Universidad Zaragoza
1542

Proyecto Fin de Carrera

Desarrollo e implantación de un sistema de adquisición de datos y monitorización para un sistema de detección de materia oscura

Autor

Alberto Peiró Val

Director

José Ramón Asensio Diago

Escuela de Ingeniería y Arquitectura

2013

DESARROLLO E IMPLANTACIÓN DE UN SISTEMA DE ADQUISICIÓN DE DATOS Y MONITARIZACIÓN PARA UN SISTEMA DE DETECCIÓN DE MATERIA OSCURA

RESUMEN

El proyecto consiste en realizar un sistema de adquisición de datos para los experimentos de detección de materia oscura que se llevan a cabo en la Facultad de Ciencias de Zaragoza.

Para este caso, por sus características, se plantea una solución en la que se construyen conjuntos con SBC[1] Raspberry Pi y SMC[2] Arduino que integran, de una forma modular, todos los equipos a una red Ethernet. Consiguiendo así un sistema con un alto grado de flexibilidad.

El proyecto también incluye el desarrollo de los programas necesarios para el uso de estos conjuntos. El software ha sido realizado en su mayor parte en Python por ser un lenguaje sencillo y fácilmente adaptable a cualquier cambio que surgiese en el proyecto. Comprende desde una interfaz gráfica de usuario, al uso de sockets TCP/IP para la red Ethernet, un servidor que hace de puente con el puerto serie y los programas para el micro-controlador Arduino, estos últimos en C, para la digitalización de las señales, comunicación, y control en bucle abierto de un motor paso a paso.

1 Single Board Computer.
2 Single Board Microcontroller.

ÍNDICE

MEMORIA

1. INTRODUCCIÓN. Problema a resolver

El proyecto, como el título adelanta, consiste en construir un sistema de adquisición de datos para los experimentos de detección de materia oscura que se realizan en la Facultad de Ciencias de Zaragoza; en concreto, por el Laboratorio de Física Nuclear y Astropartículas.

El experimento se enmarca en el proyecto TREX (Novel Developments in the Time Projection Chambers for Rare Event Searches), financiado por una *Starting Grant* del European Research Center, que fue otorgada en 2009 al investigador Igor García Irastorza. Su objetivo principal es la consolidación de las actividades del grupo en TPCs (Time Projection Chamber) y el equipamiento de un laboratorio especializado en TPCs para aplicaciones de bajo fondo en la Universidad de Zaragoza. Dentro de este proyecto, se encuentra la actividad TREX-DM, consistente en la construcción de una TPC con Micromegas para la detección de Materia Oscura con un bajo umbral de energía y un bajo nivel de fondo.

El montaje, figura 1, consta de una carcasa cilíndrica en la que hay alojados una placa y unos anillos que crean un fuerte campo eléctrico longitudinal al eje. Contenido en la carcasa hay un gas con el que las partículas buscadas interaccionando creando pares

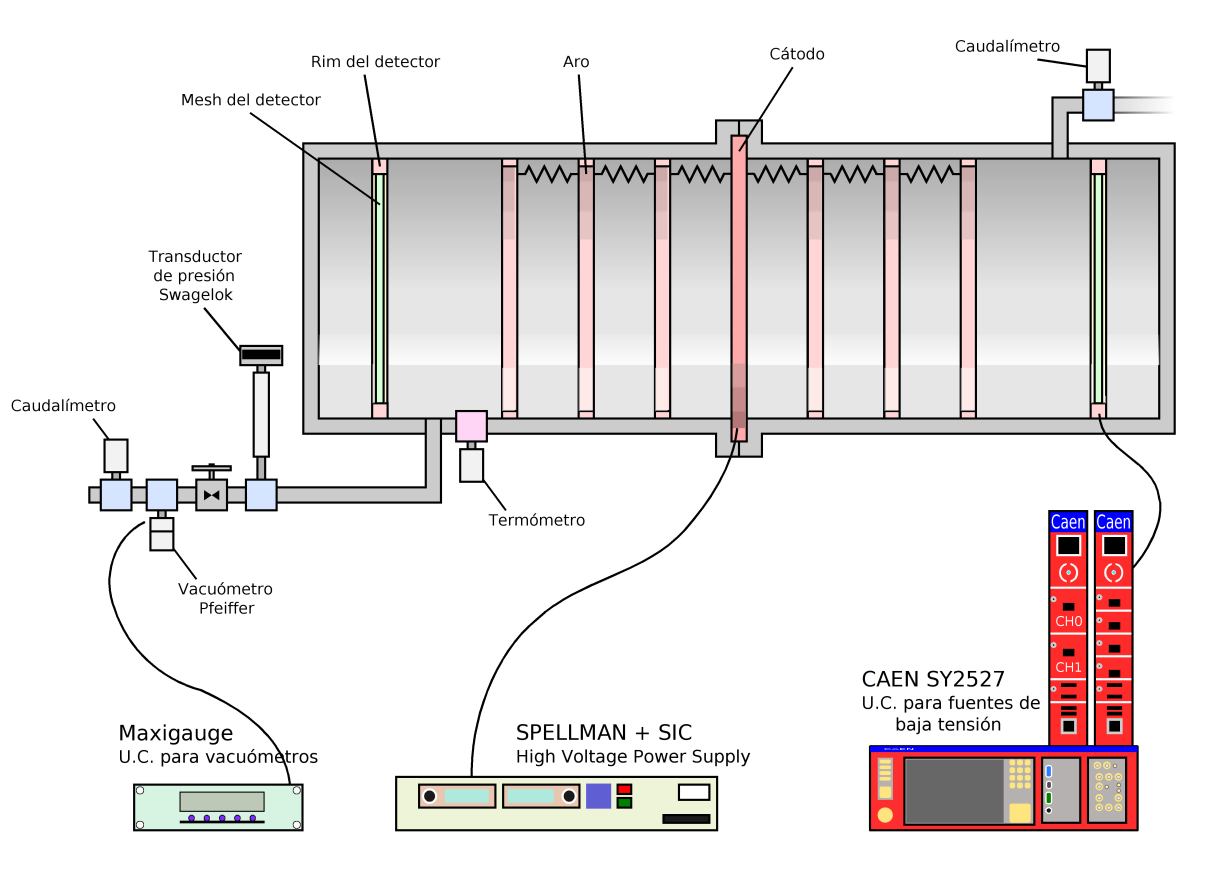


Figura 1. Esquema del experimento con los equipos involucrados

electrón-ión a lo largo de su trayectoria. Estos electrones son arrastrados por el campo eléctrico hacia uno de los detectores donde se obtiene una medida de la cantidad de energía depositada y datos que permiten la reconstrucción de la trayectoria de la partícula inicial.

Lo que se desea es monitorizar las variables físicas en las que el experimento se desarrolla (presión, temperatura, campo eléctrico), además de tener la posibilidad de interactuar con los equipos a distancia a través de una red. Estos equipos, como puede verse en la figura 1, son fuentes de tensión, caudalímetros, un vacuómetro, un manómetro y un termómetro.

Objetivos.

Las características que debía tener el sistema de adquisición de datos fueron fijadas por consenso con los encargados del experimento tras una presentación en la que se expusieron distintas soluciones. En éstas se consideraron desde usar un PC con tarjetas PCI, pasando por LabView con los módulos de hardware de National Instruments, hasta PLCs con un sistema SCADA. Ver Anexo 1.

Finalmente, con esta información y teniendo en cuenta las restricciones en cuanto a tiempo de respuesta y resolución, se decidió construir conjuntos con SBC Raspberry Pi y SBM Arduino que hiciesen de interfaz entre los equipos y una red Ethernet (ver figura 2). Cada uno ellos dotado con una interfaz gráfica de usuario independiente (GUI), consiguiéndose así una modularidad que hace al sistema fácilmente reutilizable en otros experimentos, con solo trasladar el conjunto a un nuevo experimento (o duplicarlo) y empezar a utilizarlo desde su correspondiente interfaz gráfica de usuario (GUI). Igual de fácil es conectar al sistema un nuevo equipo similar.

Como lenguaje de programación se escogió Python, por ser un lenguaje sencillo y de alto nivel que permite a cualquier usuario modificarlo y adaptarlo rápida y fácilmente a cualquier cambio en el diseño del experimento.

A pesar de no haberse dado especificaciones sobre la robustez del sistema, por no ser un aspecto crítico, se intenta que este sea lo más robusto posible con la menor probabilidad de error en la toma de las medidas. Para ello, en todo momento, se usa el protocolo con mayor inmunidad al ruido disponible en el equipo de medida, programación con tratamiento de excepciones y, además, se incorpora algún control para detectar y evitar errores.

Contenido.

La memoria se divide en dos secciones principales bien diferenciadas, una para el hardware y otra para el software.

En la primera de ellas, se hace un análisis de los equipos centrándonos en los tipos de conexión disponibles y en su inmunidad al ruido. Con esta información, definimos los conjuntos de Raspberry Pi y Arduino que hacen de interfaz con la red Ethernet.

La segunda sección, más extensa, trata del software necesario para conseguir la integración de los equipos en la red. Este constaría de las siguiente partes dependiendo del tipo de conexión existente en el equipo.

· Una interfaz gráfica de usuario que usa hilos de ejecución para evitar bloqueos en la aplicación y una cola FIFO a la que se accede en exclusión mutua para organizar la

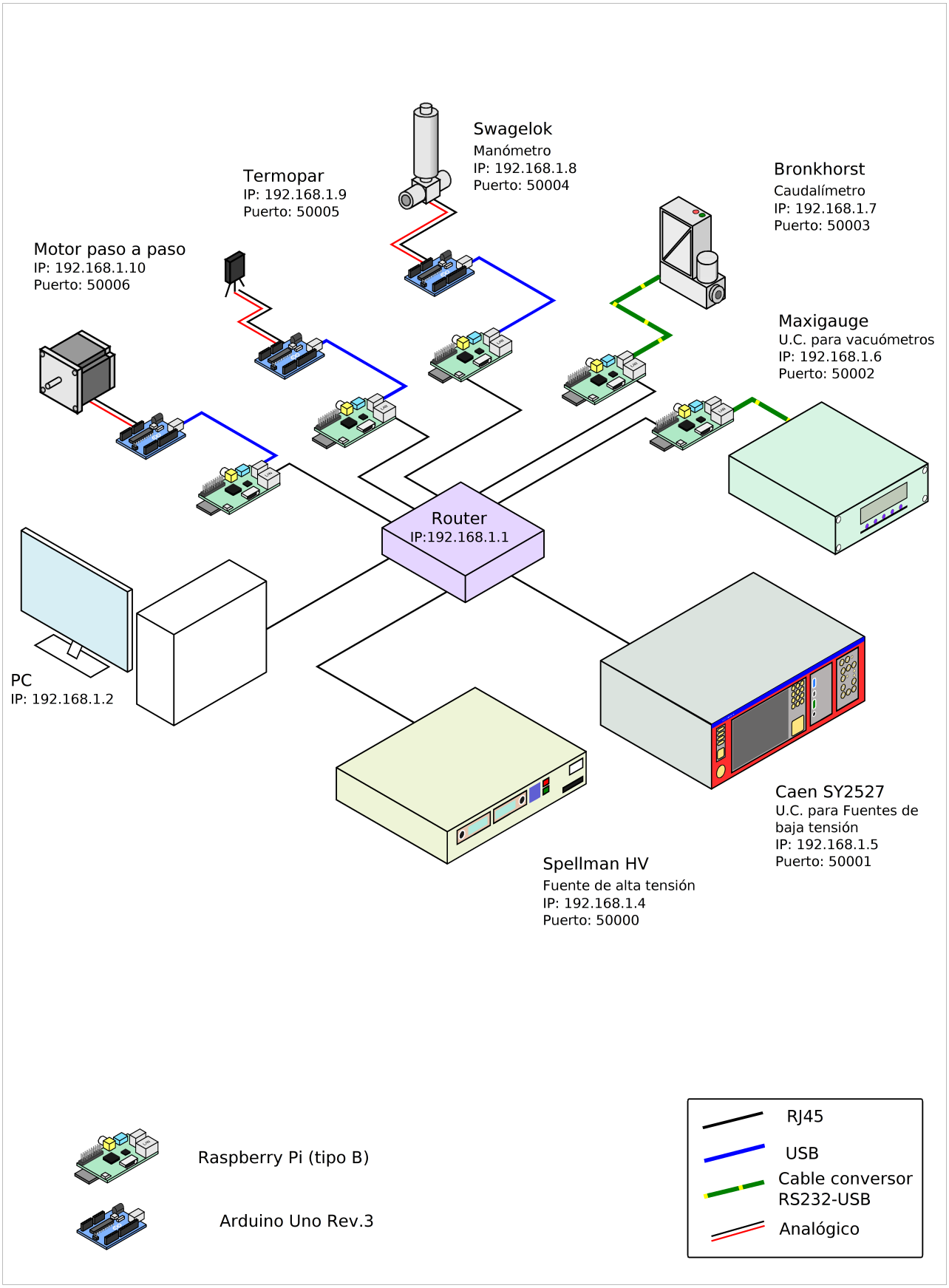


Figura 2. Equipos y conjuntos con Raspberry Pi y Arduino que integran la red Ethernet

salida por pantalla de los datos proporcionados por las tareas.

· Módulos Python, que son el equivalente a las bibliotecas, que aglutinan las instrucciones para entenderse y operar con los equipos. Casi todos ellos están constituidas por funciones que construyen mensajes en el lenguaje comprensible de cada equipo de medida. Aunque también se han utilizado otras facilidades de Python, como el módulo Ctypes que permite usar las funciones C de bibliotecas de enlace dinámico, proporcionadas por el fabricante, directamente desde Python.

· Uso de sockets TCP/IP para la comunicación Ethernet

· Un programa servidor que hace de puente con la conexión serie.

· Programas para el micro-controlador Arduino que realizan las tareas de digitalización de las señales, comunicación, y de control en bucle abierto del motor paso a paso.

2. ESTRUCTURA HARDWARE

2.1 Análisis de los equipos

Empezamos por hacer un análisis de los tipos de conexión disponibles en los equipos involucrados en el experimento. Así, teniendo presente la tabla 1, que recoge las características de cada conexión, construimos la parte sombreada en gris de la figura 3 en la que aparece el equipo, su función, y el tipo de conexión más confiable disponible. Entendiendo por confiable la más inmune al ruido, y por tanto más segura y menos propensa a la aparición de errores. Resumiendo, según la tabla 2, escogeremos como primera opción Ethernet, seguida de USB, RS232 y, finalmente, la conexión analógica.

Comunicación	Inmunidad	longitud
Ethernet	*Encapsulamiento inform. + comprobación CRC*	*Depende del tipo de medio físico*
USB	*Señal diferencial*	*1600m*
RS232	*Niveles de voltaje*	*5m típico*
Analógica	*Ninguno*	

Tabla 1: Análisis de inmunidad al ruido del protocolo de comunicación.

Para una información más detallada de los equipos se puede consultar el anexo 2.

2.2 Arquitectura de la solución

Sintetizando, la solución, consensuada con el equipo de los experimentos, consiste en construir conjuntos con SBC Raspberry Pi y SBM Arduino que hacen de interfaz entre los equipos y la red Ethernet, ver figuras 2 y 3. Este diseño modular permite una alta flexibilidad en la que es muy fácil añadir nuevos equipos o reutilizarlos en otros experimentos.

Antes de continuar con la creación de los bloques, describimos brevemente qué es un SBC Raspberry Pi y un SBM Arduino.

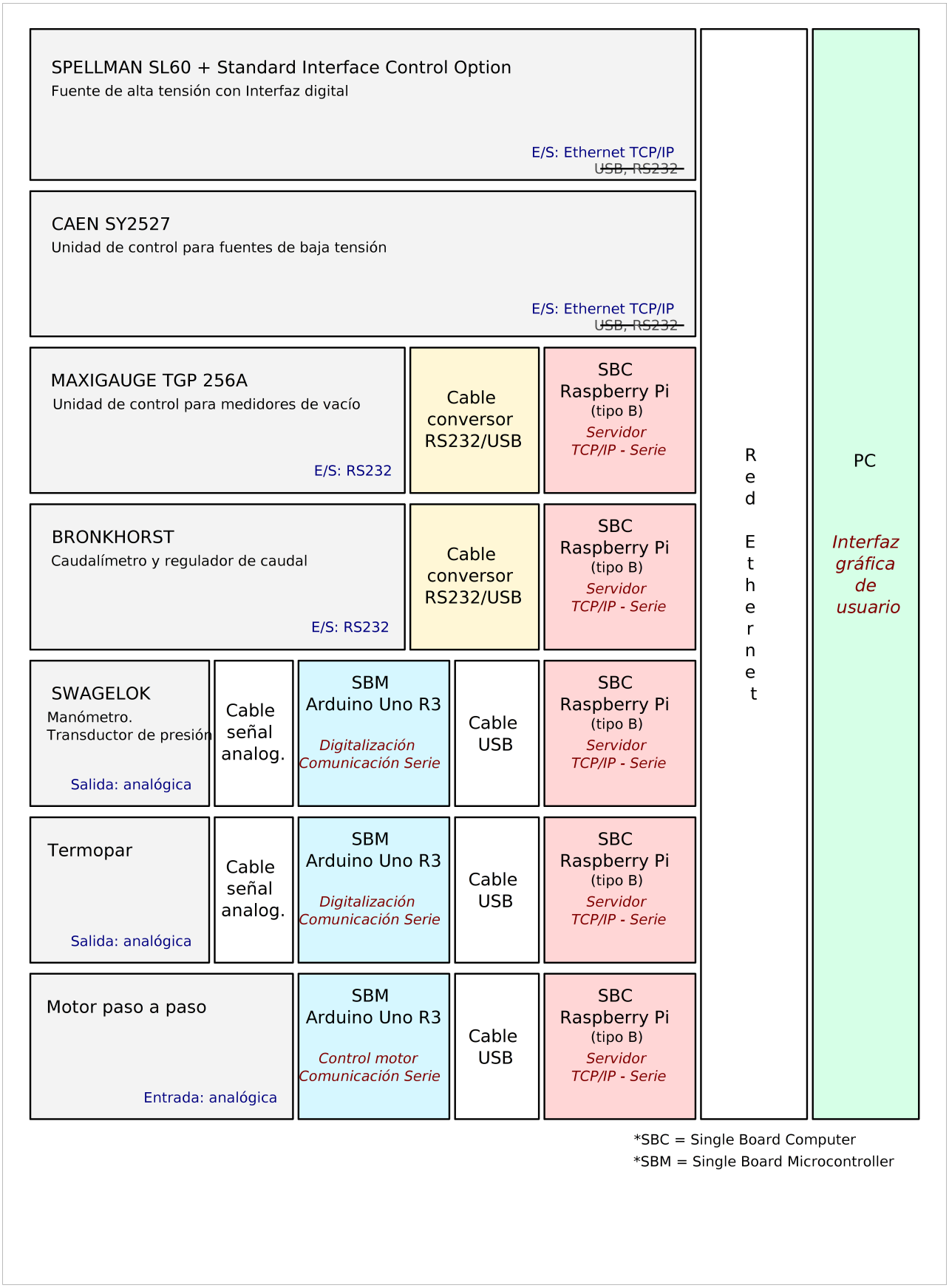


Figura 3. Hardware y software de los conjuntos

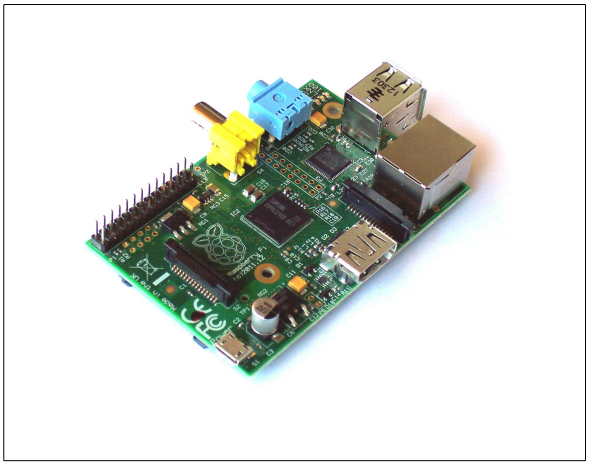


Figura 4. Raspberry pi

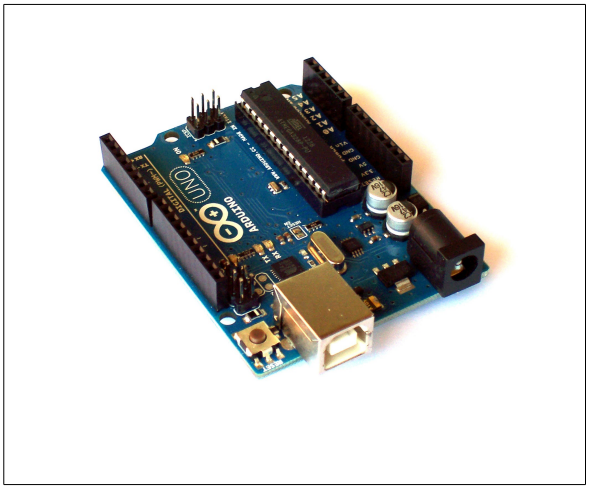


Figura 5. Arduino

Un SBC (Single Board Computer) Raspberry Pi, figura 4, es un ordenador completo construido en una única placa. El tamaño es del de una tarjeta de crédito, porta un microprocesador ARM (arquitectura RISC) y usa Linux como sistema operativo. Fue desarrollado en Reino Unido por la fundación Raspberry Pi [1], cuyos miembros están de algún modo relacionados con la empresa Broadcom.

El SBM (Single Board Microcontroller) Arduino [2], figura 5, es una plataforma de hardware libre, basada en una placa con un micro-controlador y un entorno de desarrollo. Existen distintos modelos, de entre los cuales, usaremos el Arduino Uno rev.3, que viene con un puerto USB y un micro-controlador Atmega 328 de 16 Mhz con 6 entradas multiplexadas para la digitalización con una resolución de 10 bits, que es suficiente para lo que se requería, y dos salidas PWM (Pulse With Modulation.

Volviendo a la tarea de crear los conjuntos , vemos en la figura 3 que ya hay dos equipos, Spellman y Caen, que ya vienen con conexión Ethernet, por lo que no necesitarán ningún dispositivo extra.

Los dos siguientes, Maxigauge y Bronkhorst, disponen solo de conexión serie RS232. La cual, como hemos visto, tiene poca inmunidad al ruido por lo que conviene reducir la longitud de este tipo de conexión a la menor posible para mejorar la robustez. Para este caso, usamos un cable conversor RS232-USB (con distancia RS232 aproximadamente nula) que conectaremos a un Raspberry Pi, en el que se estará ejecutando un programa servidor que hace de puente entre la conexión TCP/IP en la red Ethernet y el puerto serie del dispositivo.

Para los equipos con salida analógica, el primer paso será digitalizar la señal utilizando un Arduino. Este es capaz de digitalizar la señal con 10 bits de resolución y tomando una muestra cada 0.25ms. Puesto que el Arduino es un dispositivo pequeño, cuanto más cerca lo podamos colocar de la fuente reduciendo la distancia del cable de la señal, mayor será la calidad de la medición. En el siguiente paso conectamos el Arduino con un cable USB a un Raspberry Pi, obteniendo ya un conjunto similar al descrito al caso anterior.

Por último, en el caso del motor paso a paso, el hardware del conjunto es el mismo que el que acabamos de ver. Siendo la única diferencia el programa que corre en el Arduino, que ahora es un control en bucle abierto de un motor paso a paso.

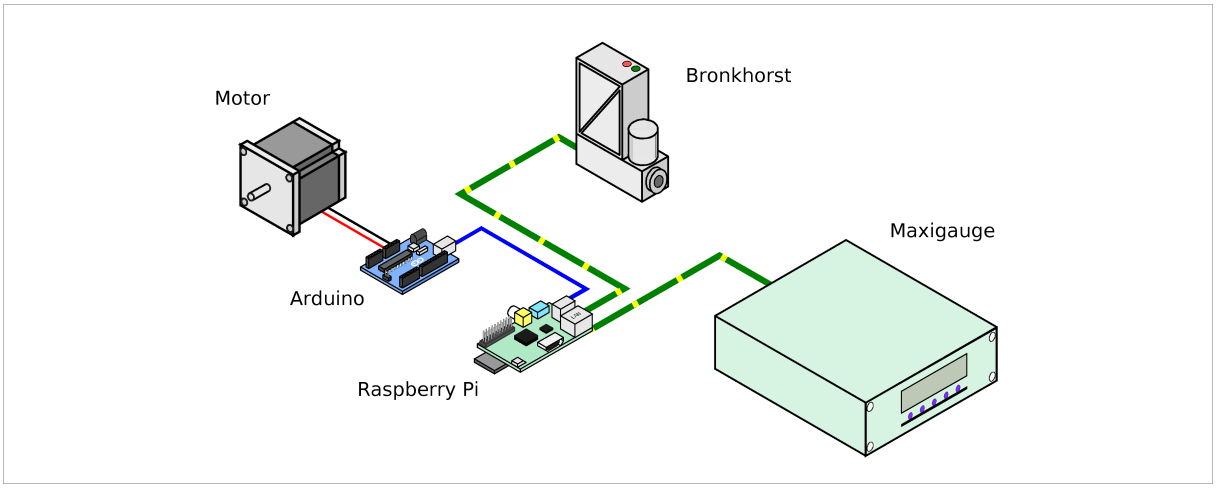


Figura 6. Alternativas de diseño a los conjuntos

Para terminar, habría que mencionar que durante la realización del proyecto han ido apareciendo otras opciones de hardware, y parece que la tendencia actual es la de seguir ampliando aún más la oferta; por mencionar dos ejemplos, el Gnublin es otro SBC similar a Raspberry Pi y el Hummingbird al Arduino.

En la misma dirección, también son posibles otras configuraciones distintas a los conjuntos presentados y que son compatibles con los programas desarrollados. Por tanto, otras configuraciones como la de la figura 6 podrían ser alternativas a los diseños realizados.

3. ESTRUCTURA SOFTWARE

3.1 Software y lenguaje de programación.

En la figura 3 aparece todo el software junto al hardware que lo aloja. Casi todo él, está realizado en Python [3][4][5], o en C en el caso de los programas para el Arduino [2].

Una de las características de Python, el nombre le viene de los Monty Python y no de la serpiente, es que es un lenguaje interpretado utilizado para prototipado rápido con multitud de bibliotecas, y que hace hincapié en una sintaxis clara,. Eso lo hace muy apropiado para casos como el que tratamos, en el que se van a efectuar muchos cambios y modificaciones, y lo mejor es poderlas hacer rápido y sin muchas complicaciones.

En los siguientes apartados vamos a ir explicando con más detalle las características más importantes de los programas que contiene el conjunto.

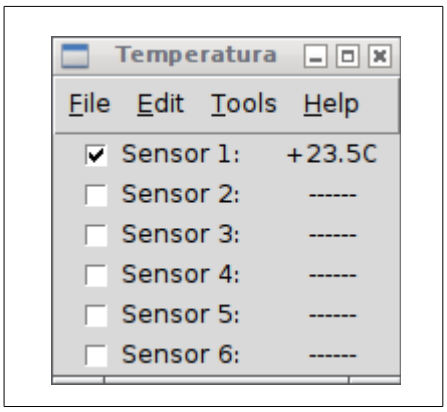


Figura 7. Ejemplo de interfaz gráfica

3.2 Interfaz gráfica del usuario

Pensando en la modularidad de la solución se han

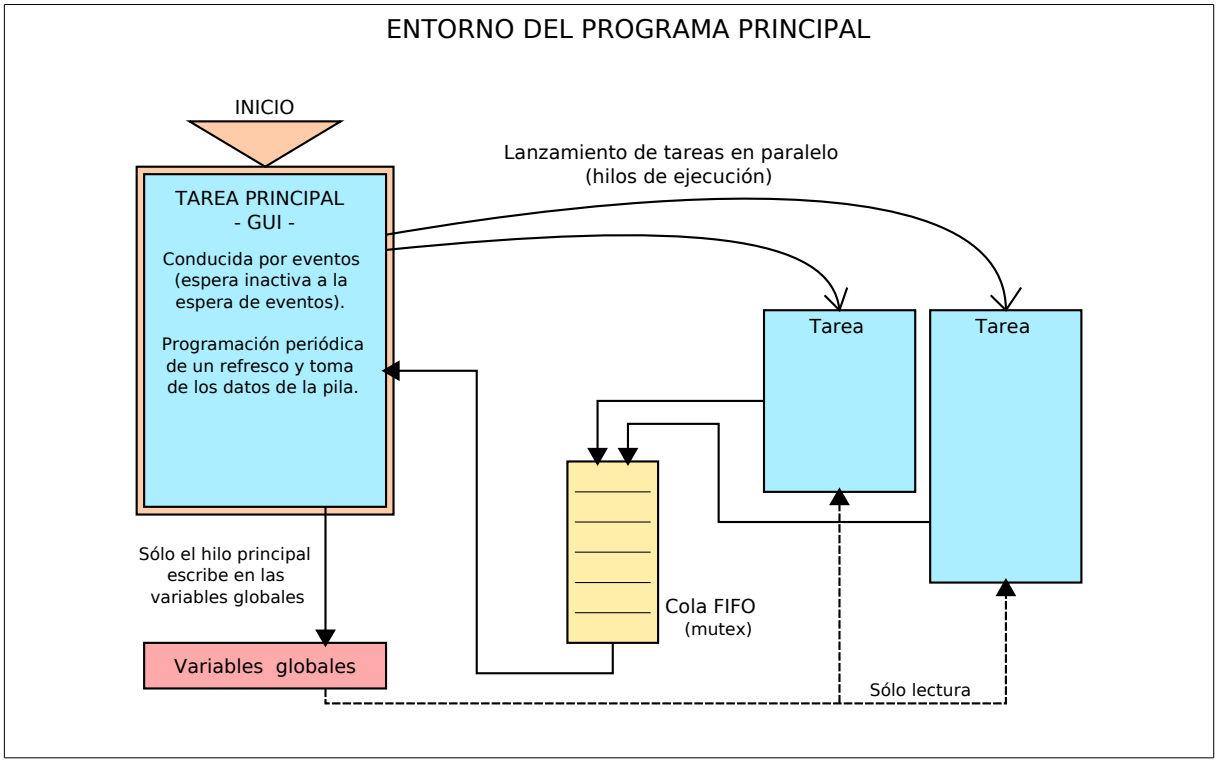


Figura 8. Alternativas de diseño a los conjuntos

construido interfaces gráficas independientes para cada equipo. Un ejemplo de cómo son estas se muestra en la figura 7. Están realizadas en Python y usan la biblioteca gráfica *tkinter*, la cual se ejecuta en un modo conducido por eventos, es decir: permanece inactiva hasta que un evento la hace despertar. Razón por la que se tienen que usar hilos para ejecutar tareas de fondo y evitar bloqueos. Además, puesto que la salida por pantalla es un recurso compartido se necesita usar una cola FIFO para acceder en exclusión mutua y organizar la salida de los datos. La estructura puede verse en la figura 8.

