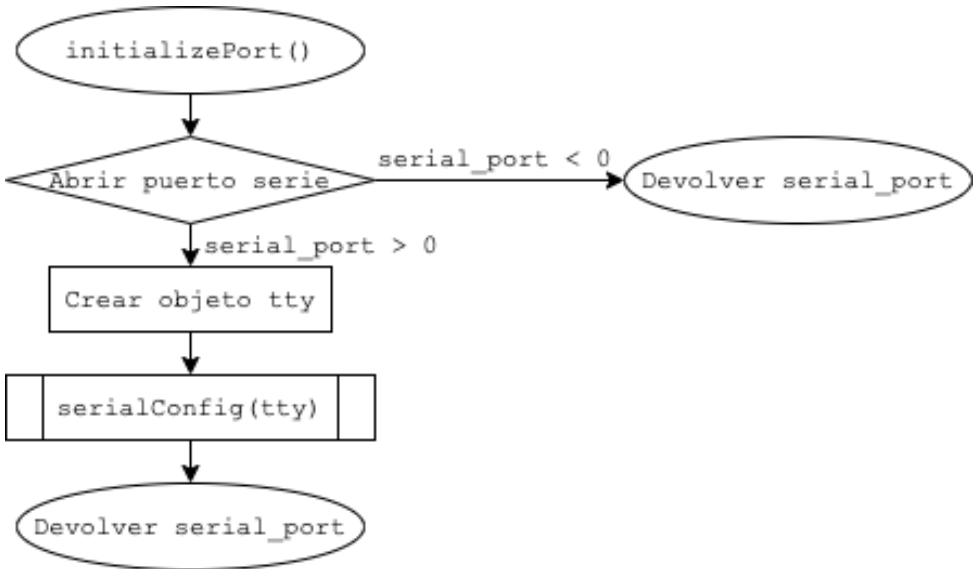
Otra idea que considerar de este sistema operativo es que no trabaja a tiempo real. Es decir, cualquier acción que le digamos que haga, lo ejecutará cuando lo vea necesario. Si es cierto que es bastante rápido, pero, como veremos, esto puede dar problemas ya que no se puede saber cuando ciertas tareas terminan de ejecutarse.

4.1.4.3 Código

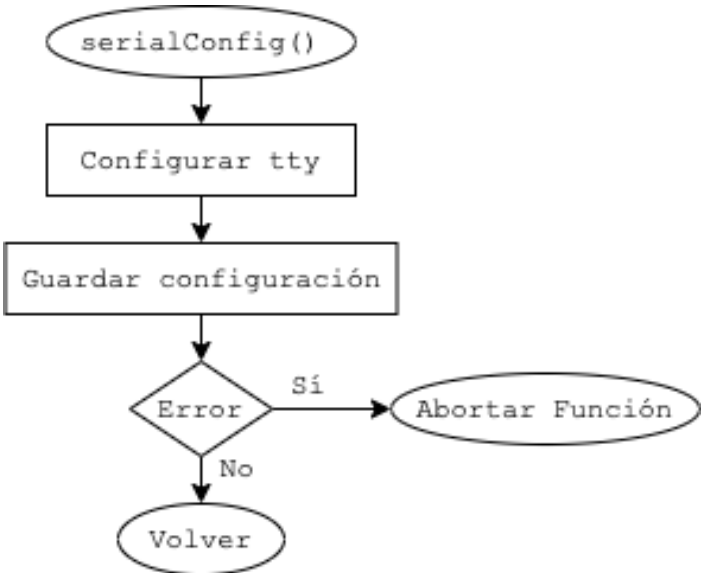
Como se comentó en el estudio de las alternativas, el lenguaje que se utilizará en este trabajo será "C".

El objetivo de este modulo es construir una función que pueda llamar el programa principal, donde se introduzca el parámetro que se quiera leer, y esta devuelve directamente un número en coma flotante.

Lo primero es inicializar la comunicación serie. Como se vio anteriormente, todo es un archivo. Por tanto, lo que se tiene que hacer es abrir el archivo donde se encuentra el puerto USB mediante la función `initializePort()`:



Lo primero que hacer esta función es abrir el puerto serie utilizando funciones de la librería estándar de Linux. Esta función devuelve un numero que equivale al puerto serie (`serial_port`). Si este numero es menor a cero, significa que hubo un error a la hora de intentar abrir el puerto indicado. En este caso, directamente se devuelve `serial_port`, que el programa principal interpretará como error. Cuando sea mayor a cero significará que el puerto se abrió correctamente. Sabiendo que el puerto esta abierto, se crea el objeto de configuración serie (`tty`) y se modifica con la función `serialConfig()`:



Esta configuración serie servirá tanto para leer del puerto USB como para escribir en el. Lo que permitirá la comunicación entre la Raspberry Pi y el conversor USB a RS-485; y, por consecuente, con el vatímetro.

Esta configuración del `tty` es una variable que contiene la configuración de la conexión al puerto. Se asignarán valores que indican el formato del mensaje y el método en el que se interpretan los bytes recibidos. Lo primero en esta parte es indicar, como dicta el protocolo Modbus, que todas la palabras enviadas y recibidas tengan un bit de paridad, un bit de parada y ocho bits de datos. También se tiene que indicar que la velocidad de transmisión sea de 9600 baudios.

Hay mas parámetros de configuración que se modifican, básicamente sobre la interpretación de los bytes recibidos. Como las comunicaciones son básicas, se deshabilitarán todos estos extras interpretativos.

Una vez este instanciada la comunicación, se guardarán estos cambios y se aplican el puerto serie abierto. Si por alguna razón hubo un error, la función `initializePort()`, que llamo a esta función, se abortará y no devolverá el puerto serie. En caso de que todo sea correcto, `serialConfig()` terminará de ejecutarse, se volverá a la función `initialzePort()`, y esta devolverá el puerto serie.

Ahora que ya está configurado el puerto serie, ya se puede leer y escribir al puerto USB y empezar las comunicaciones con el vatímetro. Pero antes, se tiene que programar qué es lo que se enviará y cómo interpretar lo que llegue del vatímetro.

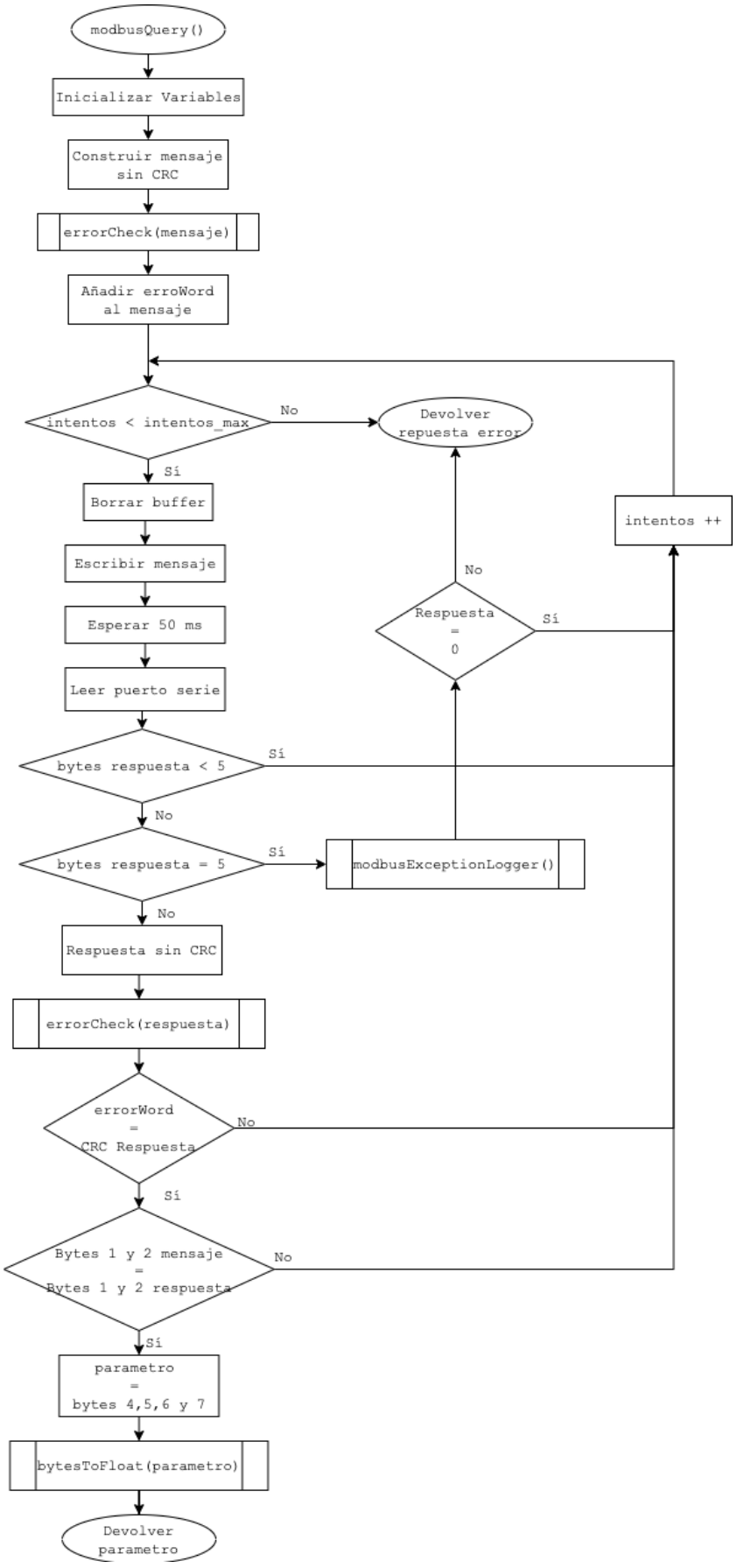
Como se explica en el protocolo MODBUS, el mensaje a enviar consiste en 8 bytes (mostrado en la *figura 33*), con 5 parámetros. El primer byte de la identificación del esclavo será 0x01. El segundo byte de código de función será 0x04. El tercer y cuarto byte de la dirección dependerá del parámetro que se quiera leer. El quinto y sexto byte del número de registros siempre será 0x0002. Y los últimos dos bytes serán el CRC que se calculará cuando el mensaje esté formado. Por tanto, los únicos bytes de interés son los de la dirección y la identificación del esclavo.

First Byte							Last Byte
Slave Address	Function Code	Start Address (Hi)	Start Address (Lo)	Number of Points (Hi)	Number of Points (Lo)	Error Check (Lo)	Error Check (Hi)

Figura 33: Orden de bytes en el mensaje para enviar.

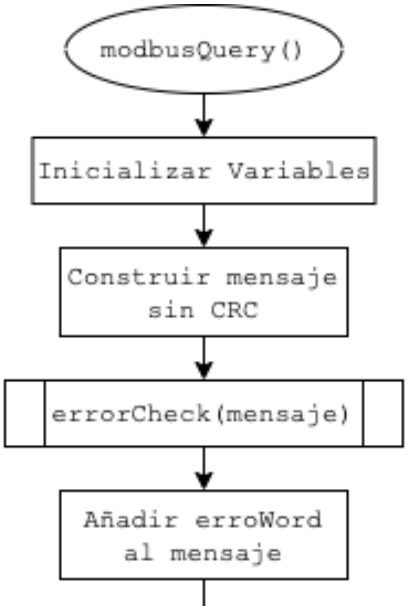
Sabiendo esto se crea la función `modbusQuery()`, que se encargará de componer el mensaje, enviarlo, recibir la respuesta, comprobar que sea correcta e interpretar el parámetro para pasarlo de simples bytes a un numero en coma flotante (`float`).

A continuación, se muestra el diagrama de flujo del funcionamiento completo de esta función, para dar una idea general de como esta todo unido. Enseguida se explicará cada parte con mas detalle.


modbusQuery()
Inicializar Variables
Construir mensaje sin CRC
errorCheck(mensaje)
Añadir erroWord al mensaje
intentos < intentos_max
No
Devolver repuesta error
Sí
Borrar buffer
Escribir mensaje
Esperar 50 ms
Leer puerto serie
intentos ++
No
Respuesta = 0
Sí
bytes respuesta < 5
Sí
No
bytes respuesta = 5
Sí
modbusExceptionLogger()
No
Respuesta sin CRC
errorCheck(respuesta)
errorWord = CRC Respuesta
No
Sí
Bytes 1 y 2 mensaje = Bytes 1 y 2 respuesta
No
Sí
parametro = bytes 4,5,6 y 7
bytesToFloat(parametro)
Devolver parametro

Empezando por el principio de esta función, la creación del mensaje, primero se necesitan inicializar todas las variables que se vayan a utilizar. Esto incluye el lugar donde se guarda el mensaje, la respuesta, el buffer del puerto serie, etc.

Como se comento anteriormente, gran parte de los bytes necesarios para el mensaje ya son conocidos. Simplemente son necesarios el ID del esclavo y la dirección que se quiera leer, que serán parámetro de entrada de esta función. También seria necesario calcular los últimos dos bytes del mensaje que corresponden al error CRC.



Para calcular el error CRC es necesario seguir un algoritmo que se encuentra en la hoja de datos como un pseudocódigo:

```
BEGIN
        Error Word = Hex (FFFF)
        FOR Each byte in message
                Error Word = Error Word XOR byte in message
                FOR Each bit in byte
                        LSB = Error Word AND Hex (0001)
                        IF LSB = 1 THEN Error Word = Error Word – 1
                        Error Word = Error Word / 2
                        IF LSB = 1 THEN Error Word = Error Word XOR Hex (A001)
                NEXT bit in byte
        NEXT Byte in message
END
```

Esto se pondrá en la función `errorCheck()`, que traducido a código "C" quedará de la siguiente forma:

```
uint16_t errorCheck(uint8_t bytes[], int n){
  uint16_t errorWord = 0xFFFF;
  for(int i = 0; i < n; i ++){
    errorWord ^= bytes[i];
    for(int j = 0; j < 8; j++){
      uint8_t LSB = errorWord & 0x1;
      if(LSB == 0x1){
        errorWord -= 0x1;
      }
      errorWord = errorWord/2;
      if (LSB == 0x1){
        errorWord ^= 0xA001;
      }
    }
  }
  return errorWord;
}
```

Una vez obtenidos los dos bytes de error CRC, se guardan en la variable del mensaje para completarlo.