Todos estos temas, hilos, estructuras de datos, y el módulo *tkinter*, son tratados en profundidad en el libro de la referencia [5], incluso apunta la solución que hemos usado, la cual se explica en detalle en el anexo 3.

3.3 Módulos Python

Los módulos de Python son el equivalente a las bibliotecas en otros lenguajes [3][4].

Se han creado módulos específicos para cada equipo que aglutinan las instrucciones para la comunicación con cada uno de ellos. Estos módulos son usados desde las interfaces gráficas para interactuar con los equipos, ya sea solicitando datos o modificando alguna salida. Su construcción es autosuficiente, es decir, que contienen todo lo necesario para efectuar por si solos una comunicación TCP/IP, enviar un mensaje y esperar su respuesta. Por tanto esto les permite ser usados desde cualquier programa, e incluso desde el intérprete de Python. Este es una especie de Shell de comandos desde el que se puede cargar un módulo y utilzar directamente sus funciones. Con lo que

disponemos de una plataforma extra para comunicarnos con los equipos sin la necesidad de una interfaz gráfica.

El caso del equipo CAEN, es un poco distinto, ya que en éste se ha utilizado el módulo *Ctypes* para usar las funciones de una biblioteca C proporcionada por el fabricante, el resto usa Python para crear los mensajes y sockets como medio de comunicación. Igualmente tenemos para dicho equipo un módulo Python autosuficiente, pero que ahora no sólo necesita Python, sino también la biblioteca C del equipo. En el anexo 4 se explica el uso de *Ctypes* para el uso de funciones C desde Python.

3.4 Comunicación por sockets TCP/IP

Como medio de comunicación entre aplicaciones dentro de una red Ethernet, usamos sockets TCP/IP. Diferenciamos dos tipos de sockets. Uno, el socket servidor que está esperando en un puerto determinado a una petición de conexión, y otro, los sockets clientes entre los que se establece la comunicación, la cual generalmente se compone del intercambio de unos pocos mensajes. Cuando se da por finalizada se destruye la conexión liberando todos los recursos utilizados.

Explicado en detalle, **figura 9, la cual no debe tomarse como una especificación formal, sino simplemente como un apoyo gráfico a la explicación**., el proceso se inicia cuando un cliente realiza una petición de conexión al socket servidor. En caso de ser aceptada, este socket servidor crea un nuevo socket, del tipo cliente, asociándolo con el primero y formando una pareja de iguales (*peer to peer*) entre los que se establece la conexión. Esta pareja está ligada a un puerto cada vez distinto, o al menos éste no se va a repetir en mucho tiempo, del ordenador del que parte la petición. El uso de puerto distinto cada vez, unido a que sólo se envía una única instrucción al equipo en cada conexión, se traduce en que hay una relación instrucción-respuesta que dota al sistema de una mayor robustez ya que no puede darse un error por el que un cliente obtenga una respuesta que no le correspondía.

La conversación, como se ha adelantado, consta del envío de un solo mensaje, donde se encuentra la instrucción, y una espera de la respuesta. Para evitar bloqueos la conversación acabará cuando se reciba una respuesta o cuando se supere un tiempo límite (los rectángulos representan transiciones temporizadas). En ambos casos cuando acaban de usarse los sockets se destruyen liberando los recursos (*buffers* de red) que el sistema operativo esté usando.

Mientras transcurría la conversación, el socket servidor sigue aceptando peticiones de conexión, las cuales quedan a la espera de que les sea asociado un socket (o sean anuladas porque el socket cliente ya no existe). La asociación se ha dispuesto de forma secuencial y no ocurrirá hasta que la anterior comunicación haya sido completada. Esta estructura permite tener a varios clientes comunicándose con una mismo dispositivo serie con lo que podemos tener varias interfaces gráficas abiertas al mismo tiempo.

3.5 Comunicación serie

Una biblioteca de Python, *PySerial*, proporciona todos los los elemento necesarios para interactuar con el puerto serie. En la página del proyecto PySerial [6] hay varios ejemplos de uso y en los cuales nos basamos para hacer el programa para la gestión de

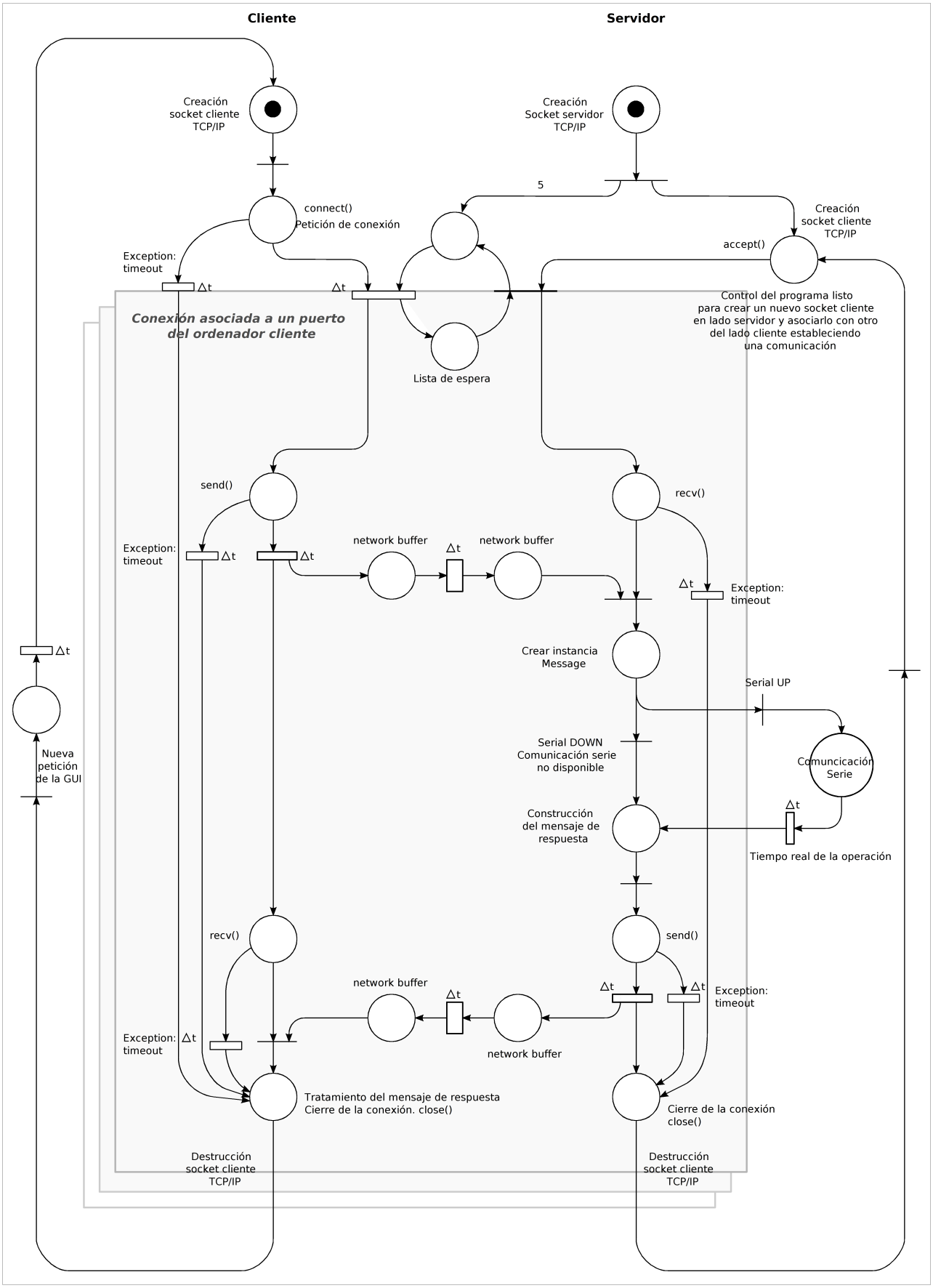


Figura 9. Comunicación entre sockets TCP/IP

la conexión serie.

Nos ayudaremos de **la figura 10, que igual que antes no debe tomarse como una especificación formal, sino simplemente como un apoyo gráfico a la explicación**. Además, para tener un modelo adecuado quizás se debería usar una Red de Petri Coloreada en la que intervendría el orden de llegada de las marcas. En una primera versión muy simple en la que el cliente hiciese una petición al dispositivo serie y esperase indefinidamente la respuesta aparecería el problema de que un mensaje corrupto (como el no encontrar el carácter fin de mensaje) bloquease el sistema.

La primera solución fue añadir un tiempo de espera límite en el cliente, similar a como funcionan los sockets. El problema que aparece es que al sobrepasarse este tiempo, el cliente puede volver a enviar otro mensaje con lo que estos se pueden ir acumulando en los *buffers* de entrada y salida. Aunque tiene la ventaja de que puede avisar aguas arriba de que el problema se encuentra en el dispositivo serie y no en la red Ethernet.

Esta acumulación de mensajes puede producir una pérdida de sincronización entre la petición y su respuesta. Para evitar esto podemos vacíar los buffers, una forma de hacerlo es leer todo el *buffer* y descartar los más antiguos quedándonos sólo con el último, como se representa en la figura 10. Sin embargo queda por resolver un problema más sutil. Si un cliente envía una petición al dispositivo serie y supera el tiempo de espera sin obtener respuesta, quizá porque la petición sigue siendo procesada, después de que el siguiente cliente haga su petición la respuesta que encontrará será la del cliente anterior y no la suya. Para evitar esta situación lo que hacemos es devolver la respuesta con el mensaje de entrada que la generó. De forma que podemos comprobar si coincide y actuar en consecuencia. Mucho mejor si añadimos un número de secuencia a la orden para el caso en que estas sean idénticas.

Todo esto último está implementado en el programa que hace de puente entre la conexión serie y la red Ethernet. También dispone de tratamiento de excepciones para evitar que el programa se cuelgue por cualquier circunstancia no esperada, por ejemplo campos de datos inexistentes, o tipos incorrectos que puedan aparecer por corrupción de los mensajes en la transmisión. También usando excepciones, el mismo programa es capaz de detectar la desconexión del dispositivo serie y lanzar un hilo para su reconexión. Mientras, el programa principal sigue atento a cualquier conexión que venga de la interfaz gráfica informando que el dispositivo serie se ha desconectado.

3.6 Digitalización de señales

El programa que se encarga de la digitalización de las señales analógicas se encuentra en los Arduino. Este posee 6 entradas multiplexadas para la digitalización de la señal con 10 bits de resolución, suficientes para lo que se requería.

El programa sigue el esquema de comunicación, que acabamos de ver en el apartado anterior, efectuando una digitalización a instancia de una petición y devolviendo la respuesta acompañada de la orden. En el caso que no entendiese la orden, simplemente, la remitiría volviendo a estado de inicial de escucha. El programa, como todos los demás, se encuentra en los anexos.

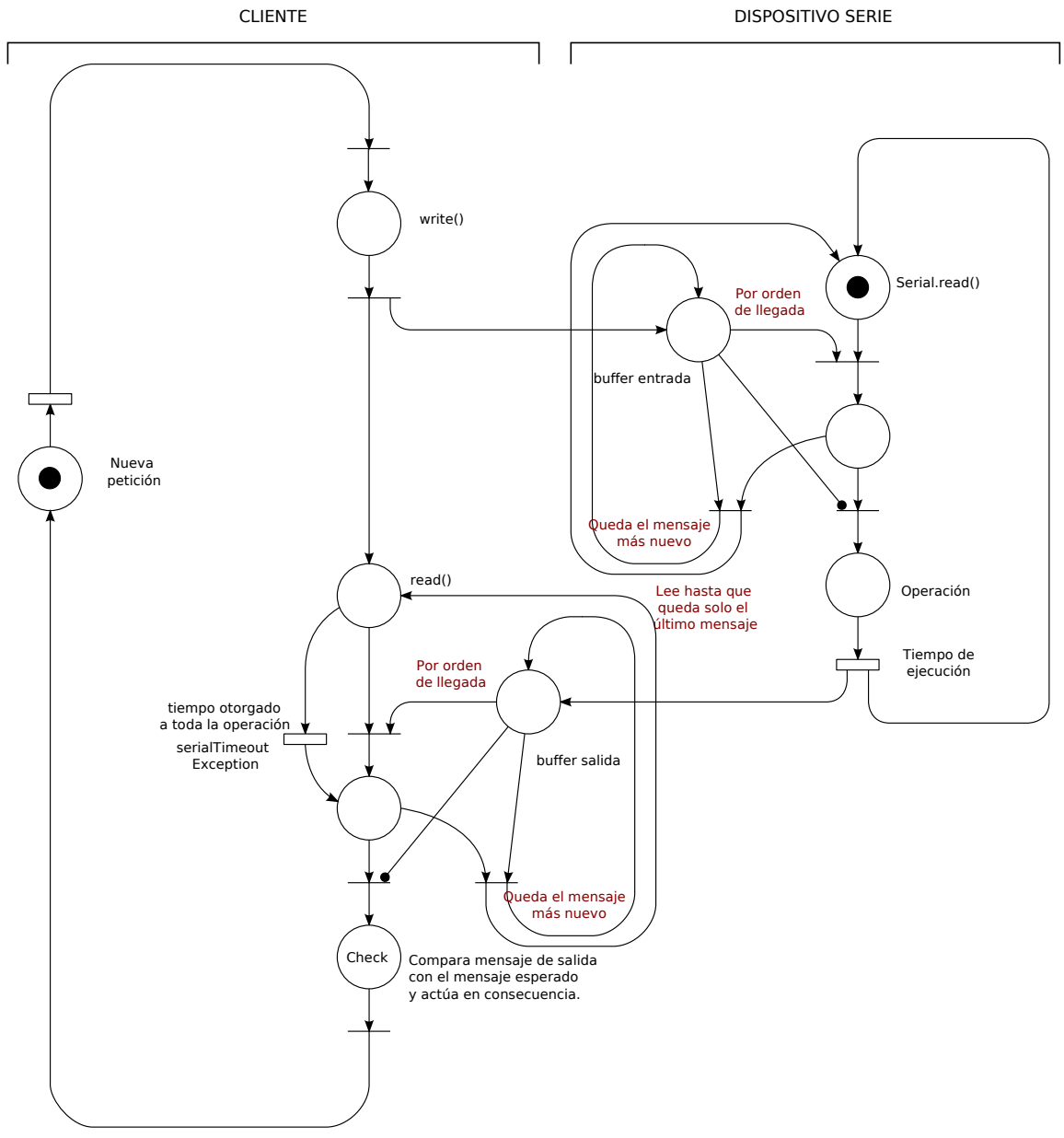


Figura 10. Comunicación por el puerto serie

3.7 Controlador del motor paso a paso

Durante la fase de desarrollo de este proyecto, el equipo de investigación sugirió una ampliación consistente en que la fuente de radiación de baja intensidad, que en un principio iba a permanecer fija, se moviese entre ocho posiciones en el interior de la carcasa.

El dispositivo para guiar el movimiento se vale de una sirga conducida por el interior de un tubo hueco de plástico, y movida por fricción a través de una polea por un motor paso a paso. El motor seleccionado debía transmitir un par de 1Nm sin un conjunto reductor, por razones de espacio, y tener una resolución aceptable. Ambos condicionantes podían cumplirse con un motor paso a paso híbrido bipolar de 200 pasos por vuelta. Figura 11.

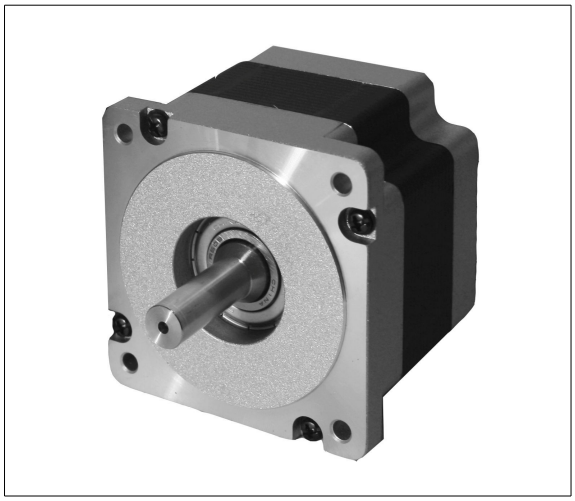


Figura 11. Motor paso a paso

Para su control se usa un Arduino Uno Rev3 acompañado de una placa Motor Shield Rev3 que es la encargada última de cortar o hacer fluir en un sentido u otro la corriente que circula por las bobinas del motor. Una síntesis del hardware relevante de esta placa y la relación con los pines del Arduino puede verse en la figura 12, en la que también se incluyen los elementos más importantes que intervienen en el proceso de control de las intensidades que hacen girar el motor. En el anexo 5 se recoge información más detallada, y tanto en este como en la página web [7] se puede encontrar el esquema eléctrico completo.

En lo respectivo al software, existe una biblioteca, pero esta no se corresponde con el hardware del Motor Shield rev3 por ser nuevo (febrero de 2013, de aquí la utilidad de la figura 12). Por tanto, se han programado las funciones específicas para el movimiento del motor. Estas se basan en generar una secuencia que va cambiando la intensidad que pasa por los distintos polos haciendo que el campo magnético gire y con él arrastre el rotor. Como no se ha instalado ningún tipo de sensor para la posición del conjunto, no hay realimentación de la posición, por lo que lo único que podemos hacer es fiarnos del ajuste experimental, de la fuerza y el tiempo de paso, y esperar que llegue a la posición correcta. Para tener al menos un punto de cero, se instaló un interruptor que hace de final de carrera, por lo que además de estas funciones, también se ha programado una interrupción en el Arduino: ésta se lanza cuando la señal del final de carrera pasa de 0 a 1, e inmediatamente interrumpe el giro del motor y reajusta la posición eliminando cualquier error en ésta que se hubiese podido ir acumulando.

4. CONCLUSIONES Y TRABAJOS FUTUROS

El aspecto más destacable de la solución, seguramente, es su diseño modular que le da una gran flexibilidad. Siendo muy fácil volver a reutilizar los conjuntos en otros experimentos, o incluso añadir otros completamente nuevos modificando solo las partes específicas del software existente, que puede tomarse como una plantilla.

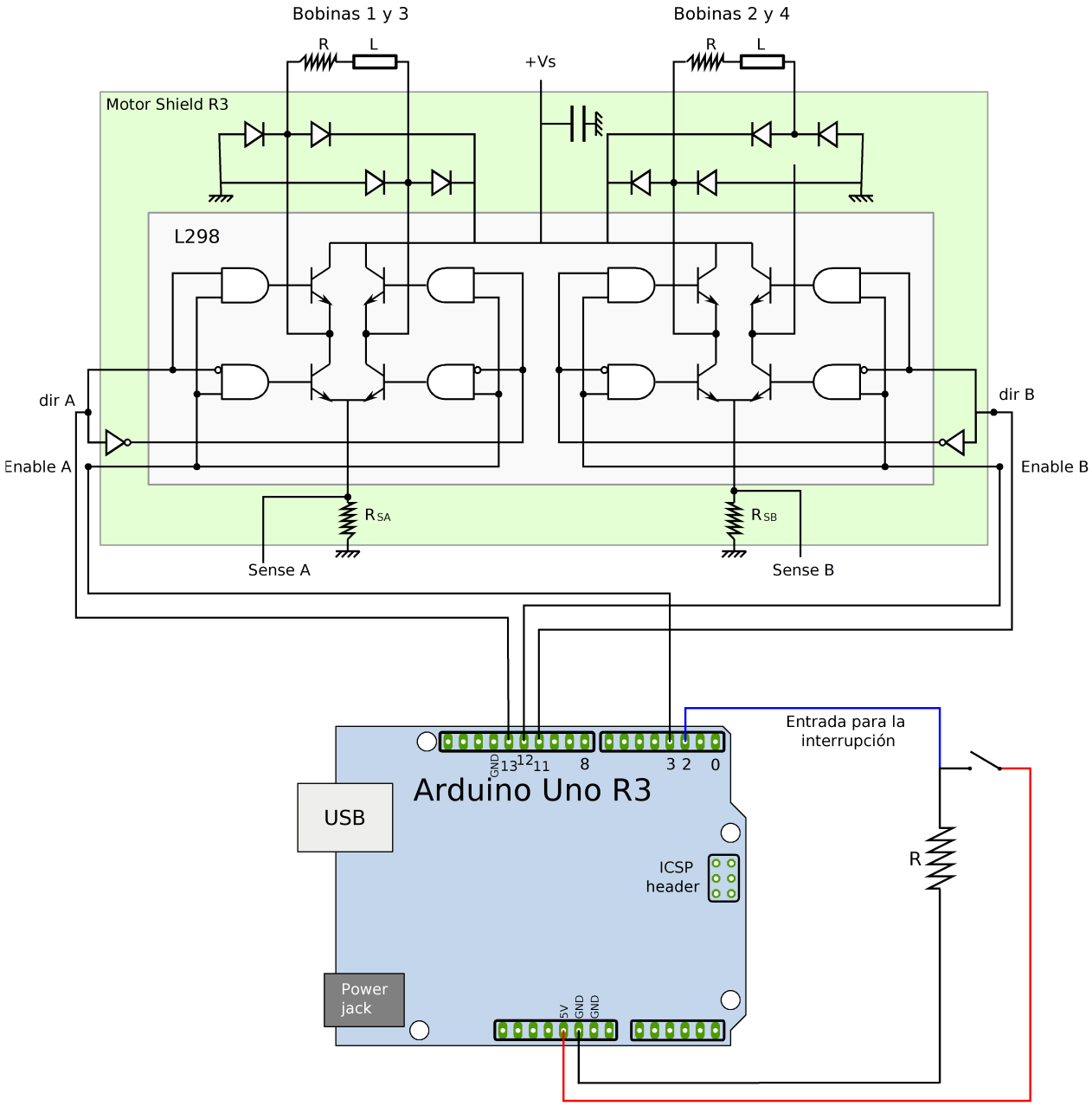


Figura 12. Esquema simplificado del Motor Shield rev3. Y su conexión al Arduino.

Otra característica a reseñar, es que el hardware utilizado es de uso general y está destinado al consumo de masas, por lo que tiende a ser más barato que cualquier otro específico. Además, estos dispositivos han aparecido hace poco, con lo que la solución es bastante novedosa. Por el ejemplo el Rasberry Pi se lanzó al mercado hace año y medio, el 29 de febrero del 2012.

Lo que faltaría hacer, o mejor, lo que no se ha podido hacer, por retrasos en la instalación y puesta en marcha del experimento, es probar el comportamiento del sistema en condiciones de trabajo continuo, y cuál sería su fiabilidad.

Este proyecto deja la puerta abierta a futuras mejoras. Desde el punto de vista de control quedaría cerrar el bucle de control del motor paso a paso, usando el PWM como modulador de la acción e incorporando algún sensor para la realimentación.

También se podrían incorporar métodos de detección y corrección de errores en los mensajes para aumentar la robustez. Un método apropiado sería el de Hamming. Así mismo se podría pensar en usar algún método de *hash*, aunque el uso de estos en mensajes cortos, como los que se envían en esta aplicación, hace que aumente de forma importante la longitud de la cadena.

5. BIBLIOGRAFÍA Y ENLACES

[1] Raspberry Pi

http://www.raspberrypi.org/

[2] Arduino

http://www.arduino.cc/

[3] Python

http://www.python.org/

[4] Learning Python 5th Ed 2013

Autor: Mark Lutz. Editorial O'Reilly Media Inc.

[5] Programming Python 4th Ed 2010

Autor: Mark Lutz. Editorial O'Reilly Media Inc.

[6] PySerial

http://pyserial.sourceforge.net/

[7] Esquema eléctrico del Arduino Motor Shield R3

http://arduino.cc/en/Main/ArduinoMotorShieldR3

6. AGRADECIMIENTOS

Quiero expresar mi agradecimiento a las personas que me han ayudado a la realización de este proyecto fin de carrera, en especial al personal del Laboratorio de Física Nuclear y Astropartículas:

A D. Igor García Irastorza, investigador principal del proyecto en el que se encuadra la solución aportada en este PFC.

A D. Francisco Iguaz Gutierrez, con quien he trabajado en el laboratorio, por su tiempo, receptividad, y con quien el que el intercambio de ideas ha sido crucial para la culminación de este PFC.

A D. Juan Castel por su continua disposición a echar una mano.

Y a D. José Ramón Asensio por toda la experiencia que ha aportado como por su trato personal y ayuda prestada en la realización de este proyecto.

ANEXOS:

ANEXO 1. INFORMACION EQUIPOS

Introducción

Ampliamos la información de los equipos involucrados en el experimento detallando el modelo y dando la página web donde se puede encontrar las características correspondientes al modelo.

Spellman

Modelo: Spellman SL. Fuente de alta tensión.

http://www.spellmanhv.com/en/Products/Rack-Supplies/SL.aspx

Standard Interface control option (SIC). Opción adicional que permite la comunicación por RS232/USB/Ethernet

http://www.spellmanhv.com/en/Products/Accessories/SIC-Option.aspx

CAEN

Modelo: CAEN SY2527. Universal Multichannel Power Supply System.

http://www.caen.it/csite/CaenProd.jsp?idmod=123&parent=20

CAEN HV Wrapper. Biblioteca C. (versión 5.20).

http://www.caen.it/csite/CaenProd.jsp?idmod=835&parent=38

MaxiGauge

Modelo: Maxigauge TPG 256 A. Vacuum measurement and control unit for Compact Gauges.

http://www.pfeiffer-vacuum.com/products/measurement/activeline/controllers/onlinecatalog.action?detailPdoId=3526#product-accessories

Bronkhorst

Modelo: Bronkhorst F-201CV-MGD-33-V

http://www.bronkhorst.com/en/products/gas_flow_meters_and_controllers/elflow_select/

Manual del protocolo FlowBus pa

http://www.bronkhorst.com/files/downloads/manuals_english/917027manual_rs232_interface.pdf

Swagelok

Modelo: Swagelok PTU. Ultra High Purity Transducer.

https://www.swagelok.com/products/measurement-devices/pressure-transducers/ultrahigh-purity-flow-through.aspx

Motor paso a paso

RS 57mm stepper motor,126Ncm 1.8deg. 2.5V

http://es.rs-online.com/web/p/motores-paso-a-paso/5350439/

ANEXO 2. PRESENTACIÓN DE OPCIONES PARA EL SISTEMA DE ADQUISICIÓN Y MONITORIZACIÓN

Objetivo del documento: presentar cinco opciones de sistema de control y toma de datos para el proyecto TREX-DM, resaltando características y diferencias para su comparación, y final selección de una de ellas.

Esquema proyecto

En el esquema que se encuentra al final del documento quedan reflejados los dispositivos usados en el proyecto de forma que sea fácil su identificación y el estudio de su conexionado. Figura 5.

Tabla aparatos-protocolo de conexión

Tabla -protocolos- RS232, RS485, USB, Ethernet, Conexión analógica.

	RS 232	RS485	USB	Ethernet	Analógica
Spellman	✅	❌	✅	✅	❌
Caen	✅	✅	✅	❌	❌
MaxiGauge	✅	❌	❌	❌	❌
Swagelok	❌	❌	❌	❌	✅

Excepto donde no sea posible descartamos el protocolo RS-232 por ser este muy sensible al ruido (no más de 5 o 10 frente a 1km del RS-485). Además, cada conexión necesitaría un puerto de la placa.

La mejor opción es usar RS-485, con este protocolo se pueden conectar hasta 32 dispositivos en cadena, formando una red y usando un solo puerto.

Observando la tabla, vemos que no podemos conseguir agruparlas en un mismo tipo de protocolo, por tanto vamos a necesitar varios tipos de tarjetas de adquisición de datos.

La fuente Spellman de alta tensión la podremos conectar usando USB o Ethernet.

Y por último, a falta de fijar el termómetro y caudalímetro nos queda una señal entrada analógica, que habrá que digitalizar.

Propuestas

1) National Instruments (PC-Módulos) + LabView

Consistiría en un bloque en el que se insertan el PC y los módulos para adquisición de datos. Se usaría LabView como software de programación. Figura 1.

Lema: "Solución empresarial"

+ Empresa Líder en el sector de la instrumentación.

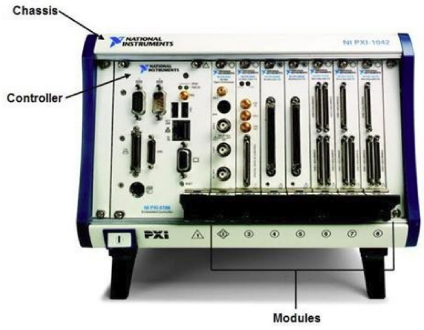


Figura 1. Bloque National Instruments

+ Solución utilizada en el ámbito empresarial. Podemos confíar en su robustez, fiabilidad, facilidad de desarrollo y mantenibilidad.

+ Usa LabView, lenguaje estándar a la hora de buscar continuidad o ampliación.

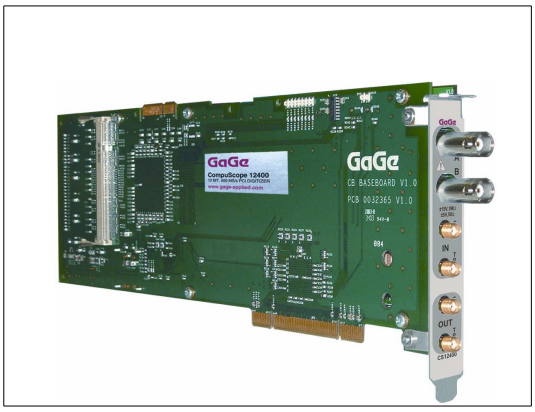


Figura 2. Tarjeta PCI

2) PC + tarjetas + LabView

Usaríamos un PC normal con las tarjetas necesarias para la captura de datos. Y como lenguaje de programación LabView.

Lema: "Solución ajustada a una situación concreta"

+ Más barata que la opción 1.

- Probable aparición de dificultades en el desarrollo (drivers tarjetas-PC) ante los cuales se producirán retrasos y en el caso más grave vías muertas en las que hubiese que cambiar hardware.

+ [LabView] Usar Labview permite una mayor facilidad a que el software sea modificado por otros.

3) PC + tarjetas + Python

Usaríamos un PC normal con las tarjetas necesarias para la captura de datos. Y como lenguaje de programación Python.

Lema: "Solución ajustada a una situación concreta usando software libre"

= Mismas consideraciones que en la opción 2 respecto al hardware.

+ [Python] Software libre. Lenguaje sencillo con amplia gama de bibliotecas usado para prototipado rápido.

- [Python] llevará muchas más horas de desarrollo del software.

- [Python] Aunque el programa resultante esté bien documentado, el uso de un software que no está estandarizado en la industria, sumado a que será un trabajo individual, conducirá, seguramente, a que sea difícil de retomar o modificar por otros.

4) Red Ethernet con Raspberry Pi y Arduinos.

Un Arduino es un microcontrolador que usaríamos para la adquisición de datos, mientras el Raspberry Pi es un ordenador del tamaño de una tarjeta de crédito en el que instalaríamos un servidor con el que nos comunicaríamos por TCP/IP desde cualquier otro ordenador. Figura 3.

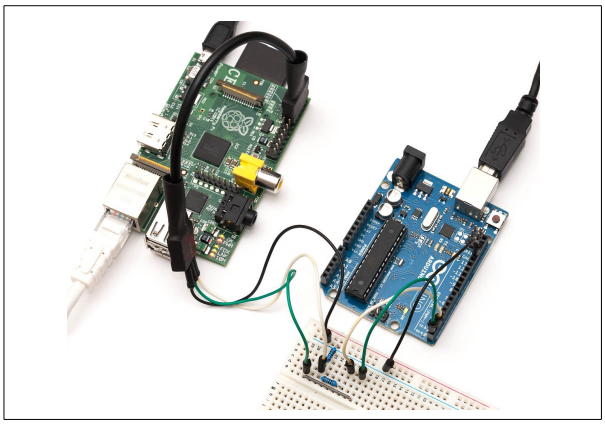


Figura 3. Raspberry Pi y Arduino

Lema: "Solución experimental e innovadora"

- Robustez y fiabilidad son una incógnita. Es algo muy nuevo

- [Python] Software libre. Desarrollo

Figura 4. PLC y Sistema SCADA

costoso en tiempo.

+ Opción barata.

+ Reducción del ruido en la adquisición de datos. TCP/IP es también muy robusto al ruido.

+ Exportable al resto del laboratorio.

+ Modularidad

5) PLC + tarjetas + SCADA

Consistiría en un PLC (Programmable Logic Controller) o autómata programable, este es un dispositivo que ejecuta secuencialmente órdenes, sin necesidad de un sistema operativo al uso (Windows, Linux, BSD..) a la vez que interactúa con el exterior. Necesitaría de las tarjetas de adquisición de datos que se pueden añadir modularmente, y finalmente, un sistema SCADA para control y monitorización remota. Figura 4.

Lema: "Solución Robusta"

+ Es la opción más robusta y fiable. “No se colgaría en 20 años”.

- El programa del PLC es más difícil de modificar o reutilizar.

- Coste, opción más cara que las anteriores, tanto en hardware como en software.

Presupuesto

PXI de N.I.		PC + LabView		PC + Python		Red Arduino Rbpi	
LabView	5.800 €	LabView	5.800 €	Python	0 €	Python	0 €
Controlador	1.700 €	PC	400 €	PC	400 €		
Módulo RS-485	592 €	Tjta C422 2xRS485	225 €	Tjta C422 2xRS485	225 €	Arduino (x2)	50 €
Módulo Analógico	1.330 €	Tjta PCI 9112 analog.	450 €	Tjta PCI 9112 analog.	450 €	Raspberry Pi (x4)	200 €
Chasis 5+1	1.209 €					Router/Switcher	100 €
Cables		Cables		Cables		Cables	
	10.630 €		6.875 €		1.075 €		350 €

Tabla resumen

	Característic.	Fiabilidad	Desarrollo	Coste	Mantenibilid.	Anotación
National Instruments	Solución empresarial	✓	✓✓✓	✗	✓	
PC + LabView	Solución al caso	✓	✓	✓	✓	
PC + Python	Solución al caso	✓	✗	✓✓✓	✗	Formación
Red Arduino y Raspberry	Solución experiment.	✗	✗	✓✓✓	✗	Ampliación
Automata programble	Solución robusta	✓✓✓	✗	✗✗✗	✗✗✗	

Fiabilidad: comprendería tanto el hardware como el software.

Desarrollo: haría mención a las horas de trabajo para instalar todo el sistema. Hardware y software.

Coste: desembolso total.

Mantenibilidad: facilidad para que otra persona modifique el sistema.

Conclusión

Propondríamos una solución mixta entre la opción 3 y 4.

Esta consistiría en ensayar la solución con Arduino y Raspberry Pi en los sensores de temperatura, la cual se adapta muy bien porque al colocar el conjunto pegado al sensor reducimos el ruido. Paso seguido valoraríamos su funcionamiento. Y según sea este, podríamos seguir con esta opción o cambiar a usar un PC central con las tarjetas de adquisición sin mucho costo añadido.

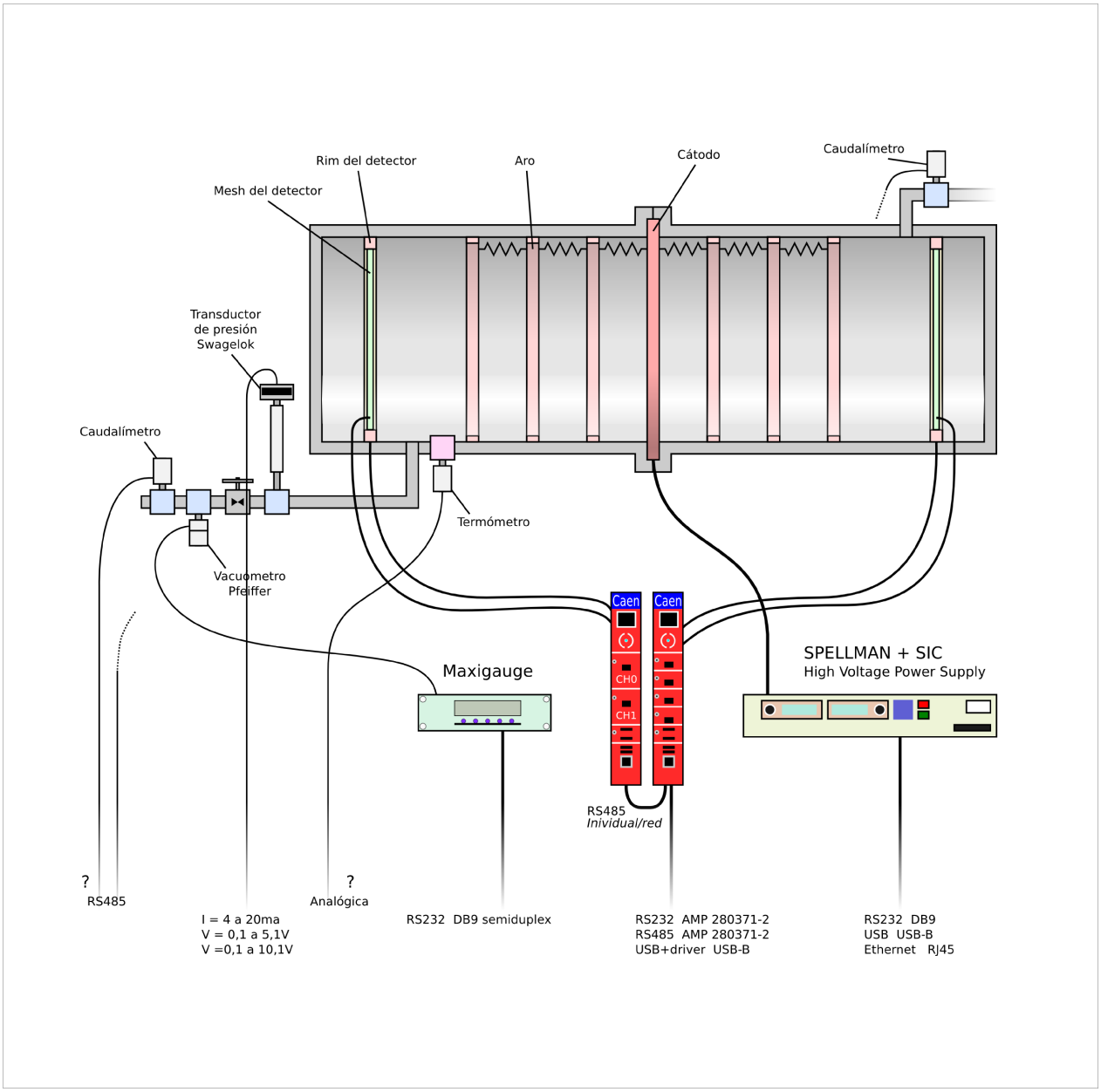


Figura 5. Esquema del sistema

ANEXO 3. ESTRUCTURA DE LA INTERFAZ GRÁFICA

Introducción

Dos ideas: (1) Tareas en paralelo par no bloquear la GUI. (2) La GUI es un recurso compartido al que se debe acceder en exclusión mutua.

La interfaz gráfica (GUI) utilizada para la comunicación con el usuario se ejecuta en un modo llamado conducido por eventos, es decir, que está durmiendo esperando, por ejemplo la pulsación con el ratón de un botón, para ejecutar el código relacionado con ese evento. Para no bloquear la GUI, en vez de pasarle el control y esperar el tiempo que tarda en realizar la tarea, la lanzamos en paralelo. Sin embargo, tener un programa con varias tareas corriendo en paralelo, impone, que tengamos un especial cuidado con recursos compartidos (variables o GUI) a la hora de escribir y leer datos.

Esquema y breve explicación de sus elementos

La estructura del programa en ejecución quedaría reflejada en la figura 1.

Explicación breve de cada elemento representado.

Entorno del programa

El entorno del programa sería el marco donde coexisten tareas con su control de ejecución, estructuras como la cola FIFO, y las varibles globales accesibles por todas las tareas.

Tarea principal

Es el bucle de la interfaz gráfica. A partir de esta, según se demande, se lanzan otras tareas en

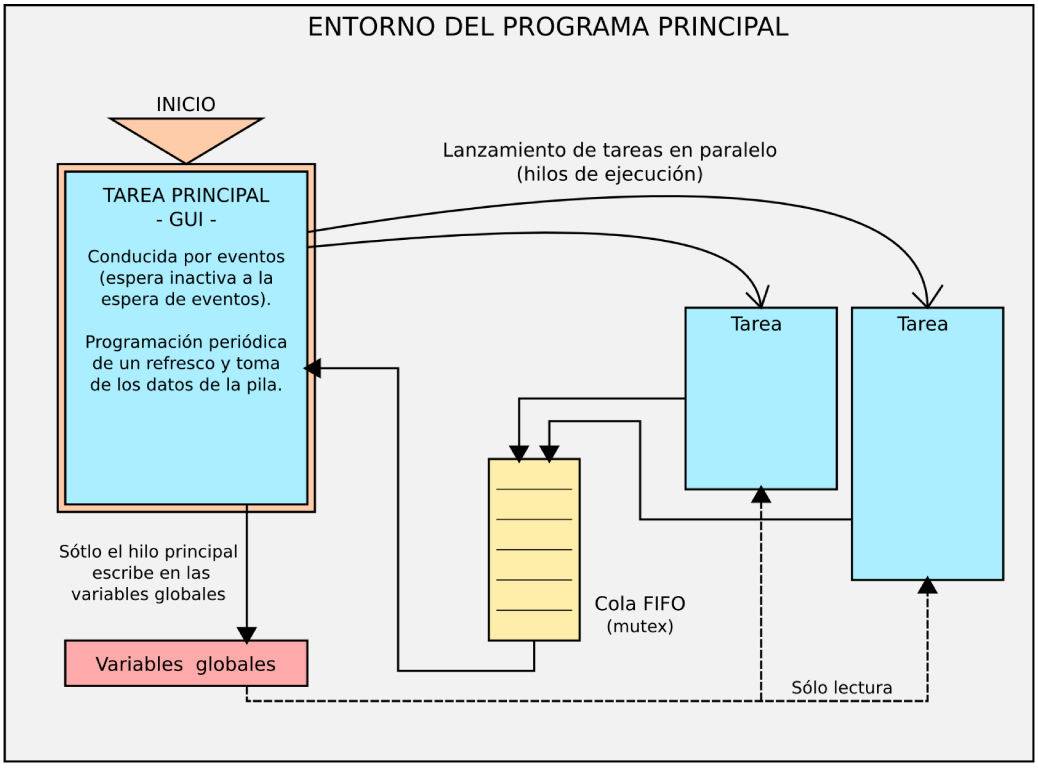


Figura 1. Estructura del programa

paralelo.

El modo en que trabaja es conducido por eventos. Espera durmiente a que ocurra un evento para ejecutar el código correspondiente. Para no congelar la interfaz, este código se lanza como tareas en paralelo.

Para actualizar la GUI programamos una acción periódica que se encargará de refrescar las variables mostradas.

Tareas en paralelo

Las tareas en paralelo son trozos de código cuya ejecución se realiza al mismo tiempo y sin parar la tarea principal. El resultado de su trabajo se guarda en una cola FIFO a la que se accede en exclusión mutua*.

* Exclusión mutua. Se trata de evitar el problema de que al usar un recurso compartido se produzcan inconsistencias. Por ejemplo, si dos tareas intentan escribir una cadena de caracteres en la salida estandar al mismo tiempo, ambas se entremezclarían perdiendo su sentido.

Cola FIFO en exclusión mutua

Es una estructura que va guardando los resultados del trabajo efectuado por las tareas en paralelo. Los datos luego son recogidos periódicamente por la tarea principal y mostrados en la GUI.

Variables globales

Son accesibles por tadas las tareas que comparten el mismo entorno. En este caso son variables booleanas. Garantizamos la consistencia de los datos permitiendo que sólo la tarea principal escriba en ellos, y el resto solo las puede leer.

Estas variables globales tienen como función finalizar las tareas en paralelo.

Salida ordenada

Una vez que al programa le llega la orden de finalizar, este sale del bucle de la GUI y ordena el cierre del resto de tareas por medio de las anteriores variables globales. Una vez que todas la tareas han finalizado se sale completamente del programa borrando de memoria el entonrno del programa.

Secuencia de funcionamiento

Para explicar como funciona vamos a seguir una ejecución del programa.

Inicio

Al iniciar el programa se crearía el entorno del programa. Este contendría la tarea principal en ejecución, y las variables, tanto las globales como la pila FIFO. Figura 2.

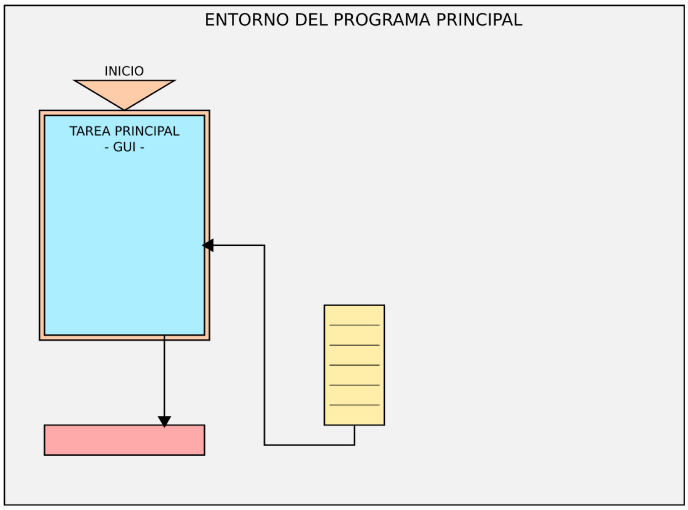


Figura 2. Entorno inicial

En este momento inicial solo habría una tarea en ejecución. Esta sería la GUI, la cual, está haciendo sólo dos cosas. Una,

esperar cualquier evento, como puede ser la pulsación con el ratón de un botón. Y dos, refrescar periódicamente los campos de las variables. Esta segunda, es una función que es llamada periódicamente por la GUI, pasándole el control, y cuya misión es leer, si los hubiese, datos de la pila FIFO y mostrarlos por pantalla.

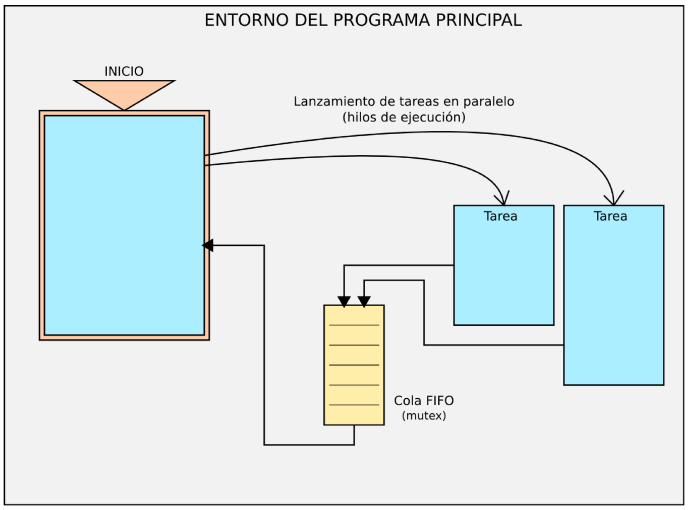


Figura 3.Lanzamiento de tareas periódicas

Lanzamiento de una tarea periódica

Ahora, imaginemos que pulsamos, con el ratón, el botón que inicia la lectura continuada de un sensor y que mostrará su valor en un campo de la GUI. Esta pulsación inicicará una nueva tarea que trabaja en paralelo, ya que en caso contrario, la GUI quedaría congelada hasta que esta tarea acabase su trabajo, y que se encargará de leer los datos del sensor y proporcionárselos a la GUI.

Puesto que podemos tener varias tareas en paralelo, hay que tener cuidado con los recursos compartidos, por ejemplo la GUI. No podemos permitir que tareas independientes la actualicen al mismo tiempo. Para salvar este problema, hacemos que todas estas tareas escriban los datos que generan en una cola FIFO a la que se accede en exclusión mutua. Periódicamente, la GUI los va tomando para representarlos en pantalla.

Salida ordenada del programa

Cuando se envía la señal de cerrar la GUI, es exactamente eso, cerrar la GUI y no el programa. Lo que ocurre es que se sigue ejecutando las instrucciones del programa principal que se encuentran a continuación del bucle de la GUI. En este trozo de programa, lo que hacemos es, por medio de unas variables globales, indicar a las tareas que se ejecutan en paralelo que deben finalizar. Una vez hecho esto la tarea principal acaba, pero no se sale del programa hasta que el resto de las tareas ha acabado. Cuando finalmente todas acaban se borra el entorno del programa con todas sus estructuras de la memoria.

ANEXO 4. ACCESO A FUNCIONES C DESDE PYTHON

Explicamos el módulo ctypes de Python y varias soluciones a puntos problemáticos que nos pueden aparecer.

Índice

El módulo ctypes

Python permite múltiples formas de usar funciones de C desde su código. Nos vamos a centrar en el uso del módulo ctypes para acceder a funciones de C compiladas en una biblioteca de enlace dinámico (DLL) u objeto compartido (.so) en *nix.

Ctypes es un módulo de la biblioteca estándar de Python, es decir que se distribuye al mismo tiempo que el intérprete de Python, y por tanto, está presente en todas las implementaciones del lenguaje. Ctypes contiene métodos que envuelven la biblioteca (dll o so) creando un objeto en el que los atributos se corresponden con las funciones de C. Permitiendo acceder a estas desde código interpretado de Python.

Como ejemplo introductorio vamos a usar la función printf de C desde el intérprete de Python.

```
Python 3.3.2
>>>import ctypes
>>>libcpy = ctypes.cdll.LoadLibrary('libc.so.6') #ruta completa
/usr/lib/libc.so.6 (o lib64)
>>>libcpy.printf(b'hola\n') # todas las cadenas en Python 3 son unicode con lo
que hay que especificar una cadena de bytes con el prefijo b.
5     # número de caracteres. Salida de la función.
hola  # la cadena esperada. Acción de la función.
```

Soluciones a puntos problemáticos

Cuando uno se pone a trabajar con funciones de C se encuentra que ciertas construcciones son extrañas, desde el punto de vista de Python, y que no tienen una solución inmediata. Para introducir estos problemas vamos a crear una biblioteca de enlace dinámico con varias funciones que ejemplifican cada uno de estos casos y sobre las que probaremos las soluciones.

Creación de una biblioteca con funciones C de prueba

Las funciones están distribuidas en cada ejemplo para facilitar la explicación, pero podemos crear un único fichero con todas ellas y compilarlas con gcc. Digamos que le hemos puesto el nombre de

simple.c y a partir de este vamos a crear con los siguientes pasos una biblioteca de enlace dinámico u objeto compartido en Linux.

1) Compilar creando código de posición independiente:

```
gcc -Wall -fpic -c simple.c
```

Siendo -c la opción para que compile y -Wall significa que muestre todos los warnings.

2) Crear la biblioteca o el objeto compartido a partir del fichero objeto simple.o

```
gcc -shared -o libsimple.so simple.o
```

Añadir el prefijo 'lib' al nombre del fichero de salida indicado con la opción -o, de output.

Función C genérica

Con la función 'suma' vamos a dar un ejemplo de como usar tipos de variables básicas, tales como enteros y apuntadores.

Código de la función C.

Función "suma" de una dll u objeto compartido

```
// simple.c

// codigo de la biblioteca de enlace dinamico libsimple.so
// conjunto de funciones

// suma dos enteros
int suma (int a, int *b) {
       return ( a + *b);
       }
```

Creamos el objeto compartido libsimple.so como se ha indicado en el apartado anterior y lo guardamos en la misma carpeta que el programa de Python desde donde se va a usar.

Programa Python que usa la función suma:

Uso de "suma" desde Python

```
# sample.py
import ctypes

# Creamos objeto CDLL que envuelve libreria de C
libSimple = ctypes.cdll.LoadLibrary('./libsimple.so')

# Funcion suma en Python
suma = libSimple.suma # suma = atributo del objeto, tipo CDLL, libSimple
suma.argtypes = (ctypes.c_int, ctypes.POINTER(ctypes.c_int))
suma.restypes = ctypes.c_int

# Llamada a la función suma
a = ctypes.c_int(34)
b = ctypes.pointer(ctypes.c_int(6))
print(suma(a,b)) # resultado 40
```

Explicación del script de Python:

Creamos el objeto libSimple que es el que envuelve la biblioteca C. Siendo cada atributo de este objeto una función de la biblioteca. Así, libSimple.suma hace referencia a la función 'suma'.

A continuación, con argtypes y restypes indicamos que tipos de parámetros deben ser pasados a la función. No es estrictamente necesario, pero en caso de no enviarle el parámetro adecuado intentará convertirlo al tipo correspondiente o dará un error dentro de Python. La lista de todos los tipos de ctypes se encuentra en la página oficial de Python (http://docs.python.org/3/library/ctypes.html).

Por último, hacemos una llamada a la función suma en el que primero hemos creado explícitamente los parámetros correspondientes en el tipo adecuado. En caso de enteros, cadenas y bytes la conversión es automática. Pero no para el resto.

En los punteros distinguimos las dos instrucciones POINTER y pointer en las que la primera crea una clase correspondiente a un tipo y la segunda una instancia del tipo correspondiente.

P = ctypes.POINTER(type) crea y devuelve un nuevo tipo de puntero de ctypes. <class ....>

p = ctypes.pointer(obj) crea una instancia del puntero que apunta al objeto del tipo POINTER(type(obj))

Función de C que usa punteros como parámetros de entrada y reusados como salida

Una técnica, comúnmente usada en C, utiliza punteros en los argumentos tanto para pasar un valor a la función como para recogerlos es similar al uso de tipos mutables de Python. En el fondo, es el mismo concepto de puntero. Sin embargo, hay que hacerlo explícito en Python para no encontrarnos con variables que siendo 'no-mutables', como números (enteros, reales) y cadenas, de repente, se comportan como 'mutables'. Para ajustarnos a la forma de actuar de Python crearemos funciones intermedias que se comportarán exteriormente como es esperable de una función de Python.

Función "division_entera" en C

```
// division entera
int division_entera (int *D, int d) {
       int resto = *D % d;
       *D = *D / d;
       return resto;
       }
```

Uso de "division_entera" desde Python

```
# Funcion division entera
_division_entera = libSimple.division_entera
_division_entera.argtypes = (ctypes.POINTER(ctypes.c_int), ctypes.c_int)
_division_entera.restypes = ctypes.c_int

# Funcion para usar en Python
def division_entera(divisor, dividendo):
       D = ctypes.c_int(divisor) # ctypes.c_int es mutable
       resto = _division_entera(D, dividendo) # enteros y cadenas son
automáticamente convertidos
```

```
        # el tipo entero de ctypes tiene el atributo value que es un entero python
        return (D.value, resto)

# Llamada a la funcion division entera
res = division_entera(7,2)
print('Conciente =', res[0], ' Resto =',res[1]) # Resultado: Cociente = 7 Resto=
1
```

Función de C con arrays

Ahora vamos a ver como tratar con arrays. En Python una conversión natural sería una lista o una tupla, así que vamos a construir otro ejemplo en el que hallaremos la media de una serie de valores guardados en una lista.

Antes, un resumen de lo básico para construir arrays.

```
Array5int = ctypes.c_int*5              # clase de un array de 5 enteros
array5 = (ctypes.c_int*5)(1,2,3,4,5)    # instancia de un array de 5 enteros
array5 = Array5int(1,2,3,4,5)           # construye la misma instancia de un array
de 5 enteros
lista = [1, 2, 3, 4, 5]                 # definimos una lista de Python
array5int = (ctypes.c_int*5)(*lista)    # *lista no es un puntero, es la forma de
pasar la lista elemento a elemento a la función.
```

Respecto al uso de punteros, en general, ctypes hace un chequeo estricto de los tipos. Esto significa que, si en la lista de argumentos de una función (.argtypes) se tiene (c_int) solo serían aceptadas instancias de ese tipo. Sin embargo, hay una excepción a esta regla en el caso de punteros, estos admiten instancias de arrays compatibles. Por ejemplo, POINTER(c_int) admite no solo enteros sino arrays de enteros. Por otro lado, POINTER(c_int*5) fija el tamaño y no admite arrays de otras dimensiones, aunque si enteros en lo que debería ser un error.

Como en los apartados anteriores se a escrito una función que recoge todo anterior a modo de modelo. En este caso la función media espera un array de enteros largos y un entero con su número para hacer la media y devolverla como otro entero largo.

Función "media" en C

```
// media
long int media(long int *lista, int n) {
        int i;
        long int total = 0;
        for (i=0; i<n; i++) {
                total += lista[i];
        }
        return total/n;
}
```

Uso de la función media desde Python creando otra función intermedia.

Uso de la función "media" desde Python

```
# Funcion media
_media = libSimple.media
# Siendo una excepción, un puntero de un tipo acepta arrays de ese mismo tipo
```

```
_media.argtypes = (ctypes.POINTER(ctypes.c_long), ctypes.c_int)
_media.restypes = ctypes.c_long

def media(lista):
       # *lista, no es un puntero, es la forma de pasar la lista elemento a elem.
       return _media( ((ctypes.c_long)*len(lista))(*lista), len(lista) )

res = media([3,5,1,7])
print ('Media = ',res)
```

Referencias:

Documentación de ctypes en la página oficial de Python.

http://docs.python.org/3/library/ctypes.html

ANEXO 5. CONTROL DE UN MOTOR PASO A PASO CON ARDUINO Y MOTOR SHIELD (rev3)

El problema que queríamos resolver utilizando un motor paso a paso era el de llevar un dispositivo, conducido por una sirga dentro de un tubo, a ocho posiciones de forma repetitiva. En este documento explicaremos el funcionamiento de un motor paso a paso híbrido, y su control con un Arduino rev.3 y un motor shield rev.3. Un "shield" es un circuito integrado preparado para conectar directamente al Arduino.

CONTENIDO

1. Esquema y breve explicación de un motor paso paso

Hay varios tipos de motores paso a paso, estos pueden ser de reluctancia variable, de imán permanente, e híbridos. Nos vamos a centrar en este último tipo que es que se ha utilizado en nuestro proyecto. El principio de funcionamiento es similar en todos ellos [ver enlace 1]. Este se basa en poder controlar de una forma diferenciada la corriente que pasa por las bobinas, y así, ir creando un campo magnético al que el rotor sigue para alinearse con él en su busca de un camino de reluctancia mínima.

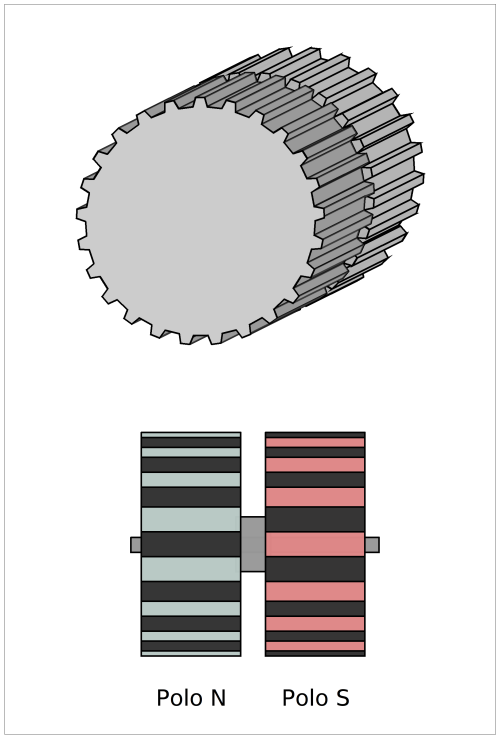
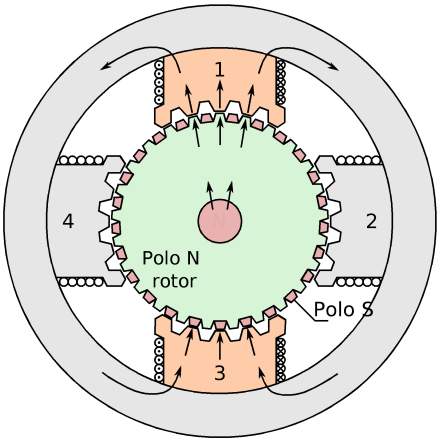


Figura 1. Rotor de 50 dientes de un motor paso a paso híbrido.