El siguiente paso es enviar el mensaje. Aquí es donde empiezan uno de los problemas con trabajar dentro de un sistema operativo que no funciona a tiempo real. Cuando se ejecuta la función `write()` de la librería estándar de Linux, que sirve para escribir bytes al puerto serie, el sistema operativo simplemente escribe los datos que se quieren enviar en el buffer del puerto serie. Esto no implica que el mensaje se haya enviado, solo que el sistema operativo lo enviará en el momento más próximo que pueda, ya que hay tareas del mismo sistema operativo que tienen prioridad. Entonces, si acto seguido de escribir el mensaje, se intenta leer del buffer, es posible que no encontremos nada ya que el mensaje ni se haya enviado.

Para mitigar esto se tiene que para le programa un cierto tiempo. Se tiene que tomar en cuenta el retardo de enviar el mensaje en sí y de recibir la respuesta; que, para una velocidad de 9600 baudios, serían un mínimo de 19,2 milisegundos (ver calculo abajo).

$$Mensaje: 8\ palabras \cdot \ 11 \frac{bits}{palabra} \cdot \frac{1}{9600\ baudios\left[\frac{bits}{segundo}\right]} = 0{,}00916 \text{ segundos}$$

$$\text{Respuesta: } 9\ palabras \cdot \ 11 \frac{bits}{palabra} \cdot \frac{1}{9600\ baudios\left[\frac{bits}{segundo}\right]} = 0{,}01031 \text{ segundos}$$

$$\boldsymbol{Total}:\ 0{,}01948\ segundos = \mathbf{19{,}2}\ \boldsymbol{mili\ segundos}$$

Si se mide en el osciloscopio (*figura 34*), se puede ver como ambos mensajes hacen los 19 milisegundos previamente calculados. Luego, la respuesta del SDM230 es casi inmediata; con un total de 26 milisegundos entre que empieza el mensaje y acaba la respuesta. Ya que no se requieren de velocidades excesivamente elevadas, se parará el programa durante 50 milisegundos para asegurar de que la respuesta esté en el buffer y se pueda leer.

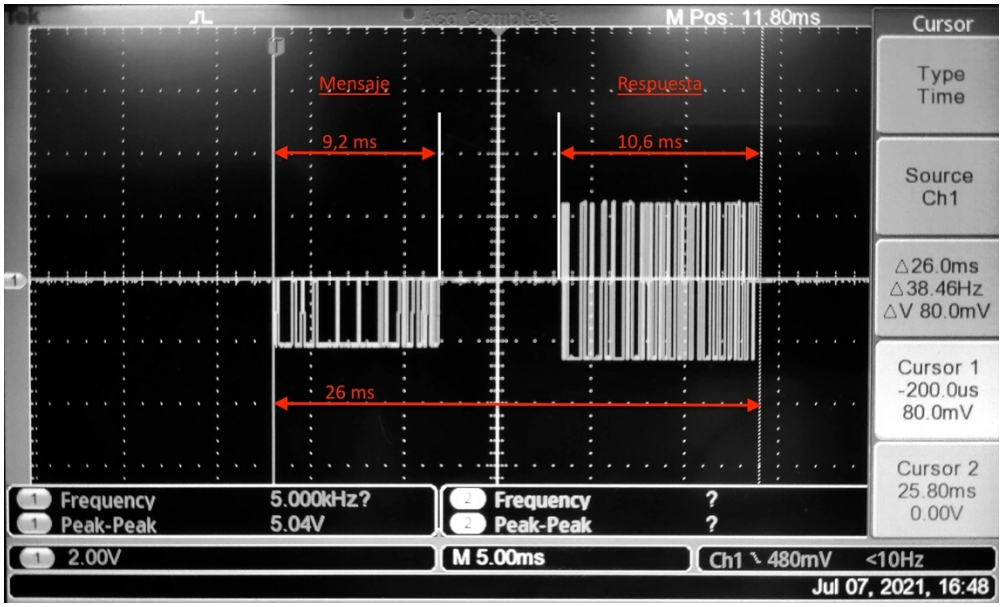
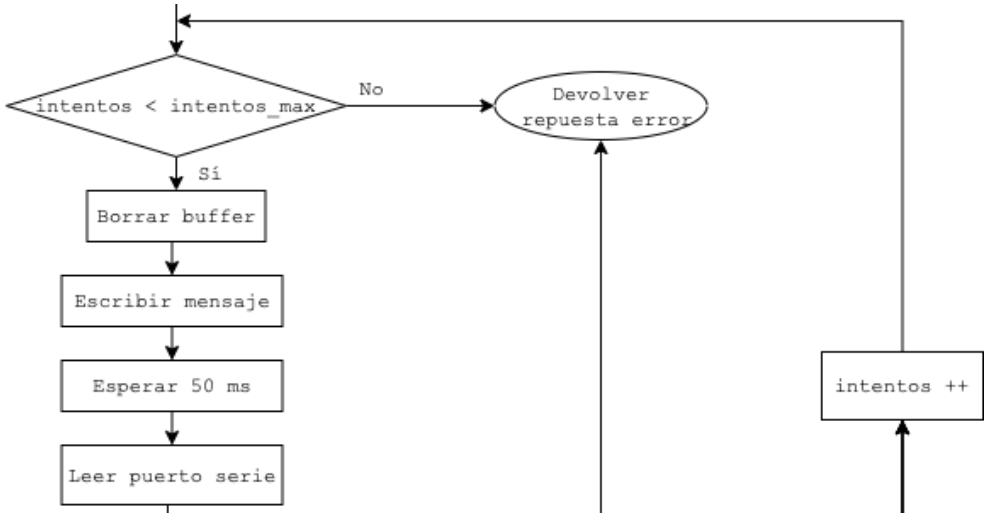


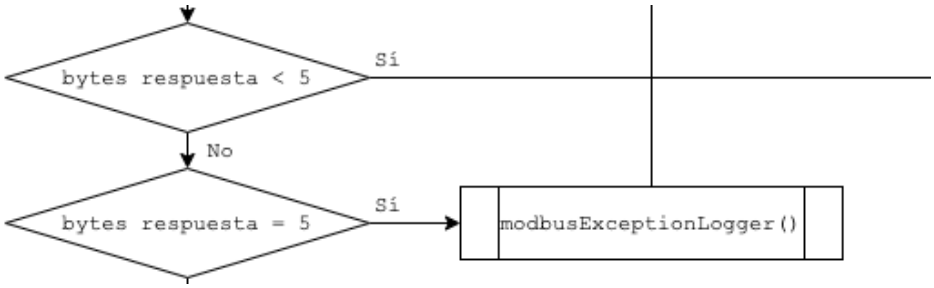
Figura 34: Imagen del osciloscopio del proceso mensaje-repuesta con los tiempos marcados en rojo.

Ya que es posible que ocurran errores de transmisión, se pondrán las próximas partes en un bucle. Cuando oscura un error en la interpretación o lectura se incrementarán los intentos (luego se verán los casos en los que ocurre esto). Cuando se llegue a un máximo, la función devolverá la respuesta error. También hay otro caso, que se verá luego, donde se envié esta respuesta error.

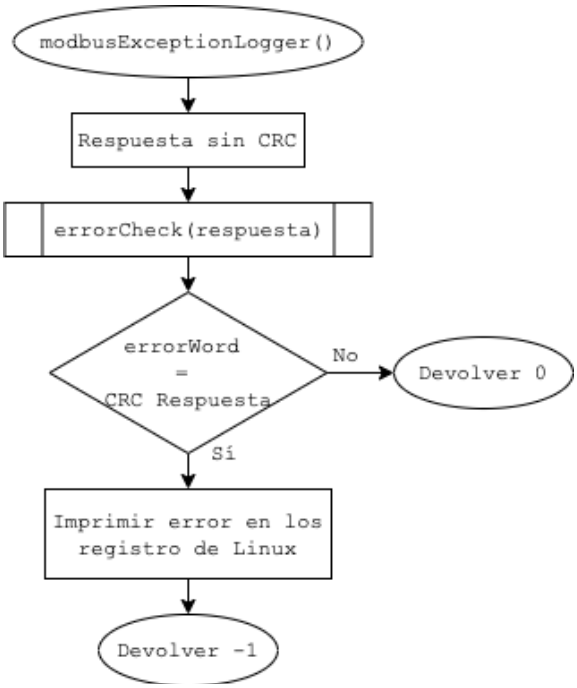


Una vez llamada la función de leer el puerto serie de la librería estándar de Linux, los bytes que se hayan acumulado serán guardados en un buffer temporal. Se detectarán cuantos bytes están allí, y dependiendo del resultado se hará una cosa u otra.

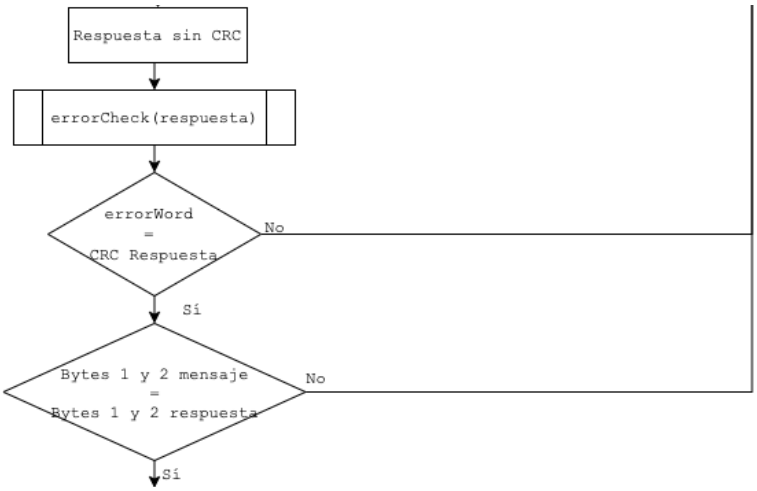
En el caso de tener menos de cinco bytes, se incrementarán los intentos y se volverá al principio de este bucle. Si el numero de bytes es exactamente cinco, eso significa que hubo un error de interpretación del lado del vatímetro. Por tanto, se llamara a al función `modbusExceptionLogger()` para que interprete la respuesta.



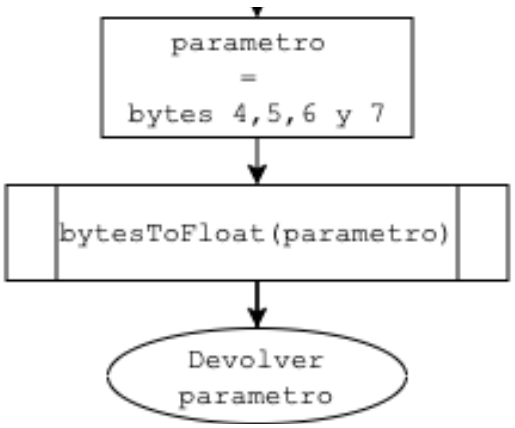
Si el mensaje de error de Modbus no es identificado o es incorrecto, la función `modbusExceptionLogger()` devolverá "*0*", que hará que incrementen los intentos y se vuelva al inicio del bucle. En caso de que sea correcto, se registrará en la Raspberry Pi para su revisión posterior y devolverá "*-1*". Esto es el otro caso que hace que la función devuelva la respuesta error.



Una vez el mensaje pase esos filtros, se considera como valido y se pasa a verificar su integridad revisando los bytes de error CRC, calculando dicho error con le función errorCheck() sin incluir los bytes de CRC del mensaje. Si no son iguales se incrementan los intentos y se vuelve al inicio del bucle. Si lo son se comprueba que los dos primeros bytes del mensaje sean iguales a los dos primeros de la respuesta. Si no lo son, se incrementan los intentos y se vuelve al inicio del bucle.



Cuando ya se haya comprobado que la respuesta recibida es correcta, se extraen los bytes del parámetro (que son el cuarto, quinto, sexto y séptimo). Antes de devolver el parámetro, se pasa a un numero de coma flotante (`float`), mediante la función `bytesToFloat()`, para que mejor interpretación en el resto del programa.



Con esto se termina la parte de comunicación serie e interpretación de mensajes Modbus. Ahora, lo único que tiene que hacer el programa principal es inicializar el puerto serie con la función `initializePort()` y cuando requiere leer un parámetro del vatímetro, tendrá que llamar a la función `modbusQuery()`, que devolverá directamente un `float`.

4.2 Comunicación MQTT

Esta parte es estrictamente software, ya que toda la comunicación se hace a través de protocolos por internet y servidores internos. Al igual que en la primera parte, se profundizará, levemente, en el protocolo MQTT que se utilizará para la comunicación entre la Raspberry Pi y el ordenador externo.

4.2.1 Protocolo MQTT

MQTT, del inglés "Message Queuing Telemetry Transport", es un protocolo ligero basado en el modelo de publicar-suscribir. Fue diseñado para conexiones remotas a dispositivos que no tienen mucha fuerza de procesamiento o donde el ancho de banda es escaso. Es uno de los protocolos más utilizados para aplicaciones IoT ya que, como se explicará, se puede conectar una red vasta de dispositivos sin interferencias entre ellos.