1.1 Motor híbrido

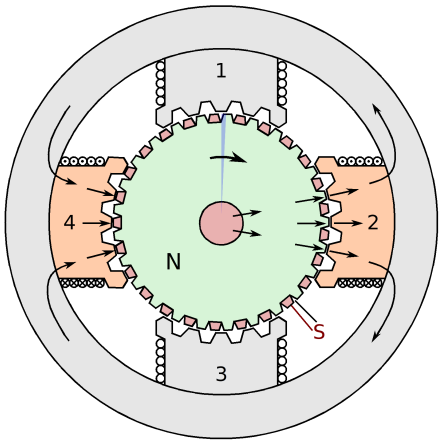
El motor híbrido consiste en un rotor, formado por un imán permanente dividido en dos discos, o ruedas, y que presenta una serie de dientes alternados, figura 1. Y en un estátor con varios polos bobinados. Este tipo de motores consiguen una resolución entre 100 y 400 pasos por vuelta.

Para entender el funcionamiento vamos a seguir la serie de pasos presentados en la figura 2. Iniciamos la secuencia desde el estado en el que la intensidad paso por las bobinas 1 y 3 del estátor, y en el que los dientes del polo N del rotor están alineados con los dientes del polo 1 del estátor, al igual que los del polo S del rotor lo

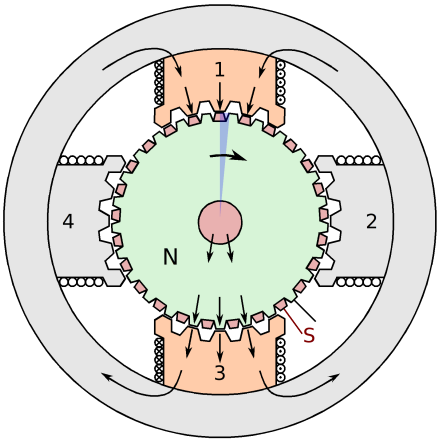


Recordamos que el rotor es un imán permanente constituido por dos ruedas dentadas en la que cada una es un polo.

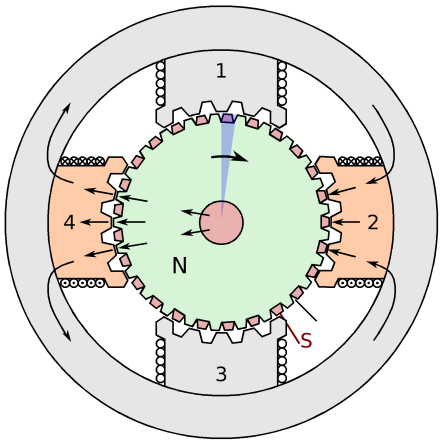
- Paso 0 -
La corriente circula por las bobinas 1 y 3.
Los dientes del polo N del rotor se alinean con los de la bobina 1 del estatpr, mientras las del polo S lo hacen con los de la bobina 3.



- Paso 1 -
La corriente circula por las bobinas 2 y 4.
El rotor gira 1/4 de diente en el sentido de las agujas del reloj para alinear el polo N con la bobina 2, y el polo S con la bobina 4. De esta forma, el rotor gira buscando crear siempre un camino en el que la reluctancia disminuya.



- Paso 2 -
La corriente circula en sentido contraria al que lo hacía en el - paso 0 - por las bobinas 1 y 3.
El rotor sigue girando para alinear el polo N del rotor con los dientes de la bobina 3 y el S con los de la 4.



- Paso 3 -
La corriente circula por las bobinas 2 y 4 en el sentido contrario al que lo hacían en el paso 1.
El rotor gira para alinear el polo N del rotor con los dientes de la bobina 4 y el Sur con los de la 2.
En cada paso se gira 1/4 de diente, y al ser un rotor de 25 dientes, cada paso equivale a 1/100 de vuelta.
El siguiente paso será equivalente al paso 0, iniciando una nueva secuencia.

Figura 2. Secuencia de paso completo de un motor híbrido.

están con los del polo 3. En el siguiente paso, la intensidad deja de pasar por las bobinas 1 y 2 activándose ahora los polos 2 y 4. El rotor se mueve en el sentido de las agujas del reloj hacia posiciones de menor energía en las que los dientes se alinean creando un circuito magnético de mínima reluctancia. La cantidad girada en cada paso es de ¼ de diente, como el rotor representado tiene 25 dientes, este habrá girado 1/100 de vuelta. Los siguientes pasos pueden seguirse en las figura 2 constituyendo una secuencia que se va repitiendo para girar el rotor.

No es la única forma de hacer girar el motor. El expuesto en la figura 2 correspondería con el denominado de paso completo. Pero existen otros modos entre los que los más comunes son el de medio paso en el que hay cuatro pasos extra intercalados y en los que todas las bobinas están activas. Y el micro-stepping, en el que las corrientes varían de una forma continua y no en escalones todo o nada; [enlace 1].

1.2 Circuito RL y par motor

Los dos términos más importantes en el par motor, son la intensidad de la corriente y la posición relativa del rotor respecto del campo magnético. [Enlaces 1 y 2].

Respecto a la intensidad, el par (o la fuerza si se fija un radio) es, en base a la Ley de Lorentz, proporcional a la intensidad que recorre las bobinas. El par máximo es un dato de diseño que aparece en las hojas de características y que depende de la cantidad de calor que puede disipar las bobinas al paso de la corriente. Por tanto, hay que tener cuidado en exceder esa intensidad. También, hay que tener en cuenta que a este valor máximo no se llega instantáneamente, sino que se hace de una forma progresiva ya que es un circuito con una alta inductancia que hace de inercia (circuito R-L). El comportamiento puede verse en la figura 3.

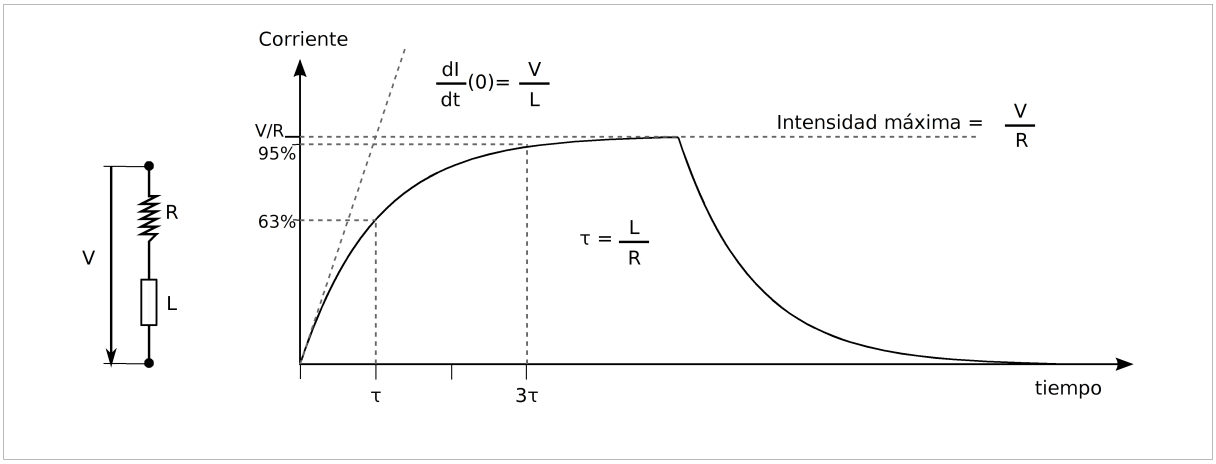


Figura 3. Intensidad en un circuito RL.

El otro factor es la posición relativa. La fuerza va desde el valor 0, cuando están perfectamente alineados los dientes del rotor con los de los polos de las bobinas, y de allí, se va incrementando hasta un máximo cuando hay un desplazamiento de ¼ de diente (que equivale a que el 50% de la superficie del diente este enfrentada), para luego disminuir hasta un mínimo inestable en el que los dientes están alternados. La figura 4 representa el par, o la fuerza, en función del desplazamiento entre el rotor y el estátor.

Adelantándonos al punto 4 en el que tratamos el ajuste del par, vamos a describir como un motor daría un paso estando bajo la acción de un par resistente. Supongamos que este par es de valor la

mitad al valor máximo y que se parte de una posición a de equilibrio. Entonces avanzamos un paso activando y desactivando las bobinas correspondientes, la nueva posición con respecto a la alineación de los dientes pasaría a ser a'. Como el par del motor es superior al resistente hace que el rotor se mueva hacia b, que será la siguiente posición de equilibrio.

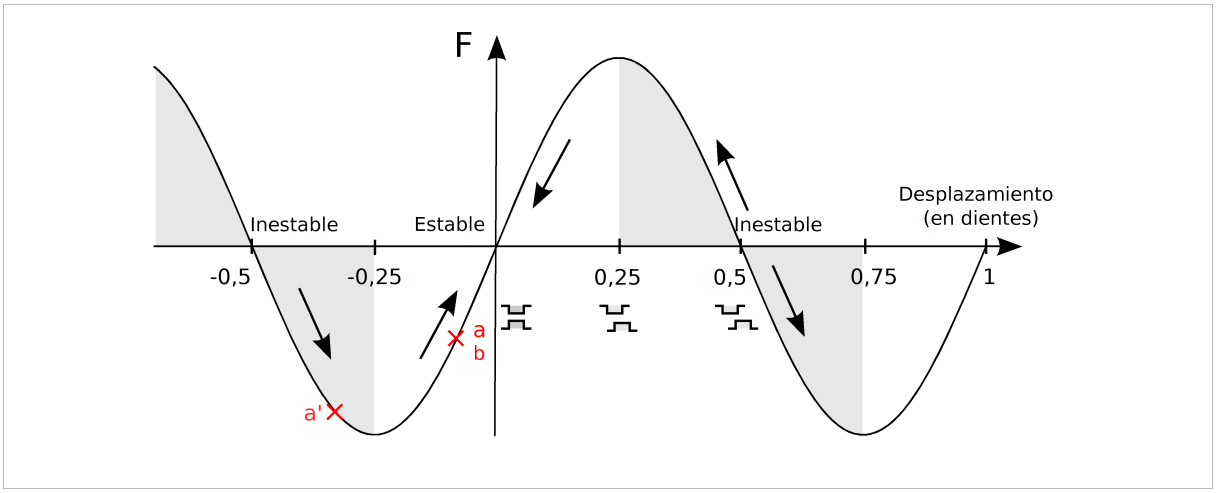


Figura 4. Fuerza en función del desplazamiento entre dientes..

Otro parámetro que aparece en las hojas de especificaciones es el par de retención (holding torque). Este hace referencia al par que muestra el motor a un cambio de posición cuando ya no pasa ninguna corriente. Este fenómeno aparece por la imanación del material ferromagnético y es interesante a la hora del diseño porque podemos saber si el rotor va a mantener la posición una vez que se corte la intensidad.

2. Arduino y Arduino Motor Shield

En el apartado anterior hemos visto como es por dentro y como funciona un motor paso a paso. Ahora, nos vamos a centrar en el dispositivo de control que gobierna la intensidad que pasa por las bobinas.

El dispositivo de control consta de un micro-controlador, en nuestro caso un Arduino Uno rev.3 [figura 5], y dos puentes completos (double full-bridge driver) que es el encargado último de cortar o dejar fluir la corriente por las bobinas. Para construir este último tendríamos tres opciones. Primera, crearlo desde los componentes básicos. Segunda, usar componentes ya empaquetados. Y tercera, que es la que se ha utilizado, es un Arduino Shield.

Un Arduino Shield [figura 6] es un placa ya preparada para conectar al Arduino que trae consigo un L298 que incorpora dos drivers de puente completo.

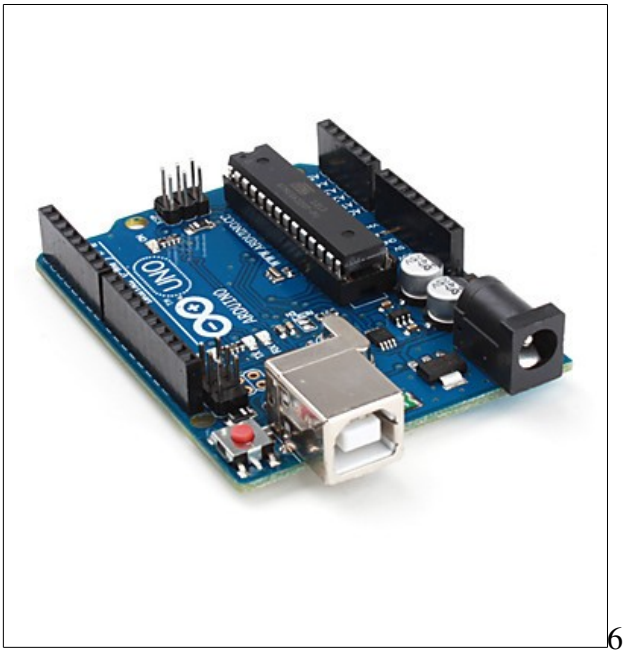


Figura 5. Arduino Uno rev.3.

2.1 Circuito del Arduino Motor Shield

El esquema del circuito completo, por ser libre, puede descargarse de la página web oficial. [Enlaces 3 y 4]. El componente principal es un L298, dual full-bridge driver, que como hemos dicho antes se encarga de dirigir el sentido de las corrientes en la bobinas y que explicaremos en el siguiente apartado. También incluye unas pequeñas resistencias de 0.15 ohmnios con sus correspondientes amplificadores operacionales que nos pueden servir para controlar la intensidad que circula (existe la posibilidad de cortocircuitarlas). La figura 6(a) resume lo más importante del esquema.

El 'Arduino Motor Shield r3' tiene sus límites, sobre todo en lo concerniente a la intensidad máxima que puede tratar. Estos vienen dado sobre todos por el componente principal que es el L298. Es conveniente mirar la hoja de características [enlace 5] antes de conectar el motor. Pero podemos tomar el valor de 2A como límite de la zona segura, para valores más altos hay que tener en cuenta la alternancia y la duración de intensidad máxima. Por ejemplo podemos tener 2.5A durante 80% de 10ms y el resto, un 20%, sin circular. Respecto al voltaje, el mismo que alimenta la fuente puede usarse para alimentar el Arduino (micro-controlador) y en ese caso el voltaje debería estar entre 5 y 12V.

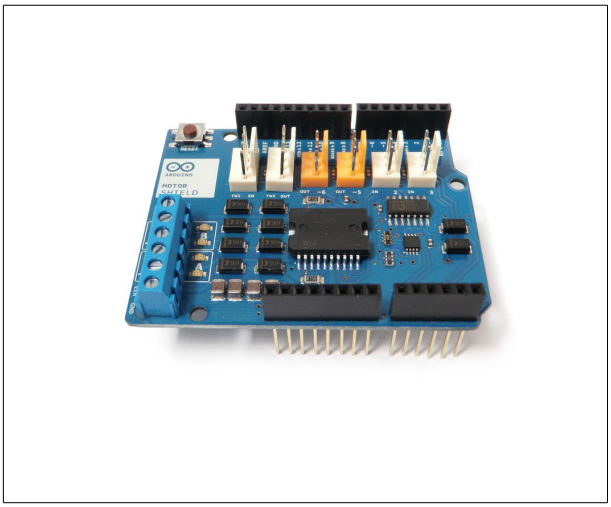


Figura 6. Arduino Motor Shield rev.3

A la hora de conectar el motor, del cual seguramente tendremos las especificaciones de par, voltaje, resistencia, inductancia, debemos tener en mente que el voltaje de la fuente no es el que se ve entre las bobinas del motor en funcionamiento. Esto se debe a que hay intercalados varios transistores y una resistencia y provocan una caída de tensión y que depende de la intensidad, por ejemplo podemos encontrarnos una caída de 1.7V cuando la intensidad es de 1.5A. Podemos encontrar los valores en las hojas de especificaciones del L298, y luego añadir los de las resistencias del 'motor Shield'.

2.2 Control del Motor Shield con el Arduino

A día de hoy, la información para controlar un motor paso a paso con un Arduino y su 'Motor Shield' es bastante escasa, por no decir inexistente. Y casi toda la que se puede encontrar trata del control de motores de corriente continua. Incluso, todas las referencias a relación entre los pines del Arduino y su función en un motor son las correspondientes a las de uno corriente continua, PWM, Brake y Direction. Por tanto, lo que ahora vamos a ver cómo funciona el puente completo y que pines del Arduino controlan cada función para luego diseñar el control de un motor paso a paso.

Función	A	B
Dirección	D12	D13
PWM/Enable	D3	D11
Brake	D9	D8

Sensor de corriente	A0	A1

En la figura 7(a) podemos ver qué relación existe entre cada pin del Arduino y el conductor de puente completo. Y en las figuras 7(b) y 7(c), se puede obtener qué valores deben tener las puertas para conducir la corriente en un sentido u en otro. Faltaría representar el estado en el que no se permite la circulación que simplemente es poner la entrada 'enable' a cero.

Es importante no olvidar, que en circuitos con altas impedancias no se puede cortar la corriente y debe proporcionarse siempre un camino. Si no se hiciese, tendríamos un voltaje muy alto debido a que V=L·dI/dt=L·infinito. Este camino viene dado por unos diodos que al cerrar los transistores crearían un circuito de descarga.

Tampoco hemos usado la función brake. Este lo que hace es crear un camino que cierra el circuito de la bobina. Si alguien está interesado y mira el esquema del circuito, la puerta NAND con una 'e' en el interior es una puerta XNOR (exclusive not OR).

Ahora, construimos la siguiente tabla en la que relacionamos cada paso de la secuencia de giro del motor, figura 3, con el conjunto de valores correspondientes a las entradas del 'motor shield'.

	Enable A	Dir A	Enable B	Dir B	Bobina A	Bobina B
Paso 0	1	1	0	0	→	X
Paso 1	0	1	1	1	X	→
Paso 2	1	0	0	1	←	X
Paso 3	0	0	1	0	X	←
STOP	0	---	0	---	X	X

La conexión entre el PWM y el enable del L298 es muy interesante, aunque no se ha usado en este proyecto, se puede utilizar para controlar el par, aunque hay que tener en cuenta que no va a ser lineal por el tipo de crecimiento de la intensidad.

3. Programación

En este punto explicamos el programa introducido en el y Arduino que se encarga de ir dando las señales necesarios para ordenar la realización de cada paso.

Para generar las señales va siguiendo la secuencia de la tabla anterior, en la que si se sigue en una dirección u otra dará un sentido de giro u otro.

3.1 Variables y funciones

Variables:

absPosStep posición absoluta en pasos, entero largo.

whatstep clase matemática del estado del paso del motor, los valores pueden ser 0, 1, 2, 3.

nsteps numero de pasos a dar, es un entero largo.

sentido sentido del giro

tstep tiempo para cada paso

Funciones:

step0, step1, step2, step3 funciones que establecen los valores en los pines de salida

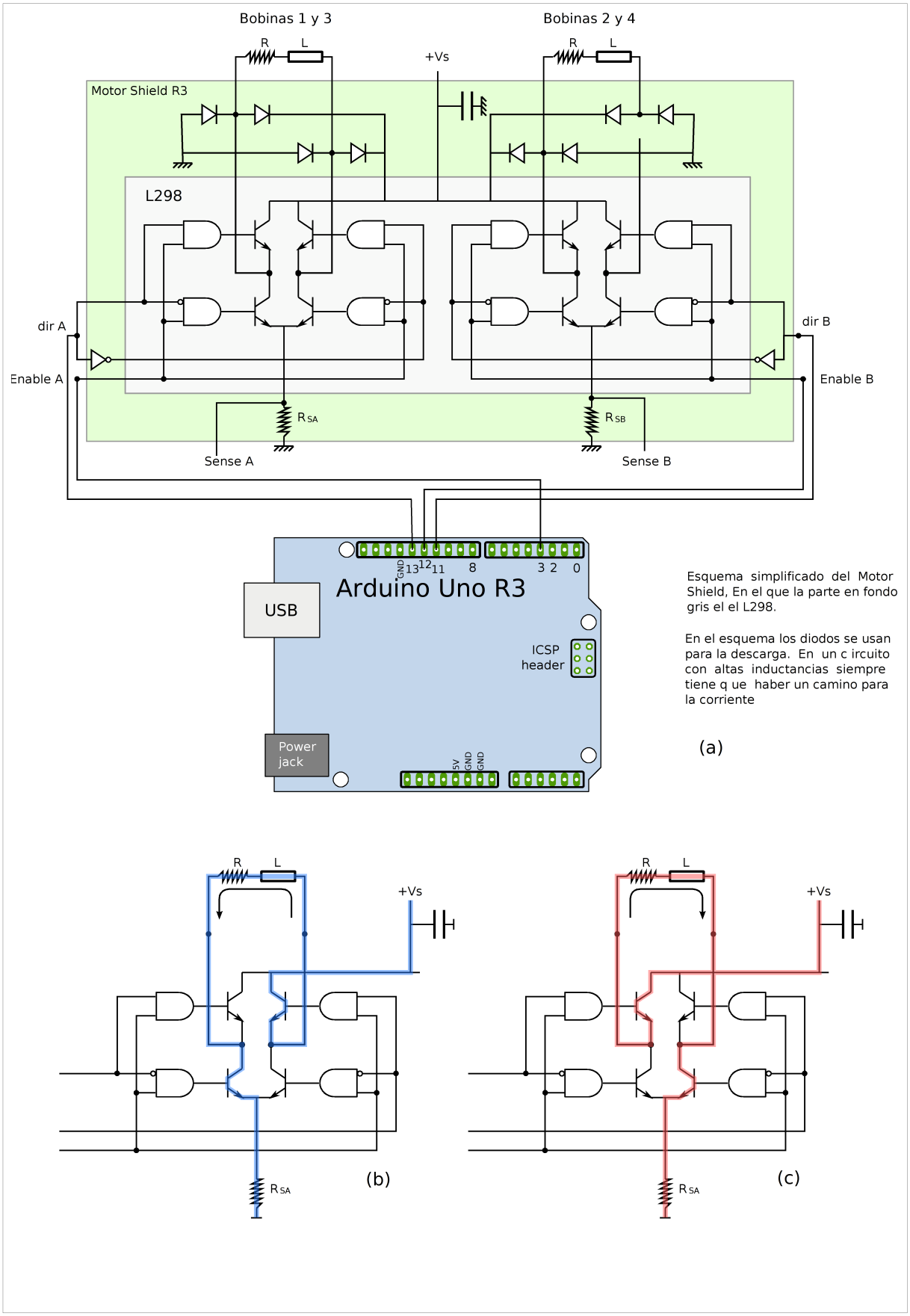


Figura 7. Conexión del Arduino Uno rev.3 al Motor Shield rev.3.

correspondientes a cada paso.

Turn() realiza el numero de pasos dado por nsteps llamando a las funciones steps. El primer paso dado es el siguiente correspondiente al último estado en la secuencia dada por el sentido de giro. Un retraso en microsegundo se intercala entre cada llamada, el tiempo entre cada llamada es el mismo.

Stop() para el motor.

2.2 Comunicación con el Arduino.

Desde la consola del IDE de Arduino, o desde un programa Python usando PySerial.

Las instrucciones que entiende el programa son cadenas ASCII compuestas de carácter de inicio, 3 caracteres de la instrucción, seguidas de una ',' cuando hay que enviar un parámetro, el valor del parámetro, y el carácter de finalización del mensaje.

Set Number of steps to turn : sns

ejemplo: \02sns,800\03

3.3 Una interrupción reajusta la posición

Es probable que durante el funcionamiento aparezcan errores de posición como consecuencia del deslizamiento entre la polea movida por el motor y la sirga. Para disminuir en lo posible este error, lo que hacemos es que cada vez que inicie la serie de ocho posiciones el sistema se mueva en una dirección buscando un punto 'cero' que produce el lanzamiento de una interrupción del micro-controlador. La función asociada a esta, detiene el motor y pone a cero la variable que almacena la posición absoluta de la sirga, reajustando así, la posición supuesta con la posición real.

Pines del Arduino y circuito

En el Arduino, son los pines 2 y 3, los correspondientes a la captura de los eventos que lanzan la interrupciones 0 y 1 respectivamente. En la figura 8 puede verse el circuito para para detectar el cierre de un pulsador. Notar que mientras el interruptor está abierto el pin 2 está a voltaje 0, si no

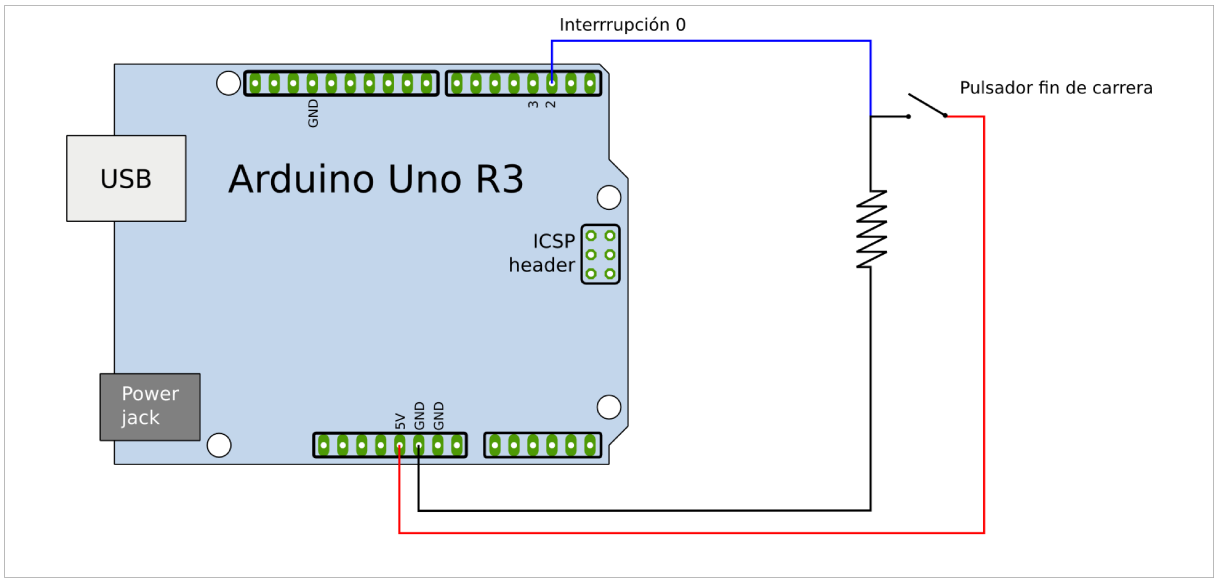


Figura 8. Circuito para la señal de interrupción.

fuese así, y no lo conectásemos a tierra, el valor del pin 2 quedaría indeterminado con lo que la interrupción podría lanzarse en cualquier momento.

También colocamos un resistencia para no conectar directamente la salida de 5V con tierra.

Programación de la interrupción

Con la instrucción attachInterrupt(interrupt, function, mode) enlazamos la interrupción 0 (pin2) ó 1 (pin3) con la función “function” que se lanzará cuando se produzca un cambio de estado en el que el pin pase de nivel bajo a alto (mode=RISE). También se encarga de deshabilitar las intererrupciones para que estas no se aniden y luego volverlas a habilitar.

Las variables usadas y que sean compartidas en el programa principal deben ser calificadas como volatiles.

Las líneas que añadiríamos al programa serían las siguientes:

```
// Variables Globales
volatile long absPosStep = 0; // absolute position in steps
volatile unsigned long nsteps = 0; // number of steps to turn

// Interrupcion
void zero_position() {
  absPosStep = 0;
  nsteps = 0;
}
```

Aunque una comparación al inicio del procedimiento que ejecuta cada paso podría ser suficiente, sobre todo pensando en que el movimiento no es continuo sino que se va pasando de un paso a otro paso, utilizamos una interrupción para evitar perder el cambio de valor en momentos intermedios, con lo que podríamos pasarnos el punto de 'cero'.

4. Ajuste del par

Con lo visto en el apartado 1.2. en el que se explica como se genera el par. Este podemos ajustarlo usando una resistencia para fijar la intensidad o usar una modulación por pulso (PWM) en la que vamos cortando la intensidad a intervalos cortos para tener un valor entre el máximo y el mínimo.

La idea del ajuste es conseguir un movimiento suave combinando par y tiempo entre pasos. De forma que activemos el siguiente paso cuando el rotor llegue a la posición del paso actual, y no antes porque si no dejaremos atrás el movimiento real y no se moverá, ni tampoco mucho después porque el movimiento podría ser una oscilación subamortiguada si el par es muy alto con lo que se produciría una fuerte vibración. La cual es fácilmente detectable por el ruido y traqueteo del motor. Entre estas dos situaciones queda un margen en el que el motor girará correctamente e incluso será capaz de absorber cambios en la magnitud de la fuerza que tiene que vencer.

Ambos factores nos pueden servir para saber si el ajuste es correcto o incluso para realizar este ajuste.

5. Bibliografía y enlaces

[enlace 1] Nociones básicas del motores paso a paso.

http://www.solarbotics.net/library/pdflib/pdf/motorbas.pdf

[enlace 2] Circuitos de control de motores paso a paso.

http://www.solarbotics.net/library/pdflib/pdf/drive.pdf

[enlace 3] Página oficial del Arduino.

http://www.arduino.cc/

[enlace 4] Página del Arduino Motor Shield Rev.3. Apuntar que en el esquema las puertas NAND marcadas con una e son puertas XNOR.

http://arduino.cc/en/Main/ArduinoMotorShieldR3

[enlace 5] Hoja de especificaciones del dual full-bridge driver L298.

http://www.st.com/web/catalog/sense_power/FM142/CL851/SC1790/SS1555/PF63147

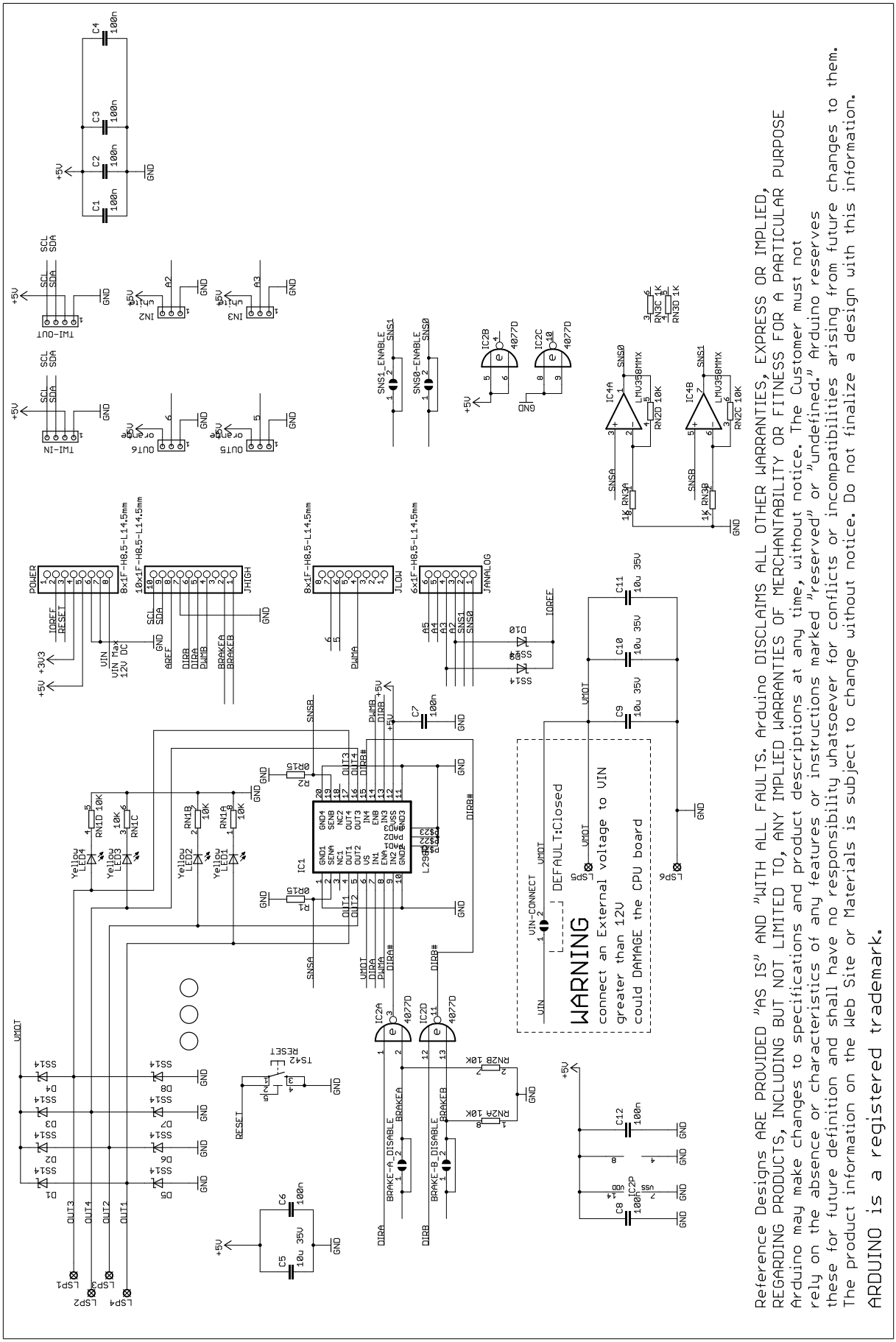


Figura 9. Esquema del Arduino Motor Shiel rev3

ANEXO 6. PROGRAMAS

Introducción

En los siguientes apartados, que se han estructurado por conjuntos de hardware, recogemos y explicamos los puntos más importantes de los programas realizados para el proyecto.

Por indicar un conjunto que puede tomarse como base o plantilla tomaría el de digitalización de la señal analógica. Por ser este el más simple, pero a la vez el más completo al reunir todos los tipos de programas que pueden aparecer.

Índice de conjuntos y programas

Conjunto Swagelok y Temperatura: Digitalización de una señal analógica.

Ambos conjuntos, el del transductor de presión Swagelok y el de temperatura, son similares ya que su misión es tomar una señal analógica, discretizarla y permitir su visualización desde la interfaz gráfica. Figura 1.

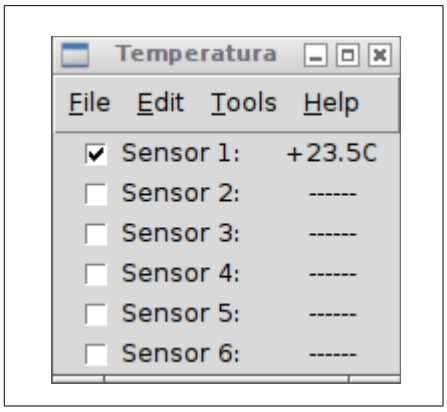


Figura 1. Interfaz gráfica analogGui

Los programas que la constituyen son los mismos y solo son necesarios algunos cambios en las opciones de estos programas. Por ejemplo la selección de la función de discretización y de dirección en la red, IP y puerto.

En los siguientes puntos desgranamos la estructura y explicamos los detalles más importantes de cada programa.

Sketch Arduino Digitalización. Programa corriendo en el Arduino que se encarga de digitalizar la señal y enviar la respuesta cuando es requerida desde la interfaz gráfica. La digitalización cuesta 0.25ms con 10 bits de resolución. Y la forma en que se realiza la comunicación serie está recogida en la sección correspondiente de la memoria siendo. En caso de no entender la instrucción, por ser errónea o fallo en la comunicación, simplemente devuelve el mensaje que le ha llegado sin hacer nada.

AnalogArduinoServer. Programa en Python que se está ejecutando en el Raspberry Pi, aunque también lo podría hacer en cualquier PC, y que hace de puente entre el puerto serie y la red Ethernet.

Entre las características que posee, es el uso de excepciones para mantener el servidor en funcionamiento en cualquier caso. Como ejemplo, el cable serie podría desconectarse y simplemente avisaría a la vez que intenta volverse a conectar cada 5s

Variables que deben ajustarse,según sea el caso, son el puerto del dispositivo serie y el puerto en el que el socket servidor está a la escucha.

SocketClient. Es un módulo Python, común a todas las interfaz gráficas, que resume la mecánica de comunicación por sockets en una función para ser utilizada desde cualquier programa Python que llame al módulo. También usa excepciones para, en cualquier caso, cerrar correctamente la conexión.

La IP y el puerto al que se conecta pueden ser especificadas en los parámetros de la función o en variables globales que se usan como valores por defecto. Estos también pueden ser modificados desde el programa que carga al módulo.

AnalogModule. Este es el módulo en el que se encuentran las instrucciones de operación con el programa de digitalización del Arduino. En este caso, solo hay una, sensor(i), que hace una petición de una medición de la señal en la entrada i devolviendo un valor entero, o una cadena de texto informativa del tipo de error.

Puede usarse la instrucción help(AnalogModule) para obtener la ayuda que consta de la explicación de cada comando, como de los parámetros de entrada y salida que intervienen.

Usa el módulo SocketClient anterior para realizar la conexión y el intercambio de mensajes.

AnalogGui. Es la interfaz gráfica. Es completamente intuitiva y las variables para ajustar IP, puertos, tiempos... se encuentran al inicio del programa con su explicación. Usa el módulo anterior para obtener los datos.

Sketch Arduino Digitalización

```
const char STX='\02';   // STX = \02
const char ETX='\03';   // ETX = \03
const char SEP=',';   // Separador
const int  str_len=25;  // max length message strings

char charin;
char entrada[str_len];
int index = 0;

int valor;            // operations integer var

// Functions / Funciones
void clear_buffer()
{
  while (Serial.read() >= 0);
}


// Setup
void setup()
{
  Serial.begin(9600);
}

// Program loop
void loop()
{
  index = 0;
  entrada[index] = '\0';
  // get command from serial
  do
  {
    while (Serial.available() == 0) {delay(10);}
    charin = Serial.read(); // get a character
    switch (charin)
    {
      case ETX:
        break;
      case STX:
        index = 0;
        entrada[index] = '\0';
        break;
      case SEP: // future
        break;
      default:
        //strcat(entrada,charin);
        entrada[index]= charin;
        index++;
```

```
        entrada[index]= '\0';
        if (index > 6){index=6;} // 7 chars + null, no sobrepasar longitud maxima
      }
    } while( (charin != ETX) || (Serial.available() > 0) );

    Serial.print(STX);      // ECHO command
    Serial.print(entrada);  //

    switch (entrada[0])
    {
      case 'r': // it's a request / es una peticion
        switch (entrada[1])
        {
          case 's': // it's a sensor / es un sensor
            switch (entrada[2]) // forma de asegurarnos que estamos dentro de los
limites
            {
              case '1':
                Serial.print(",");
                Serial.print(analogRead(0));
                break;
              case '2':
                Serial.print(",");
                Serial.print(analogRead(1));
                break;
              case '3':
                Serial.print(",");
                Serial.print(analogRead(2));
                break;
              case '4':
                Serial.print(",");
                Serial.print(analogRead(3));
                break;
              case '5':
                Serial.print(",");
                Serial.print(analogRead(4));
                break;
              case '6':
                Serial.print(",");
                Serial.print(analogRead(5));
                break;
            }
            break;
          case 'c': // clear
              switch (entrada[2])
              {
                case 'f':
                  clear_buffer(); // clear input buffer
                  break;
              }
            break;
        }
```

```
      break;
  }
  // return
  Serial.print(ETX);  // end message
  Serial.flush();     // wait send output buffer
}
```

Analog Arduino Server

```
# analogArduinoServer
# Python 3
# Programa puente
# Acepta una conexion a traves de un socket TCP/IP que hace de puente
# con un puerto serie

import socket
import serial
import time
import threading

# ---Constantes---

STX = b'\x02'
ETX = b'\x03'

# ---Variables---

actividad = {}                          # Diccionario con el estado
actividad['run']    = True  # Si False => salir del programa
actividad['serial'] = False # Estado del puerto serie
contador = 0

socketHost  = ''
socketPort  = 50000
socketGenerator = None

serialPort  = '/dev/ttyACM0'
objectSerial = None

serialTimeOut  = 2 # segundos; tiempo maximo de espera
serialTimeOp   = 0.001 # segundos; tiempo estimado de operacion


# ---Clases---
class Message():
       '''Clase Mensaje'''
       def __init__(self, byteString = b'', n=0):
              self.bytesIn = byteString
              self.bytesOut = b''
              self.time = time.time()
              self.contador = n

       def write(self, objectSerial):
              objectSerial.write(self.bytesIn)

       def read(self, objectSerial):
              self.bytesOut = b''
              time0 = time.time()
              while True:
```

```
                    data = objectSerial.read()
                    self.bytesOut += data
                    if data == STX: self.bytesOut = STX
                    if (data == ETX) and (objectSerial.inWaiting() == 0): break
                    if time.time() > time0+serialTimeOut+serialTimeOp:
                          raise serial.SerialTimeoutException

      def serie(self, objectSerial):
            print('Enviando y recibiendo por serie')      #
            try:
                  n=bytes('%03d' % (self.contador), encoding='ascii')
                  self.bytesIn = self.bytesIn[:4]+n+self.bytesIn[4:]
                  self.write(objectSerial)
                  self.read(objectSerial)
                  if self.bytesIn[:7] == self.bytesOut[:7]:
                        self.bytesOut = self.bytesOut[:4]+self.bytesOut[7:]
                  else:
                        time.sleep(1)
                        self.read(objectSerial)
                        if self.bytesIn[:7] == self.bytesOut[:7]:
                              self.bytesOut =
self.bytesOut[:4]+self.bytesOut[7:]
                        else:
                              self.bytesOut = STX + b'error' + ETX
            except serial.SerialTimeoutException:
                  print(message.bytesIn)    #
                  print(message.bytesOut)   #
                  print('SerialTimeoutException')
                  pass
            except serial.SerialException:
                  print('Conexion serie caida')
                  actividad['serial'] = False
                  objectSerial.close()
                  objectSerial == None
                  threading.Thread(target=serial_up).start() # Thread
            except Exception as x:
                  actividad['serial'] = False
                  objectSerial.close()
                  objectSerial == None
                  threading.Thread(target=serial_up).start() # Thread
                  print('Otro tipo de excepcion\nSerial down\n', x)
            else:
                  return
            self.bytesIn = STX + b'error' + ETX



# ---Funciones---

#    SERIE

def serial_up():
```

```
	'''Thread that make an object serial instance
	Hilo para establecer la conexion serie
	Crea una instancia de un objeto serial (GLOBAL objectSerial).
	El objeto esta conectado al puerto proporcionado por serialPort'''
	global objectSerial, serialPort
	while not actividad['serial'] and actividad['run']:
		try:
			objectSerial = serial.Serial(
									serialPort,
									baudrate=9600,
									timeout=1,
									writeTimeout=1)
		except serial.serialutil.SerialException:
			print('Excepcion:\n',
			'a) El dispositivo no se encuentra en el puerto indicado.\n',
			'b) o no está conectado.\n> Reintentando en 5 segundos')
			# enviar un mensaje al agente que se ha conectado
			time.sleep(5)
		else:
			time.sleep(3)# Tiempo de activacion
			print('Conexion serie up')
			actividad['serial'] = True


#    SOCKET

def socket_up():
	global genSocket
	try:
		genSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
	except OSError:
		print('Excepcion al crear una instancia del tipo socket')
		print('--- Cerrando ---')
	try:
		genSocket.bind((socketHost, socketPort))
		genSocket.listen(5)
		print('Socket up.')
		print('  Socket host:',socket.gethostname())
		print('  Socket port:',socketPort)
	except OSError:
		print('Excepcion al crear el socket receptor de peticiones de
conexion',
			'puede ser debido a que el puerto o la direccion IP',
socketPort,
			'estan siendo usados por otra aplicación')
		print('--- Cerrando ---')
		genSocket.close()
		actividad['run'] = False


# ----- INICIO PROGRAMA -----
print('--- iniciando ---')
```

```
threading.Thread(target=serial_up).start() # Thread
socket_up()
while actividad['run']: # Bucle
	try:
		# Crea un socket cliente tras una peticion de conexion
		print('Waiting nueva conexion TCP/IP...')
		conexionRed, address = genSocket.accept()
		conexionRed.settimeout(2)
		print('Conexion con', address)
		# recepcion del mensaje
		data = b''
		while not ETX in data: # reconstruye el mensaje
			trozo = conexionRed.recv(1024)
			if not trozo: break
			data += trozo
		message = Message(data, contador)
		contador = (contador + 1) % 1000
		if actividad['serial']: message.serie(objectSerial)
		else: message.bytesOut = STX + b'error,serial_down' + ETX
		conexionRed.sendall(message.bytesOut)
	except socket.timeout:
		pass
	except Exception:
		pass
	finally:
		conexionRed.close()
```

SocketClient

```
# socketClient
# Modulo para la comunicacion como cliente por socket
# especifico para el ArduinoServer.py
# Caracteristicas:
#     Propaga excepciones
#     Cierra siempre el socket
import socket
import time

def send_recv(message, ip='localhost', port=50008, tOp=0, tOut=1):
      '''
      Envia un mensaje y espera su respuesta.
      Sintaxis:
            send_recv(message, ip, port, t_operacion, t_out)
      Parametros pasados a la funcion:
                  message = cadena Unicode o ASCII
                  ip = cadena de texto de la direccion IP del servidor
                  port = entero del puerto de escucha del servidor
                  t_operacion = tiempo de estimado
                  t_out = tiempo de envio
                  t_operacion + t_out + t_out = t_max
      Respuesta:
            cadena Unicode.
      '''
      try:
            so = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            so.settimeout(tOut)
            so.connect((ip,port))
            #print(so.getpeername())
            #print(so.getsockname())
            so.sendall(bytes(message, encoding='ascii'))
            so.shutdown(socket.SHUT_WR)
            so.settimeout(tOut+tOp)
            resp = b''
            while True:
                  trozo = so.recv(1024)
                  if not trozo: break
                  resp += trozo
      finally:
            so.close()
      return str(resp, encoding='ascii')
```

AnalogModule

```
# analogicModule.py
# Python 3
# Modulo con las instrucciones del Arduino conversor de la señal analogica a
# digital

import sys
import socketClient as sc

# Direccion y puerto donde se encuentra el servidor
serverHost = 'localhost' # ejemplos: 'localhost', '192.168.2.3'
serverPort = 50000

STX = '\x02' # Start of Text character
ETX = '\x03' # End of Text character
# Sintaxis del protocolo
# STX + Command + ',' + Argumentos + ETX

# Funciones basicas

def sensor(i = 1):
	'''Request sensor i value.
		i must be 1 to 6.
	Pide la medición del sensor i
		i debe ser un valor de 1 a 6 (valor entero o caracter)
		La orden devuelve un entero entre 0 y 1023,
			o una cadena de error
	'''
	cmd = 'rs' + str(i) + '_'
	arg = None
	message = STX + cmd[:3] + ETX
	# socket
	try:
		ans = sc.send_recv(message, serverHost, serverPort ,tOp=1, tOut=1)
	except sc.socket.timeout:
		return 'timeout'
	except Exception:
		return 'socket error'
	# comprobacion de la respuesta
	try:
		ans = ans.strip(STX+ETX+',').split(',')
		if cmd[:3] == ans[0]:
			resp = int(ans[1])
			if 0<=resp<1024: return resp # correcto
		elif 'error' == ans[0]:
			return ans[1] # mensaje asociado al error
		else:
			return 'corrupcion del mensaje'
	except Exception:
		return 'corrupcion del mensaje'
```

AnalogGui

```
# analogGui
# Python 3
# Interfaz grafica
import tkinter as tk
import tkinter.messagebox as tkm
import threading, queue, time
import analogModule as ana

# CONSTANTES GLOBALES

# Localizacion del servidor volcadas en el modulo analogicModule
ana.serverHost = 'localhost' # 'localhost' o '192.168.2.13'
ana.serverPort = 50000

t_intervalo = 1 # intervalo entre peticiones (en segundos)

# VARIABLES

# Variables globales

buttons = {} # diccionario de objetos graficos; campos a actualizar.
labels  = {} # diccionario de objetos graficos; campos a actualizar.
checkbuttons  = {}
checks      = {} # variables del entorno de ventanas
checkValues = {} # valores de las variables anteriores
interpols   = {} # conjunto de funciones de interpolacion a aplicar
actividad = {'play':True, 'record':False} # bit, si False => close thread
dataqueue = queue.Queue() # Cola de datos de acceso en exclusion.

# valores iniciales
checkValues['sensor_1'] = 0
checkValues['sensor_2'] = 0
checkValues['sensor_3'] = 0
checkValues['sensor_4'] = 0
checkValues['sensor_5'] = 0
checkValues['sensor_6'] = 0

# Funciones matematicas
def interpol(x):
       '''Devuelve un valor entero'''
       x0=0
       y0=0
       x1=1023
       y1=1023
       m = (y1-y0)/(x1-x0)
       x = int(x)
       return  m*(x-x0) + y0

def interpol_LM35(x):
       '''Devuelve un numero
```

```
        T[C] = 0 + 0.01T[V]'''
        x0=0
        y0=0
        x1=1023
        y1=500 # 500C
        m = (y1-y0)/(x1-x0)
        x =  m*(x-x0) + y0
        return '%+.1FC' % (x)

interpols['sensor_1'] = interpol_LM35
interpols['sensor_2'] = interpol
interpols['sensor_3'] = interpol
interpols['sensor_4'] = interpol
interpols['sensor_5'] = interpol
interpols['sensor_6'] = interpol


# Funciones

def repeater(t_refresco=200):
        try:
                data = dataqueue.get(block=False)
        except queue.Empty:
                pass
        else:
                # actualizar GUI
                labels[data[0]].config(text=data[1])
        winroot.after(t_refresco, repeater) # llamada periodica ms

def clean_labels():
        time.sleep(1)
        for key in checks:
                if not checkValues[key]: dataqueue.put((key, '-----'))

def checker():
        '''Actualiza la varible global checks
        con los valores de las variables del entorno grafico'''
        for key in checks:
                checkValues[key] = checks[key].get()
        threading.Thread(target=clean_labels).start()


def record():
        # No implementado. Seguir el ejemplo MaxiGauge
        threading.Thread(target=record_thread).start()

def record_thread():
        fichero = open('temperature_data.txt', 'a')
        fichero.write('----- Inicio de sesion -----\n')
        while actividad['record']:
                pass
        fichero.close()
```

```
def play():
	# lanzador del hilo play_thread
	threading.Thread(target=play_thread).start()

def play_thread():
	t = time.time()
	while actividad['play']:
		t0 = t
		for key in checkValues:
			if checkValues[key]:
				try:
					resp = ana.sensor(int(key[-1]))
					if type(resp) == int:
						resp = interpols[key](resp)
				except Exception:
					resp = 'error'
				dataqueue.put((key, resp))
				try:
					t = t + t_intervalo
					time.sleep(t - time.time())
				except Exception: # el tiempo era negativo, no espera.
					resp = 'sobrepasamiento'
		if t0 == t:
			t = t + t_intervalo
			try:
				time.sleep(t - time.time())
			except:
				pass



# Definicion de las ventanas

def notdone():
	tkm.showerror('NO implementado', 'NO disponible.\nEn desarrollo')

def makemenu(win):
	menubar = tk.Menu(win)
	win.config(menu=menubar)

	filemenu = tk.Menu(menubar)
	filemenu.add_command(label='Save...', command=notdone, underline=0)
	filemenu.add_command(label='Save as', command=notdone, underline=0)
	filemenu.add_separator()
	filemenu.add_command(label='Quit...', command=win.quit,underline=0)
	menubar.add_cascade(label='File', menu=filemenu,   underline=0)

	editmenu = tk.Menu(menubar)
	editmenu.add_command(label='Preferences', command=notdone)
	menubar.add_cascade(label='Edit', menu=editmenu, underline=0)
```

```
        toolmenu = tk.Menu(menubar)
        toolmenu.add_command(label='Init', command=notdone, underline=0)
        menubar.add_cascade(label='Tools', menu=toolmenu, underline=0)

        helpmenu = tk.Menu(menubar)
        helpmenu.add_command(label='About',command=notdone)
        menubar.add_cascade(label='Help', menu=helpmenu, underline=0)


def makecheck(win, checkbuttons=checkbuttons, labels=labels):
        '''Construye los checkbuttons'''
        frame = tk.Frame(win)
        frame.pack()
        # Creacion objeto grafico: Checkbutton del sensor 1
        checks['sensor_1'] = tk.IntVar()
        check_1 = tk.Checkbutton(frame, command=checker,
               text=' Sensor 1:', variable=checks['sensor_1'], width=10)
        check_1.pack(side='left')
        checkbuttons['check_1'] = check_1
        # Creacion objeto grafico: Etiqueta del sensor 1
        sensor_1 = tk.Label(frame, text=' ------ ', width=8)
        sensor_1.pack(side='right')
        labels['sensor_1'] = sensor_1

        frame = tk.Frame(win)
        frame.pack()
        # Creacion objeto grafico: Checkbutton del sensor 2
        checks['sensor_2'] = tk.IntVar()
        check_2 = tk.Checkbutton(frame,  command=checker,
               text=' Sensor 2:', variable=checks['sensor_2'], width=10)
        check_2.pack(side='left')
        checkbuttons['check_2'] = check_2
        # Creacion objeto grafico: Etiqueta del sensor 2
        sensor_2 = tk.Label(frame, text=' ------ ', width=8)
        sensor_2.pack(side='right')
        labels['sensor_2'] = sensor_2

        frame = tk.Frame(win)
        frame.pack()
        # Creacion objeto grafico: Checkbutton del sensor 3
        checks['sensor_3'] = tk.IntVar()
        check_3 = tk.Checkbutton(frame,  command=checker,
               text=' Sensor 3:', variable=checks['sensor_3'], width=10)
        check_3.pack(side='left')
        checkbuttons['check_3'] = check_3
        # Creacion objeto grafico: Etiqueta del sensor 3
        sensor_3 = tk.Label(frame, text=' ------ ', width=8)
        sensor_3.pack(side='right')
        labels['sensor_3'] = sensor_3

        frame = tk.Frame(win)
        frame.pack()
```

```
        # Creacion objeto grafico: Checkbutton del sensor 4
        checks['sensor_4'] = tk.IntVar()
        check_4 = tk.Checkbutton(frame, command=checker,
               text=' Sensor 4:', variable=checks['sensor_4'], width=10)
        check_4.pack(side='left')
        checkbuttons['check_4'] = check_4
        # Creacion objeto grafico: Etiqueta del sensor 4
        sensor_4 = tk.Label(frame, text=' ------ ', width=8)
        sensor_4.pack(side='right')
        labels['sensor_4'] = sensor_4

        frame = tk.Frame(win)
        frame.pack()
        # Creacion objeto grafico: Checkbutton del sensor 5
        checks['sensor_5'] = tk.IntVar()
        check_5 = tk.Checkbutton(frame, command=checker,
               text=' Sensor 5:', variable=checks['sensor_5'], width=10)
        check_5.pack(side='left')
        checkbuttons['check_5'] = check_5
        # Creacion objeto grafico: Etiqueta del sensor 5
        sensor_5 = tk.Label(frame, text=' ------ ', width=8)
        sensor_5.pack(side='right')
        labels['sensor_5'] = sensor_5

        frame = tk.Frame(win)
        frame.pack()
        # Creacion objeto grafico: Checkbutton del sensor 6
        checks['sensor_6'] = tk.IntVar()
        check_6 = tk.Checkbutton(frame, command=checker,
               text=' Sensor 6:', variable=checks['sensor_6'], width=10)
        check_6.pack(side='left')
        checkbuttons['check_6'] = check_6
        # Creacion objeto grafico: Etiqueta del sensor 6
        sensor_6 = tk.Label(frame, text=' ------ ', width=8)
        sensor_6.pack(side='right')
        labels['sensor_6'] = sensor_6




def maketable(win):
        marco = tk.Frame(win)
        marco.pack()
        # cabecera
        titulo_status = tk.Label(marco, text='st', relief='flat',
                                                          width=3, bg='beige')
        titulo_sensor = tk.Label(marco, text='sensor', relief='flat',
                                                          width=9, bg='beige')
        titulo_data   = tk.Label(marco, text='data', relief='flat',
                                                         width=13, bg='beige')
        titulo_status.pack(side='left')
        titulo_sensor.pack(side='left')
```

```
        titulo_data.pack(side='left')
        # tabla
        tabla = [(titulo_status, titulo_sensor, titulo_data)]
        for i in range(1,6+1):
                marco = tk.Frame(win)
                marco.pack()
                status = tk.Label(marco, text='--', relief='ridge', width=3)
                sensor = tk.Label(marco, text=str(i), relief='ridge', width=9)
                data = tk.Label(marco, text='----', relief='sunken', width=13)
                status.pack(side='left')
                sensor.pack(side='left')
                data.pack(side='left')
                tabla.append((status, sensor, data))
        return tabla


#   INICIO PROGRAMA

if __name__=='__main__':
        winroot = tk.Tk()
        winroot.title('Temperatura')
        winroot.after(500,repeater) # actualizacion de la GUI
        makemenu(winroot)
        checkbuttons = makecheck(winroot)
        play()
        # lanzamiento del sistema de eventos
        winroot.mainloop()
        # cierre de hilos
        print('cerrando')
        actividad['play']=False
        actividad['record']=False
```

Conjunto motor paso a paso

La estructura de los programas es similar al caso anterior. Cambiando la interfaz gráfica, el módulo de instrucciones (motorModule), y algunos cambios mínimos en el motorArduinoServer respecto al analogArduinoServer.

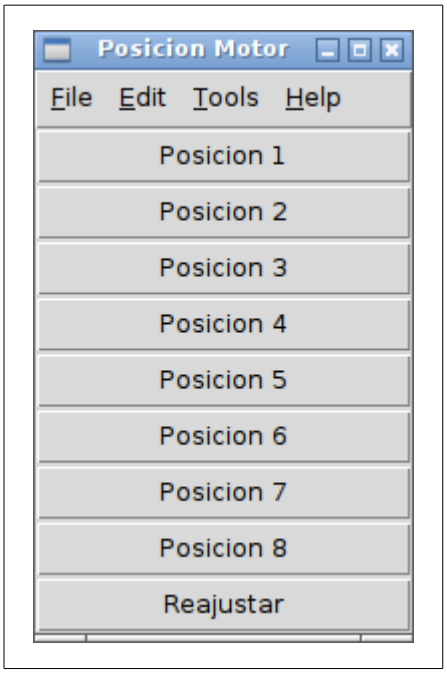


Figura 2. Interfaz gráfica motorGui

Sketch Arduino motor. Programa que corre en el Arduino, que genera la secuencia de señales que el Motor Shield utiliza para cerrar o dejar pasar la corriente en un sentido u otro. Hay una explicación más extensa en el anexo 5.

motorArduinoServer. Programa que hace de puente entre la red Ethernet y el dispositivo serie, en este caso el Arduino.

SocketClient. El mismo módulo que en el caso anterior, es un módulo que se encarga de hacer las conexiones TCP/IP de la interfaz gráfica al servidor (motorArduinoServer).

motorShieldModule. Instrucciones para operar con el motor. La más importante es turn(integer, boolean) por la que se ordena al motor a girar el número de pasos indicados por el número entero en una dirección u otra según el booleano.

Otras instrucciones son set_step_duration(microsegundos), get_abs_pos().
Con help(motorShieldModule) se puede obtener más información en el intérprete de Python.

MotorGui. Interfaz gráfica donde seleccionamos la siguiente posición. Figura 2.