Para que funcione este protocolo, solo se necesitan 3 entidades, como se muestra en la abstracción de la *figura 35*:

- **Broker:** es el que tiene la mayor carga de procesamiento. Se encarga de recibir todas las publicaciones y enviárselas a los suscriptores correspondientes.
- **Publicador:** entidad que publica un mensaje a un canal.
- **Suscriptor:** entidad que recibe mensajes de un canal al que está suscrito.

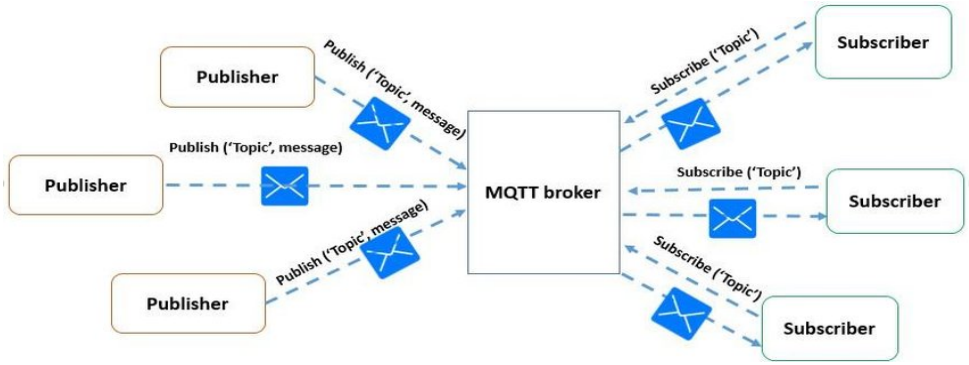


Figura 35: Imagen abstracta para funcionamiento de la comunicación MQTT.

Para entenderlo mejor: un Publicador se conecta al Broker y publica un mensaje a un canal. El Broker recibe el mensaje y verifica si hay alguna entidad suscrita a ese canal. Si lo hay, este mensaje se le envía al Suscriptor. Un Publicador puede enviar tantos mensajes como quiera a los canales que quiera una vez conectado al Servidor. Y un Suscriptor se puede suscribir a todos los canales que quiera.

Lo bueno de este sistema es que no es necesario que cada una de estas editadas esté vinculada a un dispositivo concreto. En el caso de este trabajo, la Raspberry Pi es Broker, Publicador y Suscriptor. Tanto a los Publicadores como a los Suscriptores, se les llama Clientes. Por lo que todas las conexiones deben ser entre un Broker y un Cliente. Por tanto, sólo puede existir un Broker en una red.

Para utilizar MQTT en la Raspberry Pi, se instalará un servidor llamado "Mosquitto". Para ello se inicia sesión mediante SSH y se introduce el siguiente comando:

```
$ sudo apt install -y mosquitto mosquitto-clients
```

Una vez instalado, se activa el servicio utilizando el siguiente comando, también hace que el servicio se active automáticamente cuando se encienda la Raspberry Pi:

```
$ sudo systemctl enable mosquitto.service
```

Con esto instalado y funcionando, se puede comprobar su funcionamiento tanto internamente, como desde un ordenador externo con los comandos para publicar y suscribirse a un canal. En este caso al canal `test/`. En las *figuras 36 y 37* se muestra el funcionamiento de esta comunicación publicador-suscriptor.

```
$ mosquitto_sub -h localhost -t test/
$ mosquitto_pub -h localhost -t test/ -m "hello world"
```

Figura 36: Ejemplo de la Raspberry Pi publicando un mensaje al canal "test/".

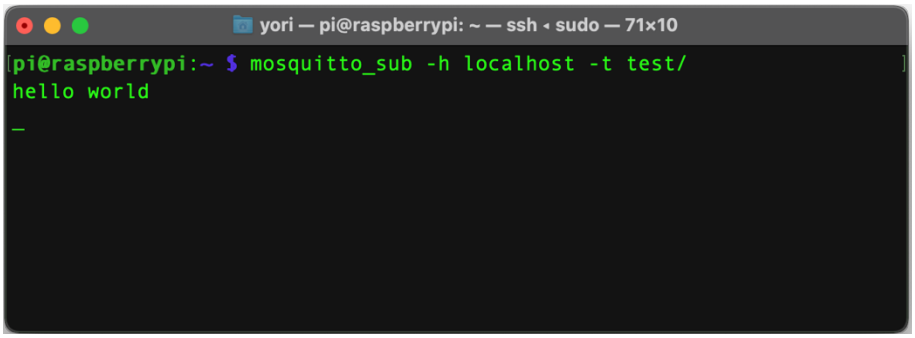


Figura 37: Ejemplo de la Raspberry Pi suscribiéndose al canal "test/" y recibiendo el mensaje publicado.

En el caso de querer comprobar desde un dispositivo externo, se sustituye "localhost" por la dirección IP de la Raspberry Pi. Que se puede averiguar con el siguiente comando:

```
$ hostname -I
```

En la *figura 38* se muestra el uso del comando e indica cual es la dirección IP del dispositivo.

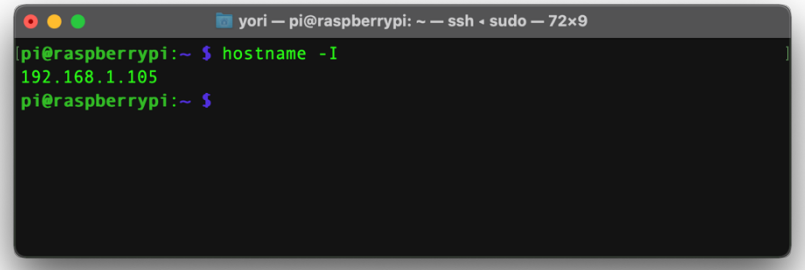


Figura 38: Ejemplo de como averiguar la dirección IP desde la Raspberry Pi.

4.2.2 Código

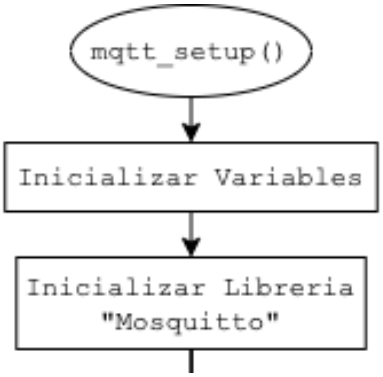
Esta parte es mucho menos extensa y complicada que la anterior porque la instalación de *"Mosquitto"* incluye una librería en C con funciones ya hechas. Lo único que se hará es inicializar y configurarla.

Otro dato importante es que, a la hora de compilar este programa para la Raspberry Pi, se tiene que incluir un argumento nuevo en el comando `gcc` para incluir la librería de *"Mosquitto"* instalada:

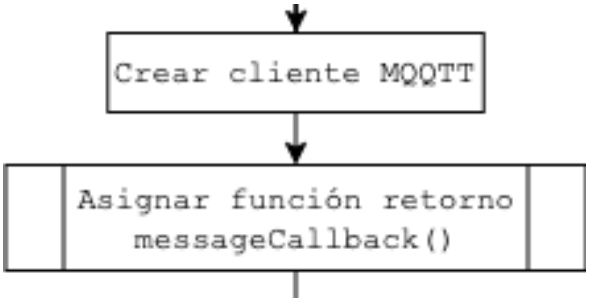
```
$ gcc *.c -l mosquitto -o modbus_mqtt.exec
```

Lo primero es configurar el cliente MQTT. Para ello se creará una función de configuración que se pueda ejecutar al inicio de el programa principal que no requiere ni devuelve nada.

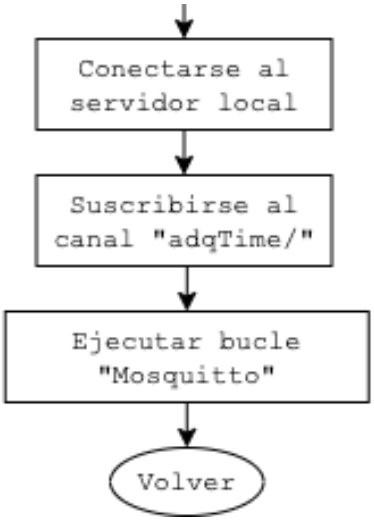
Lo primeo es inicializar las variables necesarias para el funcionamiento de esta librería. Luego, para esta librería en concreto, es necesario inicializarla, como se indica en la documentación de dicha librería.



Luego se debe crear el objeto que representa la conexión con el servidor MQTT. Esta variable contiene la configuración, que se tendrá que modificar para cumplir con las necesidades de este trabajo. También se le tendrá que asignar una función de retorno, que se ejecutará cuando llegue un mensaje de los canales a los que se han suscrito.

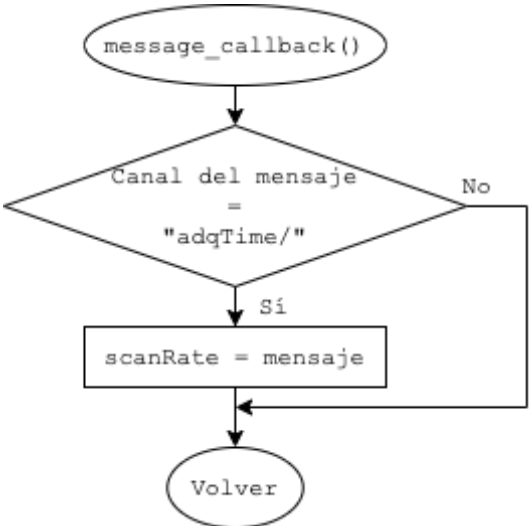


Una vez hecha toda la configuración, se conecta al servidor. Que en este caso será el servidor local, ya que la Raspberry Pi hace la tarea de Broker como se comento en el apartado anterior. También se suscribirá al canal "`adqTime/`" donde se encontrará el tiempo de adquisición para el programa principal y se ejecutara el bucle *"Mosquitto"* que es necesario para el funcionamiento de esta librería.



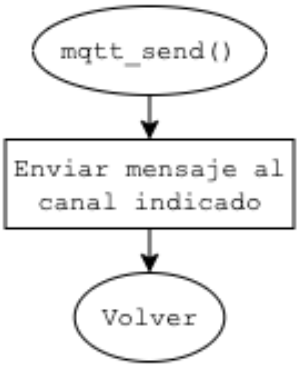
Otra parte importante de este fichero es la función `messageCallback()`, que es la que se ejecuta cuando llega un mensaje al canal suscrito. Como es el caso de este fichero, que esta suscrito al canal "`adqTime/`", es necesario implementar esta función.

El objetivo de suscribirse a ese canal es para poder modificar el tiempo de muestreo desde la aplicación externa. Por tanto, llegarán mensajes desde fuera desde la Raspberry Pi, y es necesario interpretarlos. Primero se asegurará que el canal del que llega el mensaje es al que se esta suscrito ("`adqTime/`"), si no es así, la función termina y no hace nada. Si es correcto, se modificará la variable `scanRate` con el contenido del mensaje.

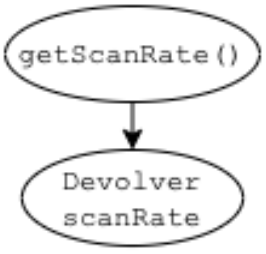


También, a este fichero se le han añadido un par de funciones que facilitaran el uso del servidor MQTT.

La primera es para enviar mensajes a un canal determinado. La librería ya incluye dicha función, pero tiene muchos parámetros que no son muy relevantes para el uso que se le dará. Por tanto, se simplificará para solo aceptar dos parámetros; el mensaje que se quiere enviar y el canal al que se quiere enviar.



La otra función simplemente es para que la función principal pueda acceder a la variable scanRate. Es posible hacerla una variable global, pero de esta manera se asegura que esta variable solo puede ser modificada por el fichero de comunicaciones MQTT.



4.3 Ejecutable de la Raspberry Pi

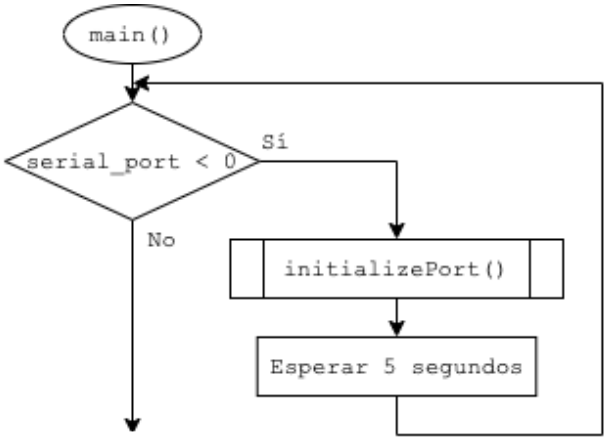
El programa principal es el que recopila todas las funciones creadas en los dos apartados anteriores y donde se ejecutarán las instrucciones en sí; a excepción de la función `message_callback()` que se ejecutará cuando llegue un mensaje, una vez inicializado el cliente MQTT.

También es el lugar donde se controlará lo qué ocurrirá con esta aplicación una vez se esté ejecutando en la Raspberry Pi.

4.3.1 Código

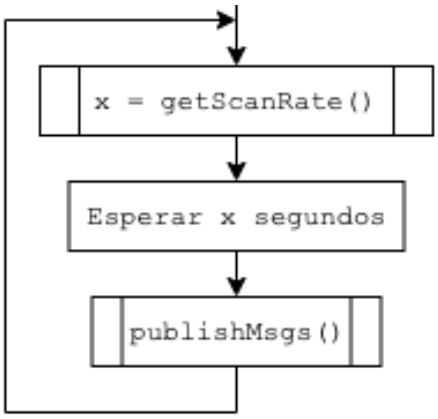
Se empezará describiendo la función `main()`. Esta es la función que el compilador entenderá como la función principal y la primera que se deba ejecutar cuando se inicie el programa.