Sketch Arduino motor

```
/* Stepper Motor, Arduino uno motor shield R3 */
// Include Stepper Library
#include <Stepper.h>

// Map pins to constants
const int pwmA=3;
const int pwmB=11;
const int brakeA=9;
const int brakeB=8;
const int dirA=12;
const int dirB=13;

// steps for a revolution
const int STEPS=200;
const int periodo=20;

// Initialize the stepper class
//Stepper myStepper(STEPS,pwmA,pwmB,dirA,dirB);
void step1(){
  digitalWrite(pwmA,HIGH);
  digitalWrite(pwmB,LOW);
  digitalWrite(dirA,HIGH);
  digitalWrite(dirB,LOW);
}
void step2(){
  digitalWrite(pwmA,LOW);
  digitalWrite(pwmB,HIGH);
  digitalWrite(dirA,HIGH);
  digitalWrite(dirB,HIGH);
}
void step3(){
  digitalWrite(pwmA,HIGH);
  digitalWrite(pwmB,LOW);
  digitalWrite(dirA,LOW);
  digitalWrite(dirB,LOW);
}
void step4(){
  digitalWrite(pwmA,LOW);
  digitalWrite(pwmB,HIGH);
  digitalWrite(dirA,LOW);
  digitalWrite(dirB,LOW);
}
void stop(){
  digitalWrite(pwmA,LOW);
  digitalWrite(pwmB,LOW);
}

void setup() {
```

```
  //RPM of the motor
//  myStepper.setSpeed(1);
  // Enable
  pinMode(pwmA, OUTPUT);
  digitalWrite(pwmA, LOW);
  pinMode(pwmB, OUTPUT);
  digitalWrite(pwmB, LOW);

  pinMode(dirA, OUTPUT);
  digitalWrite(dirA, LOW);
  pinMode(dirB, OUTPUT);
  digitalWrite(dirB, LOW);

  pinMode(brakeA, OUTPUT);
  digitalWrite(brakeA, LOW);
  pinMode(brakeB, OUTPUT);
  digitalWrite(brakeB, LOW);
}

void loop() {
//  delay(2000);
//  digitalWrite(pwmA,HIGH);
//  digitalWrite(pwmB,LOW);
//  digitalWrite(dirA,LOW);
//  digitalWrite(dirB,LOW);
//  delay(2000);
//  digitalWrite(pwmA,LOW);
//  digitalWrite(pwmB,HIGH);
//  digitalWrite(dirA,LOW);
//  digitalWrite(dirB,LOW);
  delay(0000);
  step1();
  delay(periodo);
  step2();
  delay(periodo);
  step3();
  delay(periodo);
  step4();
  delay(periodo);
  stop();
  //digitalWrite(pwmA, LOW);
  //delay(1000);
  //digitalWrite(pwmA, HIGH);
  //delay(1000);
  //myStepper.step(STEPS);
//  digitalWrite(pwmA, HIGH);
//  digitalWrite(pwmB, HIGH);
//  digitalWrite(dirA, HIGH);
//  digitalWrite(dirB, HIGH);
//  delay(1000);
//  digitalWrite(pwmA, LOW);
//  digitalWrite(pwmB, LOW);
```

```
//  digitalWrite(dirA, LOW);
//  digitalWrite(dirB, LOW);
//  delay(1000);
//  //myStepper.step(-STEPS);
//  digitalWrite(pwmA, HIGH);
//  digitalWrite(pwmB, HIGH);
//  digitalWrite(dirA, HIGH);
//  digitalWrite(dirB, HIGH);
//  delay(1000);
//  digitalWrite(pwmA, LOW);
//  digitalWrite(pwmB, LOW);
//  digitalWrite(dirA, LOW);
//  digitalWrite(dirB, LOW);
//  delay(1000);
}
```

motorArduinoServer

```
# motorArduinoServer
# Python 3
# Programa puente
# Acepta una conexion a traves de un socket TCP/IP que hace de puente
# con un puerto serie

import socket
import serial
import time
import threading

# ---Constantes---

STX = b'\x02'
ETX = b'\x03'

# ---Variables---

actividad = {}                          # Diccionario con el estado
actividad['run']    = True  # Si False => salir del programa
actividad['serial'] = False # Estado del puerto serie
contador = 0

socketHost  = ''
socketPort  = 50000
socketGenerator = None

serialPort  = '/dev/ttyACM0'
objectSerial = None

serialTimeOut  = 30 # segundos; tiempo maximo de espera
serialTimeOp   = 0.005 # segundos; tiempo estimado de operacion


# ---Clases---
class Message():
	'''Clase Mensaje'''
	def __init__(self, byteString = b'', n=0):
		self.bytesIn = byteString
		self.bytesOut = b''
		self.time = time.time()
		self.contador = n

	def write(self, objectSerial):
		objectSerial.write(self.bytesIn)

	def read(self, objectSerial):
		self.bytesOut = b''
```

```
            time0 = time.time()
            while True:
                  data = objectSerial.read()
                  self.bytesOut += data
                  if data == STX: self.bytesOut = STX
                  if (data == ETX) and (objectSerial.inWaiting() == 0): break
                  if time.time() > time0+serialTimeOut+serialTimeOp:
                        raise serial.SerialTimeoutException

      def serie(self, objectSerial):
            print('Enviando y recibiendo por serie')      #
            try:
                  n=bytes('%03d' % (self.contador), encoding='ascii')
                  self.bytesIn = self.bytesIn[:4]+n+self.bytesIn[4:]
                  self.write(objectSerial)
                  self.read(objectSerial)
                  if self.bytesIn[:7] == self.bytesOut[:7]:
                        self.bytesOut = self.bytesOut[:4]+self.bytesOut[7:]
                  else:
                        time.sleep(1)
                        self.read(objectSerial)
                        if self.bytesIn[:7] == self.bytesOut[:7]:
                              self.bytesOut =
self.bytesOut[:4]+self.bytesOut[7:]
                        else:
                              self.bytesOut = STX + b'error' + ETX
            except serial.SerialTimeoutException:
                  print(message.bytesIn)    #
                  print(message.bytesOut)   #
                  print('SerialTimeoutException')
                  pass
            except serial.SerialException:
                  print('Conexion serie caida')
                  actividad['serial'] = False
                  objectSerial.close()
                  objectSerial == None
                  threading.Thread(target=serial_up).start() # Thread
            except Exception as x:
                  actividad['serial'] = False
                  objectSerial.close()
                  objectSerial == None
                  threading.Thread(target=serial_up).start() # Thread
                  print('Otro tipo de excepcion\nSerial down\n', x)
            else:
                  return
            self.bytesIn = STX + b'error' + ETX



# ---Funciones---

#    SERIE
```

```
def serial_up():
	'''Thread that make an object serial instance
	Hilo para establecer la conexion serie
	Crea una instancia de un objeto serial (GLOBAL objectSerial).
	El objeto esta conectado al puerto proporcionado por serialPort'''
	global objectSerial, serialPort
	while not actividad['serial'] and actividad['run']:
		try:
			objectSerial = serial.Serial(
								serialPort,
								baudrate=9600,
								timeout=1,
								writeTimeout=1)
		except serial.serialutil.SerialException:
			print('Excepcion:\n',
			'a) El dispositivo no se encuentra en el puerto indicado.\n',
			'b) o no está conectado.\n> Reintentando en 5 segundos')
			# enviar un mensaje al agente que se ha conectado
			time.sleep(5)
		else:
			time.sleep(3)# Tiempo de activacion
			print('Conexion serie up')
			actividad['serial'] = True


#    SOCKET

def socket_up():
	global genSocket
	try:
		genSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
	except OSError:
		print('Excepcion al crear una instancia del tipo socket')
		print('--- Cerrando ---')
	try:
		genSocket.bind((socketHost, socketPort))
		genSocket.listen(5)
		print('Socket up.')
		print('  Socket host:',socket.gethostname())
		print('  Socket port:',socketPort)
	except OSError:
		print('Excepcion al crear el socket receptor de peticiones de
conexion',
			'puede ser debido a que el puerto o la direccion IP',
socketPort,
			'estan siendo usados por otra aplicación')
		print('--- Cerrando ---')
		genSocket.close()
		actividad['run'] = False
```

```
# ----- INICIO PROGRAMA -----
print('--- iniciando ---')
threading.Thread(target=serial_up).start() # Thread
socket_up()
while actividad['run']: # Bucle
	try:
		# Crea un socket cliente tras una peticion de conexion
		print('Waiting nueva conexion TCP/IP...')
		conexionRed, address = genSocket.accept()
		conexionRed.settimeout(2)
		print('Conexion con', address)
		# recepcion del mensaje
		data = b''
		while not ETX in data: # reconstruye el mensaje
			trozo = conexionRed.recv(1024)
			if not trozo: break
			data += trozo
		message = Message(data, contador)
		contador = (contador + 1) % 1000
		if actividad['serial']: message.serie(objectSerial)
		else: message.bytesOut = STX + b'error,serial_down' + ETX
		conexionRed.sendall(message.bytesOut)
	except socket.timeout:
		pass
	except Exception:
		pass
	finally:
		conexionRed.close()
```

SocketClientt

```
# socketClient
# Modulo para la comunicacion como cliente por socket
# especifico para el ArduinoServer.py
# Caracteristicas:
#     Propaga excepciones
#     Cierra siempre el socket
import socket
import time

def send_recv(message, ip='localhost', port=50008, tOp=0, tOut=1):
       '''
       Envia un mensaje y espera su respuesta.
       Sintaxis:
              send_recv(message, ip, port, t_operacion, t_out)
       Parametros pasados a la funcion:
                     message = cadena Unicode o ASCII
                     ip = cadena de texto de la direccion IP del servidor
                     port = entero del puerto de escucha del servidor
                     t_operacion = tiempo de estimado
                     t_out = tiempo de envio
                     t_operacion + t_out + t_out = t_max
       Respuesta:
              cadena Unicode.
       '''
       try:
              so = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
              so.settimeout(tOut)
              so.connect((ip,port))
              #print(so.getpeername())
              #print(so.getsockname())
              so.sendall(bytes(message, encoding='ascii'))
              so.shutdown(socket.SHUT_WR)
              so.settimeout(tOut+tOp)
              resp = b''
              while True:
                     trozo = so.recv(1024)
                     if not trozo: break
                     resp += trozo
       finally:
              so.close()
       return str(resp, encoding='ascii')
```

motorShieldModule

```
# motorShieldModule
# Python 3
# Modulo para interaccionar con el sketch del motor

import sys
import socketClient as sc

# Direccion y puerto donde se encuentra el servidor
serverHost = 'localhost' # ejemplos: 'localhosts', '192.168.2.3'
serverPort = 50000

STX = '\x02' # Start of Text character
ETX = '\x03' # End of Text character
# Sintaxis
# STX + Command + ',' + Argumentos + ETX


# Funciones basicas

def turn(i,d):
	'''Hace girar el motor i pasos en la direccion d
	i es un entero.
	d es interpretado como un booleano.
	'''
	if d:
		cmd = 'sns' + ','+ str(i) + '+'
	else:
		cmd = 'sns' + ','+ str(i) + '-'
	arg = None
	message = STX + cmd + ETX
	# socket
	try:
		ans = sc.send_recv(message, serverHost, serverPort, tOp=1 ,tOut=60)
	except sc.socket.timeout:
		return 'timeout'
	except Exception:
		return 'socket error'
	# comprobacion de la respuesta
	if message == ans:
		pass


def set_step_duration(i):
	'''Fija la duracion del paso
	i es un entero interpretado como microsegundos
	'''
	cmd = 'str' + ',' + str(i)
	arg = None
	message = STX + cmd + ETX
	# socket
```

```
        try:
                ans = sc.send_recv(message, serverHost, serverPort, tOp=2 ,tOut=2)
        except sc.socket.timeout:
                return 'timeout'
        except Exception:
                return 'socket error'
        # comprobacion de la respuesta
        if message == ans:
                return 'ok'

def set_abs_pos(i):
        '''Fija la posicion absoluta del motor
        '''
        cmd = 'sap' + ',' + str(i)
        arg = None
        message = STX + cmd + ETX
        # socket
        try:
                ans = sc.send_recv(message, serverHost, serverPort, tOp=2 ,tOut=2)
        except sc.socket.timeout:
                return 'timeout'
        except Exception:
                return 'socket error'
        # comprobacion de la respuesta
        if message == ans:
                return 'ok'

def get_abs_pos():
        '''Obtiene la posicion absoluta del motor en pasos
        '''
        cmd = 'gap'
        arg = None
        message = STX + cmd + ETX
        # socket
        try:
                ans = sc.send_recv(message, serverHost, serverPort, tOp=2 ,tOut=2)
        except sc.socket.timeout:
                return 'timeout'
        except Exception:
                return 'socket error'
        # comprobacion de la respuesta
        try:
                ans = ans.strip(STX+ETX+',').split(',')
                if cmd[:3] == ans[0]:
                        resp = int(ans[1])
                        return resp
        except Exception:
                return 'corrupcion del mensaje'
```

motorGui

```
# motorGui
# python 3
import tkinter as tk
import tkinter.messagebox as tkm
import threading, queue, time
import motorShieldModule as motor

# CONSTANTES GLOBALES

# Localizacion del servidor volcadas en el modulo analogicModule
motor.serverHost = 'localhost' # 'localhost' o '192.168.2.13'
motor.serverPort = 50000

t_intervalo = 1 # intervalo entre peticiones (en segundos)

# VARIABLES

# Variables globales

buttons = {} # diccionario de objetos graficos; campos a actualizar.
labels  = {} # diccionario de objetos graficos; campos a actualizar.
actividad = {'play':False, 'record':False} # bit, si False => close thread
dataqueue = queue.Queue() # Cola de datos de acceso en exclusion.

posicion = ''
#posicion = 'pos_1'
pasosEntrePos = 740
pasosPos0 = 500
posiciones = {
                          'pos_1': pasosPos0 + 0*pasosEntrePos,
                          'pos_2': pasosPos0 + 1*pasosEntrePos,
                          'pos_3': pasosPos0 + 2*pasosEntrePos,
                          'pos_4': pasosPos0 + 3*pasosEntrePos,
                          'pos_5': pasosPos0 + 4*pasosEntrePos,
                          'pos_6': pasosPos0 + 5*pasosEntrePos,
                          'pos_7': pasosPos0 + 6*pasosEntrePos,
                          'pos_8': pasosPos0 + 7*pasosEntrePos,
                          'reajustar':0
                          }
# Funciones

def steps_to_new(newPos):
      global posicion
      if posicion == '':
            try:
                  get()
                  time.sleep(1)
            except Exception:
                  pass
      else:
```

```
            try:
                  steps = posiciones[newPos] - posiciones[posicion]
                  posicion = newPos
                  if steps > 0: return (steps,1)
                  else: return (abs(steps),0)
            except Exception:
                  return (0,0)


def get():
      global posicion
      resp = motor.get_abs_pos()
      if type(resp) == int:
            for key in posiciones:
                  buttons[key].config(relief='raise')
                  if posiciones[key] == resp:
                        posicion = key
                        buttons[key].config(relief='sunken')
      else:
            return


def pos_1():
      for key in posiciones: buttons[key].config(relief='raise')
      buttons['pos_1'].config(relief='sunken')
      stepd = steps_to_new('pos_1')
      motor.turn(stepd[0],stepd[1])
def pos_2():
      for key in posiciones: buttons[key].config(relief='raise')
      buttons['pos_2'].config(relief='sunken')
      stepd = steps_to_new('pos_2')
      motor.turn(stepd[0],stepd[1])
def pos_3():
      for key in posiciones: buttons[key].config(relief='raise')
      buttons['pos_3'].config(relief='sunken')
      stepd = steps_to_new('pos_3')
      motor.turn(stepd[0],stepd[1])
def pos_4():
      for key in posiciones: buttons[key].config(relief='raise')
      buttons['pos_4'].config(relief='sunken')
      stepd = steps_to_new('pos_4')
      motor.turn(stepd[0],stepd[1])
def pos_5():
      for key in posiciones: buttons[key].config(relief='raise')
      buttons['pos_5'].config(relief='sunken')
      stepd = steps_to_new('pos_5')
      motor.turn(stepd[0],stepd[1])
def pos_6():
      for key in posiciones: buttons[key].config(relief='raise')
      buttons['pos_6'].config(relief='sunken')
      stepd = steps_to_new('pos_6')
      motor.turn(stepd[0],stepd[1])
```

```
def pos_7():
	for key in posiciones: buttons[key].config(relief='raise')
	buttons['pos_7'].config(relief='sunken')
	stepd = steps_to_new('pos_7')
	motor.turn(stepd[0],stepd[1])
def pos_8():
	for key in posiciones: buttons[key].config(relief='raise')
	buttons['pos_8'].config(relief='sunken')
	stepd = steps_to_new('pos_8')
	motor.turn(stepd[0],stepd[1])
def reajustar():
	for key in posiciones: buttons[key].config(relief='raise')
	buttons['reajustar'].config(relief='sunken')
	stepd = steps_to_new('reajustar')
	if stepd[1] == 0: motor.turn(100,1)
	time.sleep(1.5)
	motor.turn(stepd[0]+500,stepd[1])
	time.sleep(10)
	motor.set_abs_pos(0)
	time.sleep(1)
	get()


def repeater(t_refresco=500):
	try:
		data = dataqueue.get(block=False)
	except queue.Empty:
		pass
	else:
		# actualizar GUI
		#labels[data[0]].config(text=data[1])
		pass
	winroot.after(t_refresco, repeater) # llamada periodica ms


def play():
	# lanzador del hilo play_thread
	threading.Thread(target=play_thread).start()

def play_thread():
	# Futuro uso
	while actividad['play']:
		time.sleep(1)
		motor.get_abs_pos()


# Definicion de las ventanas

def notdone():
	tkm.showerror('NO implementado', 'NO disponible.\nEn desarrollo')

def makemenu(win):
```

```
        menubar = tk.Menu(win)
        win.config(menu=menubar)

        filemenu = tk.Menu(menubar)
        filemenu.add_command(label='Save...', command=notdone, underline=0)
        filemenu.add_command(label='Save as', command=notdone, underline=0)
        filemenu.add_separator()
        filemenu.add_command(label='Quit...', command=win.quit,underline=0)
        menubar.add_cascade(label='File', menu=filemenu,   underline=0)

        editmenu = tk.Menu(menubar)
        editmenu.add_command(label='Preferences', command=notdone)
        menubar.add_cascade(label='Edit', menu=editmenu, underline=0)

        toolmenu = tk.Menu(menubar)
        toolmenu.add_command(label='Get position', command=get, underline=0)
        menubar.add_cascade(label='Tools', menu=toolmenu, underline=0)

        helpmenu = tk.Menu(menubar)
        helpmenu.add_command(label='About',command=notdone)
        menubar.add_cascade(label='Help', menu=helpmenu, underline=0)

def makebuttons(win, buttons=buttons, labels=labels):
        '''Construye los botones'''
        frame = tk.Frame(win)
        frame.pack()
        bttn_pos_1 = tk.Button(frame, text='Posicion 1', command=pos_1, width=20)
        bttn_pos_2 = tk.Button(frame, text='Posicion 2', command=pos_2, width=20)
        bttn_pos_3 = tk.Button(frame, text='Posicion 3', command=pos_3, width=20)
        bttn_pos_4 = tk.Button(frame, text='Posicion 4', command=pos_4, width=20)
        bttn_pos_5 = tk.Button(frame, text='Posicion 5', command=pos_5, width=20)
        bttn_pos_6 = tk.Button(frame, text='Posicion 6', command=pos_6, width=20)
        bttn_pos_7 = tk.Button(frame, text='Posicion 7', command=pos_7, width=20)
        bttn_pos_8 = tk.Button(frame, text='Posicion 8', command=pos_8, width=20)
        bttn_reajustar = tk.Button(frame, text='Reajustar', command=reajustar,
width=20)
        bttn_pos_1.pack()
        bttn_pos_2.pack()
        bttn_pos_3.pack()
        bttn_pos_4.pack()
        bttn_pos_5.pack()
        bttn_pos_6.pack()
        bttn_pos_7.pack()
        bttn_pos_8.pack()
        bttn_reajustar.pack()
        buttons['pos_1'] = bttn_pos_1
        buttons['pos_2'] = bttn_pos_2
        buttons['pos_3'] = bttn_pos_3
        buttons['pos_4'] = bttn_pos_4
        buttons['pos_5'] = bttn_pos_5
        buttons['pos_6'] = bttn_pos_6
        buttons['pos_7'] = bttn_pos_7
```

```
        buttons['pos_8'] = bttn_pos_8
        buttons['reajustar'] = bttn_reajustar
        return buttons, labels


def makebar(win, buttons=buttons, labels=labels):
        '''Construye los campos donde se representan las lecturas de los
sensores'''
        # Creacion objeto grafico: Etiqueta del sensor 1
        sensor_1 = tk.Label(frame, text='lectura sensor 1 : ------ ', width=20)
        sensor_1.pack(side='left')
        labels['sensor_1'] = sensor_1
        # Creacion objeto grafico: Etiqueta del sensor 2
        sensor_2 = tk.Label(frame, text='lectura sensor 2 : ------ ', width=20)
        sensor_2.pack()
        labels['sensor_2'] = sensor_2
        # Creacion objeto grafico: Etiqueta del sensor 3
        sensor_3 = tk.Label(win, text='lectura sensor 3 : ------ ', width=20)
        sensor_3.pack()
        labels['sensor_3'] = sensor_3
        return buttons, labels


def maketable(win):
        marco = tk.Frame(win)
        marco.pack()
        # cabecera
        titulo_status = tk.Label(marco, text='st', relief='flat',
                                                        width=3, bg='beige')
        titulo_sensor = tk.Label(marco, text='sensor', relief='flat',
                                                        width=9, bg='beige')
        titulo_data   = tk.Label(marco, text='data', relief='flat',
                                                       width=13, bg='beige')
        titulo_status.pack(side='left')
        titulo_sensor.pack(side='left')
        titulo_data.pack(side='left')
        # tabla
        tabla = [(titulo_status, titulo_sensor, titulo_data)]
        for i in range(1,6+1):
                marco = tk.Frame(win)
                marco.pack()
                status = tk.Label(marco, text='--', relief='ridge', width=3)
                sensor = tk.Label(marco, text=str(i), relief='ridge', width=9)
                data = tk.Label(marco, text='----', relief='sunken', width=13)
                status.pack(side='left')
                sensor.pack(side='left')
                data.pack(side='left')
                tabla.append((status, sensor, data))
        return tabla


#   INICIO PROGRAMA
```

```
if __name__=='__main__':
        winroot = tk.Tk()
        winroot.title('Posicion Motor')
        winroot.after(500,repeater) # actualizacion de la GUI
        makemenu(winroot)
        buttons, labels = makebuttons(winroot)
        #buttons, labels = makebar(winroot)
        # lanzamiento del sistema de eventos
        winroot.mainloop()
        # cierre de hilos
        print('cerrando')
        actividad['play']=False
        actividad['record']=False
```

Conjunto Bronkhorst

Sigue siendo similar a los programas anteriores necesitando como ellos un progama que haga de puente entre la red Ethernet y el dispositivo serie.

bronkhorstServer. Programa que hace de puente entre la red Ethernet y el dispositivo serie.

SocketClient. El mismo módulo que en los casos anteriores, que se encarga de hacer las conexiones TCP/IP entre la interfaz gráfica y el servidor que hace de puente.

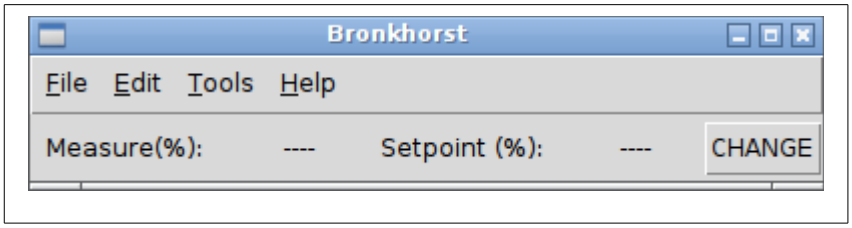


Figura 3. Interfaz gráfica bronkhorstGui

bronkohorstModule. Instrucciones para operar con el caudalímetro Bronkhorst.

bronkhorstGui. Interfaz gráfica donde se puede seleccionar el punto de trabajo (como válvula) y conocer el flujo (Measure). Figura 3.

bronkhorstServer

```
# bronkhorstServer
# Python 3
# Programa puente
# Acepta una conexion a traves de un socket TCP/IP que hace de puente
# con un puerto serie

import socket
import serial
import time
import threading

# ---Constantes---

STX = b':'
ETX = b'\n'

# ---Variables---

actividad = {}                          # Diccionario con el estado
actividad['run']    = True  # Si False => salir del programa
actividad['serial'] = False # Estado del puerto serie

socketHost  = ''
socketPort  = 50000
socketGenerator = None

serialPort  = '/dev/ttyUSB0'
#serialPort  = '/dev/ttyACM0'
objectSerial = None

serialTimeOut  = 2 # segundos; tiempo maximo de espera
serialTimeOp   = 0.05 # segundos; tiempo estimado de operacion


# ---Clases---
class Message():
       '''Clase Mensaje'''
       def __init__(self, byteString = b'', n=0):
              self.bytesIn = byteString
              self.bytesOut = b''
              self.time = time.time()
              self.contador = n

       def write(self, objectSerial):
              objectSerial.write(self.bytesIn)

       def read(self, objectSerial):
              self.bytesOut = b''
              time0 = time.time()
              while True:
```

```
                data = objectSerial.read()
                self.bytesOut += data
                if data == STX: self.bytesOut = STX
                if (data == ETX) and (objectSerial.inWaiting() == 0): break
                if time.time() > time0+serialTimeOut:
                        raise serial.SerialTimeoutException

        def serie(self, objectSerial):
                print('Enviando y recibiendo por serie')      #
                try:
                        self.write(objectSerial)
                        time.sleep(serialTimeOp)
                        self.read(objectSerial)
                except serial.SerialTimeoutException:
                        print(message.bytesIn)    #
                        print(message.bytesOut)   #
                        print('SerialTimeoutException')
                        pass
                except serial.SerialException:
                        print('Conexion serie caida')
                        actividad['serial'] = False
                        objectSerial.close()
                        objectSerial == None
                        threading.Thread(target=serial_up).start() # Thread
                except Exception as x:
                        actividad['serial'] = False
                        objectSerial.close()
                        objectSerial == None
                        threading.Thread(target=serial_up).start() # Thread
                        print('Otro tipo de excepcion\nSerial down\n', x)
                else:
                        return
                self.bytesIn = STX + b'error' + ETX


# ---Funciones---

#    SERIE

def serial_up():
        '''Thread that make an object serial instance
        Hilo para establecer la conexion serie
        Crea una instancia de un objeto serial (GLOBAL objectSerial).
        El objeto esta conectado al puerto proporcionado por serialPort
        '''
        global objectSerial, serialPort
        while not actividad['serial'] and actividad['run']:
                try:
                        objectSerial = serial.Serial(
                                                                serialPort,
                                                                baudrate=38400,
                                                                timeout=1,
```

```
                                                                writeTimeout=1)
            # baudrate=38400
            except serial.serialutil.SerialException:
                  print('Excepcion:\n',
                  'a) El dispositivo no se encuentra en el puerto indicado.\n',
                  'b) o no está conectado.\n> Reintentando en 5 segundos')
                  # enviar un mensaje al agente que se ha conectado
                  time.sleep(5)
            else:
                  time.sleep(3)# Tiempo de activacion
                  print('Conexion serie up')
                  actividad['serial'] = True


#    SOCKET

def socket_up():
      global genSocket
      try:
            genSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
      except OSError:
            print('Excepcion al crear una instancia del tipo socket')
            print('--- Cerrando ---')
      try:
            genSocket.bind((socketHost, socketPort))
            genSocket.listen(5)
            print('Socket up.')
            print('  Socket host:',socket.gethostname())
            print('  Socket port:',socketPort)
      except OSError:
            print('Excepcion al crear el socket receptor de peticiones de
conexion',
                  'puede ser debido a que el puerto o la direccion IP',
socketPort,
                  'estan siendo usados por otra aplicación')
            print('--- Cerrando ---')
            genSocket.close()
            actividad['run'] = False


# ----- INICIO PROGRAMA -----
print('--- iniciando ---')
threading.Thread(target=serial_up).start() # Thread
socket_up()
while actividad['run']: # Bucle
      try:
            # Crea un socket cliente tras una peticion de conexion
            print('Waiting nueva conexion TCP/IP...')
            conexionRed, address = genSocket.accept()
            conexionRed.settimeout(2)
            print('Conexion con', address)
            # recepcion del mensaje
```

```
        data = b''
        while not ETX in data: # reconstruye el mensaje
                trozo = conexionRed.recv(1024)
                if not trozo: break
                data += trozo
        message = Message(data)
        if actividad['serial']: message.serie(objectSerial)
        else: message.bytesOut = STX + b'error,serial_down' + ETX
        # contestacion
        conexionRed.sendall(message.bytesOut)
except socket.timeout:
        pass
except Exception:
        print('Exception')
finally:
        conexionRed.close()
```

socketClient

```
# socketClient
# Modulo para la comunicacion como cliente por socket
# especifico para el ArduinoServer.py
# Caracteristicas:
#     Propaga excepciones
#     Cierra siempre el socket
import socket
import time

def send_recv(message, ip='localhost', port=50008, tOp=0, tOut=1):
      '''
      Envia un mensaje y espera su respuesta.
      Sintaxis:
            send_recv(message, ip, port, t_operacion, t_out)
      Parametros pasados a la funcion:
                  message = cadena Unicode o ASCII
                  ip = cadena de texto de la direccion IP del servidor
                  port = entero del puerto de escucha del servidor
                  t_operacion = numero real con el tiempo de duracion de la
operacion
                  t_out = numero real con el tiempo de espera en las operaciones
de
                        conexion, envio, y recepcion.
                  t_operacion + t_out + t_out = t_max
      Respuesta:
            cadena Unicode.
      '''
      try:
            so = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            so.settimeout(tOut)
            so.connect((ip,port))
            #print(so.getpeername())
            #print(so.getsockname())
            so.sendall(bytes(message, encoding='ascii'))
            so.shutdown(socket.SHUT_WR)
            so.settimeout(tOut+tOp)
            resp = b''
            while True:
                  trozo = so.recv(1024)
                  if not trozo: break
                  resp += trozo
      finally:
            so.close()
      return str(resp, encoding='ascii')
```