Dentro de esta función, lo primero que se hará es inicializar el puerto serie con la función `initializePort()` del fichero de comunicaciones Modbus. Esta se meterá dentro de un bucle infinito. Por tanto, hasta que el puerto serie no este inicializado el programa seguirá intentándolo. Se ha puesto un retardo de 5 segundos entre cada intento para no saturar el sistema.

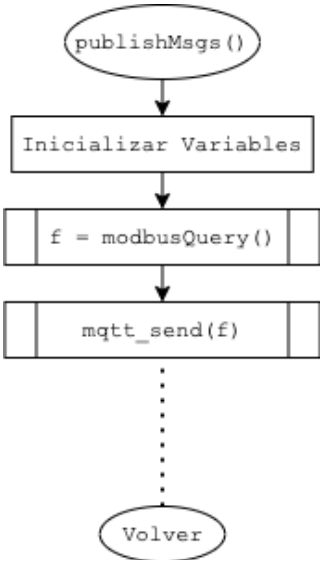


Si el puerto se inicializa correctamente se pasará a inicializar el cliente MQTT.

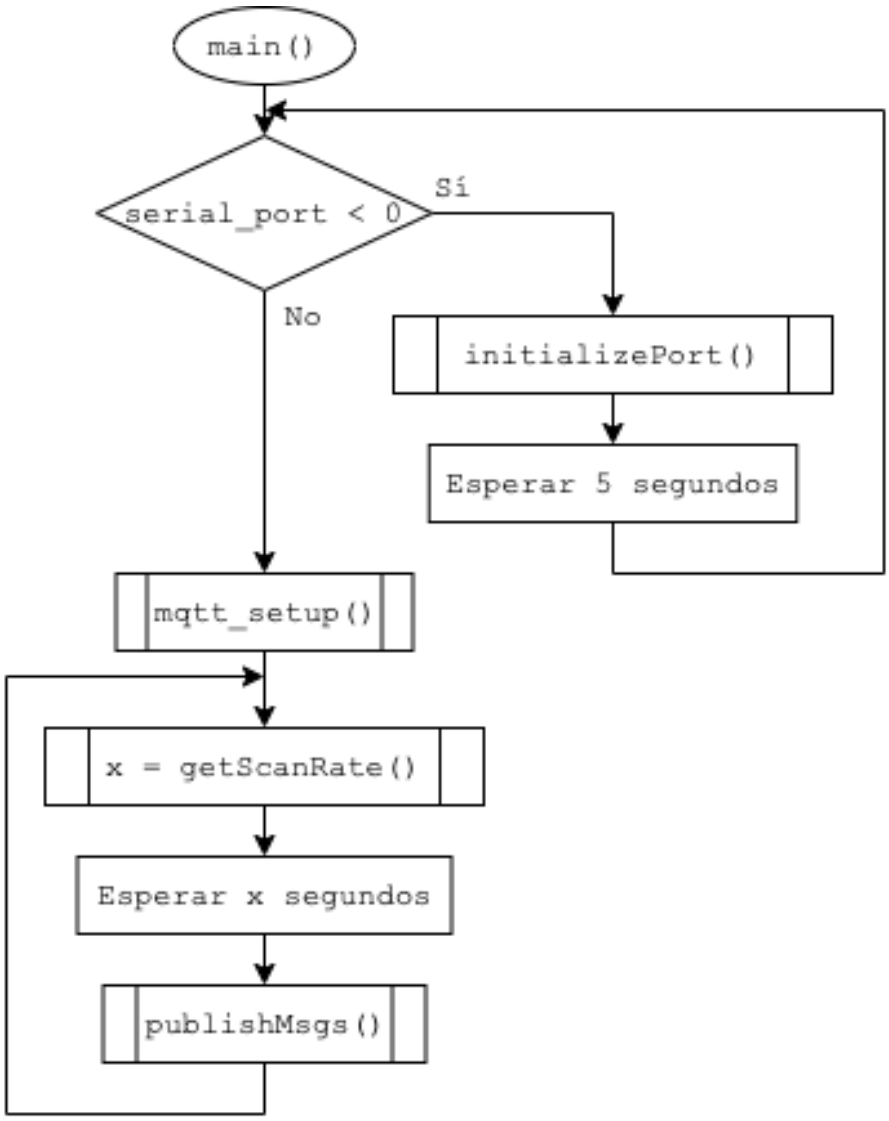
Con el cliente MQTT activo y el puerto serie inicializado, se puede pasar al bucle infinito principal, donde se leerán los parámetros del vatímetro mediante la función `modsbusQuery()`, y se publicarán los mensajes a sus respectivos canales mediante la función `mqtt_send()`. Este bucle tendrá un retardo que será determinado por la aplicación externa. Este valor esta guardado en el fichero de comunicaciones MQTT, por tanto se utilizará la función `getScanRate()` para adquirirlo.



La parte de coger parámetros del vatímetro y publicarlos al servidor MQTT se separo a una función aparte para mejorar la visibilidad del programa. Esta función es una repetición del mismo formato: leer el parámetro con la función `modbusQuery()` y publicarlo al canal correspondiente con la función `mqtt_send()`.



A continuación, se muestra el diagrama completo de la función principal para ilustrar en un solo diagrama como esta todo conectado.



4.3.2 Ejecutable

Para compilar este programa, se tiene que copiar todos los archivos .c y .h creados a un mismo directorio de la Raspberry Pi con el comando:

```
$ scp *.c *.h pi@raspberrypi.local:~/modbus/
```

Luego, se conecta a la Raspberry Pi con SSH y se ejecutan los dos comandos siguientes, uno para moverse al directorio `~/modbus` y el siguiente para compilar los archivos copiados con `gcc`:

```
$ cd ~/modbus/
$ gcc *.c -l mosquitto -o modbus_mqtt.exec
```

Esto creará el ejecutable `modbus_mqtt.exec` si el compilador no encontró ningún error. Mediante el comando `ls` se podrá ver si efectivamente se encuentra en el directorio. Y para ejecutar el programa se introduce el comando siguiente:

```
$ ./modbus_mqtt.exec
```

En la *figura 39* se muestra la compilación y se muestra el archivo ejecutable creado.

Figura 39: Ejemplo de compilación de todos los archivos en el área de trabajo.

Para no tener que conectarse a la Raspberry Pi para ejecutar el programa, se tiene que modificar un archivo de configuración que permitirá ejecutar el programa automáticamente cuando se enciende la Raspberry Pi.

El archivo que se tiene que modificar es uno llamado `.bashrc`. Es un archivo oculto y con permisos restringidos. Para editarlo se tiene que introducir el siguiente comando y dar la contraseña de usuario:

```
$ sudo nano /home/pi/.bashrc
```

Este documento tiene muchos datos de configuración. Para no interrumpir ninguno de esos procesos, se escribirá `./modbus/modbus_mqtt.exec` al final del documento y se guardará. En la *figura 40* se muestra el archivo con la modificación que se debe hacer.

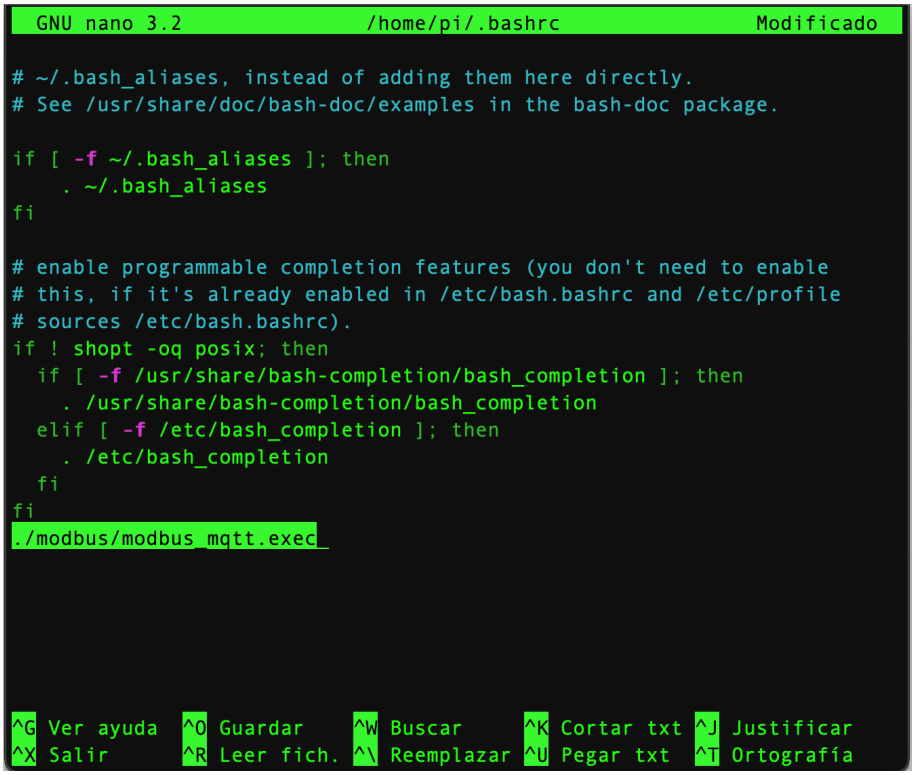


Figura 40: Archivo ".bashrc" visto desde la terminal con el comando añadido al final.

Para guardar el archivo se pulsa la combinación de teclas "Crtl+O" y luego Enter para confirmar.

Esta configuración se hará efectiva cuando se inicie sesión en el usuario "pi", que ocurrirá después de toda la inicialización básica del sistema operativo. Esto es deseable, ya que así se asegura de que, tanto el servidor "Mosquitto" como los puertos USB, están activos.

4.4 Interfaz Gráfica de Usuario (GUI)

Como se comentó en el estudio de alternativas, para la interfaz de usuario se utilizará Java.

Esta interfaz se ejecutará en los ordenadores externos que quieran diagnosticar los parámetros eléctricos del vatímetro SDM230. Para ello, primero aparecerá una ventana para conectarse al servidor MQTT de la Raspberry Pi. Una vez conectado, aparecerá la ventana con todos los parámetros a simple vista que se irán actualizando conforme la Raspberry Pi vaya publicando los valores. Se tendrá la opción de cambiar este tiempo de adquisición dentro del menú principal.

En esta parte no se comentará el código detalladamente porque gran parte de la programación en Java es muy repetitiva. El código se basa mucho en modificar objetos para que se vean correctamente en la pantalla y no aportan nada sustancial al entendimiento de la aplicación. Funciones que sí son relevantes, se comentarán en más en detalle.

Para todas las ventanas, elementos gráficos y objetos interactivos, se utilizaran las librerías estándar de Java: `java.swing`, `java.awt` y `java.beans`.

4.4.1 Ventana de Conexión

Esta es la primera ventana que salta al usuario cuando primero se abre la aplicación. Consta de un texto instruyendo al usuario, un cuadro de texto donde introducir la dirección IP y un botón para realizar la conexión, como se muestra en la *figura 41*.

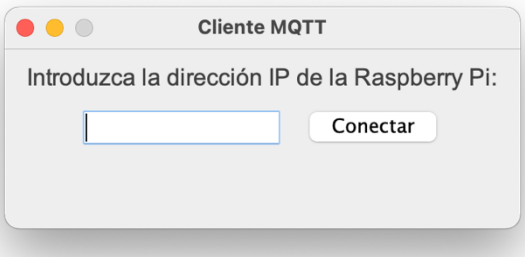


Figura 41: Imagen de la ventana de conexión.

Una vez se introduce la dirección y se pulsa el botón o Enter, se ejecuta el método implementado de la interfaz `actionListener`:

```
  @Override
  public void actionPerformed(ActionEvent e) {
    if(e.getSource() == tfIP || e.getSource() == buttonIP) {
      frameIP.setEnabled(false);
      frameIP.setTitle("Connectando...");
      Main.setPiIP(tfIP.getText().replace(" ", ""));
      try {
        Main.getMqttSub().connectClient(Main.getPiIP());
        frameIP.dispose();
        updateScreen();
        Main.getMqttSub().sendMessage(String.valueOf(Main.getAdqTime()),
"adqTime/");
      } catch (MqttException e1) {
        labelErr.setVisible(true);
        e1.printStackTrace();
        frameIP.setEnabled(true);
        frameIP.setTitle("Cliente MQTT");
      }
    }
  }
```

Este método detecta cuando se recibe una acción y ejecuta el código que se le indique. En este caso, lo primero que hace es cambiar la ventana para indicar que se está conectando (mostrado en la *figura 42*) al servidor, ya que es un proceso que puede llegar a tardar unos segundos. Luego se ejecutan los métodos de conectar al Servidor y enviar un mensaje con el tiempo de adquisición inicial al canal `adqTime/`, para sincronizar la aplicación con el servidor.

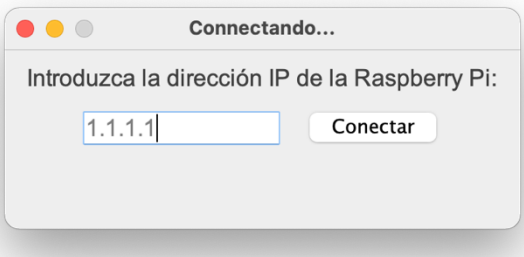


Figura 42: Imagen de la ventana de conexión cuando de esta conectado.

Estos dos métodos están dentro de un `try{}catch{}`; si se detecta algún error en la ejecución de alguno de los dos métodos, la aplicación ejecutará lo que está dentro del `catch{}`. Si no, lo ignorará.

En el caso de que todo sea correcto, esta ventana se cerrará y se abrirá la pantalla principal. El proceso de cómo se abre la pantalla principal se explicará en la parte de control.

Si alguno de los métodos da error, la ventana se actualizará y mostrará un texto indicando que no se pudo conectar al servidor (mostrado en la *figura 43*), donde el usuario tendrá que volver a introducir los datos.