bronkhorstModule

```
# bronkhorstModule
# python 3
# Funciones basicas
#

import socketClient as sc

# Direccion y puerto donde se encuentra el servidor
serverHost = 'localhost' #
serverPort = 50000

STX = ':'
ETX = '\n'

def set_point(n):
	'''
	Selecciona el punto de trabajo, set_point(n) con n entero de 0 a 32000.
	Que se corresponde al 0 y al 100% de la abertura de la valvula
	'''
	#n = bytes(hex(n), encoding='ascii')[2:] # valor hexadecimal 16000=3E80
	if (0 > n and n > 3200): return 'fuera de rango'
	if type(n) == int: n = int(n)
	else: return 'type error'
	n = ('0000' + hex(n)[2:])[-4] # 4 digitos hexadecimales
	l = '06'
	nodo = '03'
	comando = '01'
	proceso = '01'
	tipo = '21'
	message = ':'+ l + nodo + comando + proceso + tipo + n + '\r\n'
	# socket
	try:
		ans = sc.send_recv(message, serverHost, serverPort, tOp=1, tOut=1)
	except sc.socket.timeout:
		return 'timeout'
	except Exception:
		return 'socket error'
	# comprobacion de la respuesta
	try:
		if int(ans[7:9]) == 0: ans = 0
		else:
			ans = 'error'
	except Exception:
		ans = 'exception'
	return ans

def get_point():
	# try o rango de n
	l = '06'
	nodo = '03'
```

```
        comando = '04'
        proceso = '01'
        tipo = '21'
        message = (':'+ l + nodo + comando + proceso + tipo
                                                + proceso + tipo + '\r\n')
        # socket
        try:
                ans = sc.send_recv(message, serverHost, serverPort, tOp=1, tOut=1)
        except sc.socket.timeout:
                return 'timeout'
        except Exception:
                return 'socket error'
        # comprobacion de la respuesta
        try:
                ans = int(ans[11:15],16) # de hexadecimal string a entero
        except Exception:
                ans = 'exception'
        return ans

def measure():
        '''
        Hace una medida del flujo.
        valor entero que puede tomar valores en ter -23593 a 41942, aunque
        32000 equivale al 100%
        '''
        l = '06'
        nodo = '03'
        comando = '04' # request parameter
        proceso = '01' # proceso
        tipo = '21'    # entero + index (tipo del valor que me a a devolver)
        proceso = '01' # proceso
        parametro = '20'  # tipo entero + FBnr (tipo del parametro)
        message = (':'+ l + nodo + comando + proceso + tipo +
                                                proceso + parametro + '\r\n')
        # socket
        try:
                ans = sc.send_recv(message, serverHost, serverPort, tOp=1, tOut=1)
        except sc.socket.timeout:
                return 'timeout'
        except Exception:
                return 'socket error'
        # comprobacion de la respuesta
        try:
                ans = int(ans[11:15],16) # de hexadecimal string a entero
        except Exception:
                ans = 'exception'
        return ans

# ---- funciones no usadas

def close():
        '''
```

```
        tipo string de 8 caracters
        '''
        l = '0C' # 12
        nodo = '03'
        comando = '02' # send parameter sin peticion de estado
        proceso = '01' # proceso
        parametro = '06'  # tipo caracter (0) + FBnr (parametro)
        valor = '0101010101010101'
        message = (':'+ l + nodo + comando + proceso + parametro
                           + valor + '\r\n')

def open():
        '''
        tipo string de 8 caracters
        '''
        l = '0C' # 12
        nodo = '03'
        comando = '02' # send parameter sin peticion de estado
        proceso = '01' # proceso
        parametro = '06'  # tipo caracter (0) + FBnr (parametro)
        valor = '0000000000000000'
        message = (':'+ l + nodo + comando + proceso + parametro
                           + valor + '\r\n')

def status():
        '''
        Estado de la valvula open/close
        '''
        l = '0C' # 12
        nodo = '03'
        comando = '04' # send parameter sin peticion de estado
        proceso = '01' # proceso
        parametro = '06'  # tipo caracter (0) + FBnr (parametro)
        valor = '0000000000000000'
        message = (':'+ l + nodo + comando + proceso + parametro
                           + valor + '\r\n')
```

bronkhorstGui

```
# bronkhorstGui
# python 3
# interfaz grafica para caudalimetro Bronkhorst
import tkinter as tk
import tkinter.messagebox as tkm
import threading, queue, time
import bronkhorstModule as br

# CONSTANTES GLOBALES

# VARIABLES

# Variables globales

botones = {} # diccionario de objetos graficos Button. record, play, stop.
labels = {}  # diccionario de objetos graficos Labels, campos a actualizar.
dialogos = {}
actividad = {'run':True, 'save':False } # boolean, if False => close thread
dataqueue = queue.Queue() # Cola de datos de acceso en exclusión mutua


# Funciones

def repeater():
	'''Actualizacion periodica'''
	try:
		data = dataqueue.get(block=False)
	except queue.Empty:
		pass
	else:
		# actualizar GUI
		labels[data[0]].config(text=data[1])
	winroot.after(100, repeater) # programar nueva llamada

def reading():
	threading.Thread(target=reading_thread).start()
def reading_thread():
	while actividad['run']:
		try:
			ans = br.get_point()
			ans =  ans
			dataqueue.put(('point','%.1f' % (ans/32)))
			time.sleep(0.5)
			ans = br.measure()
			dataqueue.put(('measure','%.1f' % (ans/32)))
			time.sleep(0.5)
		except Exception:
			print('excepcion')
			time.sleep(0.5)
```

```
def dialog_change_point():
	def push():
		valor = dialogo.get()
		change_point(valor)
		popup.destroy()
	popup = tk.Toplevel()
	dialogo = tk.Entry(popup)
	dialogo.pack(side='top')
	tk.Button(popup, text='Cancel', command=popup.destroy).pack(side='left')
	tk.Button(popup, text = 'Apply',command=push).pack(side='right')
def change_point(valor):
	threading.Thread(target=change_point_thread,args=(valor,)).start()
def change_point_thread(valor):
	try:
		valor = int(valor)
	except:
		pass
	else:
		if ((valor >= 0) and (valor < 100)):
			try:
				br.set_point(int(valor/32))
			except Exception:
				pass


# Definicion de las ventanas

def notdone():
	tkm.showerror('NO implementado', 'NO disponible.\nEn desarrollo')

def makemenu(win):
	menubar = tk.Menu(win)
	win.config(menu=menubar)

	filemenu = tk.Menu(menubar)
	filemenu.add_command(label='Save...', command=notdone, underline=0)
	filemenu.add_command(label='Save as', command=notdone, underline=0)
	filemenu.add_separator()
	filemenu.add_command(label='Quit...', command=win.quit,underline=0)
	menubar.add_cascade(label='File', menu=filemenu,   underline=0)

	editmenu = tk.Menu(menubar)
	editmenu.add_command(label='Preferences', command=notdone)
	menubar.add_cascade(label='Edit', menu=editmenu, underline=0)

	toolmenu = tk.Menu(menubar)
	menubar.add_cascade(label='Tools', menu=toolmenu, underline=0)

	helpmenu = tk.Menu(menubar)
	helpmenu.add_command(label='About',command=notdone)
	menubar.add_cascade(label='Help', menu=helpmenu, underline=0)
```

```
def makeBar(win, botones, labels):
       measure_text= tk.Label(win, text='Measure(%): ', width=12, height=2)
       measure_label = tk.Label(win, text='----', width=8)
       measure_text.pack(side='left')
       measure_label.pack(side='left')
       point_text= tk.Label(win, text='Setpoint (%): ', width=12, height=2)
       point_label = tk.Label(win, text='----', width=8)
       point_text.pack(side='left')
       point_label.pack(side='left')

       boton_point = tk.Button(win, text='CHANGE',
                                    command=dialog_change_point, width=4)
       boton_point.pack(side='left')
       labels['measure']=measure_label
       labels['point']=point_label
       botones['point']=boton_point
       return botones, labels



#   INICIO PROGRAMA

if __name__=='__main__':
       winroot = tk.Tk()
       winroot.title('Bronkhorst')
       winroot.after(100,repeater) # actualizacion de la GUI
       makemenu(winroot)
       topmarco=tk.Frame()
       topmarco.pack()
       #botones, labels = makecanalbar(topmarco, botones, labels)
       botones, labels = makeBar(winroot, botones, labels)
       reading()
       # lanzamiento del sistema de eventos
       winroot.mainloop()
       # cierre de hilos
       actividad['record']=False
       actividad['run'] = False
```

Conjunto MaxiGauge

Otro dispositivo serie con lo que la estructura general a los vistos anteriormente.

maxiGaugeServer. Programa que hace de puente entre la red Ethernet y el dispositivo serie.

SocketClient. El mismo módulo que en los casos anteriores, que se encarga de hacer las conexiones TCP/IP entre la interfaz gráfica y el servidor que hace de puente.

maxiGaugeModule. Instrucciones para operar con la MaxiGauge.

maxiGaugeGui. Interfaz gráfica en la que aparece el estado de los sensores y su medida. Figura 4.

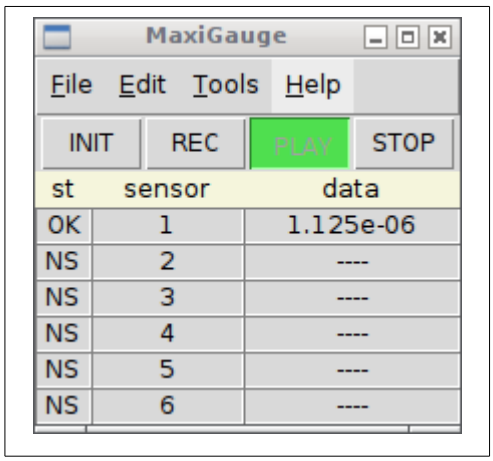


Figura 4. Interfaz gráfica maxiGaugeGui

maxiGaugeServer

```
# mauxiGaugeServer
# Python 3
# Programa puente
# Acepta una conexion a traves de un socket TCP/IP que hace de puente
# con un puerto serie

import socket
import serial
import time
import threading

# ---Constantes---

STX= b'\x02'
ETX= b'\x03' # End of Text. Reset buffer.
CR = b'\x0D' # Carriage Return.
LF = b'\x0A' # Line Feed. Better don't send.
ENQ= b'\x05' # Enquiry. Request fo data transmission
ACK= b'\x06' # Acknowledge.
NAK= b'\x15' # No acknowledge.
ESC= b'\x1B' # Escape.

# ---Variables---

actividad = {}                          # Diccionario con el estado
actividad['run']    = True  # Si False => salir del programa
actividad['serial'] = False # Estado del puerto serie

socketHost  = ''
socketPort  = 50000
socketGenerator = None

serialPort  = '/dev/ttyUSB0'
#serialPort  = '/dev/ttyACM0'
objectSerial = None

serialTimeOut  = 2 # segundos; tiempo maximo de espera
serialTimeOp   = 0.5 # segundos; tiempo estimado de operacion


# ---Clases---
class Message():
       '''Clase Mensaje'''
       def __init__(self, byteString = b'', n=0):
              self.bytesIn = byteString
              self.bytesOut = b''
              self.time = time.time()
              self.contador = n

       def write(self, objectSerial):
```

```
            objectSerial.write(self.bytesIn)

        def read(self, objectSerial):
            self.bytesOut = b''
            time0 = time.time()
            while True:
                data = objectSerial.read()
                self.bytesOut += data
                if (data == LF) and (objectSerial.inWaiting() == 0): break
                if time.time() > time0+serialTimeOut+serialTimeOp:
                    raise serial.SerialTimeoutException

        def serie(self, objectSerial):
            print('Enviando y recibiendo por serie')      #
            try:
                lineas = self.bytesIn.strip(STX+CR+ETX).split(CR)
                for linea in lineas:
                    self.bytesIn = linea+CR
                    self.write(objectSerial)
                    self.read(objectSerial)
                    if self.bytesOut == ACK+CR+LF: pass
                    elif self.bytesOut == NAK+CR+LF: break
                    else: break
            except serial.SerialTimeoutException:
                print(message.bytesIn)    #
                print(message.bytesOut)   #
                print('SerialTimeoutException')
                pass
            except serial.SerialException:
                print('Conexion serie caida')
                actividad['serial'] = False
                objectSerial.close()
                objectSerial == None
                threading.Thread(target=serial_up).start() # Thread
            except Exception as x:
                actividad['serial'] = False
                objectSerial.close()
                objectSerial == None
                threading.Thread(target=serial_up).start() # Thread
                print('Otro tipo de excepcion\nSerial down\n', x)
            else:
                self.bytesOut = STX + self.bytesOut + ETX
                return
            self.bytesIn = STX + b'error' + ETX


# ---Funciones---

#    SERIE

def serial_up():
    '''Thread that make an object serial instance
```

```
	Hilo para establecer la conexion serie
	Crea una instancia de un objeto serial (GLOBAL objectSerial).
	El objeto esta conectado al puerto proporcionado por serialPort
	'''
	global objectSerial, serialPort
	while not actividad['serial'] and actividad['run']:
		try:
			objectSerial = serial.Serial(
									serialPort,
									baudrate=9600,
									timeout=1,
									writeTimeout=1)
		# baudrate=38400
		except serial.serialutil.SerialException:
			print('Excepcion:\n',
			'a) El dispositivo no se encuentra en el puerto indicado.\n',
			'b) o no está conectado.\n> Reintentando en 5 segundos')
			# enviar un mensaje al agente que se ha conectado
			time.sleep(5)
		else:
			time.sleep(3)# Tiempo de activacion
			print('Conexion serie up')
			actividad['serial'] = True


#    SOCKET

def socket_up():
	global genSocket
	try:
		genSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
	except OSError:
		print('Excepcion al crear una instancia del tipo socket')
		print('--- Cerrando ---')
	try:
		genSocket.bind((socketHost, socketPort))
		genSocket.listen(5)
		print('Socket up.')
		print('  Socket host:',socket.gethostname())
		print('  Socket port:',socketPort)
	except OSError:
		print('Excepcion al crear el socket receptor de peticiones de
conexion',
			'puede ser debido a que el puerto o la direccion IP',
socketPort,
			'estan siendo usados por otra aplicación')
		print('--- Cerrando ---')
		genSocket.close()
		actividad['run'] = False


# ----- INICIO PROGRAMA -----
```

```
print('--- iniciando ---')
threading.Thread(target=serial_up).start() # Thread
socket_up()
while actividad['run']: # Bucle
	try:
		# Crea un socket cliente tras una peticion de conexion
		print('Waiting nueva conexion TCP/IP...')
		conexionRed, address = genSocket.accept()
		conexionRed.settimeout(2)
		print('Conexion con', address)
		# recepcion del mensaje
		data = b''
		while not ETX in data: # reconstruye el mensaje
			trozo = conexionRed.recv(1024)
			if not trozo: break
			data += trozo
		message = Message(data)
		if actividad['serial']: message.serie(objectSerial)
		else: message.bytesOut = STX + b'error,serial_down' + ETX
		# contestacion
		conexionRed.sendall(message.bytesOut)
	except socket.timeout:
		pass
	except Exception:
		print('Exception')
	finally:
		conexionRed.close()
```

socketClient

```
# socketClient
# Modulo para la comunicacion como cliente por socket
# especifico para el ArduinoServer.py
# Caracteristicas:
#     Propaga excepciones
#     Cierra siempre el socket
import socket
import time

def send_recv(message, ip='localhost', port=50008, tOp=0.1, tOut=1):
      '''
      Envia un mensaje y espera su respuesta.
      Sintaxis:
            send_recv(message, ip, port, t_operacion, t_out)
      Parametros pasados a la funcion:
                  message = cadena Unicode o ASCII
                  ip = cadena de texto de la direccion IP del servidor
                  port = entero del puerto de escucha del servidor
                  t_operacion = numero real con el tiempo de duracion de la
operacion
                  t_out = numero real con el tiempo de espera en las operaciones
de conexion, envio, y recepcion.
                  t_operacion + t_out + t_out = t_max
      Respuesta:
            cadena Unicode.
      '''
      try:
            so = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            so.settimeout(tOut)
            so.connect((ip,port))
            #print(so.getpeername())
            #print(so.getsockname())
            so.sendall(bytes(message, encoding='ascii'))
            so.shutdown(socket.SHUT_WR)
            so.settimeout(tOut+tOp)
            resp = b''
            while True:
                  trozo = so.recv(1024)
                  if not trozo: break
                  resp += trozo
      finally:
            so.close()
      return str(resp, encoding='ascii')
```

maxiGaugeModule

```
# MaxiGaugeModule
# Python 3
# - Traduce ordenes a la maxigauge en cadenas comprensibles por esta
# y las envia a un servidor que se encarga de mandarlas a la MaxiGauge.

import socketClient as sc

# direccion IP del servidor
serverHost = 'localhost'
serverPort = 50000

STX= '\x02'
ETX= '\x03' # End of Text. Reset buffer.
CR = '\x0D' # Carriage Return.
LF = '\x0A' # Line Feed. Better don't send.
ENQ= '\x05' # Enquiry. Request fo data transmission
ACK= '\x06' # Acknowledge.
NAK= '\x15' # No acknowledge.
ESC= '\x1B' # Escape.

# Funciones basicas

def read_press(sensor):
	''' sensor es un valor de 1 a 6'''
	# devuelve una tupla con el estado (tipo entero) y un valor (tipo float)
	message = STX + 'PR' + str(sensor) + CR + ENQ + CR + ETX
	# socket
	try:
		ans = sc.send_recv(message, serverHost, serverPort ,tOp=1, tOut=1)
	except sc.socket.timeout:
		return 'timeout'
	except Exception:
		return 'socket error'
	# comprobacion de la respuesta
	try:
		if ans == NAK: return 'error'
		ans = ans.strip(STX+ETX+','+CR+LF).split(',')
		return (int(ans[0]), float(ans[1]))
	except Exception:
		return 'corrupcion del mensaje'

def sensor_on(i):
	try:
		i = i-1
		lista = ['0','0','0','0','0','0']
		lista[i] = '2'
		y=''
		for x in lista:
			y += ','+x
	except Exception:
```

```
        return 'Out of range'
    message = STX + 'SEN' + y + CR + ENQ + CR + ETX
    try:
        ans = sc.send_recv(message, serverHost, serverPort ,tOp=1, tOut=1)
    except sc.socket.timeout:
        return 'timeout'
    except Exception:
        return 'socket error'
    # comprobacion de la respuesta
    try:
        ans = ans.strip(STX+ETX+','+CR+LF).split(',')
        if ans[0] == NAK:
            return 'NAK'
    except Exception:
        return 'corrupcion del mensaje'
    return ans
```

maxiGaugeGui

```
import tkinter as tk
import tkinter.messagebox as tkm
import threading, queue, time
import maxiGaugeModule as mxg

# CONSTANTES GLOBALES
estado_sensor = ['OK','Und','Ove','Err','Off','NS','IdE']

# VARIABLES
# Variables globales

botones = {} # diccionario de objetos graficos, botones record, play, stop.
tabla = []   # lista de objetos graficos, ventanas de actualizacion
actividad = {'play':False, 'record':False} # bit, si False => close thread
dataqueue = queue.Queue() # Cola de datos de acceso

# Funciones

def repeater():
	try:
		data = dataqueue.get(block=False)
	except queue.Empty:
		pass
	else:
		# actualizar GUI
		tabla[data[0]][0].config(text=estado_sensor[data[1]])
		tabla[data[0]][2].config(text=str(data[2]))
	winroot.after(100, repeater) # llamada periodica

def record():
	botones['record'].config(relief='sunken')
	botones['play'].config(relief='sunken')
	botones['record'].config(state='disabled',bg='#df4f4f')
	botones['play'].config(state='disabled', bg='#4fdf4f')
	if actividad['record'] == True:
		pass
	elif actividad['record'] == False:
		actividad['record']=True
		actividad['play']=False
		# lanza el hilo
		threading.Thread(target=record_thread).start()
		print('record')

def record_thread():
	fichero = open('maxiGauge_data.txt', 'a')
	fichero.write('----- Inicio de sesion -----\n')
	while actividad['record']:
		linea = ''
		for i in range(1,7):
			try:
```

```
                        ans = mxg.read_press(i)
                        (st, valor) = ans
                    except Exception:
                        pass
                    dataqueue.put((i,st,valor))
                    #tabla[i][0].config(text=estado_sensor[st])
                    #tabla[i][2].config(text=str(valor))
                    linea += str(valor) + ','
            fichero.write(linea + str(time.time()) + '\n')
            fichero.flush()
            time.sleep(1)
      fichero.close()

def play():
      botones['play'].config(relief='sunken')
      botones['play'].config(state='disabled', bg='#4fdf4f')
      if actividad['play']==True or actividad['record']== True:
            pass
      elif actividad['play'] == False:
            actividad['play']=True
            threading.Thread(target=play_thread).start()
            print('play')

def play_thread():
      while actividad['play']:
            for i in range(1,7):
                  try:
                        ans = mxg.read_press(i)
                        (st, valor) = ans
                        if st==5: valor ='----'
                        dataqueue.put((i,st,valor))
                  except Exception:
                        pass
                  #tabla[i][0].config(text=estado_sensor[st])
                  #tabla[i][2].config(text=str(valor))
            time.sleep(1)

def stop():
      botones['record'].config(relief='raised')
      botones['play'].config(relief='raised')
      botones['record'].config(state='normal', bg='light grey')
      botones['play'].config(state='normal', bg='light grey')
      actividad['record']=False
      actividad['play']=False
      print('stop')

def init():
      ''' Encendido de los sensores de la maxigauge
      '''
      for i in [1,2,3,4,5,6]:
            try:
                  ans = mxg.sensor_on(i)
```

```
            except:
                  print('exception')

# Definicion de las ventanas

def notdone():
      tkm.showerror('NO implementado', 'NO disponible.\nEn desarrollo')

def makemenu(win):
      menubar = tk.Menu(win)
      win.config(menu=menubar)

      filemenu = tk.Menu(menubar)
      filemenu.add_command(label='Save...', command=notdone, underline=0)
      filemenu.add_command(label='Save as', command=notdone, underline=0)
      filemenu.add_separator()
      filemenu.add_command(label='Quit...', command=win.quit,underline=0)
      menubar.add_cascade(label='File', menu=filemenu,   underline=0)

      editmenu = tk.Menu(menubar)
      editmenu.add_command(label='Preferences', command=notdone)
      menubar.add_cascade(label='Edit', menu=editmenu, underline=0)

      toolmenu = tk.Menu(menubar)
      toolmenu.add_command(label='Init', command=notdone, underline=0)
      toolmenu.add_command(label='Play', command=notdone, underline=0)
      menubar.add_cascade(label='Tools', menu=toolmenu, underline=0)

      helpmenu = tk.Menu(menubar)
      helpmenu.add_command(label='About',command=notdone)
      menubar.add_cascade(label='Help', menu=helpmenu, underline=0)

def makecommandbar(win):
      marco = tk.Frame(win)
      marco.pack()
      boton_init = tk.Button(marco, text='INIT', command=init, width=3)
      boton_rec  = tk.Button(marco, text='REC', command=record, width=3)
      boton_play = tk.Button(marco, text='PLAY', command=play, width=3)
      boton_stop = tk.Button(marco, text='STOP', command=stop, width=3)
      boton_init.pack(side='left')
      boton_rec.pack(side='left')
      boton_play.pack(side='left')
      boton_stop.pack(side='left')
      return { 'init': boton_init,
               'record': boton_rec,
               'play': boton_play,
               'stop': boton_stop}

def maketable(win):
      marco = tk.Frame(win)
      marco.pack()
      # cabecera
```

```
        titulo_status = tk.Label(marco, text='st', relief='flat',
                                                              width=3, bg='beige')
        titulo_sensor = tk.Label(marco, text='sensor', relief='flat',
                                                              width=9, bg='beige')
        titulo_data   = tk.Label(marco, text='data', relief='flat',
                                                            width=13, bg='beige')
        titulo_status.pack(side='left')
        titulo_sensor.pack(side='left')
        titulo_data.pack(side='left')
        # tabla
        tabla = [(titulo_status, titulo_sensor, titulo_data)]
        for i in range(1,6+1):
               marco = tk.Frame(win)
               marco.pack()
               status = tk.Label(marco, text='--', relief='ridge', width=3)
               sensor = tk.Label(marco, text=str(i), relief='ridge', width=9)
               data = tk.Label(marco, text='----', relief='sunken', width=13)
               status.pack(side='left')
               sensor.pack(side='left')
               data.pack(side='left')
               tabla.append((status, sensor, data))
        return tabla


#   INICIO PROGRAMA

if __name__=='__main__':
        winroot = tk.Tk()
        winroot.title('MaxiGauge')
        winroot.after(100,repeater) # actualizacion de la GUI
        makemenu(winroot)
        botones = makecommandbar(winroot)
        tabla = maketable(winroot)
        # lanzamiento del sistema de eventos
        winroot.mainloop()
        # cierre de hilos
        actividad['play']=False
        actividad['record']=False
```

Conjunto Spellman

Primer dispositivo que aparece con conexión Ethernet. Por lo que no se va a necesitar el programa servidor que hacía de puente con el dispositivo serie.

spellmanModule. Módulo donde se recogen las instrucciones para operar con la Spellman.

spellamGui. Interfaz gráfica para monitorizar y actuar sobre la Spellman. Figura 5.

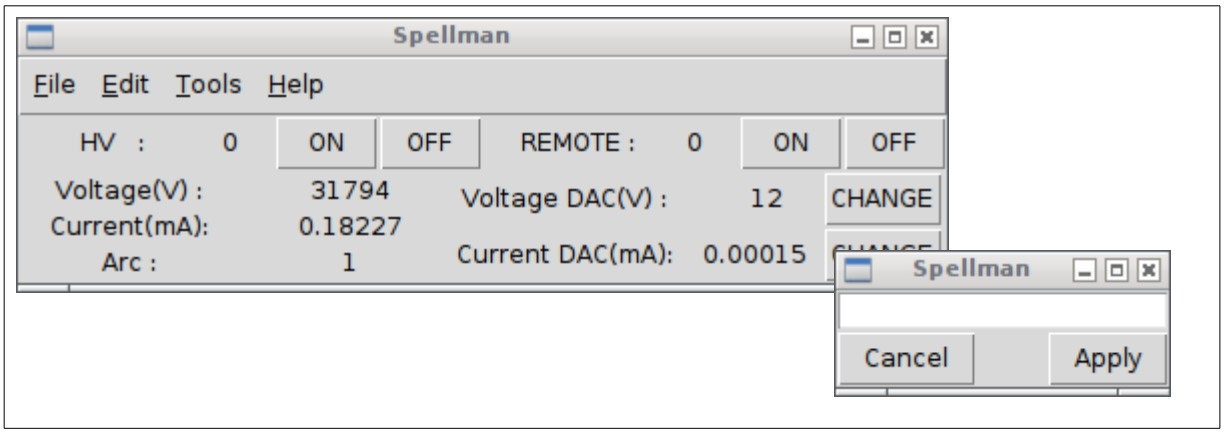


Figura 5. Interfaz gráfica spellmaGui

spellmanModule

```
# spellmanModule
# python 3
# modulo para enviar comandos por sockets TCP/IP
import sys
import socket
import time

serverHost = '192.168.1.4'
serverHost = 'localhost' # Conexion serie
serverPort = 50000

STX = b'\x02' # Start of Text character
ETX = b'\x03' # End of Text character
SUCCESS = b'\x24' # caracter $
# STX + Command + ',' + Argumentos + ',' + ETX


# Funciones basicas

def conect():
	sockobject = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
	sockobject.settimeout(3)
	sockobject.connect((serverHost, serverPort))
	return sockobject

def send(sockobject, byte_string):
	sockobject.send(byte_string)

def receive(sockobject):
	data = b''
	while True:
		try:
			data += sockobject.recv(1024)
		except socket.timeout:
			return data
		if ETX in data:
			return data

def close_socket(sockobject):
	sockobject.close()

# Funciones compuestas

def command(cmd,arg=None):
	cmd = bytes(str(cmd), encoding = 'ascii')
	if arg == None:
		message = STX + cmd + b'\x2C' + ETX
	else:
		arg = bytes(str(arg), encoding = 'ascii')
		message = STX + cmd + b'\x2C' + arg + b'\x2C' + ETX
```

```
        print('sending:',message)
        try:
                conexion = conect()
                time.sleep(0.001) # tiempo extra para crear la conexion
                send(conexion,message)
                data = receive(conexion)
                if (STX+cmd) in data:
                        data = data[data.find(STX+cmd):]
                        if ETX in data:
                                data = data[:data.find(ETX)]
                        else: data = b''
                else:data = b''
        except Exception:
                data = b''
        finally:
                close_socket(conexion)
        print('receiving',data)
        return data

# Funciones especificas

def set_DAC(i,n):
        '''Set digital analog converter
        i=0 Voltage, arg is a integer = max_V/4096
        i=1 Current, arg is a integer = max_I/4096'''
        cmd = 10+i
        arg = n
        try:
                ans = command(cmd,arg).strip(STX+ETX+b',').split(b',')
                return ans[0],ans[1]
        except Exception:
                return b''

def request_DAC(i):
        '''Redquest digital analog converter
        i=0 Voltage, arg is a integer = max_V/4096
        i=1 Current, arg is a integer = max_I/4096
        return a tupla with command, value'''
        cmd = 14+i
        arg = None
        try:
                ans = command(cmd,arg).strip(STX+ETX+b',').split(b',')
                return ans[0],ans[1]
        except Exception:
                return b''

def analog():
        '''Request analog inputs
        return a tupla with command and 7 args
        ans[3] = voltage, integer = max_V/4096
        ans[4] = current, integer = max_I/4096'''
        cmd = 20
```

```
	arg = None
	try:
		ans = command(cmd,arg).strip(STX+ETX+b',').split(b',')
		return ans[0],ans[1],ans[2],ans[3],ans[4],ans[5],ans[6],ans[7]
	except Exception:
		return b''

def system():
	'''Request system status
	return a tupla with command, HV on/off, interlock, fault
	ans[1] High Voltage on/off'''
	cmd = 22
	arg = None
	try:
		ans = command(cmd,arg).strip(STX+ETX+b',').split(b',')
		return ans[0],ans[1],ans[2],ans[3]
	except Exception:
		return b''

def status():
	'''Request digital input status
	return a tupla with command and 7 args
	ans[3] remote on(1)/off(0)
	ans[7] ARC'''
	cmd = 76
	arg = None
	try:
		ans = command(cmd,arg).strip(STX+ETX+b',').split(b',')
		return
ans[0],ans[1],ans[2],ans[3],ans[4],ans[5],ans[6],ans[7],ans[8]
	except Exception:
		return b''

def remote_on():
	'''Turn on remote mode
	return a tupla with command code and error code ($=>ok)'''
	cmd = 85
	arg = 1
	try:
		ans = command(cmd,arg).strip(STX+ETX+b',').split(b',')
		return ans[0],ans[1]
	except Exception:
		return b''

def remote_off():
	'''Turn off remote mode
	return a tupla with command code and error code ($=>ok)'''
	cmd = 85
	arg = 0
	try:
		ans = command(cmd,arg).strip(STX+ETX+b',').split(b',')
		return ans[0],ans[1]
```

```
        except Exception:
                return b''

def hv_on():
        '''Turn on high voltage
        return a tupla with command code and error code ($=>ok)'''
        cmd = 99
        arg = 1
        try:
                ans = command(cmd,arg).strip(STX+ETX+b',').split(b',')
                return ans[0],ans[1]
        except Exception:
                return b''

def hv_off():
        '''Turn off high voltage
        return a tupla with command code and error code ($=>ok)'''
        cmd = 99
        arg = 0
        try:
                ans = command(cmd,arg).strip(STX+ETX+b',').split(b',')
                return ans[0],ans[1]
        except Exception:
                return b''

def version():
        cmd = 25
        arg = None
        command(cmd,arg)

def request_network_settings():
        cmd = 50
        arg = None
        try:
                ans = command(cmd,arg).strip(STX+ETX+b',').split(b',')
                return ans[0],ans[1],ans[2],ans[3],ans[4],ans[5],ans[6]
        except Exception:
                return b''

def set_network_settings(name,ip,port,mask,gateway,MAC):
        '''Change the network settings.
        Take 6 strings args:
          Device Name
          Remote Address
          Remote Port
          Subnet Mask
          Default Gateway
          MAC Address
        Use request_network_settings() to see values.
        Use str(b'___', encoding='ascii') to feed parameters'''
        cmd = 51
        try:
```

```
            arg = name +','+ ip +','+ port +','+ mask +','+ gateway +','+ MAC
            command(cmd,arg)
      except Exception:
            return b''

def set_ip(ip):
      cmd = 51
      arg = request_network_settings()
      arg = [ str(x,encoding='ascii') for x in arg ]
      print( 'Old settings: ', arg)
      arg[2]=ip
      print( 'New settings: ', arg)
      arg = arg[1] +','+ arg[2] +','+ arg[3] +','+ arg[4] +','+ arg[5] +','+
arg[6]
      command(cmd,arg)
```

spellmanGui

```
# spellmanGui
# python 3

import tkinter as tk
import tkinter.messagebox as tkm
import threading, queue, time
import spellmanModule as spll
#import alerta_correo as alerta

# CONSTANTES GLOBALES
max_V=5E4 #50.000,00 V
coef_V=max_V/4095
max_I=0.6 #0.6 mA
coef_I=max_I/4095

# VARIABLES

alerta_activa = False

# Variables globales

botones = {} # diccionario de objetos graficos Button. record, play, stop.
labels = {}  # diccionario de objetos graficos Labels, campos a actualizar.
dialogos = {}
actividad = {'connected':False, 'save':False, 'remote':False} # boolean, if False
=> close thread
dataqueue = queue.Queue() # Cola de datos de acceso en exclusión mutua


# Funciones

def init():
	'''Inicializa el programa'''
	# try connection
	actividad['connected']=True
	reading()

def repeater():
	'''Actualizacion periodica'''
	try:
		data = dataqueue.get(block=False)
	except queue.Empty:
		pass
	else:
		# actualizar GUI
		labels[data[0]].config(text=data[1])
	winroot.after(100, repeater) # programar nueva llamada

def reading():
	threading.Thread(target=reading_thread).start()
```

```
def reading_thread():
	high_voltage_anterior = 0
	while actividad['connected']:
		ans = spll.analog()
		if ans != b'':
			voltage = int(ans[3])*coef_V
			current = int(ans[4])*coef_I
			dataqueue.put(('voltage','%05i' % voltage))   #1
			dataqueue.put(('current','%06.5f' % current)) #2
		ans = spll.system()
		if ans != b'':
			high_voltage = str(int(ans[1]))
			dataqueue.put(('hv', high_voltage))           #3
			if (high_voltage_anterior == '1') and (high_voltage == '0'):
				print ('====CAMBIO====',alerta_activa)
				if alerta_activa == True:

	threading.Thread(target=alerta.send_email).start()
			high_voltage_anterior = high_voltage
		ans = spll.status()
		if ans != b'':
			remote = str(int(ans[3]))
			arc = str(int(ans[8]))
			dataqueue.put(('remote',remote))              #4
			dataqueue.put(('arc', arc))                   #5
		ans = spll.request_DAC(0)
		if ans != b'':
			voltage_dac = int(ans[1])*coef_V
			dataqueue.put(('voltage_dac','%5i' %voltage_dac))#6
		ans = spll.request_DAC(1)
		if ans != b'':
			current_dac = int(ans[1])*coef_I
			dataqueue.put(('current_dac','%06.5f' %current_dac))#7
		time.sleep(1)

def remote_on():
	threading.Thread(target=remote_on_thread).start()
def remote_on_thread():
	ans = spll.remote_on()

def remote_off():
	threading.Thread(target=remote_off_thread).start()
def remote_off_thread():
	ans = spll.remote_off()

def hv_on():
	threading.Thread(target=hv_on_thread).start()
def hv_on_thread():
	ans = spll.hv_on()

def hv_off():
```

```
        threading.Thread(target=hv_off_thread).start()
def hv_off_thread():
        ans = spll.hv_off()

def dialog_change_voltage_dac():
        def push_dac():
                valor = dialogo.get()
                change_voltage_dac(valor)
                popup.destroy()
        popup = tk.Toplevel()
        dialogo = tk.Entry(popup)
        dialogo.pack(side='top')
        tk.Button(popup, text='Cancel', command=popup.destroy).pack(side='left')
        tk.Button(popup, text = 'Apply',command=push_dac).pack(side='right')
def change_voltage_dac(valor):
        threading.Thread(target=change_voltage_dac_thread,args=(valor,)).start()
        print ('CHANGING DAC!!!!!!!!!!!')
def change_voltage_dac_thread(valor):
        try:
                valor = int(valor)
        except:
                pass
                print('PASSSSSS')
        else:
                if (valor >= 0) and (valor < (4095*coef_V)):
                        valor = int(valor/coef_V)
                        print('***************',valor)
                        spll.set_DAC(0,valor)

def dialog_change_current_dac():
        def push_dac():
                valor = dialogo.get()
                change_current_dac(valor)
                popup.destroy()
        popup = tk.Toplevel()
        dialogo = tk.Entry(popup)
        dialogo.pack(side='top')
        tk.Button(popup, text='Cancel', command=popup.destroy).pack(side='left')
        tk.Button(popup, text = 'Apply',command=push_dac).pack(side='right')
def change_current_dac(valor):
        threading.Thread(target=change_current_dac_thread,args=(valor,)).start()
        print ('CHANGING DAC!!!!!!!!!!!')
def change_current_dac_thread(valor):
        print('valor de entrada',type(valor))
        try:
                valor = float(valor)
                print(valor)
        except:
                pass
                print('PASSSSSS')
        else:
                if (valor >= 0) and (valor < (4095*coef_I)):
```

```
                    valor = int(valor/coef_I)
                    print('*************** CURRENT',valor)
                    spll.set_DAC(1,valor)

def alerta_correo_on():
       global alerta_activa
       alerta_activa = True
def alerta_correo_off():
       global alerta_activa
       alerta_activa = False

# Definicion de las ventanas

def notdone():
       tkm.showerror('NO implementado', 'NO disponible.\nEn desarrollo')

def makemenu(win):
       menubar = tk.Menu(win)
       win.config(menu=menubar)

       filemenu = tk.Menu(menubar)
       filemenu.add_command(label='Save...', command=notdone, underline=0)
       filemenu.add_command(label='Save as', command=notdone, underline=0)
       filemenu.add_separator()
       filemenu.add_command(label='Quit...', command=win.quit,underline=0)
       menubar.add_cascade(label='File', menu=filemenu,   underline=0)

       editmenu = tk.Menu(menubar)
       editmenu.add_command(label='Preferences', command=notdone)
       menubar.add_cascade(label='Edit', menu=editmenu, underline=0)

       toolmenu = tk.Menu(menubar)
       toolmenu.add_command(label='Alerta correo on', command=alerta_correo_on)
       toolmenu.add_command(label='Alerta correo off', command=alerta_correo_off)
       toolmenu.add_command(label='Remote on', command=remote_on)
       toolmenu.add_command(label='Remote off', command=remote_off)
       menubar.add_cascade(label='Tools', menu=toolmenu, underline=0)

       helpmenu = tk.Menu(menubar)
       helpmenu.add_command(label='About',command=notdone)
       menubar.add_cascade(label='Help', menu=helpmenu, underline=0)

def makehvbar(win, botones, labels):
       marco = tk.Frame(win)
       marco.pack(side='left')
       etiqueta_text = tk.Label(marco, text='    HV   : ',width=10)
       etiqueta = tk.Label(marco, text=' -- ',width=5)
       boton_hv_on = tk.Button(marco, text='ON', command=hv_on, width=3)
       boton_hv_off= tk.Button(marco, text='OFF',command=hv_off, width=3)
       etiqueta_text.pack(side='left')
       etiqueta.pack(side='left')
       boton_hv_on.pack(side='left')
```

```
        boton_hv_off.pack(side='left')
        labels['hv']=etiqueta
        botones['hv_on']=boton_hv_on
        botones['hv_off']=boton_hv_off
        return botones, labels


def makeremotebar(win, botones, labels):
        marco = tk.Frame(win)
        marco.pack(side='left')
        etiqueta_text = tk.Label(marco, text='    REMOTE : ',width=10)
        etiqueta = tk.Label(marco, text=' -- ',width=5)
        boton_remote_on = tk.Button(marco, text='ON', command=remote_on, width=3)
        boton_remote_off= tk.Button(marco, text='OFF',command=remote_off, width=3)
        etiqueta_text.pack(side='left')
        etiqueta.pack(side='left')
        boton_remote_on.pack(side='left')
        boton_remote_off.pack(side='left')
        labels['remote']=etiqueta
        botones['remote_on']=boton_remote_on
        botones['remote_off']=boton_remote_off
        return botones, labels

def maketable(win):
        marco = tk.Frame(win)
        marco.pack(side='left')
        # filas
        marco1 = tk.Frame(marco)
        marco1.pack()
        marco2 = tk.Frame(marco)
        marco2.pack()
        marco3 = tk.Frame(marco)
        marco3.pack()
        voltage_text = tk.Label(marco1, text='Voltage(V) : ', width=14)
        voltage_label = tk.Label(marco1, text='Voltage', width=12)
        current_text = tk.Label(marco2, text='Current(mA): ', width=14)
        current_label = tk.Label(marco2, text='Current', width=12)
        arc_text = tk.Label(marco3, text='Arc : ', width=14)
        arc_label = tk.Label(marco3, text='Arc', width=12)
        voltage_text.pack(side='left')
        voltage_label.pack(side='left')
        current_text.pack(side='left')
        current_label.pack(side='left')
        arc_text.pack(side='left')
        arc_label.pack(side='left')
        diccionario_labels={}
        diccionario_labels['voltage']=voltage_label
        diccionario_labels['current']=current_label
        diccionario_labels['arc']=arc_label
        return diccionario_labels

def makedac(win, botones, labels):
```

```
        marco = tk.Frame(win)
        marco.pack(side='bottom')
        marco1 = tk.Frame(marco)
        marco1.pack()
        marco2 = tk.Frame(marco)
        marco2.pack()
        voltage_dac_text= tk.Label(marco1, text='Voltage DAC(V) : ', width=14)
        voltage_dac = tk.Label(marco1, text='----', width=8)
        voltage_dac_text.pack(side='left')
        voltage_dac.pack(side='left')
        boton_voltage_dac = tk.Button(marco1, text='CHANGE',
                                      command=dialog_change_voltage_dac, width=4)
        boton_voltage_dac.pack(side='left')
        current_dac_text= tk.Label(marco2, text='Current DAC(mA): ', width=14)
        current_dac = tk.Label(marco2, text='----', width=8)
        current_dac_text.pack(side='left')
        current_dac.pack(side='left')
        boton_current_dac = tk.Button(marco2, text='CHANGE',
                                      command=dialog_change_current_dac, width=4)
        boton_current_dac.pack(side='left')
        labels['voltage_dac']=voltage_dac
        labels['current_dac']=current_dac
        botones['voltage_dac']=boton_voltage_dac
        botones['current_dac']=boton_current_dac
        return botones, labels



#   INICIO PROGRAMA

if __name__=='__main__':
        winroot = tk.Tk()
        winroot.title('Spellman')
        winroot.after(100,repeater) # actualizacion de la GUI
        makemenu(winroot)
        topmarco=tk.Frame()
        topmarco.pack()
        botones, labels = makehvbar(topmarco, botones, labels)
        botones, labels = makeremotebar(topmarco, botones, labels)
        labels.update(maketable(winroot))
        botones, labels = makedac(winroot, botones, labels)
        # lanzamiento del sistema de eventos
        init()
        winroot.mainloop()
        # cierre de hilos
        actividad['connected']=False
        actividad['record']=False
```

Conjunto CAEN

En este equipo se usa una biblioteca C a la que accedemos usando ctypes.

caenModule. Módulo con instrucciones que llaman a funciones C usando ctypes para comunicarse con el equipo CAEN.

caenGui. Interfaz gráfica para operar con el eqeuipo CAEN. Figura 6.

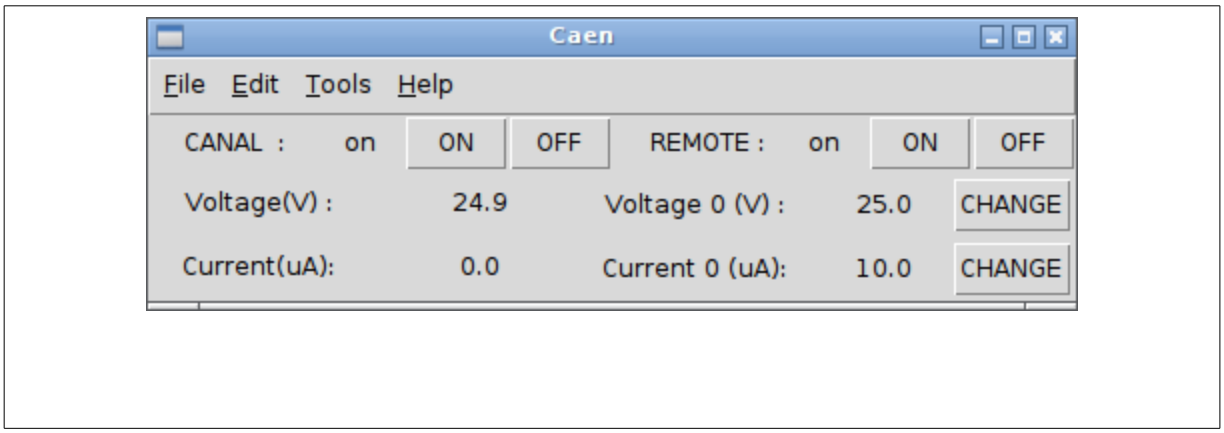


Figura 6. Interfaz gráfica caenGui

caenModule

```
# caenModule
# python 3
# Modulo con instrucciones para CAEN

import ctypes

path = './x64/libcaenhvwrapper.so.5.20' # lugar de la biblioteca
path = './libcaenhvwrapper.so.5.20' # lugar de la biblioteca
libCaen = ctypes.cdll.LoadLibrary(path)

# variables
slot = 3

# Establecimiento de la conexion
# Devuelve un entero que recoge el estado de la conexion
# int CAENHVInitSystem(
#                   const char   *SystemName, //in
#                   int                LinkType,          //in
#                   char         *Arg,              //in
#                   const char   *UserName,         //in
#                   const char   *Password          //in
#                   );

_initSystem = libCaen.CAENHVInitSystem
_initSystem.argtypes = (
                          ctypes.POINTER(ctypes.c_char),
                          ctypes.c_int,
                          ctypes.POINTER(ctypes.c_char),
                          ctypes.POINTER(ctypes.c_char),
                          ctypes.POINTER(ctypes.c_char)
                          )
_initSystem.restype = (ctypes.c_int)

def initSystem(     systemName = b'System2',
                          linkType = 0,
                          adress = b'192.168.0.132',
                          userName = b'admin',
                          password = b'admin'):
       return _initSystem(systemName, linkType, adress, userName, password)

# DeinitSystem
_deinitSystem = libCaen.CAENHVDeinitSystem
_deinitSystem.argtypes = (
                          ctypes.POINTER(ctypes.c_char),
                          )
_deinitSystem.restype = (ctypes.c_int)

def deinitSystem(   systemName = b'System2'):
       return _deinitSystem(systemName)
```

```
# int CAENHVSetChParam(
#                   const char          *SystemName, //in
#                   unsigned short      slot,               //in
#                   const char          *Parname,           //in
#                   unsigned short      ChNum,              //in
#                   cnst unsg short     *ChList,            //in
#                   const char          *ParValue           //in
def SetChParam(     systemName,
                        slot,
                        parName,
                        chNum,
                        chList,
                        parValue):
       # definicion de los tipos de entrada
       _SetChParam= libCaen.CAENHVSetChParam
       _SetChParam.argtypes = (
                        ctypes.POINTER(ctypes.c_char),
                        ctypes.c_ushort,
                        ctypes.POINTER(ctypes.c_char),
                        ctypes.c_ushort,
                        ctypes.POINTER(ctypes.c_ushort),
                        ctypes.POINTER(ctypes.c_float)
                        )
       _SetChParam.restype = (ctypes.c_int)
       # conversion de los parametros de entrada
       systemName = ctypes.c_char_p(systemName)
       slot = ctypes.c_ushort(slot)
       parName = ctypes.c_char_p(parName)
       chNum = ctypes.c_ushort(chNum)
       chList = (ctypes.c_ushort*len(chList))(*chList)
       #chList = (ctypes.c_ushort*len(chList))(*chList)
       #chList = ctypes.c_ushort(4)
       #print(chList)
       parValue = ctypes.c_float(parValue)
       return _SetChParam(systemName, slot, parName, chNum, chList, parValue)

# int CAENHVGetChParam(
#                   const char          *SystemName, //in
#                   unsigned short      slot,               //in
#                   const char          *Parname,           //in
#                   unsigned short      ChNum,              //in
#                   cnst unsg short     *ChList,            //in
#                   const char          *ParValList         //out

def GetChParam(     systemName,
                        slot,
                        parName,
                        chNum,
                        chList):
       _GetChParam= libCaen.CAENHVGetChParam
       _GetChParam.argtypes = (
                        ctypes.POINTER(ctypes.c_char),
```

```
                        ctypes.c_ushort,
                        ctypes.POINTER(ctypes.c_char),
                        ctypes.c_ushort,
                        ctypes.POINTER(ctypes.c_ushort),
                        ctypes.POINTER(ctypes.c_float)
                        )
        _GetChParam.restype = (ctypes.c_int)
        systemName = ctypes.c_char_p(systemName)
        slot = ctypes.c_ushort(slot)
        parName = ctypes.c_char_p(parName)
        chNum = ctypes.c_ushort(chNum)
        chList = (ctypes.c_ushort*len(chList))(*chList)
        parValue = (ctypes.c_float*chNum.value)()
        return(      _GetChParam(systemName, slot, parName, chNum, chList,
parValue),
                    [parValue[i] for i in range(chNum.value)] )

# int CAENHVSetChParam(
#                   const char          *SystemName, //in
#                   unsigned short      slot,               //in
#                   const char          *Parname,           //in
#                   unsigned short      ChNum,              //in
#                   cnst u short *ChList,            //in
#                   const int           *ParValue           //in #boolean
def SetChBool(      systemName,
                        slot,
                        parName,
                        chNum,
                        chList,
                        parValue):
        _SetChBool= libCaen.CAENHVSetChParam
        _SetChBool.argtypes = (
                        ctypes.POINTER(ctypes.c_char),
                        ctypes.c_ushort,
                        ctypes.POINTER(ctypes.c_char),
                        ctypes.c_ushort,
                        ctypes.POINTER(ctypes.c_ushort),
                        ctypes.POINTER(ctypes.c_int)
                        )
        _SetChBool.restype = (ctypes.c_int)
        systemName = ctypes.c_char_p(systemName)
        slot = ctypes.c_ushort(slot)
        parName = ctypes.c_char_p(parName)
        chNum = ctypes.c_ushort(chNum)
        chList = (ctypes.c_ushort*len(chList))(*chList)
        parValue = ctypes.c_int(parValue)
        return _SetChBool(systemName, slot, parName, chNum, chList, parValue)

# Funciones especificas
def init():
        '''Conexion a la CAEN
        0x0 no hay error'''
```

```
        return initSystem()

def out():
        '''Desconexion'''
        return deinitSystem()

def setV(n,x):
        '''Fija el valor de voltaje del canal n en x
                chNum es el numero de canales pasados en chList'''
        return SetChParam(b'System2', slot, b'V0Set', 1, [n], x)

def setI(n,x):
        '''Fija el valor de la intensidad del canal n en x'''
        return SetChParam(b'System2', slot, b'I0Set', 1, [n], x)

def getV(n):
        return GetChParam(b'System2', slot, b'V0Set', 1, [n] )

def getI(n):
        return GetChParam(b'System2', slot, b'I0Set', 1, [n] )

def getVMon(n):
        return GetChParam(b'System2', slot, b'VMon', 1, [n] )

def getIMon(n):
        return GetChParam(b'System2', slot, b'IMon', 1, [n] )

def on(n):
        return SetChBool(b'System2', slot, b'Pw', 1, [n], 1 )

def off(n):
        return SetChBool(b'System2', slot, b'Pw', 1, [n], 0 )
```

caenGui

```
# caenGui
# python 3
# interfaz grafica para CAEN
import tkinter as tk
import tkinter.messagebox as tkm
import threading, queue, time
import caenModule as caen

# CONSTANTES GLOBALES

# VARIABLES

# Variables globales

botones = {} # diccionario de objetos graficos Button. record, play, stop.
labels = {}  # diccionario de objetos graficos Labels, campos a actualizar.
dialogos = {}
actividad = {'run':True, 'connected':False, 'save':False, 'remote':False} #
boolean, if False => close thread
dataqueue = queue.Queue() # Cola de datos de acceso en exclusión mutua

caen.slot =3
canal = 5


# Funciones

def repeater():
	'''Actualizacion periodica'''
	try:
		data = dataqueue.get(block=False)
	except queue.Empty:
		pass
	else:
		# actualizar GUI
		labels[data[0]].config(text=data[1])
	winroot.after(100, repeater) # programar nueva llamada

def reading():
	threading.Thread(target=reading_thread).start()
def reading_thread():
	while actividad['run']:
		if actividad['connected']:
			try:
				ans = caen.getV(canal)
				dataqueue.put(('voltage0','%.1f' % (ans[1][0])))
				time.sleep(0.5)
				ans = caen.getI(canal)
				dataqueue.put(('current0','%.1f' % (ans[1][0])))
				time.sleep(0.5)
```

```
                        ans = caen.getVMon(canal)
                        dataqueue.put(('voltage','%.1f' % (ans[1][0])))
                        time.sleep(0.5)
                        ans = caen.getIMon(canal)
                        dataqueue.put(('current','%.1f' % (ans[1][0])))
                        time.sleep(0.5)
                  except Exception:
                        pass
            else:
                  time.sleep(0.5)

def remote_on():
#     threading.Thread(target=remote_on_thread).start()
#def remote_on_thread():
      try:
            ans = caen.init()
            if ans == 0:
                  dataqueue.put(('remote', 'on'))
                  actividad['connected']=True
            else:
                  dataqueue.put(('remote', 'err'))
      except Exception:
            pass

def remote_off():
#     threading.Thread(target=remote_off_thread).start()
#def remote_off_thread():
      try:
            ans = caen.out()
            if ans == 0:
                  dataqueue.put(('remote', 'off'))
                  actividad['connected']=False
            else:
                  dataqueue.put(('remote', 'err'))
      except Exception:
            pass

def canal_on():
#     threading.Thread(target=canal_on_thread).start()
#def canal_on_thread():
      try:
            ans = caen.on(canal)
            if ans == 0:
                  dataqueue.put(('canal', 'on'))
            else:
                  pass
      except Exception:
            pass


def canal_off():
#     threading.Thread(target=canal_off_thread).start()
```

```
#def canal_off_thread():
	try:
		ans = caen.off(canal)
		if ans == 0:
			dataqueue.put(('canal', 'off'))
		else:
			pass
	except Exception:
		pass

def dialog_change_voltage():
	def push():
		valor = dialogo.get()
		change_voltage(valor)
		popup.destroy()
	popup = tk.Toplevel()
	dialogo = tk.Entry(popup)
	dialogo.pack(side='top')
	tk.Button(popup, text='Cancel', command=popup.destroy).pack(side='left')
	tk.Button(popup, text = 'Apply',command=push).pack(side='right')
def change_voltage(valor):
	threading.Thread(target=change_voltage_thread,args=(valor,)).start()
def change_voltage_thread(valor):
	try:
		valor = int(valor)
	except:
		pass
	else:
		if ((valor >= 0) and (valor < 10000)):
			try:
				caen.setV(canal,valor)
			except Exception:
				pass

def dialog_change_current():
	def push():
		valor = dialogo.get()
		change_current(valor)
		popup.destroy()
	popup = tk.Toplevel()
	dialogo = tk.Entry(popup)
	dialogo.pack(side='top')
	tk.Button(popup, text='Cancel', command=popup.destroy).pack(side='left')
	tk.Button(popup, text = 'Apply',command=push).pack(side='right')
def change_current(valor):
	threading.Thread(target=change_current_thread,args=(valor,)).start()
def change_current_thread(valor):
	try:
		valor = float(valor)
	except:
		pass
	else:
```

```
            if ((valor >= 0) and (valor < 10000)):
                  try:
                        caen.setI(canal,valor)
                  except Exception:
                        pass


# Definicion de las ventanas

def notdone():
      tkm.showerror('NO implementado', 'NO disponible.\nEn desarrollo')

def makemenu(win):
      menubar = tk.Menu(win)
      win.config(menu=menubar)

      filemenu = tk.Menu(menubar)
      filemenu.add_command(label='Save...', command=notdone, underline=0)
      filemenu.add_command(label='Save as', command=notdone, underline=0)
      filemenu.add_separator()
      filemenu.add_command(label='Quit...', command=win.quit,underline=0)
      menubar.add_cascade(label='File', menu=filemenu,   underline=0)

      editmenu = tk.Menu(menubar)
      editmenu.add_command(label='Preferences', command=notdone)
      menubar.add_cascade(label='Edit', menu=editmenu, underline=0)

      toolmenu = tk.Menu(menubar)
      toolmenu.add_command(label='Remote on', command=remote_on)
      toolmenu.add_command(label='Remote off', command=remote_off)
      menubar.add_cascade(label='Tools', menu=toolmenu, underline=0)

      helpmenu = tk.Menu(menubar)
      helpmenu.add_command(label='About',command=notdone)
      menubar.add_cascade(label='Help', menu=helpmenu, underline=0)

def makecanalbar(win, botones, labels):
      marco = tk.Frame(win)
      marco.pack(side='left')
      etiqueta_text = tk.Label(marco, text='  CANAL  : ',width=10)
      etiqueta = tk.Label(marco, text=' -- ',width=5)
      boton_canal_on = tk.Button(marco, text='ON', command=canal_on, width=3)
      boton_canal_off= tk.Button(marco, text='OFF',command=canal_off, width=3)
      etiqueta_text.pack(side='left')
      etiqueta.pack(side='left')
      boton_canal_on.pack(side='left')
      boton_canal_off.pack(side='left')
      labels['canal']=etiqueta
      botones['canal_on']=boton_canal_on
      botones['canal_off']=boton_canal_off
      return botones, labels
```

```
def makeremotebar(win, botones, labels):
	marco = tk.Frame(win)
	marco.pack(side='left')
	etiqueta_text = tk.Label(marco, text='    REMOTE : ',width=10)
	etiqueta = tk.Label(marco, text=' -- ',width=5)
	boton_remote_on = tk.Button(marco, text='ON', command=remote_on, width=3)
	boton_remote_off= tk.Button(marco, text='OFF',command=remote_off, width=3)
	etiqueta_text.pack(side='left')
	etiqueta.pack(side='left')
	boton_remote_on.pack(side='left')
	boton_remote_off.pack(side='left')
	labels['remote']=etiqueta
	botones['remote_on']=boton_remote_on
	botones['remote_off']=boton_remote_off
	return botones, labels

def maketable(win):
	marco = tk.Frame(win)
	marco.pack(side='left')
	# filas
	marco1 = tk.Frame(marco)
	marco1.pack()
	marco2 = tk.Frame(marco)
	marco2.pack()
	marco3 = tk.Frame(marco)
	marco3.pack()
	voltage_text = tk.Label(marco1, text='Voltage(V) : ', width=14, height=2)
	voltage_label = tk.Label(marco1, text='---', width=12)
	current_text = tk.Label(marco2, text='Current(uA): ', width=14, height=2)
	current_label = tk.Label(marco2, text='---', width=12)
	voltage_text.pack(side='left')
	voltage_label.pack(side='left')
	current_text.pack(side='left')
	current_label.pack(side='left')
	diccionario_labels={}
	diccionario_labels['voltage']=voltage_label
	diccionario_labels['current']=current_label
	return diccionario_labels

def makedac(win, botones, labels):
	marco = tk.Frame(win)
	marco.pack(side='bottom')
	marco1 = tk.Frame(marco)
	marco1.pack()
	marco2 = tk.Frame(marco)
	marco2.pack()
	voltage0_text= tk.Label(marco1, text='Voltage 0 (V) : ', width=14,
height=2)
	voltage0 = tk.Label(marco1, text='----', width=8)
	voltage0_text.pack(side='left')
	voltage0.pack(side='left')
```

```
        boton_voltage0 = tk.Button(marco1, text='CHANGE',
                                   command=dialog_change_voltage, width=4)
        boton_voltage0.pack(side='left')
        current0_text= tk.Label(marco2, text='Current 0 (uA): ', width=14,
height=2)
        current0 = tk.Label(marco2, text='----', width=8)
        current0_text.pack(side='left')
        current0.pack(side='left')
        boton_current0 = tk.Button(marco2, text='CHANGE',
                                   command=dialog_change_current, width=4)
        boton_current0.pack(side='left')
        labels['voltage0']=voltage0
        labels['current0']=current0
        botones['voltage0']=boton_voltage0
        botones['current0']=boton_current0
        return botones, labels



#   INICIO PROGRAMA

if __name__=='__main__':
        winroot = tk.Tk()
        winroot.title('Caen')
        winroot.after(100,repeater) # actualizacion de la GUI
        makemenu(winroot)
        topmarco=tk.Frame()
        topmarco.pack()
        botones, labels = makecanalbar(topmarco, botones, labels)
        botones, labels = makeremotebar(topmarco, botones, labels)
        labels.update(maketable(winroot))
        botones, labels = makedac(winroot, botones, labels)
        reading()
        # lanzamiento del sistema de eventos
        winroot.mainloop()
        # cierre de hilos
        actividad['connected']=False
        actividad['record']=False
        actividad['run'] = False
```