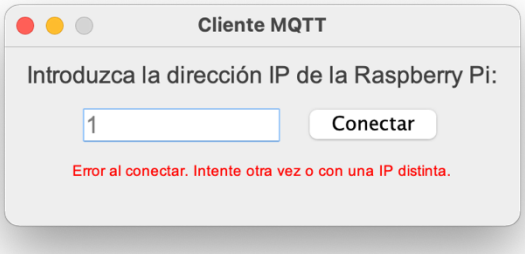


Figura 43: Imagen de la ventana de conexión cunado hubo un error.

4.4.2 Ventana Principal

La ventana principal es donde el usuario podrá visualizar todos los parámetros eléctricos de la instalación. Se encuentran en el centro, con una columna de parámetros a la izquierda y un par de gráficas a la derecha. En la parte inferior se podrá ver a qué dispositivo está conectado y cuál es el tiempo de adquisición. Y en la parte superior se encuentra el menú. En la *figura 44* se muestra dicha interfaz.

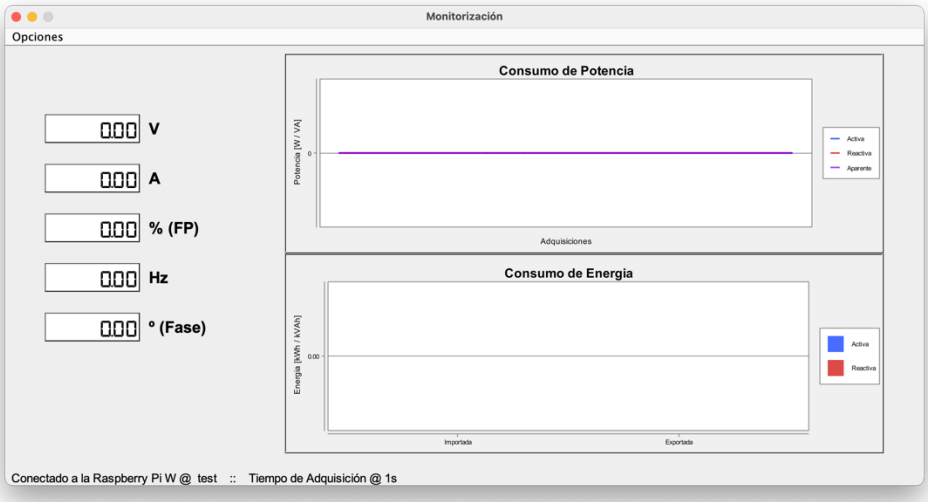


Figura 44: Imagen de la ventana principal.

En el menú se podrá configurar el tiempo de adquisición, reconectarse al mismo u otro dispositivo, o salir de la aplicación, como se muestra en la *figura 45*

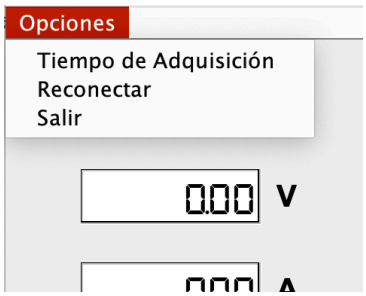


Figura 45: Ampliación de el menú superior de la ventana principal.

Esta ventana también contiene gráficas que están hechas con las librería `org.knowm.xgraph`, incluidas en el repositorio de *Maven*. Su construcción y formato están descritas en la documentación, donde además se tienen varios ejemplos.

En la figura 46 se muestra indicados las distintas partes de la interfaz grafica. Los parámetros de la izquierda simplemente representan el valor actual del voltaje, corriente, factor de potencia, fase y frecuencia; en ese orden, de arriba a abajo. Estos se van actualizando conforme lleguen los datos del servidor MQTT. La gráfica superior muestra el cambio, a tiempo real, de los tres tipos de potencia; activa (azul), reactiva (rojo) y aparente (morado). La gráfica de abajo muestra la energía consumida (importada) o generada (exportada). También con ambos tipos de energía; activa (azul) y reactiva (rojo). Y en la parte inferior se puede ver la dirección IP del dispositivo y el tiempo de adquisición, que se puede ajustar en uno de los menús.

Figura 46: Imagen indicando los paneles de la ventana principal.

Como se mencionó anteriormente, en el menú se encuentran tres opciones a las que el usuario puede acceder. La primera, "Tiempo de Adquisición", que redirige al usuario a otra ventana (mostrada en la *figura 47*) donde se puede cambiar esta variable. Esta ventana incluye un deslizador que representa el tiempo de adquisición. Abajo hay un cuadro de texto, que indica el valor en el que está el deslizador, y un botón que aplica los cambios. También se puede introducir los segundos deseados en el cuadro de texto y pulsar Enter para actualizar el deslizador. Al final, el valor que actualizará la variable del tiempo de adquisición es la del deslizador.

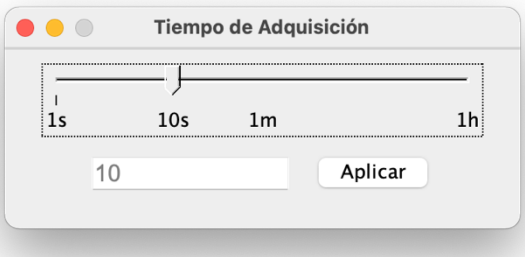


Figura 47: Imagen de la ventana para cambiar el tiempo de adquisición.

Cuando se pulsa el botón para aplicar los cambios, se ejecuta el método de `actionPerformed()`; similar a la ventana de conexión. En este método se envía un mensaje por MQTT con la variable del tiempo de adquisición, que será recogida por la Raspberry Pi.

```
   @Override
   public void actionPerformed(ActionEvent e) {
     if(e.getSource() == adqButton) {
       Main.setAdqTime(adqSlider.getValue()*1000);
       try {
         Main.getMqttSub().sendMessage(String.valueOf(Main.getAdqTime()),
"adqTime/");
         dqFrame.dispose();
         updateParameters();
       } catch (MqttException e1) {
         adqFrame.setTitle("Vuelva a Intentarlo");
         e1.printStackTrace();
       }
     }
   }
```

La segunda opción en el menú, "Reconectar", cierra la ventana principal y devuelve al usuario a la ventana de conexión. En esta ventana, el usuario tiene la opción de volverse a conectar al mismo servidor, ya que la dirección IP reciente estará escrita en el cuadro de texto, o escribir una IP nueva y conectarse a un servidor distinto.

La última opción es la de "Salir". Que, como el nombre indica, simplemente cierra la aplicación entera; lo mismo que haría si se pulsara la cruz en la parte superior de la ventana.

Para implementar las acciones de los botones y opciones de menú, se implementa el método de la interfaz `actionListener`; igual que en la ventana de conexión.

Además, esta clase también tiene dos métodos que actualizan todos los elementos gráficos de la ventana: `updatePrameters()` y `updateGraphs()`. Estos métodos se utilizarán en la parte de control del servidor MQTT, cuando le lleguen los datos publicados.

4.4.3 Control

El control se divide en dos partes: el control del servidor MQTT y el control de la interfaz gráfica.

4.4.3.1 Clase MqttSubscriber

El control del servidor MQTT lo maneja la clase `MqttSubscriber`. Una vez creado el objeto, el primer método que se tiene que llamar es el `connectClient()`, para conectarse al servidor.

```
public void connectClient(String hostIP) throws MqttException {
  client = new MqttClient("tcp://" + hostIP, "JavaApp");
  System.out.println("Setting up");
  connOpt = new MqttConnectOptions();
  connOpt.setCleanSession(true);
  connOpt.setConnectionTimeout(2);
  client.connect(connOpt);
  System.out.println("Connected");
  client.setCallback(this);
}
```

Al igual que en C, Java tiene su propia librería para conectarse a clientes "Mosquitto" de MQTT; incluida en el repositorio *Maven*. En la documentación de esta, se pueden encontrar los métodos necesarios para configurar y conectarse a un servidor.

Los dos métodos siguientes manejan las suscripciones y el envío de mensajes:

```
public void  subscribeToTopic(String t) throws MqttException {
  client.subscribe(t);
}

public void sendMessage(String msg, String topic)
                        throws MqttPersistenceException, MqttException
{
  MqttMessage message = new MqttMessage(msg.getBytes());
  message.setQos(0);
  client.publish(topic, message);
}
```

Está implementada la interfaz "`MqttCallback`", que permitirá añadir los métodos que se ejecutarán cuando se reciba un mensaje de los canales suscritos. El más relevante para este trabajo es el de mensajes recibidos `messageArrived()`. Lo primero que se hace en este método es verificar que el mensaje sea válido; se ignora el resto del método. Después se actualizan las variables internas en función del canal del mensaje. En el código de abajo se simplificó con un solo canal; por tanto, si llega un mensaje del canal `parameters/voltage` sólo se actualizará la variable que almacena el voltaje `Voltage`. Así con el resto de los canales y sus respectivas variables. Al final se llama al método de actualizar la interfaz gráfica `updateUI()`:

```
   @Override
   public void messageArrived(String topic, MqttMessage message) throws Exception
{
     if(message.toString().contains("n") || message.toString().contains("N"))
       return;
     if(topic.contains("parameters/voltage"))
       Main.setVoltage(Double.parseDouble(message.toString()));
     [...]
     Main.getUI().updateUI();
```

4.4.3.2 Clase UserInterface

Para controlar la interfaz de usuario está la clase `UserInterface`. Esta clase crea los objetos correspondientes a las ventanas de conexión y la principal. Cuando se crea un objeto de esta clase, automáticamente se llama al método `startScreen()`:

Este método se encarga de crear la ventana de conexión. Como se comentó, para que esta clase pueda saber cuando cambiar de ventana, se implementa la interfaz de `PropertyChangeListener`. Por lo que, cuando el objeto de la ventana de conexión dé la señal, esta clase ejecutará el método `propertyChange()`, llamando al método `mainScreen()`:

```
static StartScreen ss;

private void startScreen() {
  ss = new StartScreen();
  ss.addPropertyChangeListener(this);
}

public void propertyChange(PropertyChangeEvent evt) {
  if(evt.getSource() == ss) {
    mainScreen();
  }
  if(evt.getSource() == ms) {
    startScreen();
  }
}
```

Cuando esto ocurra, el método creará el objeto de la ventana principal e iniciará las suscripciones a los canales del servidor MQTT:

```
static MainScreen ms;
private void mainScreen() {
  ms = new MainScreen();
  try {
    Main.getMqttSub().subscribeToTopic("parameters/#");
    Main.getMqttSub().subscribeToTopic("energy/#");
    Main.getMqttSub().subscribeToTopic("power/#");
  } catch (MqttException e) {
    e.printStackTrace();
  }
  ms.addPropertyChangeListener(this);
}
```

Este objeto de la ventana principal también tiene implementado `PropertyChangeListener`, por lo que cuando le llegue la señal, esta ventana se cerrará y se volverá a abrir la ventana de conexión.

Por último, está el método que actualiza la interfaz de la ventana principal. Este es el que se llama de la clase `MqttSubscriber` cuando llega un mensaje nuevo:

```
public void updateUI() {
  ms.updateGraphs();
  ms.updateParameters();
}
```

4.4.3.3 Clase Main

Esta clase, aunque no sea estrictamente de control, es la más importante para que la aplicación se ejecute. Es la que contiene el método `main()` que, al igual que C, es el primer método que se ejecuta cuando se abre la aplicación. También es el lugar donde se almacenan todas las variables globales utilizadas.

El método `main()` solo consiste en crear los dos objetos de las clases de control `MqttSubscriber` y `UserInterface`.

```
static MqttSubscriber mqttSub;
static UserInterface UI;
public static void main(String[] args) {
  mqttSub = new MqttSubscriber();
  UI = new UserInterface();
}
```

4.5 Desarrollo Experimental

Esta es la última fase del proyecto, donde se verificará que todas las partes funcionan correctamente y son compatibles entre ellas.

Para realizarlo, se construirá un circuito eléctrico simple, que representará la instalación que se quiere medir. Este circuito tendrá una carga activa y otra reactiva; la activa consiste en una resistencia de potencia a 100 ohmios. Y la carga reactiva de tres inductores de alterna en serie, para un total de 9 mili henrios. En la *figura 48* se muestra el esquema de esta carga, y en la *figura 49* una fotografía del sistema.

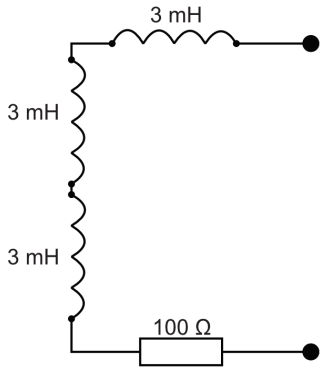


Figura 48: Esquemático de la carga eléctrica que representa la instalación.

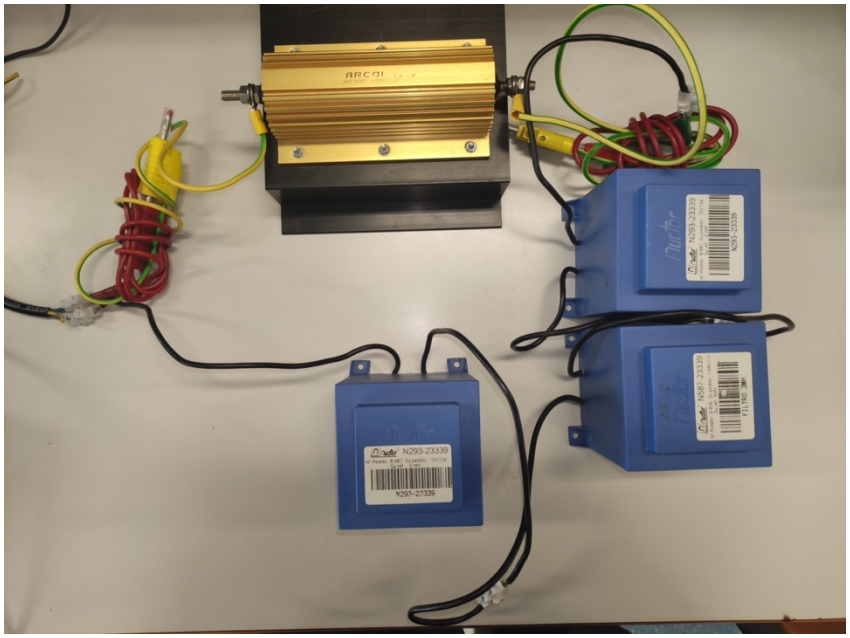


Figura 49: Imagen del circuito experimental montado en el laboratorio.

Luego se conectará el SDM230. Este tiene que estar conectado entre la carga y la entrada de la red eléctrica. En la hoja de datos se puede ver como se tiene que instalar (parte relevante mostrada en la *figura 50*).

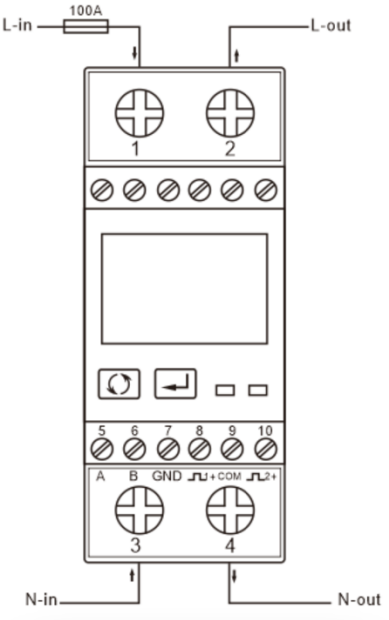


Figura 50: Diagrama de instalación del vatímetro SDM230.

El SDM230 y la Raspberry PI se conectan con el adaptador RS-485 a USB, como se explicó en el primer apartado. Una vez se conecte la Raspberry Pi a una fuente de alimentación, el programa se ejecutará automáticamente y los datos se empezarán a actualizar en el servidor MQTT. En al *figura 51* se muestra una abstracción de las conexiones que se deba hacer.

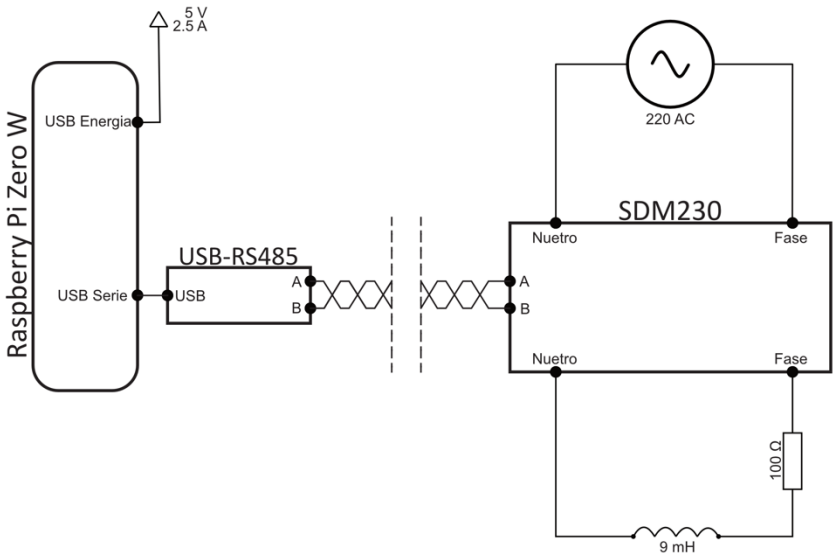


Figura 51: Esquemático de todas las conexiones necesarias.

Antes de abrir la aplicación Java en el ordenador externo, se tiene que saber la dirección IP de la Raspberry Pi. La manera más sencilla es utilizando el comando `arp` en la terminal, como se demuestra en la *figura 52*. La dirección MAC de todas las Raspberry Pi empiezan por "b8:27:eb..". Por tanto, añadiendo el comando `grep`, se puede encontrar más fácilmente la Raspberry Pi en la red. El comando completo se quedaría así:

```
$ sudo arp -a | grep "b8:27:eb"
```

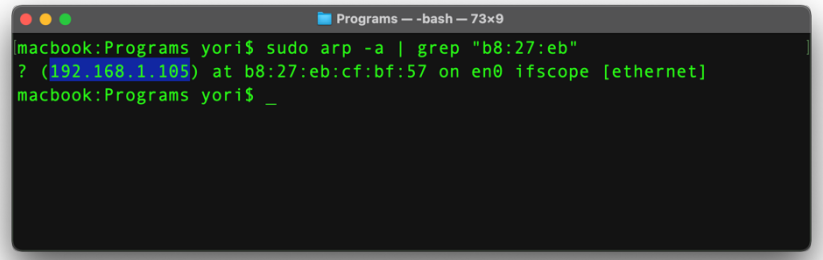


Figura 52: Ejemplo del comando "arp" para averiguar la dirección IP de la Raspberry Pi.

Sabiendo la dirección IP de la Raspberry Pi, se puede abrir la aplicación Java, mostrado en la *figura 53*, e introducir esta dirección en el cuadro de texto.

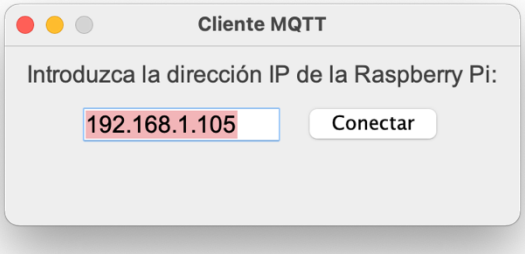


Figura 53: Imagen de la ventana de conexión con la dirección IP de la Raspberry Pi.

Una vez conectado, se pasará a la ventana principal y se verán todos los datos provenientes del SDM230, como se muestra en la *figura 54*.

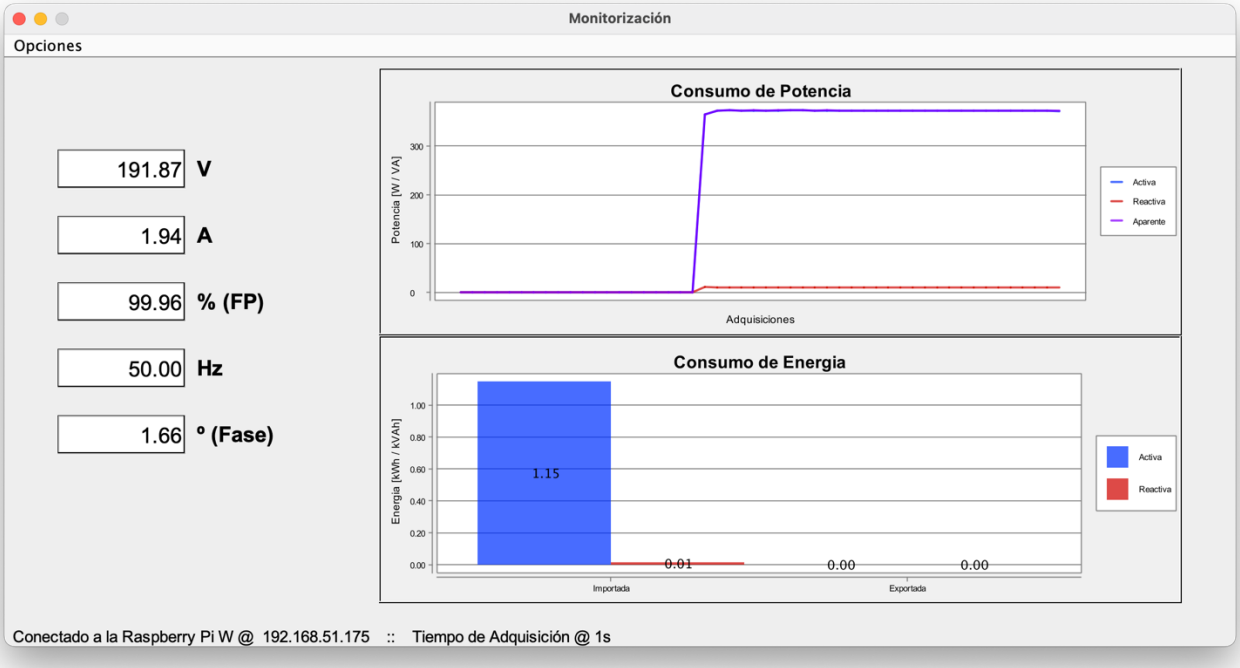


Figura 54: Imagen de la ventana principal en funcionamiento.

Para asegurarse de que el sistema esta funcionando correctamente, se medirán los parámetros eléctricos a los que están conectados la carga, se calcularan el resto de los parámetros (principalmente las potencias) y se contrastaran con los obtenidos en la aplicación.

Lo primero que podemos comprobar que es correcto es la frecuencia de 50 Hercios. En la hoja de datos del SDM230 se puede comprobar que tiene una precisión 0.2% de la frecuencia media. Que, con una lectura de 50 hercios, es exactamente lo que se espera.

Entrando en la teoría, sabiendo que tenemos una carga inductiva, se puede sacar el factor de potencia y la fase:

$$R = 100\,\Omega, \qquad L = 9\,mH = 0'009\,H, \qquad f = 50\,Hz$$

$$Z = \sqrt{R^2 + {X_L}^2} = \sqrt{R^2 + (\omega \cdot L)^2} = \sqrt{R^2 + (2 \cdot \pi \cdot f \cdot L)^2} = 100'034\,\Omega$$

$$PF = \frac{R}{Z} = 0'9725 = \mathbf{99'96\%}, \qquad \phi = \arccos(PF) = \mathbf{1'62º}$$

Continuado, la carga esta conectada a un transformador que proporciona unos 190 voltios (según medidas del osciloscopio son 192 voltios, como indica la interfaz gráfica). Sabiendo la impedancia del circuito, calculada anteriormente, se puede calcular la corriente:

$$I = \frac{V}{Z} = \mathbf{1'92\ A}$$

La corriente se sale un poco del rango de medida. Pero, tomando en cuenta que el vatímetro tiene una precisión nominal de 0.5% del valor nominal (que corresponderían a 0,025 amperios), este valor entra dentro del rango. Tomando el valor que recoge le vatímetro como la corriente, se calculan las potencias activa, reactiva y aparente:

$$P = V \cdot I \cdot cos(\phi) = \mathbf{376'06\ W}$$
$$Q = V \cdot I \cdot sen(\phi) = \mathbf{10'63\ VAr}$$
$$S = V \cdot I = \mathbf{376'21\ VA}$$

Aunque sea un poco difícil de visualizar el valor exacto en las graficas de la interfaz, si que se puede observar que los valores de la grafica apuntan a los valores que se esperaban.

Con esto se puede concluir que el vatímetro tiene un error por su precisión, pero los valores que devuelve están de acuerdo con lo que se espera dentro de esos rangos de error de precisión.

5 Conclusiones

El presente trabajo de fin de grado fue capaz de cumplir con los objetivos propuestos de modernizar el vatímetro SDM230 a un dispositivo IoT y utilizar este nuevo sistema visualizando los parámetros eléctricos en otro ordenador conectado a la red mediante una aplicación grafica.

Este trabajo fue, más que nada, una prueba de concepto para comprobar la viabilidad de integrar este sensor a la red de IoT. Con un poco más de configuración, podría ser posible conectar el dispositivo a internet y poder acceder a él remotamente desde cualquier parte del mundo, no necesariamente limitado a dispositivos de la misma red local.

El mundo está tendiendo a tener todo conectado a internet ya que facilita la automatización y el control de muchos procesos, tanto en la industria, como en muchos otros sectores. Aún estamos en el inicio de esta innovación tecnológica, por lo que no es común encontrar este tipo de sensores que ya tengan la tecnología IoT integrada. Y aún quedan décadas hasta que se modernicen muchas de las instalaciones ya construidas.

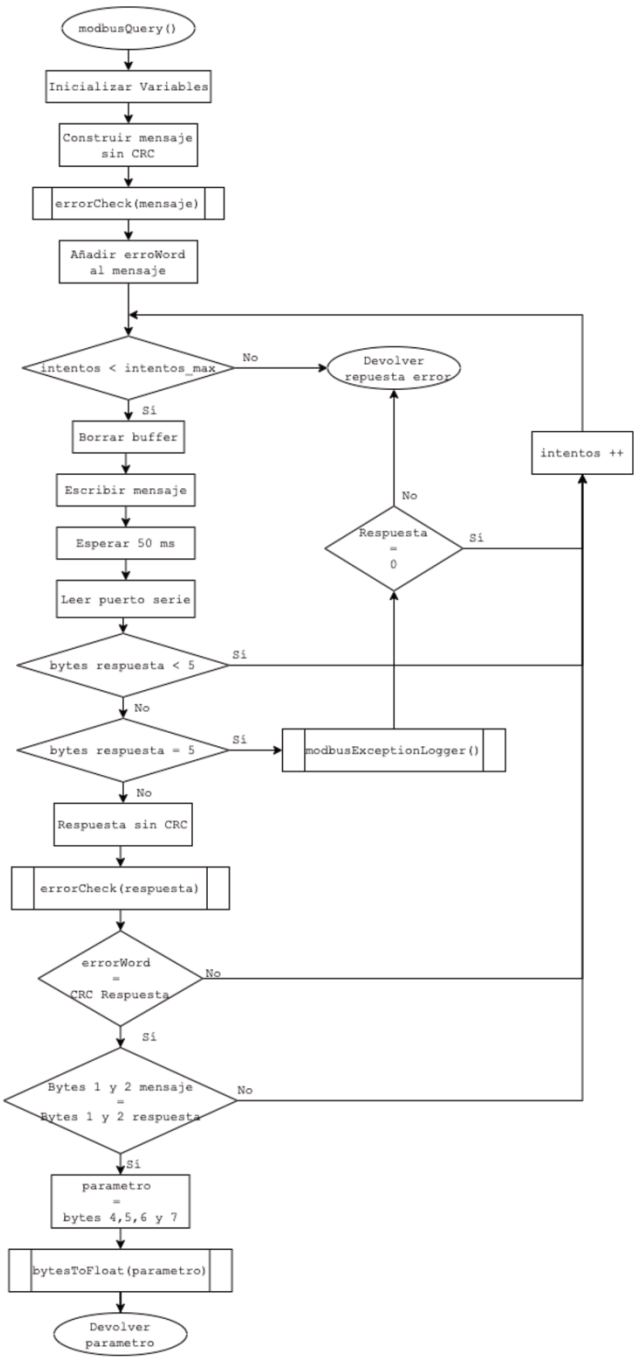
Este trabajo demuestra la posibilidad de seguir adelante con la modernización de muchos sistemas ya existentes de una manera relativamente fácil y económica, mientras nos encontramos en este periodo de transición.

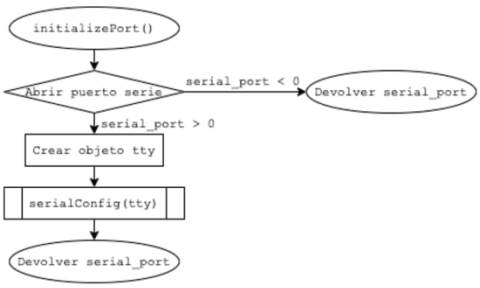
PLANOS

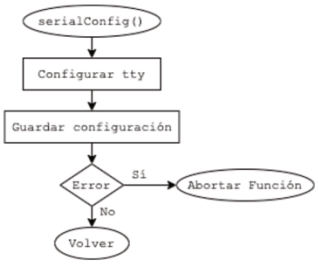
Y

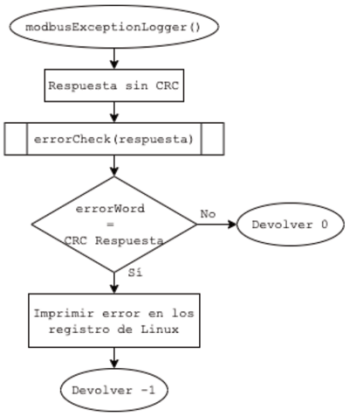
DIAGRAMAS

Índice de Planos y Diagramas

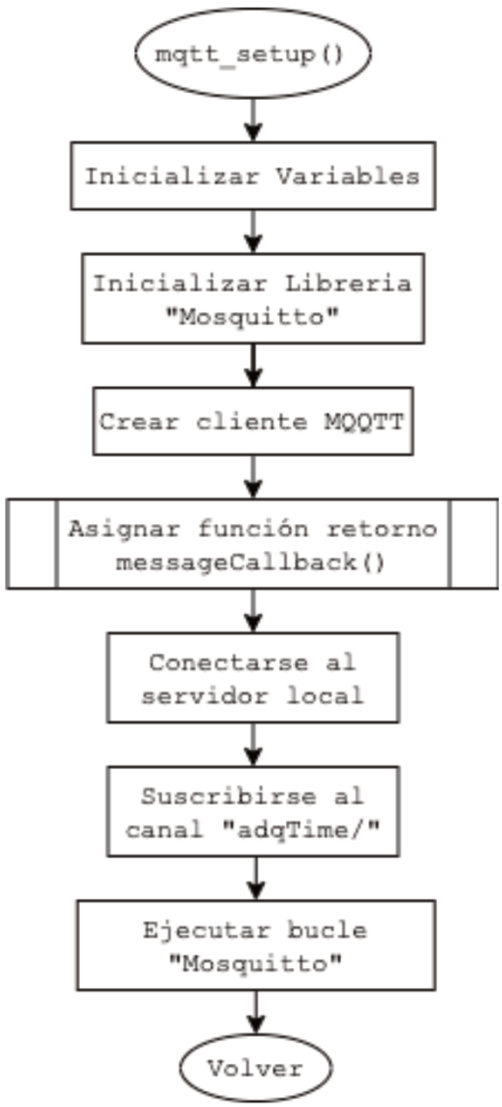


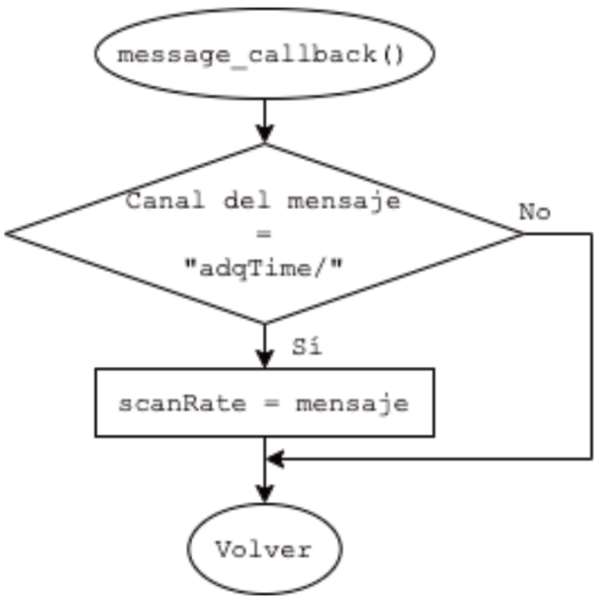


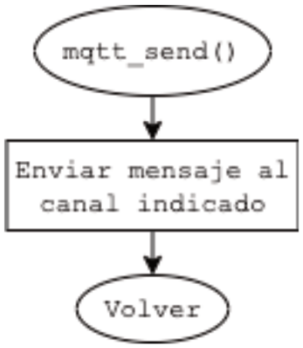


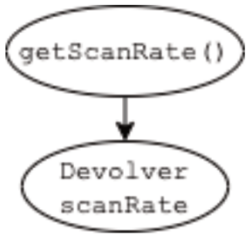


Autor:	**Fecha:**
Rogelio Adrian Rodriguez Rudametkin	8/07/2021
Titulo: Flujogramas de las comunicaciones Modbus.	



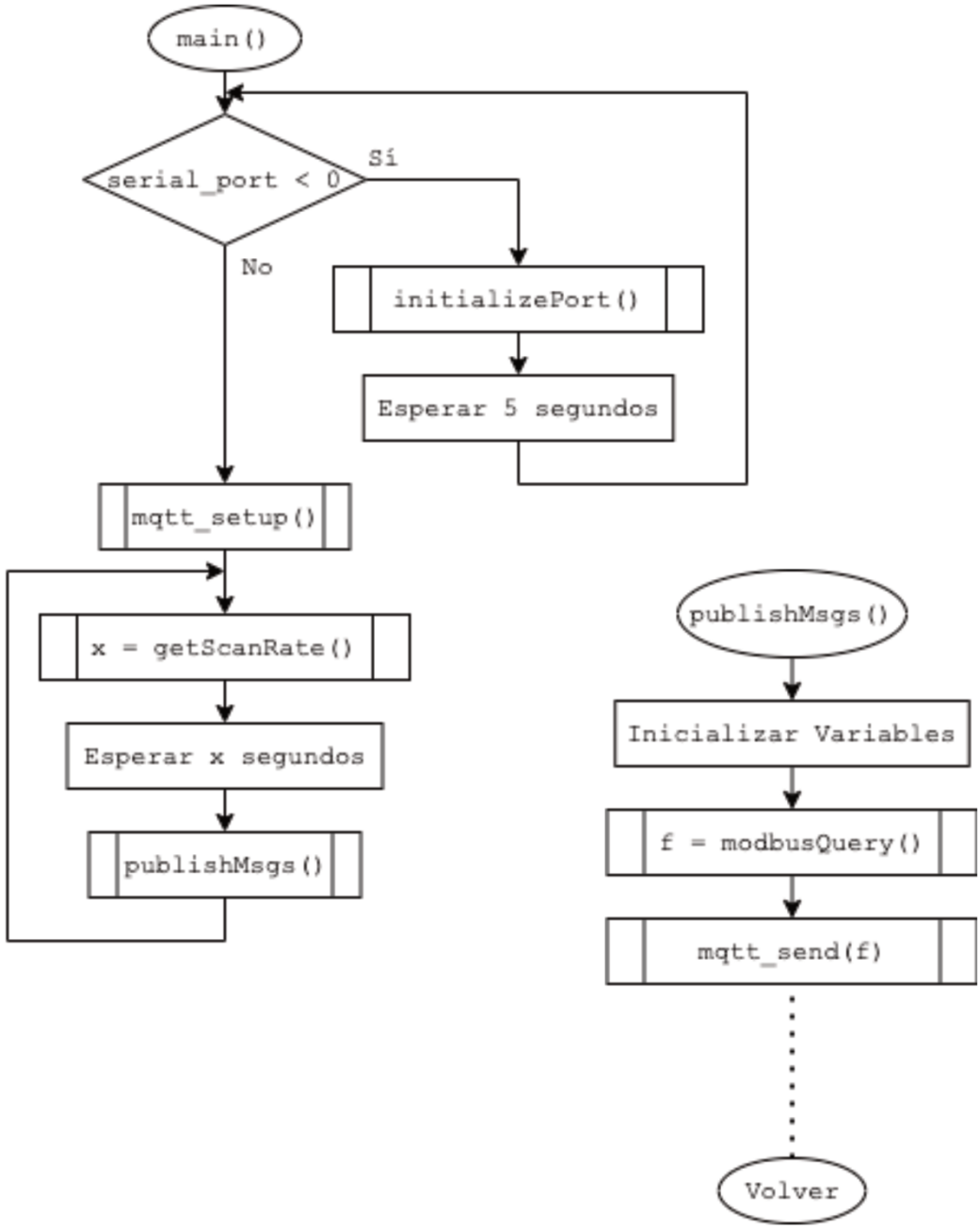




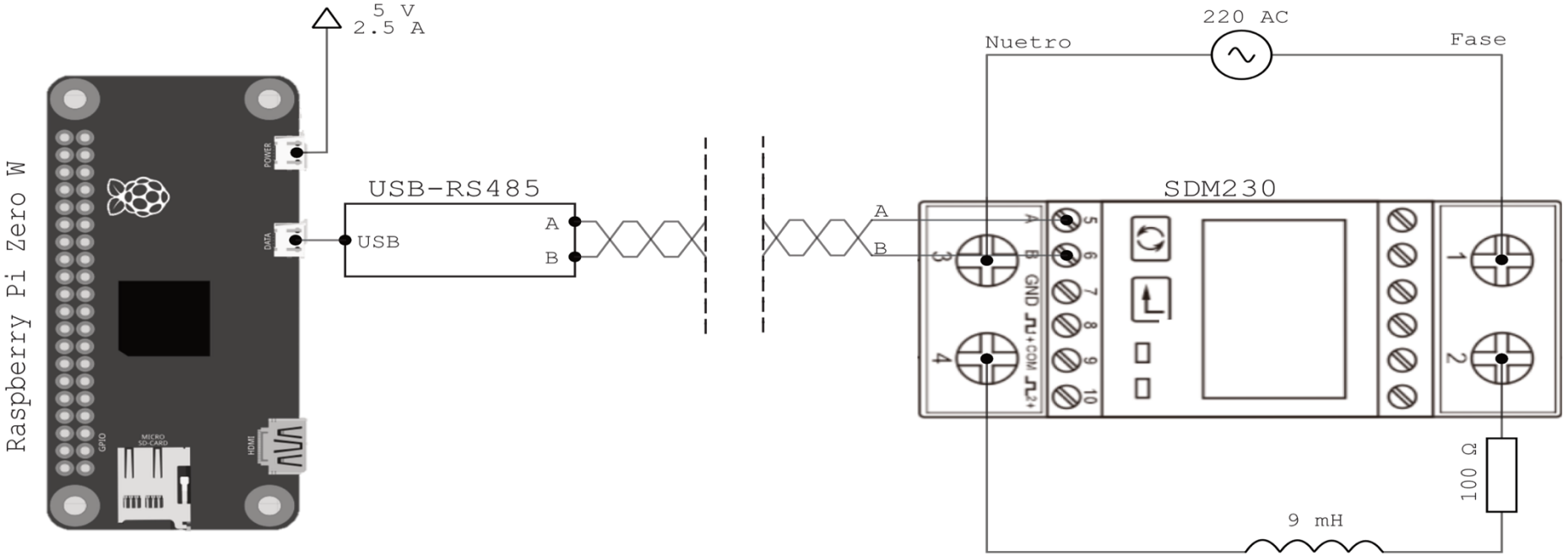


Autor:	Fecha:
Rogelio Adrian Rodriguez Rudametkin	8/07/2021

Titulo: Flujogramas de las comunicaciones MQTT.



Autor:	Fecha:
Rogelio Adrian Rodriguez Rudametkin	8/07/2021
Titulo: Flujogramas del programa principal.	



Autor:	Fecha:
Rogelio Adrian Rodriguez Rudametkin	8/07/2021
Titulo: Plano de conexiones.	

PLIEGO

DE

CONDICIONES

Índice de Pliego de Condiciones

1 Definición y Alcance del Pliego

El objetivo de este documento es establecer las condiciones necesarias para poder llevar a cabo el presente trabajo: diseño del programa y su verificación en hardware.

El ámbito de aplicación de este documento se extiende a todos los equipos eléctricos y electrónicos que forman parte de este proyecto.

2 Condiciones de Materiales

Los materiales mínimos necesarios para completar este trabajo, como se mencionado previamente en la memoria de dicho trabajo, son:

- Kit de inicio para la Raspberry Pi Zero W.
- Vatímetro SDM230 Modbus.
- Conversor USB a RS-485.
- Cable de par trenzado, UTP Cat 6.

Estos materiales son los que se tomaran en cuenta para realizar el presupuesto del trabajo. Además, se deben proveer los siguientes equipos para poder realizar correctamente el desarrollo del programa y poder verificar su funcionamiento:

- Equipo informático (ordenador, pantalla, teclado y ratón) capaz de ejecutar los programas mencionados en la memoria de este trabajo: Visual Studio Code, Eclipse IDE y Raspberry Pi Imager. Es preferible que el sistema operativo sea macOS o Linux, pero no es obligatorio. Lo que si es necesario es que los componentes de este equipo (pantalla, teclado y ratón) este por separado, ya que se utilizaran para la inicialización de la Raspberry Pi Zero W
- Equipo de osciloscopio para poder verificar el funcionamiento de la comunicación.

También se requerirá de un espacio adaptado para trabajar con instalaciones eléctricas, y con acceso a componentes eléctricos y herramientas adecuadas para poder comprobar y experimentar con distintas cargas el sistema propuesto en este trabajo. Este espacio debe tener implementada una red inalámbrica wifi (acceso a internet no requerida) para comprobar el funcionamiento IoT del trabajo propuesto.

3 Condiciones de la Ejecución

El trabajo tiene varias partes que son independientes del resto, por lo que no es completamente necesario cumplir todas las condiciones de materiales al mismo tiempo.

El trabajo puede empezar a ejecutarse una vez se tenga el mínimo de la Raspberry Pi Zero W y el equipo informático. Con esto se puede cumplir la primera etapa de inicialización del dispositivo y se puede empezar a desarrollar el programa propuesta en este trabajo.

Para poder continuar con el desarrollo del programa será imprescindible, eventualmente, el espacio adaptado con sus componentes y el resto de material. Con esto se puede verificar el programa y continuar su desarrollo; tanto su parte electrónica de comunicación serie, como su parte IoT de comunicación inalámbrica.

Una vez terminados todos los aspectos del presente trabajo, el espacio adaptado, junto con todo su material, el equipo informático, y el equipo de osciloscopio puede devolverse al proveedor.

El resto de material será entregado al cliente como el producto final del trabajo terminado. Esto, junto con el trabajo laboral, es lo que se contará dentro del presupuesto.

Una vez finalizado todo, el responsable del presente trabajo se encargará de instalar el dispositivo en la instalación eléctrica del cliente.

PRESUPUESTO

Índice del Presupuesto

1 Introducción

En el presente documento se detallará el desglose del presupuesto necesitado para llevar a cabo el trabajo expuesto. Se encuentra dividido en las diferentes partes requeridas de materiales y mano de obra. Cada parte muestra los diferentes materiales necesarios para el proyecto junto con el coste en cada caso.

2 Presupuesto de Materiales

A continuación de muestran los materiales necesarios para construir la parte física del trabajo. Esto no incluye los materiales de la red eléctrica, que se supone ya existente, ni el material necesario para configurar la Raspberry Pi por primera vez.

MATERIALES						
Parte	**Uds.**	**Descripción**	**Fabricante**	**Cantidad**	**Precio**	**Total**
Kit Raspberry Pi	ud.	Raspberry Pi Zero W	Raspberry Pi	1	42,19 €	42,19 €
	ud.	microSD 16 GB	SanDisk	1		
	ud.	Fuente de alimentacion 12.5W	Raspberry Pi	1		
	ud.	Adaptador USB-microB a USB-A	CannaKit	1		
	ud.	Adaptador mini-HDMI a HDMI	CannKit	1		
Vatímetro	ud.	SDM230	Eastron	1	96,23 €	96,23 €
Cable de comunicación	m	UTP Cat 6	PcComponentes	3	0,92 €	2,76 €
Software	lca.	Visual Studio Code	Microsoft	1	0 €	0 €
	lca.	Eclipse IDE	Eclipse Foundation	1	0 €	0 €
PC y Espacio de trabajo	día	Alquiler de PC y laboratorio	-	5	20 €	100 €
				Subtotal Materiales		**241,18 €**

3 Presupuesto de Mano de Obra

A continuación, se muestra el coste de la mano de obra para desarrollar el trabajo y ponerlo en funcionamiento.

MANO DE OBRA					
Profesional	**Tarea**	**Uds.**	**Cantidad**	**Precio**	**Total**
Ingeniero	Estudio de alternativas, elección de materiales y desarrollo de la aplicación.	h.	30	20,00 €	600,00 €
Ingeniero	Puesta en marcha y verificacion del sistema.	h.	5	20,00 €	100,00 €
Peón	Instalacion de los componentes.	h.	1	15,00 €	15,00 €
				Subtotal Mano de Obra	**715,00 €**

4 Presupuesto Total

RESUMEN DEL PRESUPESTO	
Total Materiales	241,18 €
Total Mano de Obra	715,00 €

Total sin IVA	956,18 €
IVA 21%	200,80 €
Total con IVA	**1.156,98 €**

El total para llevar a cabo este trabajo es de mil ochenta y siete euros y tres céntimos.

MANUAL

DE

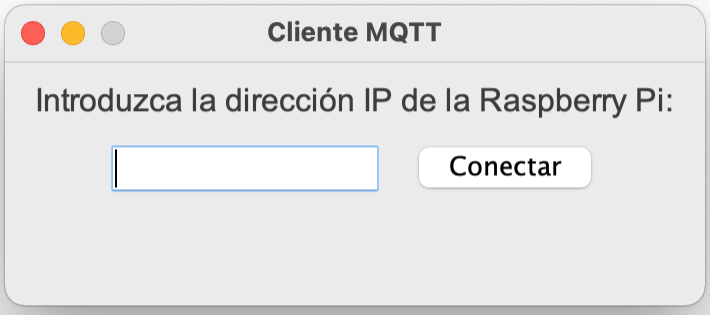
USUARIO

Índice del Manual de Usuario

1 Conectar la Aplicación al Dispositivo

La aplicación entregada, "Monitor SDM230.jar", es una aplicación Java compatible con los sistemas operativos más comunes (Linux, macOS y Windows). Si al abrir la aplicación o darle doble clic, no se abre o da un error, es posible que la aplicación no sea compatible con su dispositivo. En algunas ocasiones el error le indicará que es necesario instalar software adicional; si es el caso, siga esas instrucciones.

Una vez abierta la aplicación, se encontrará con le siguiente pantalla:

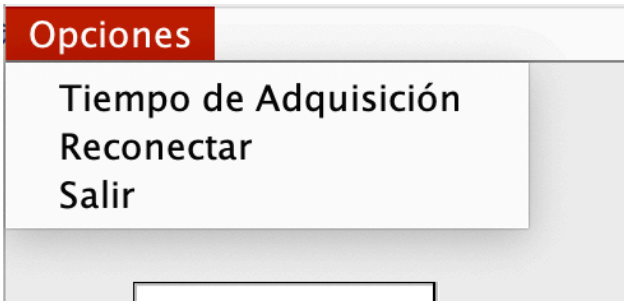


Le pedirá que introduzca la dirección IP del dispositivo instalado en su instalación. Esta dirección que debe introducir es el numero que el responsable de la instalación le otorgo al finalizar la instalación. En caso de no tener este numero, debe contactar con el responsable de la instalación o averiguarlo manualmente con el administrador de su red Wi-Fi.

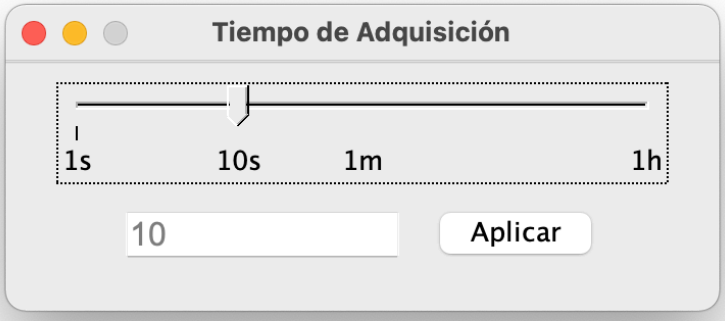
Una vez introducido la dirección IP, la aplicación se conectará al dispositivo y mostrará la ventana principal. Este proceso puede tardar varios segundos.

En la ventana principal puede encontrar los parámetros de su instalación eléctrica:

Si pulsa el menú "Opciones", se sobrepondrá una lista de opciones:

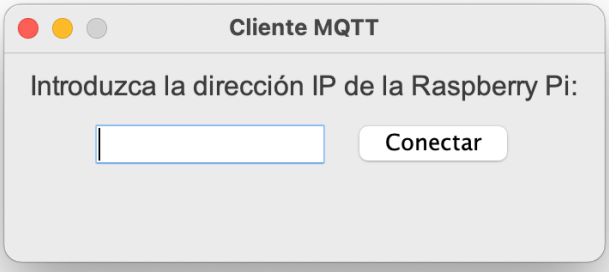


La primera opción, Tiempo de Adquisición, le llevará a otra ventana donde podrá cambiar el tiempo al que se actualizan los parámetros de la ventana principal:



Por defecto, el programa siempre comienza con un tiempo de adquisición de 1 segundo, que es el mínimo.

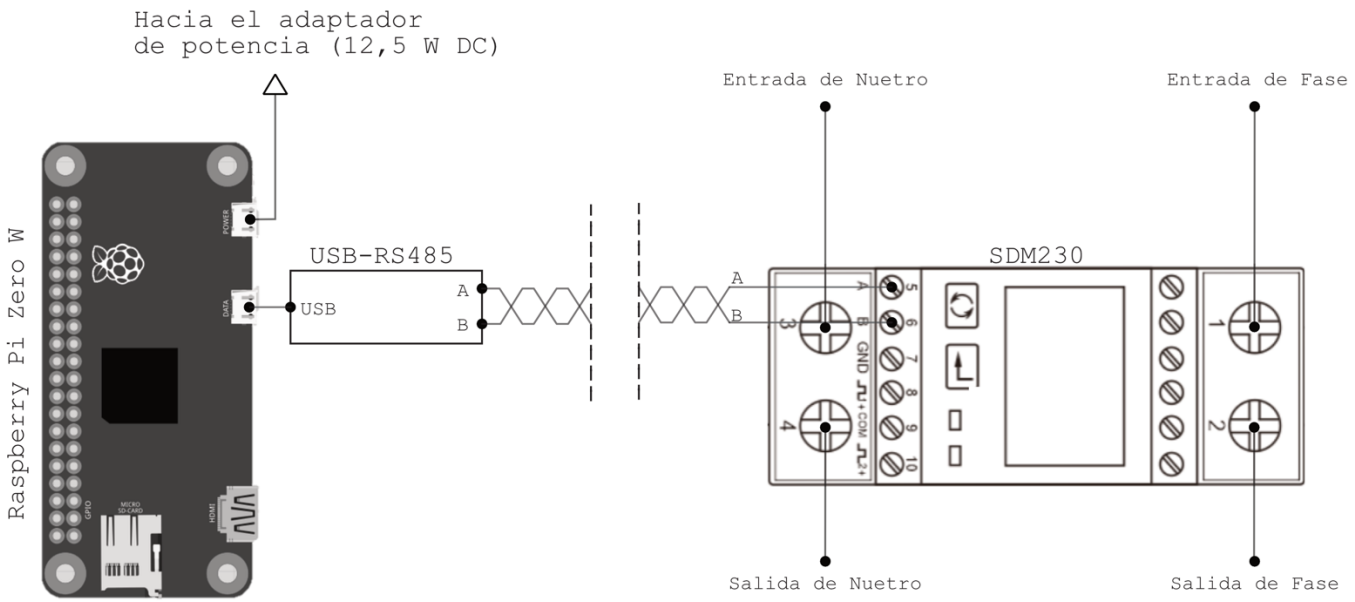
La segunda opción, "Reconectar", le llevará a la ventana inicial, donde tendrá que introducir la dirección IP del dispositivo al que se quiera conectar:



La última opción, "Salir", parará el programa y lo cerrará. Esto también se puede hacer simplemente con pulsar la cruz roja en la parte superior de la ventana.

2 Instalación del Dispositivo

En el caso de que, por cualquier motivo, algunas de las conexiones del dispositivo hayan sido modificadas con posterioridad a su instalación; y, por consecuencia, hagan que la aplicación no funcione correctamente, se presenta el siguiente esquema eléctrico simplificado para ejecutar diagnósticos o reparaciones rápidas:



Es recomendable que cualquier modificación o alteración de la instalación la haga una persona competente en electricidad.

3 Información Adicional

Para problemas en los que la dirección IP no sea reconocida por la aplicación, diríjase con el administrador de la red Wi-Fi donde se encuentra el dispositivo instalado. Puede darse el caso de que cualquier modificación de la red, cambie la dirección IP del dispositivo.

Para problemas con medidas erróneas en la aplicación, se recomienda primero verificar que todas las conexiones del dispositivo sean correctas. En casos muy extremos, podría ser necesario reiniciar el dispositivo; en cuyo caso, se debe dirigir con el competente informático para hacer un reinicio correcto. En ninguna circunstancia corte la electricidad al dispositivo, y que puede corromper el programa interno y requerirá una sustitución parcial del sistema instalado.

En el caso de que haya cualquier otro problema con la instalación o el funcionamiento de la aplicación no sea la correcta o esperada, se recomienda contactar con el responsable que instaló el dispositivo en su instalación eléctrica.

Para cualquier duda sobre el sistema o la aplicación, se recomienda consultar con el responsable del trabajo. Cabe la posibilidad de hacer leves modificaciones para las circunstancias concretas del